Introduction to R

DS Part IV – Python Hands on

Agenda

- Introduction to R
- R and Rstudio set up
- Basic data types
- Data Structures in R
- Variables and Control structures
- Functions
- Modules and Packages

Sasken training

What is R ?

- R is a programming language for statistical Computing and graphics
- R is free and open source , allowing anyone to use it and, importantly, to modify it.
- It was originated from S language which was developed by John Chambers in the Bell laboratories in the 80's
- R is cross platform, it can run on many operating systems and different hardwares
- R is extensible and offers rich functionality for developers to build their own tools and methods for analyzing data

Advantages :

- It provides vast number of packages
- R has over 4800 packages available from multiple repositories specializing in topics like econometrics, data mining, spatial analysis, and bio-informatics.
- R has active user groups where questions can be asked and are often quickly responded to

Disadvantages :

- Its main disadvantage is it's not scalable

Rstudio

- RStudio is an open-source Integrated Development Environment (code editor) for R
- Rstudio makes R easier to use, it includes code editor, debugging & visualization tools
- It has easy access to R help

xx

- Provides completion facility
- Rstudio offers many handy keyboard features

https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts

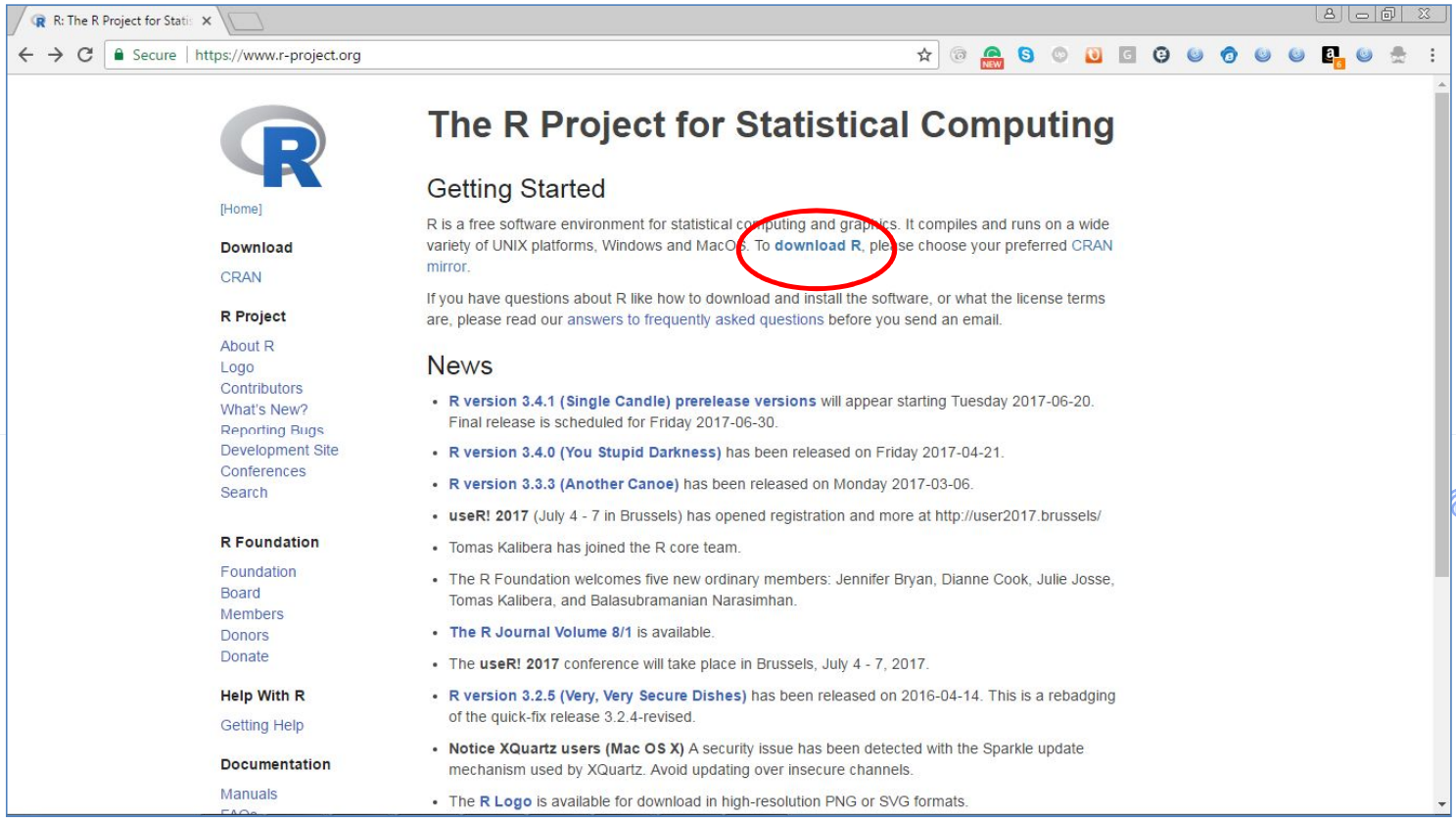
R & Rstudio Installation

- Download & Install R from this below URL
  https://www.r-project.org/

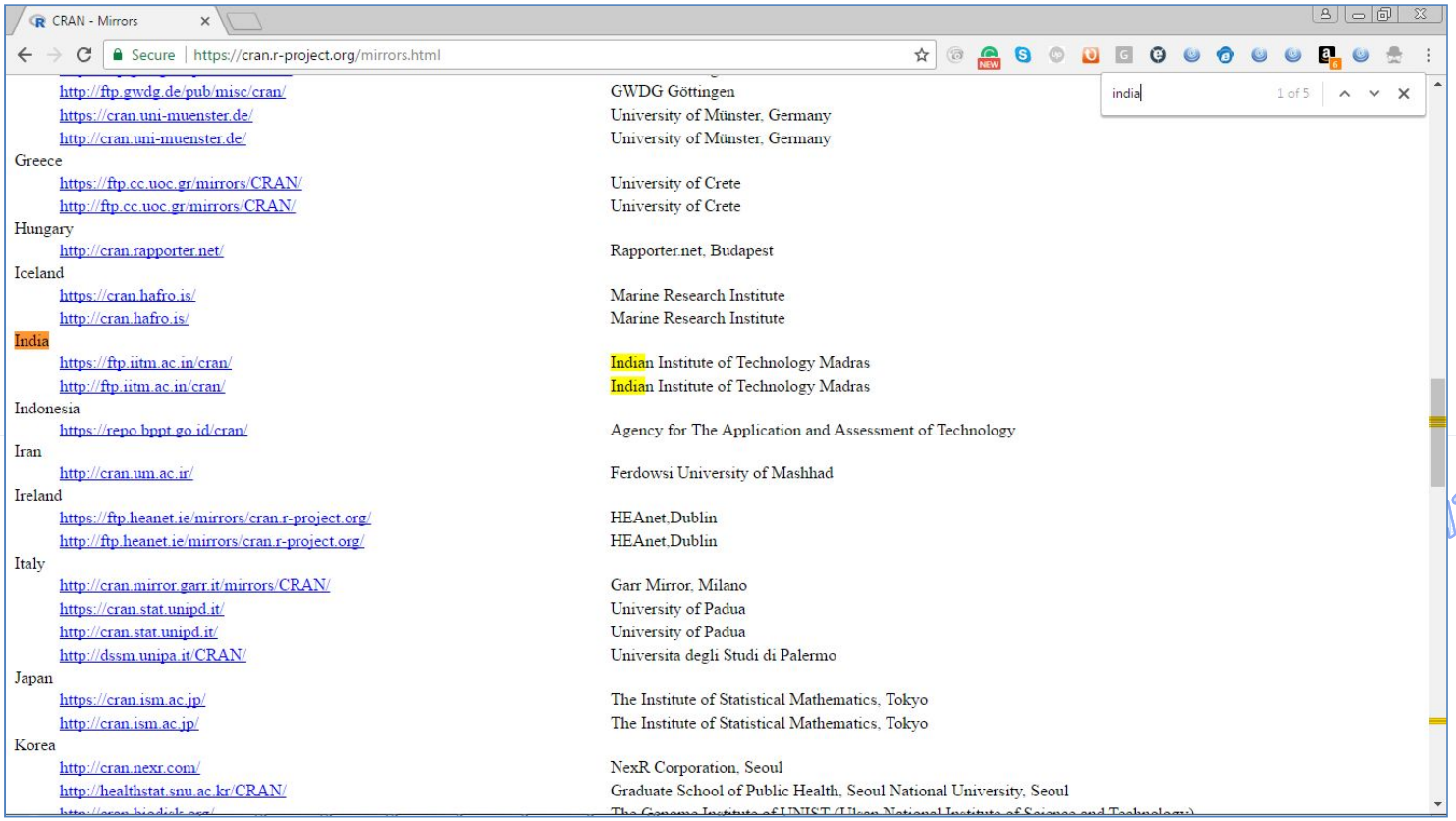
- Download & Install Rstudio from this below URL
  https://www.rstudio.com/products/rstudio/download/

Sasken training, a

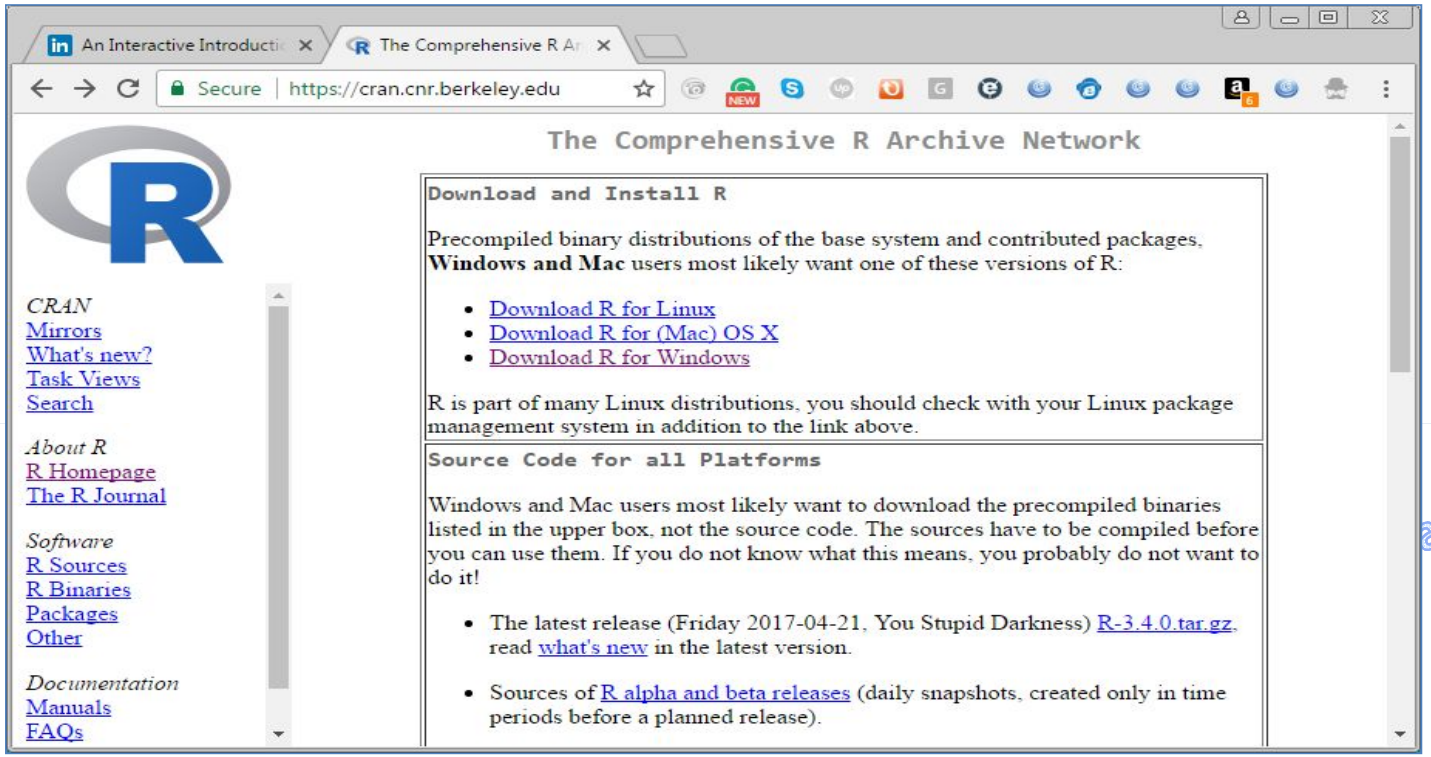
R software download site



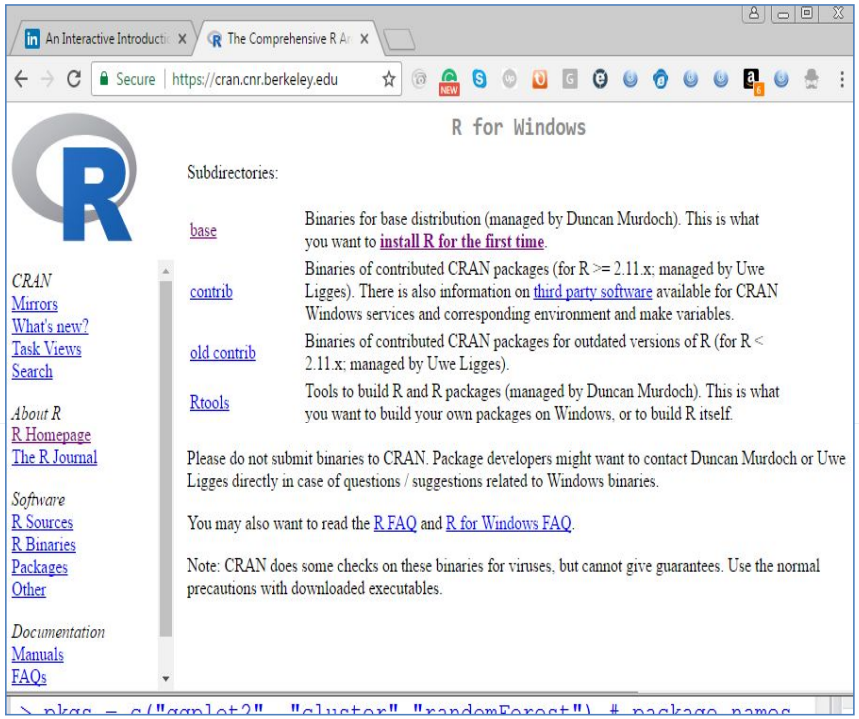
R Software CRAN mirror

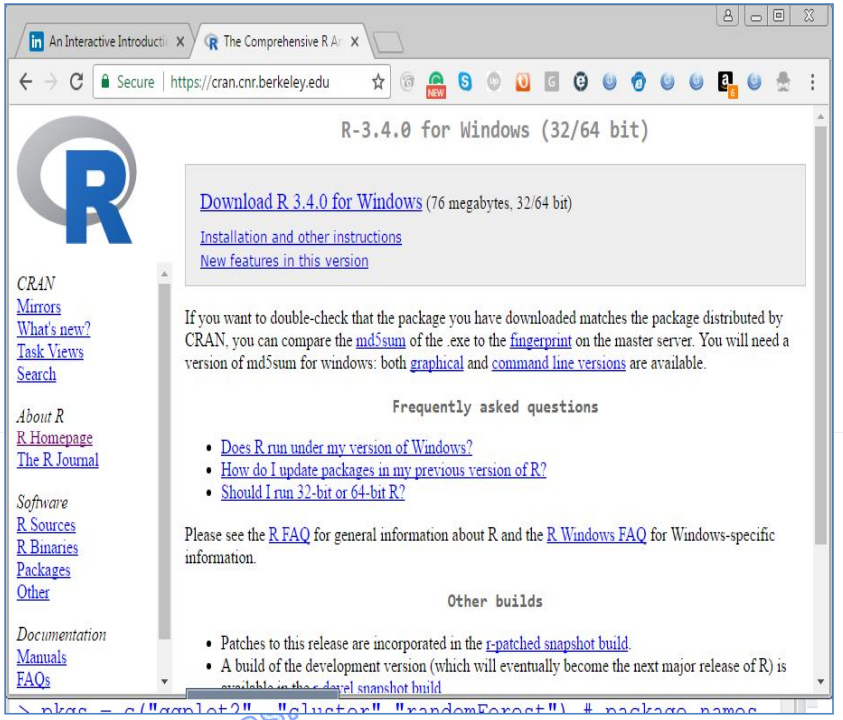


http://ftp.gwdg.de/pub/misc/cran/	GWDG Göttingen
https://cran.uni-muenster.de/	University of Münster, Germany
http://cran.uni-muenster.de/	University of Münster, Germany
Greece	
https://ftp.cc.uoc.gr/mirrors/CRAN/	University of Crete
http://ftp.cc.uoc.gr/mirrors/CRAN/	University of Crete
Hungary	
http://cran.rapporter.net/	Rapporter.net, Budapest
Iceland	
https://cran.hafro.is/	Marine Research Institute
http://cran.hafro.is/	Marine Research Institute
India	
https://ftp.iitm.ac.in/cran/	Indian Institute of Technology Madras
http://ftp.iitm.ac.in/cran/	Indian Institute of Technology Madras
Indonesia	
https://repo.bppt.go.id/cran/	Agency for The Application and Assessment of Technology
Iran	
http://cran.um.ac.ir/	Ferdowsi University of Mashhad
Ireland	
https://ftp.heanet.ie/mirrors/cran.r-project.org/	HEAnet,Dublin
http://ftp.heanet.ie/mirrors/cran.r-project.org/	HEAnet,Dublin
Italy	
http://cran.mirror.garr.it/mirrors/CRAN/	Garr Mirror, Milano
https://cran.stat.unipd.it/	University of Padua
http://cran.stat.unipd.it/	University of Padua
http://dssm.unipa.it/CRAN/	Universita degli Studi di Palermo
Japan	
https://cran.ism.ac.jp/	The Institute of Statistical Mathematics, Tokyo
http://cran.ism.ac.jp/	The Institute of Statistical Mathematics, Tokyo
Korea	
http://cran.nexr.com/	NexR Corporation, Seoul
http://healthstat.snu.ac.kr/CRAN/	Graduate School of Public Health, Seoul National University, Seoul

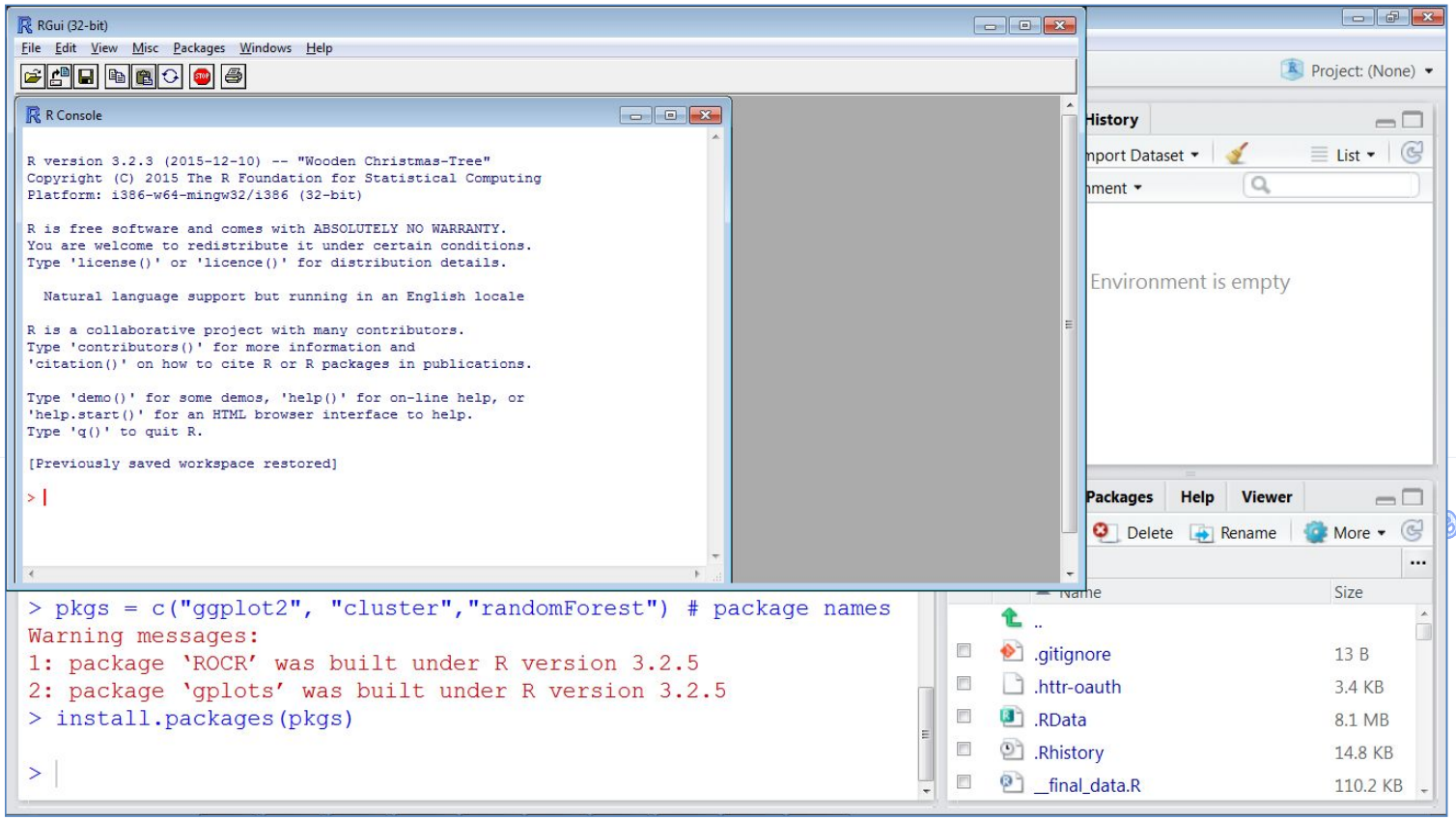

An Interactive Introducti
The Comprehensive R Ar
Secure | https://cran.cnr.berkeley.edu
The Comprehensive R Archive Network
Download and Install R
Precompiled binary distributions of the base system and contributed packages, Windows and Mac users most likely want one of these versions of R:
Download R for Linux
Download R for (Mac) OS X
Download R for Windows
R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.
Source Code for all Platforms
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!
The latest release (Friday 2017-04-21, You Stupid Darkness) R-3.4.0.tar.gz, read what's new in the latest version.
Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
CRAN
Mirrors
What's new?
Task Views
Search
About R
R Homepage
The R Journal
Software
R Sources
R Binaries
Packages
Other
Documentation
Manuals
FAQs

adyar

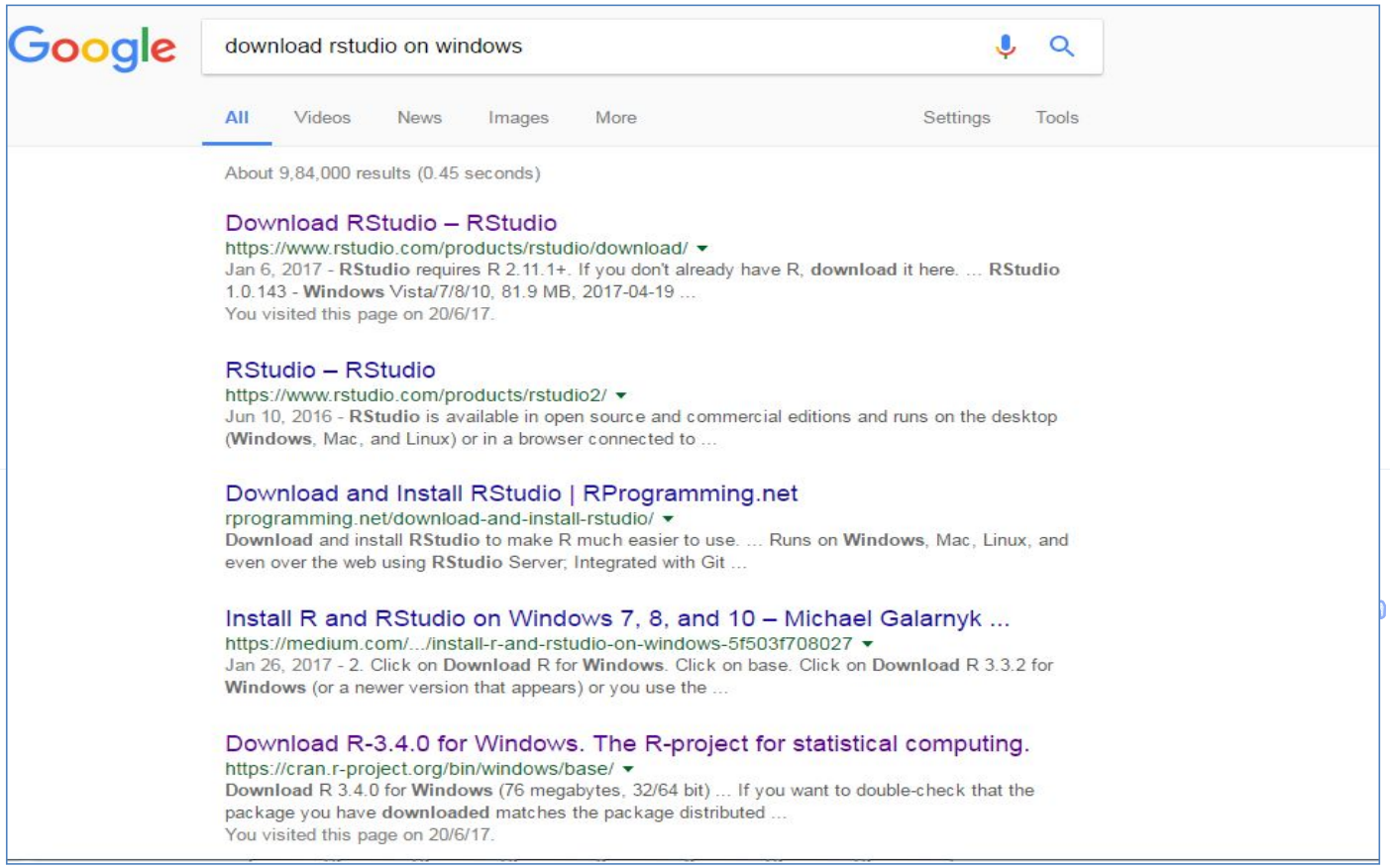

An Interactive Introducti
The Comprehensive R Ar
Secure
https://cran.cnr.berkeley.edu
R for Windows
Subdirectories:
base
Binaries for base distribution (managed by Duncan Murdoch). This is what you want to install R for the first time.
contrib
Binaries of contributed CRAN packages (for R >= 2.11.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib
Binaries of contributed CRAN packages for outdated versions of R (for R < 2.11.x; managed by Uwe Ligges).
Rtools
Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.
Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch or Uwe Ligges directly in case of questions / suggestions related to Windows binaries.
You may also want to read the R FAQ and R for Windows FAQ.
Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.
CRAN
Mirrors
What's new?
Task Views
Search
About R
R Homepage
The R Journal
Software
R Sources
R Binaries
Packages
Other
Documentation
Manuals
FAQs


An Interactive Introducti
The Comprehensive R Ar
Secure
https://cran.cnr.berkeley.edu
R-3.4.0 for Windows (32/64 bit)
Download R 3.4.0 for Windows (76 megabytes, 32/64 bit)
Installation and other instructions
New features in this version
If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.
Frequently asked questions
Does R run under my version of Windows?
How do I update packages in my previous version of R?
Should I run 32-bit or 64-bit R?
Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.
Other builds
Patches to this release are incorporated in the r-patched snapshot build.
A build of the development version (which will eventually become the next major release of R) is
CRAN
Mirrors
What's new?
Task Views
Search
About R
R Homepage
The R Journal
Software
R Sources
R Binaries
Packages
Other
Documentation
Manuals
FAQs

R Console

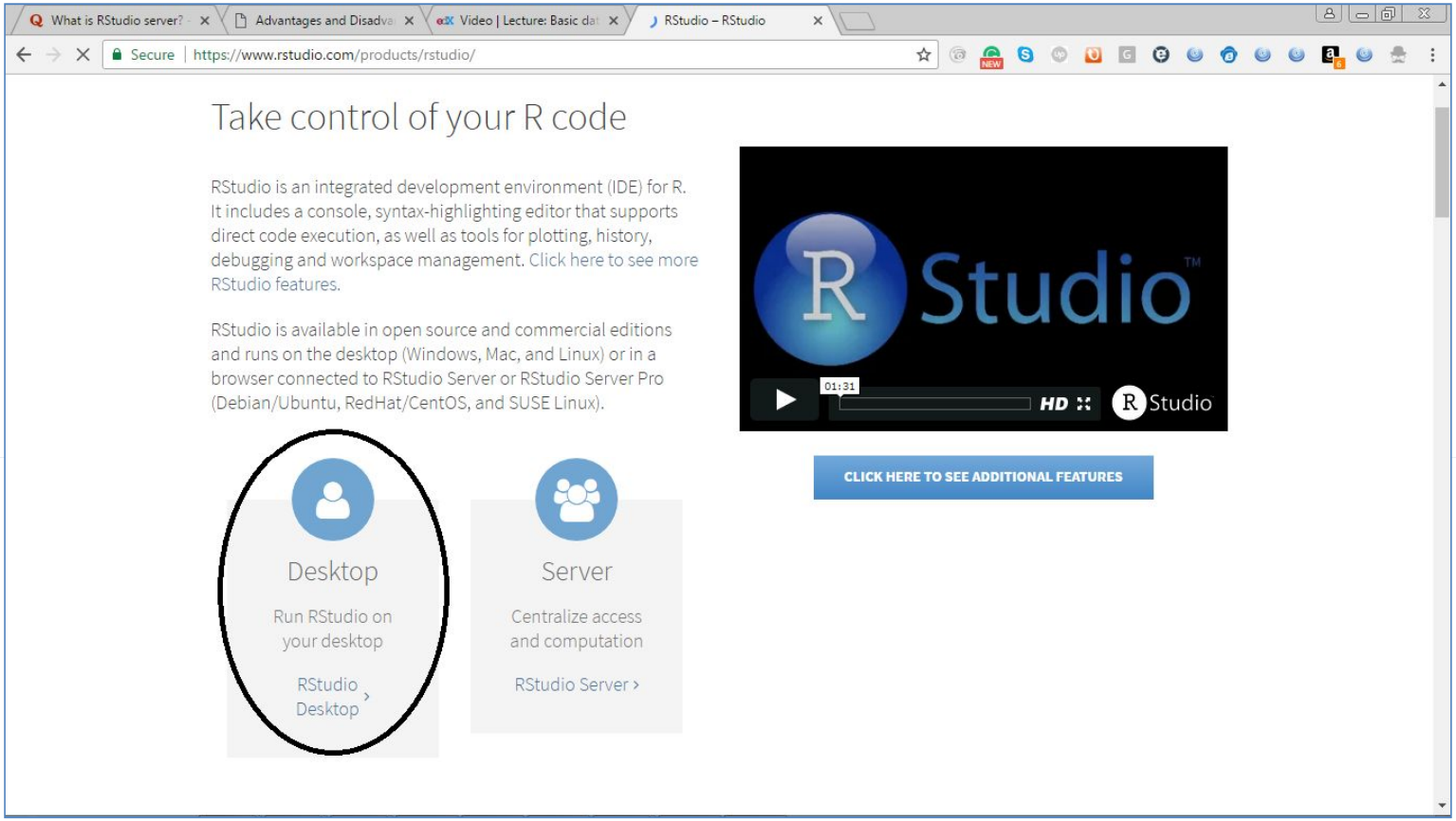

RGui (32-bit)
File Edit View Misc Packages Windows Help
R Console
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to 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]
> pkgs = c("ggplot2", "cluster","randomForest") # package names
Warning messages:
1: package 'ROCR' was built under R version 3.2.5
2: package 'gplots' was built under R version 3.2.5
> install.packages(pkgs)
Project: (None)
History
Import Dataset
List
Environment is empty
Packages
Help
Viewer
Delete
Rename
More
Name
Size
.gitignore 13 B
.httr-oauth 3.4 KB
.RData 8.1 MB
.Rhistory 14.8 KB
__final_data.R 110.2 KB

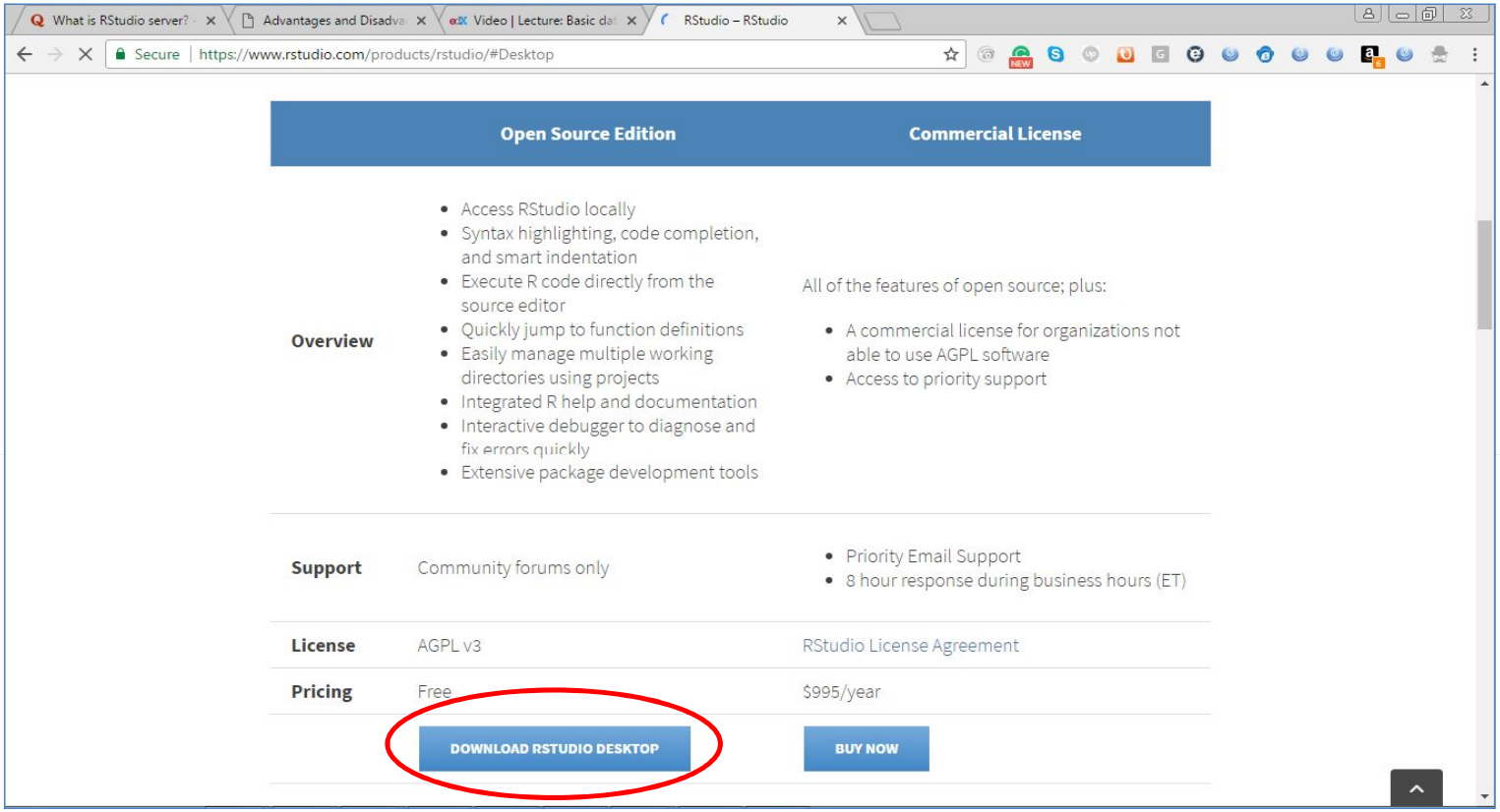
Rstudio Download



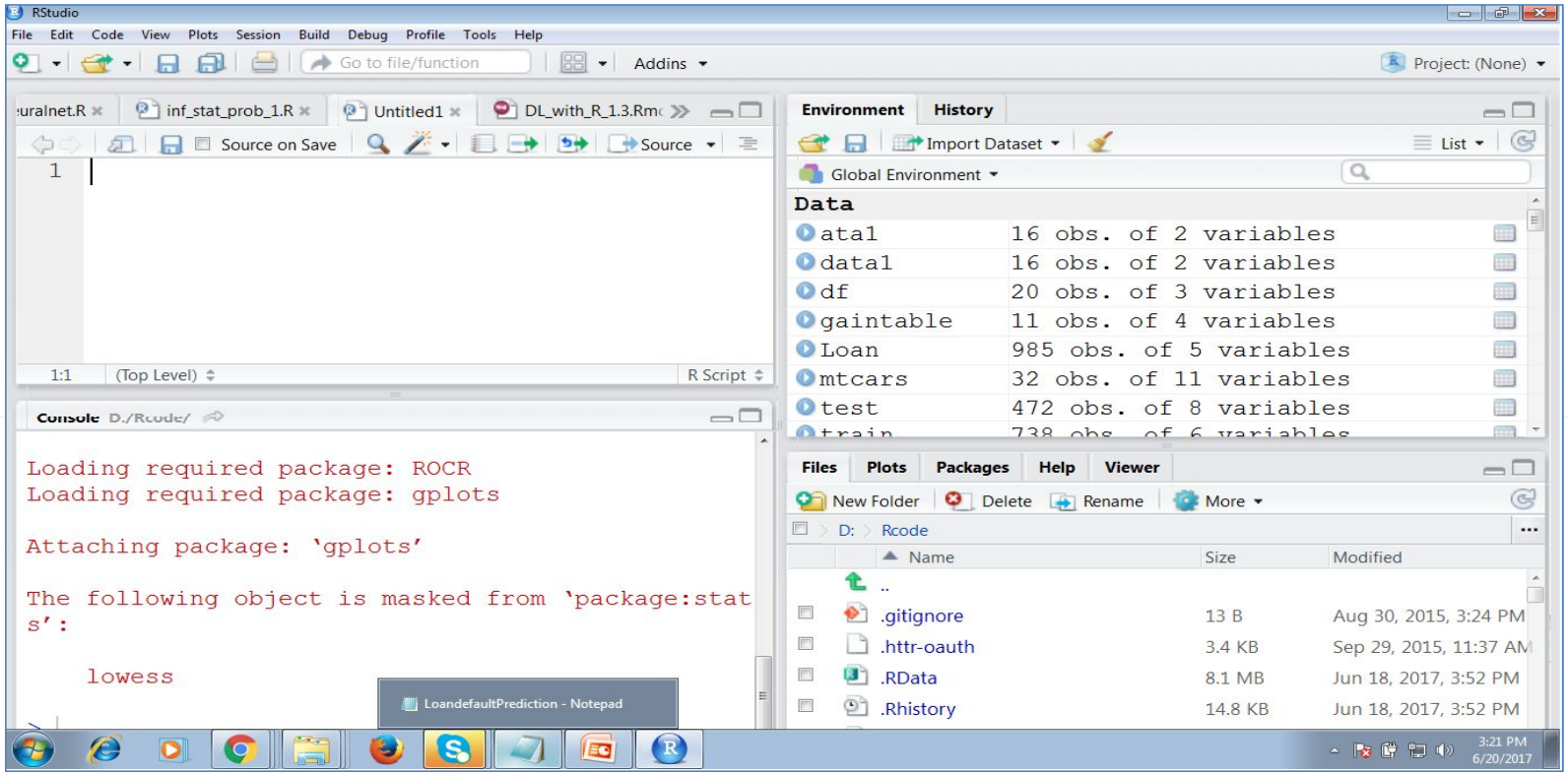
adyar

Rstudio Download page




What is RStudio server?
Advantages and Disadva
Video | Lecture: Basic da
RStudio – RStudio
Secure | https://www.rstudio.com/products/rstudio/#Desktop
Open Source Edition
Commercial License
Overview
Access RStudio locally
Syntax highlighting, code completion, and smart indentation
Execute R code directly from the source editor
Quickly jump to function definitions
Easily manage multiple working directories using projects
Integrated R help and documentation
Interactive debugger to diagnose and fix errors quickly
Extensive package development tools
All of the features of open source; plus:
A commercial license for organizations not able to use AGPL software
Access to priority support
Support
Community forums only
Priority Email Support
8 hour response during business hours (ET)
License
AGPL v3
RStudio License Agreement
Pricing
Free
$995/year
DOWNLOAD RSTUDIO DESKTOP
BUY NOW

RStudio - Window pane Layout



Installing packages in R

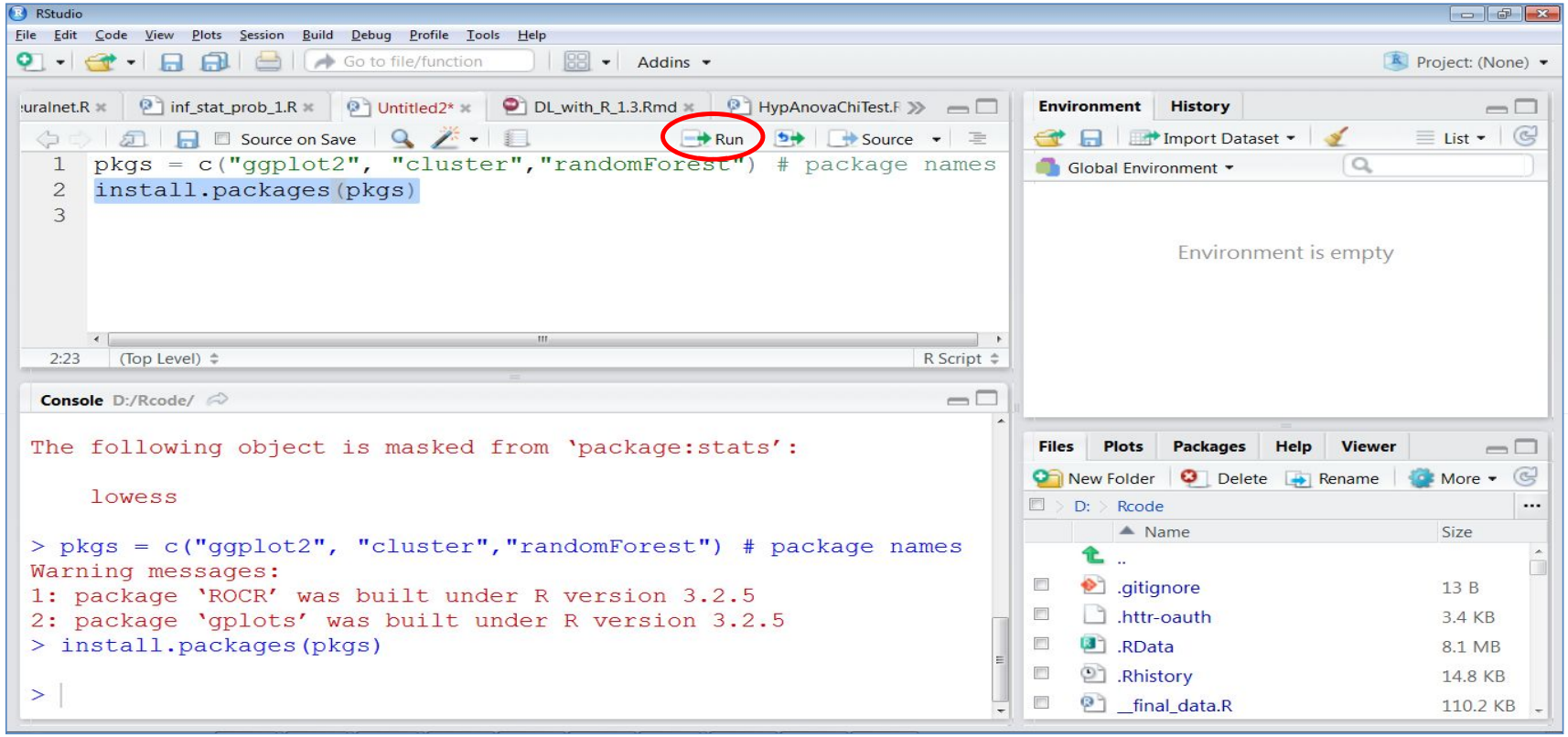
- What is a package ?
  - Packages are collections of R functions, data , and complied code in a well-defined format.
  - The directory where packages are stored is called the library
  - R comes with a standard set of packages , others are available for download and installation.
- How to install packages in R?

```
pkgs = c("ggplot2", "cluster", "randomForest") # package names
install.packages(pkgs)
```

- How to update R packages ?
  - update.packages()

Sasken train

Installing packages in RStudio



Getting Started with R

DS Part IV – Python Hands on

Sasken Training, Adyar , 9840014739 Chennai – 600 020

Working Directory

- Is the directory where all input to and output from R are stored

Function getwd() is used to get the current working directory.

- Usage: Type getwd() at the command line

- Function setwd() is used to assign a desired path as working directory
  - Usage: Type setwd({Path}) at the command line.
  - Example (Mac): setwd("D:\workingSet")
    Example (Windows): setwd("C:/Program Files/R")
- In R Studio, access Tools > Set Working Directory from the menu –

Three options available.

Select "Choose Directory" to assign a desired path

What are packages ?

- Collections of R functions, data , and compiled code in a well-defined format
- The directory where packages are stored on your computer is called the library
- To view the installed packages ?
  library()
- How to install packages or install a package with its dependencies ?
  install.packages("MASS", dependencies = TRUE)
- How to load the package ?
  - library(MASS)
- How to update the package ?
  - update.packages()

Getting help in R

- Just type ? Prior to the command

? as.integer()

Assignment

Assignment

➡ To assign a value or a formula to a variable, the assignment operator [<- or =] is used
- Usage: a {Value} or a {Formula}. Example: a <- 3

✓ Continuation prompt
- ➡ In some cases, an assignment needs to be made over a couple of lines. For example, when using two input vectors in the same formula
- ➡ In these cases, the Enter key may be used to add a line to the command.

✓ Case Sensitivity
- ➡ All function names, variable names, keywords etc in R are case-sensitive
- ➡ Example: a 3 and A 4 will retain their case-specific values

✓ Assignment rules
- ➡ Blank spaces are ignored
- ➡ No standard names or key words should be used as object names or variables

Comments

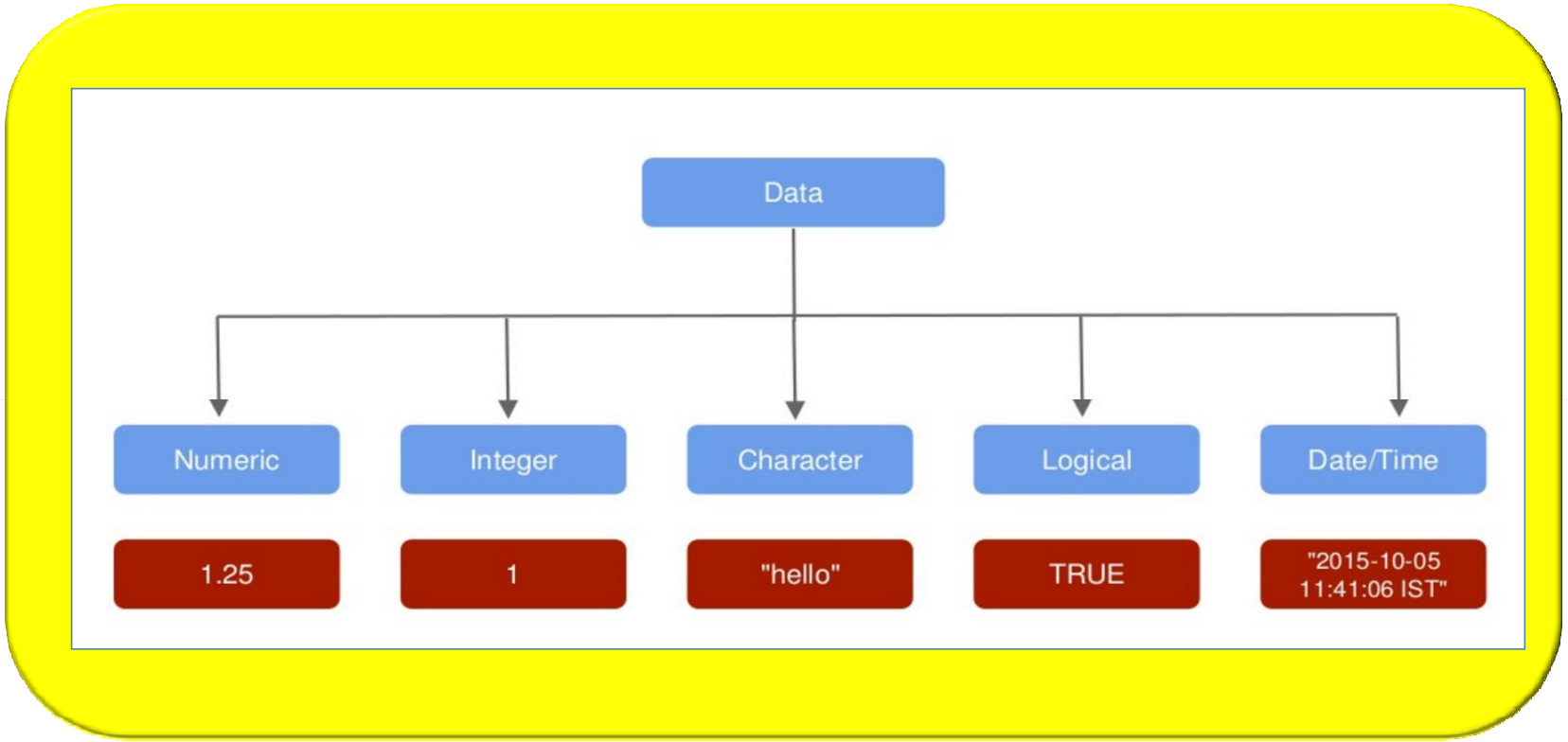
Comments may be added to a function or assignment using the # key

- ➡ Anything from the # to the end of the line is ignored
- ➡ May be used in conjunction with a Continuation prompt
- ➡ Example: a <- 3 #assign the value of 3 to a

Last expression

- ➡ The last value or expression returned by R is stored in .Last.value
- ➡ This may be used as a variable is a function or in an assignment
- ➡ NOTE: if the last expression was a function called, .Last.value will behave like a function , Example: a .Last.value

Data Types in R



What is scalar ?

- A scalar is the one element vector, for instance
  - x <- 10

Vector

- A vector is the most common and basic data structure in R . Its a sequence of data elements of same basic type.

- It's the simple and fundamental data structure in R, these are one dimensional array

  a <- c(1,2,3,4)

- members in a vector is called components, it is represented as c() ?
  - To hold the numeric values c(7,8,11)
  - To hold the character values c('Ram','Ashok','Robert')
  - To hold the logical values c(TRUE,TRUE,FALSE)

- using c() ,you can combine the vectors
- b <- (5,4,a)

Vector Arithmetic

a = c(1, 3, 5, 7)
b = c(1, 2, 4, 8)

- multiply it by 5, we would get a vector with each of its members multiplied by 5.s

```
> 5 * a
[1]  5 15 25 35
```

- Add a and b together, the sum would be a vector whose members are the sum of the corresponding members from a and b.

```
> a + b
[1]  2  5  9 15
```

Vector Arithmetic

```
> x <- c(1,2,3,4,5,13,1,1)
> length(x)
[1] 8
> prod(x)
[1] 1560
> sum(x)
[1] 30
> min(x)
[1] 1
> max(x)
[1] 13
> median(x)
[1] 2.5
> sd(x)
[1] 4.026697
> cumsum(x)
[1]  1  3  6 10 15 28 29 30
```

How to check the data type of a vector

class() - what kind of object is it (high-level)?

typeof() - what is the object's data type (low-level)?

length() - how long is it? What about two dimensional objects?

attributes() - does it have any metadata?

str()

How to name a vector

```
x <- c(10,13,14.5,12)

names(x) <- c('Quiz1', 'Quiz3', 'Quiz2', 'Quiz4')
```

Integer

- **Integers are treated as type numeric.**

  a <- c(1,2,3,4)
  Here, a is a Numeric vector and are treated as double precision real numbers. It can also be converted as an integer using as.integer(a)

- To explicitly create integers , add a L at the end
  x1 <- c(1L, 2L, 3L)

- To verify if this x1 vector is an integer type
  is.integer(x1)

Character

- Words, group of words are represented by type character
- All data of type character must be enclosed in single or double quotation marks
- we can use is.character () to check if a variable is of type character

Sasken training

Logical

- Logical takes only two values. Either TRUE or FALSE
- we coerce other data types to type logical using as.logical()

Sasken training, adyar

Date and Time

- Date and time are represented by the data type date.
- Use as.Date() to coerce data type to type date.

Sasken training, adyar

Coercion

- as.character(5)
- as.integer(100.82)
- as.numeric(“20.8”)
- as.numeric(“Hello World”)

Sasken tr

Summary

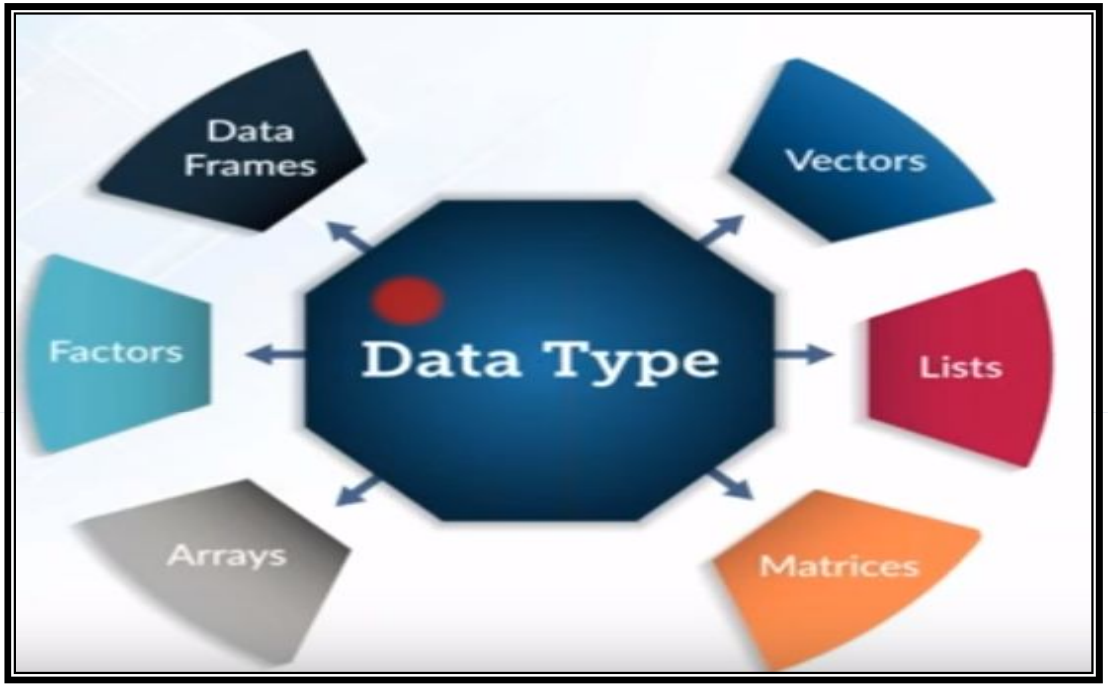
- Numeric, Integer, Character, Logical and Date are basic data types in R

- class/typeof functions returns the data type of an object
- is.data_type tests whether an object is of the specified data type
  - is.numeric()
  - is.integer()
  - is.character() &
  - Is.logical()
- as.data_type is used to coerce objects to the specified data type
  - as.numeric()
  - as.integer()
  - as.characater()
  - as.logical()

Data Structures in R

DS Part IV – Python Hands on

Sasken Training, Adyar , 9840014739 Chennai – 600 020


Data Frames
Vectors
Factors
Data Type
Lists
Arrays
Matrices

Data Structures in R

- Scalar
- Vector
  - One dimensional Array
- Matrix
  - Two Dimensional array
- Array
  - Multidimensional Array
- List
  - Collection of objects
- Factors
- Data Frame
  - Identical to spreadsheet or table data

Vectors

A vector is the most common and basic data structure in R

How to create a vector ?

```
x <- c(1,2,3,4)   # numeric vector

y  <- c(TRUE, TRUE, FALSE, FALSE)   # Logical vector

z <- c("Alec", “David", "Ravi", “Ram")  # Character Vector
```

Lists

- A list is a collection of elements , we also call this as recursive vectors, because a list can contain other lists or data frames.

- A list can have different types of data

```
n = c(2, 3, 5)
s = c("aa", "bb", "cc", "dd", "ee")
b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
x = list(n, s, b, 3)    # x contains copies of n, s, b
print(x)
```

List Slicing

```
> x[2]
```

To access a specific member from a list

```
x[[2]][1]
```

To modify a content in a List

```
x[[2]][1]='AA1'
```

Lists

```
> g <- list(Firstname=c("Ashok","Malik","Joe"), Age=c(26,32,38))
> g
$Firstname
[1] "Ashok" "Malik" "Joe"

$Age
[1] 26 32 38

> g$Firstname
[1] "Ashok" "Malik" "Joe"
> g$Age
[1] 26 32 38
> g$Firstname[3]
[1] "Joe"
>
```

```
> g[[2]][3]
[1] 38
> g
$Firstname
[1] "Ashok" "Malik" "Joe"

$Age
[1] 26 32 38
```

Lists

```
> g <- append(g,"Shiva",after=1)
> g
$Firstname
[1] "Ashok" "Malik" "Joe"

[[2]]
[1] "Shiva"

$Age
[1] 26 32 38

> names(g) <- c("Firstname","LastName","Age")
> g
$Firstname
[1] "Ashok" "Malik" "Joe"

$LastName
[1] "Shiva"

$Age
[1] 26 32 38
```

Arrays

- Arrays is a structure that contains data of the same type, whether that's strings or characters , or integers.
- Arrays can be multi-dimensional as well, so that the data can be contained in multiple rows and columns

```
> test_data <- c("Akhil","Bhaskar",
+               "Chandra","Akshaya","Mihir","Dakshit","Harsha","Gajodhaar")
>
> my_array <- array(test_data,dim=c(3,4))
> my_array
     [,1]      [,2]      [,3]       [,4]
[1,] "Akhil"   "Akshaya" "Harsha"   "Bhaskar"
[2,] "Bhaskar" "Mihir"   "Gajodhaar" "Chandra"
[3,] "Chandra" "Dakshit" "Akhil"    "Akshaya"
>
```

rows, cols

Arrays

```
> c <- array(c(1:6),dim = c(2,3))
> c
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

arr1 <- c(1:12)
dim(c) <- c(2,3)

```
> d <- array(c(7:12),dim=c(2,3))
> d
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
>
```

Array Slicing

```
> a1[1:5]
[1]  1  2  3  4  5
> a1[,1]
[1]  1  2
> a1 <- array(c(1:6), dim=c(2,3))
> a1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> a1[,1]
[1]  1  2
> a1[1]
[1]  1
> a1[1]=10
> a1
     [,1] [,2] [,3]
[1,]   10    3    5
[2,]    2    4    6
```

Matrices

- Matrix is a special kind of vector and it has the elements in a two dimensional layout
- A matrix is a vector with two additional attributes : The number of rows and the number of columns.

matrix(data, nrow, ncol, byrow,dimnames)

nrow is the number of rows to be created
ncol is the number of columns to be created
byrow is a logical representation. If TRUE then the input vector elements are arranged by row
dimname is the names assigned to the rows and columns

```
x <- matrix(c(1,2,3,4), nrow=2, ncol=2)
 x
```

Accessing members of Matrices

```
> m <- matrix(c('Ashok',8,10.2,4),nrow=2,ncol=2)
> print(m)
     [,1]    [,2]
[1,] "Ashok" "10.2"
[2,] "8"     "4"
> class(m)
[1] "matrix"
> m[1]
[1] "Ashok"
> m[1][1]
[1] "Ashok"
> m[2][1]
[1] "8"
```

Can I add an array and Matrix with different dimensions ?

```
> a <- matrix(c(3,4,5,8), nrow=2,ncol=2)
> a1 <- array(c(1:6), dim=c(2,3))
> a1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> a
     [,1] [,2]
[1,]    3    5
[2,]    4    8
> a1+a
Error in a1 + a : non-conformable arrays
```

Accessing members of Matrices

```
# Access the element at 3rd column and 1st row.
Print(tmatrix[1,3])

# Access the element at 2nd column and 3rd row.
print(tmatrix [3,2])

# Access only the  2nd row.
print(tmatrix [2,])

# Access only the 3rd column.
print(tmatrix [,3])
```

Accessing members of Matrices

```
# Access the element at 3rd column and 1st row.
Print(tmatrix[1,3])

# Access the element at 2nd column and 3rd row.
print(tmatrix [3,2])

# Access only the  2nd row.
print(tmatrix [2,])

# Access only the 3rd column.
print(tmatrix [,3])
```

Arithmetic operations

Can I perform addition with the combination of array and Matrix ?

```
> a1 <- array(c(1:6), dim=c(2,2))
> a <- matrix(c(3,4,5,8), nrow=2,ncol=2)
> a1
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> a
     [,1] [,2]
[1,]    3    5
[2,]    4    8
> a1+a
     [,1] [,2]
[1,]    4    8
[2,]    6   12
>
```

Accessing members of Matrices

```
> m <- matrix(c('Ashok',8,10.2,4),nrow=2,ncol=2)
> print(m)
     [,1]    [,2]
[1,] "Ashok" "10.2"
[2,] "8"     "4"
```

```
> m[3][1]
[1] "10.2"
```

```
> is.numeric(m[3][1])
[1] FALSE
> is.character(m[3][1])
[1] TRUE
>
```

Matrix Arithmetic

```
matrix1 <- matrix(c(1, 3, 5, 7), nrow = 2)
matrix2 <- matrix(c(2, 4, 6, 8), nrow = 2)

# Add the matrices.
Mat.Add <- matrix1 + matrix2

# Subtract the matrices
Mat.Sub <- matrix1 - matrix2

# Multiply the matrices.
Mat.Multi <- matrix1 * matrix2

# Divide the matrices
Mat.Div <- matrix1 / matrix2
```

Array, Matrix dim & attributes()

```
> b <- matrix(c(2.54,1,6,3), nrow=2,ncol=2)
> b
     [,1] [,2]
[1,] 2.54    6
[2,] 1.00    3
> attributes(b)
$dim
[1] 2 2

> dim(b)
[1] 2 2
> dim(b) <- NULL
> b
[1] 2.54 1.00 6.00 3.00
> dim(b)
NULL
>
```

```
> a1 <- array(c(1:6), dim=c(2,2))
> a1
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> attributes(a1)
$dim
[1] 2 2

> dim(a1)
[1] 2 2
> dim(a1) <- NULL
> a1
[1] 1 2 3 4
```

Data Frames

- A data frame is collections of data arranged in rows and columns.
- Data frames are used to store spreadsheet or table–like data

Policy #	Claim #	Claim date	State code	Claim Amt	Patient id
8977-AMJED	clA0J01	20160102	FL	22485.35	c00151
8981-KMEED	clA0J02	20160102	Tx	32423.23	jx0201
8981-MAEJ	DlA0J03	20160202	Ny	12134.23	kx0012

Data Frames

To delete a row use –ve values inside data frame and assign NULL to a column

```
> df <- df[-3]
> df
  name length
1 Mike     21
2 Lucy     32
3 John     35
4 <NA>     34
> df["length"] <- null
Error: object 'null' not found
> df["length"] <- NULL
> df
  name
1 Mike
2 Lucy
3 John
4 <NA>
>
```

Data Frames

Data frame creation

```
> name <- c("Mike", "Lucy", "John")
> name
[1] "Mike" "Lucy" "John"
> df <- data.frame(name)
> df
  name
1 Mike
2 Lucy
3 John
```

Adding a column to an existing data frame

```
> df['length'] <- c(21,32,35)
> df
  name length
1 Mike     21
2 Lucy     32
3 John     35
> df['age'] <- c(21,32,35)
> df
  name length age
1 Mike     21  21
2 Lucy     32  32
3 John     35  35
~
```

Data Frames

To delete a row use –ve values inside data frame and assign NULL to a column

```
> df <- df[-3]
> df
   name length
1 Mike      21
2 Lucy      32
3 John      35
4 <NA>      34
> df["length"] <- null
Error: object 'null' not found
> df["length"] <- NULL
> df
   name
1 Mike
2 Lucy
3 John
4 <NA>
>
```

Data Frames

Creating a data frame

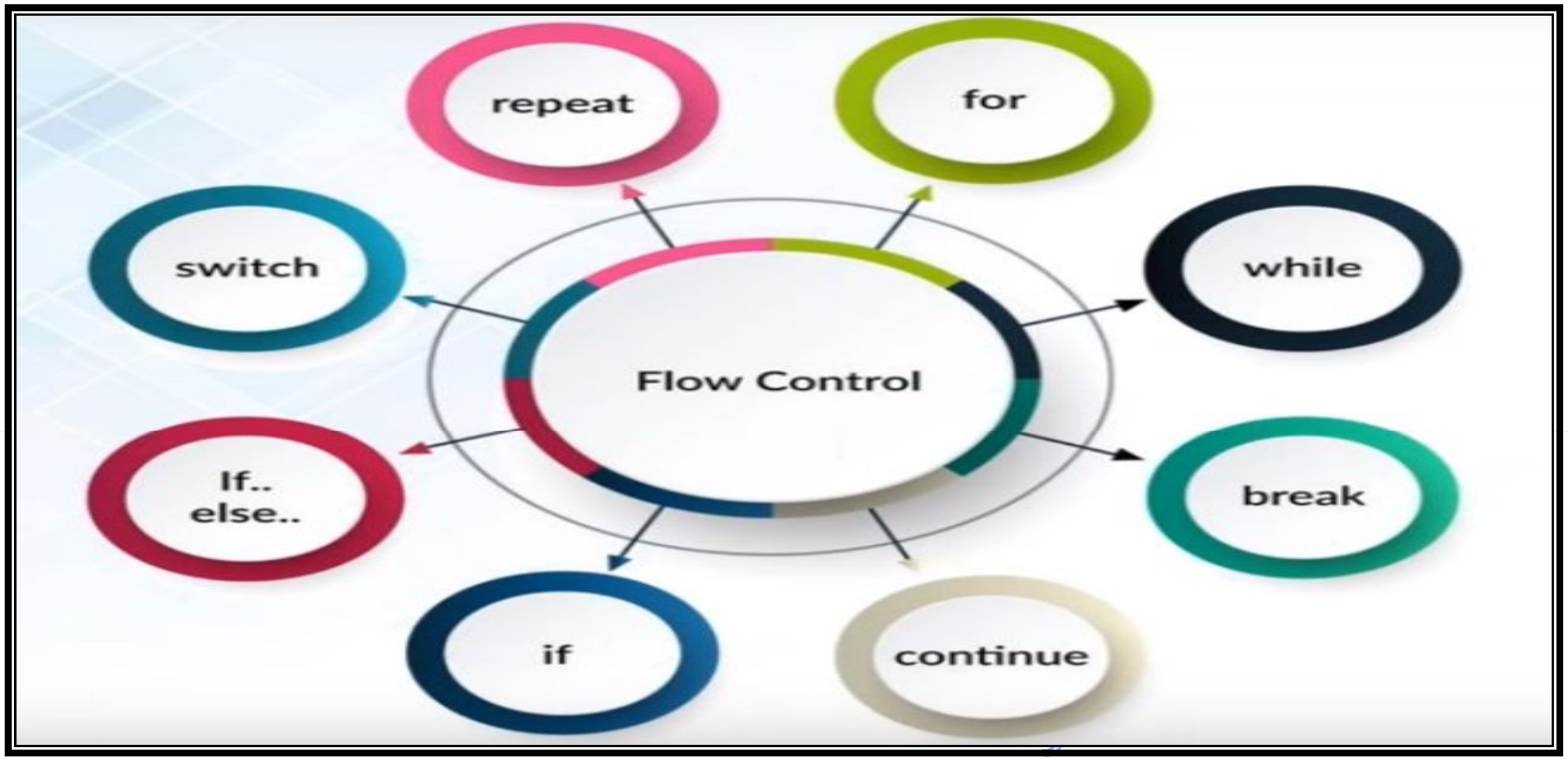
```
> employee <- c('John Doe','Peter Gynn','Jolie Hope')
> salary <- c(21000, 23400, 26800)
> startdate <- as.Date(c('2010-11-1','2008-3-25','2007-3-14'))
>
> df1 <- data.frame(employee,salary,startdate)
> df1
     employee salary  startdate
1    John Doe  21000 2010-11-01
2 Peter Gynn  23400 2008-03-25
3 Jolie Hope  26800 2007-03-14
> is.data.frame(df1)
[1] TRUE
>
```

```
> name <- c("Mike", "Lucy", "John")
> age <- c(20, 25, 30)
> student <- c(TRUE, FALSE, TRUE)
> df = data.frame(name, age, student)
>
> df
  name age student
1 Mike  20    TRUE
2 Lucy  25   FALSE
3 John  30    TRUE
>
```

Data Frames

- Extracting the dimensions of a table
  - dim(df)
- To find out the number of rows and columns
  - nrow(df)
  - ncol(df)
- To access the variable/column names
  - names(df)
- The unique name of all 1000 persons are stored in character vector
  - unique(df[,"col3"])

Control Structures



Control Structures

if statement An **if** statement consists of a Boolean expression followed by one or more statements.

```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true.
}
```

```
x <- 30L
if(is.integer(x)) {
   print("X is an Integer")
}
```

if...else statement An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.

```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true.
} else {
   // statement(s) will execute if the boolean expression is false.
}
```

```
if("Truth" %in% x) {
   print("Truth is found")
} else {
   print("Truth is not found")
}
```

Control structures – if and for loop

If…then…else

➡ This construct takes the form if (expression 1) {construct 1} else {construct 2}

➡ Examples:

```
> x <- 8
>
> if(x >4) {
+     y <- x + 10
+ } else {
+     y <- x
+ }
> y
[1] 18
>
```

✓ For ➡ This construct takes the form for (name in expression 1) {construct 1} ➡ Examples:

```
> x <- c(5:19)
> y <- numeric(15)
> for (i in 1:length(x))
+ {
+       if (x[i] < mean(x))
+       {
+          y[i] <- x[i] + mean(x)
+       }
+       else
+       {
+         y[i] <- x[i]
+       }
+ }
> y
 [1] 17 18 19 20 21 22 23 12 13 14 15 16 17 18 19
>
```

Control structures – for loop

 # for loop in R

```
for(j in 1:5)
{
print (j^2)
}
```

 # R nested for loop

```
 for(i in 1:5)
{
for(j in 1:2)
{
print(i*j);
}
}
```

For loop with break statement

```
x <- 1:5
for (i in x) {
  if (i == 3){
    break
  }
  print(i)
}
```

 # R for loop with next statement

```
x <- 1:5
for (i in x) {
 if (i == 2){
 next
 }
 print(i)
}
```

Control structures – while loop

```
> x <- 10
> while (x > 5) {
+    y <- x + 20
+    break
+ }
> y
[1] 30
>
```

✓ Repeat ➡ This construct takes the form repeat {construct 1} ➡ Example:

```
> x <- 10
> i <- 1
> repeat {
+    i <- i + 1
+    y <- x + i
+    if (i > 30) {
+      break
+    }
+ }
> y
[1] 41
```

Control structures – while loop

 # while loop in R

```
i <- 1
while (i <=8) {
print(i*i)
i = i+1
}
```

 # while loop in R with next statement

```
 i <- 1
while (i <= 6) {
if (i==4)
{
i=i+1
next;
}
print(i*i);
i = i+1;
}
```

 # R while loop with break statement

```
i <- 1
while (i <= 6) {
if (i==4)
break;
print(i*i)
i = i+1
}
```

Your first R function creation

```
> compute_sal <- function(bas,hra,leave) {
+  net_sal = (bas+hra - leave)
+  print(net_sal)
+ }
> compute_sal(10000,2000,500)
[1] 11500
>
```

Factors

Factor stores the nominal values as a vector of integers in the range

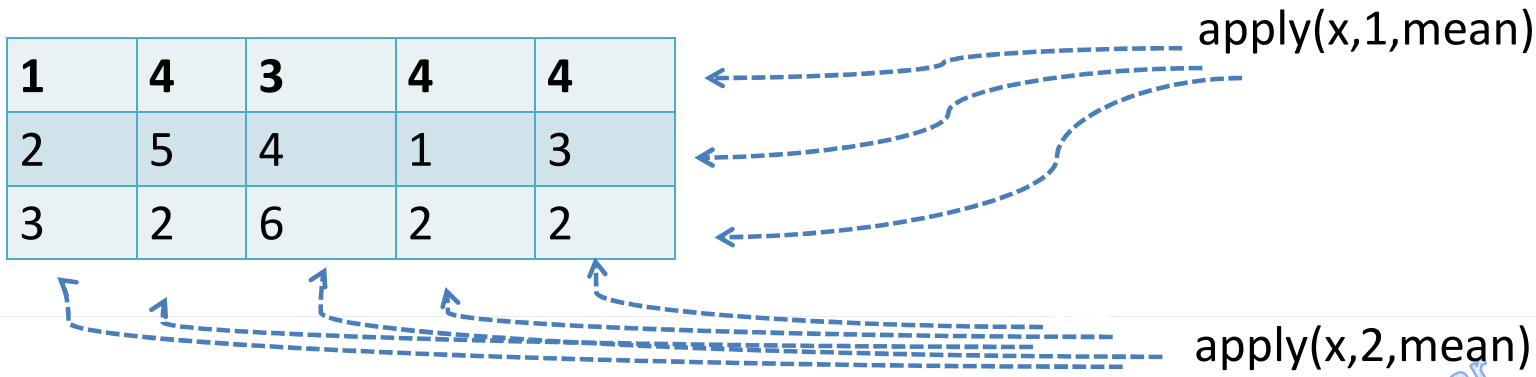
```
> fbak <- read.csv("D:/Datasets/feedback.csv")
> str(fbak)
'data.frame':   9 obs. of  3 variables:
 $ ratings: Factor w/ 3 levels "A","B","C": 1 2 3 1 3 2 2 1 1
 $ product: Factor w/ 9 levels "P1","P2","P3",..: 1 2 3 4 5 6 7 8 9
 $ topsale: int  5000 2000 5600 4800 6000 6500 7000 2800 2100
> table(fbak$ratings)

A B C
4 3 2
> fbak$ratings <- factor(fbak$ratings)
> table(fbak$ratings)

A B C
4 3 2
> # fbak$ratings <- factor(c("good","poor","very poor"))
> fbak$ratingsc
NULL
> ## re level the ratings
> # fbak$ratings = factor(fbak$ratings, levels=c("A","B","C","D"))
> # levels(fbak$ratings ) = c("good","poor","Very Poor","Horrible")
> # alternative to the above 2 lines, you can use the single line code
> fbak$ratings = factor(fbak$ratings, levels=c("A","B","C","D"),labels=c("good","poor","Ver
y Poor","Horrible"))
> fbak$ratings
[1] good       poor       Very Poor good       Very Poor poor       poor       good
[9] good
Levels: good poor Very Poor Horrible
> unique(fbak$ratings)
[1] good       poor       Very Poor
Levels: good poor Very Poor Horrible
> table(fbak$ratings)

     good       poor Very Poor   Horrible
        4          3          2          0
```

what is apply function ?



Sasken training, adyar

lapply() and sapply()

Both of them work in a similar way , sapply will work on a vector whereas lapply will return a list rather than a vector.

tapply() applies a function or operation on subset of the vector broken down by a given factor variable.

```
> tapply(mtcars$mpg,list(mtcars$cyl,mtcars$am),mean)
       0        1
4 22.900 28.07500
6 19.125 20.56667
8 15.050 15.40000
> sapply(1:3,function(x) x^2)
[1] 1 4 9
> lapply(1:3,function(x) x^2)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9
```

simplify=F & unlist

```
> sapply(1:3,function(x) x^2, simplify=F)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9

> unlist(lapply(1:3,function(x) x^2))
[1] 1 4 9
```


Convert the vector to list

Convert the list to Vector

mapply(), vapply() & rapply ()

mapply - *For when you have several data structures (e.g. vectors, lists) and you want to apply a function to the 1st elements of each, and then the 2nd elements of each, etc., coercing the result to a vector/array as in sapply.*

This is multivariate in the sense that your function must accept multiple arguments.

```
> mapply(sum, 1:5, 1:5, 1:5)
[1]  3  6  9 12 15
> mapply(rep, 1:4, 4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4
```

#Sums the 1st elements, the 2nd elements, etc.

vapply

vapply - *When you want to use sapply but perhaps need to squeeze some more speed out of your code.*

exactly the same as sapply in terms of functionality
use to squeeze some more speed out of your code by telling R what sort of stuff your function will return;
can also be safer to use since you know what will be returned
forces you to think about what your function returns

```
> cities <- c("New York", "London", "Cape Town")
> vapply(cities, nchar, numeric(1))
 New York    London Cape Town
        8         6         9
```

```
dat <- list(a = 1:10, b = 11:20)
vapply(dat, function(x) {return(c(min = min(x), max = max(x)))}, c(min = 0, max = 0)) #returns minimum and maximum
```

Dealing with Date-Time Classes

POSIXlt and POSIXct represents calendar dates and times

Date Function

Conversion specification	Description	Example
%a	Abbreviated weekday	Sun, Thu
%A	Full weekday	Sunday, Thursday
%b or %h	Abbreviated month	May, Jul
%B	Full month	May, July
%d	Day of the month 01-31	27, 07
%j	Day of the year 001-366	148, 188
%m	Month 01-12	05, 07
%U	Week 01-53 with Sunday as first day of the week	22, 27
%w	Weekday 0-6 Sunday is 0	0, 4
%W	Week 00-53 with Monday as first day of the week	21, 27
%x	Date, locale-specific	
%y	Year without century 00-99	84, 05
%Y	Year with century on input: 00 to 68 prefixed by 20 69 to 99 prefixed by 19	1984, 2005
%C	Century	19, 20
%D	Date formatted %m/%d/%y	05/27/84, 07/07/05
%u	Weekday 1-7 Monday is 1	7, 4

ining, adyar

Agenda

- Introduction to dplyr
- Selecting , filtering and creating derived columns using mutate()
- creating derived columns using mutate()
- Aggregation of data using summarise() , summarise_each() and group_by()
- Using window() functions

Sasken training, adyar

dplyr

What is dplyr?

The dplyr is a powerful R-package to manipulate, clean and summarize the data. In short, it makes data exploration and data manipulation easy and fast in R.

What's special about dplyr?

The package "dplyr" comprises many functions that perform mostly used data manipulation operations such as applying filter, selecting specific columns, sorting data, adding or deleting columns and aggregating data. Another most important advantage of this package is that it's very easy to learn and use dplyr functions.
Also easy to recall these functions. For example, filter() is used to filter row

dplyr vs. Base R Functions

dplyr functions process faster than base R functions. It is because dplyr functions were written in a computationally efficient manner.

dplyr

dplyr works only with data frames and it provides many features to perform common data manipulation tasks.

```
# installing dplyr package
install.packages('dplyr')

# importing the dplyr package after installation
library(dplyr)
```

Important dplyr verbs to remember

filter()	Filtering rows to fetch subset of the data
select()	Selecting specific columns of data ((i.e., selecting variables)
group_by()	Allows for group operations in the 'split-apply-combine' concept
arrange()	sorting
Summarize()	Aggregating Summarize the data
mutate()	Including a derived variables to the existing data set

split the data into groups, apply some analysis to each group, and then combine the results.

dplyr - Including new columns

Adding columns to data using mutate(),

Adding a new column, incToSpendRatio

```
head(mutate(df, incToSpendRatio =Salary/AmountSpent))
```

dplyr - Ordering Data

Ordering data using order_by(),

Order whole data by Salary in ascending order

head(arrange(df,desc(Salary)))

dplyr - Summarizing the Data

Summarizing data using summarize() and group_by()

group_by() makes grouped table, summarize() can take this grouped table and produce summaries for different columns

Mean level of AmountSpent and standard deviation of AmountSpent for each type of Parent

```
df_child <- group_by(df,Children)
summarise(df_child,Total=sum(AmountSpent))
```

To select all columns except a specific column

head(select(mkt, -Married)

To select range of columns

head(select(mkt,Married:Location))

To select a specific column

head(select(mkt,Married))

To filter specific rows

filter(mkt, Salary >= 10000)

filter(mkt, Salary >= 10000, Married == 'Single')

filter(mkt, order %in% c("Location", "AmountSpent"))

dplyr - pipes

- dplyr becomes a powerful tool when combined with %>% (pipe) operator
- Several data manipulation tasks can be accomplished in just one line of code

Traditionally functional composition is achieved by using nested function calls

For example, Find the mean AmountSpent by all people whose Salary is >=10000

```
mean(df[df$Salary >= 10000,'AmountSpent'])
filter(Salary >= 10000) %>% summarise(mean(AmountSpent)
```

Helpers	Description
starts_with()	Starts with a prefix
ends_with()	Ends with a prefix
contains()	Contains a literal string
matches()	Matches a regular expression
num_range()	Numerical range like x01, x02, x03.
one_of()	Variables in character vector.
everything()	All variables.

Combining Data sets

dplyr::lef_join(a, b, by = "x1")	Join matching rows from b to a.
dplyr::right_join(a, b, by = "x1")	Join matching rows from a to b.
dplyr::inner_join(a, b, by = "x1")	Join data - Retain only rows in both sets.
dplyr::full_join(a, b, by = "x1")	Join data - Retain all values, all rows.

Filtering Joins

dplyr::semi_join(a, b, by = "x1") - All rows in a that have a match in b.

dplyr::anti_join(a, b, by = "x1") - All rows in a that do not have a match in b.

Set Operators

dplyr::intersect(y, z) Rows that appear in both y and z.

dplyr::union(y, z) Rows that appear in either or both y and z.

dplyr::setdiff(y, z) Rows that appear in y but not z

Summary

- dplyr: better manipulation functionality
- Sub-setting data using filter()
- Selecting columns using select()
- Adding new columns using mutate()
- Ordering data using arrange()
- Summarizing using summarize() and group_by()
- Using functional pipelines to do more than one manipulation task

Data Analysis Case Study

BigBox is a leading seller of Laptops, Personal computers, stereo equipment, and other electronic products. The company advertises entirely by mailing catalogs to its customers, and all of its orders are taken over the telephone. Since the company spends a great deal of resource on its catalog mailings, it wishes to find out if this is paying off in sales.

For this purpose, the company collected data from 250 customers.

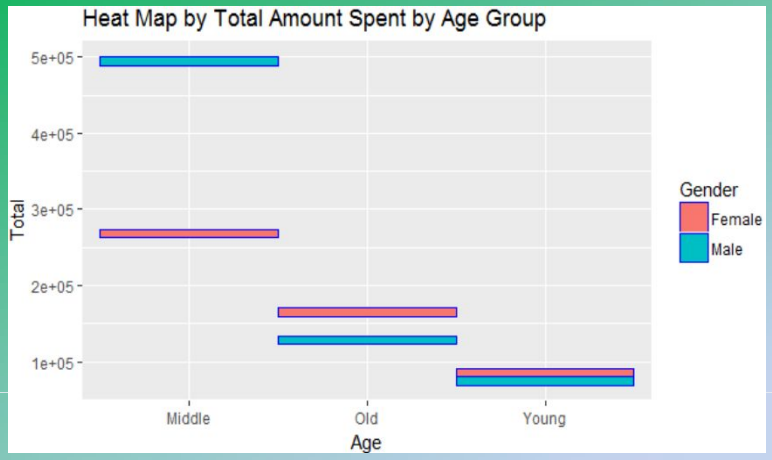
The last column gives the total amount of purchases made by each customer.

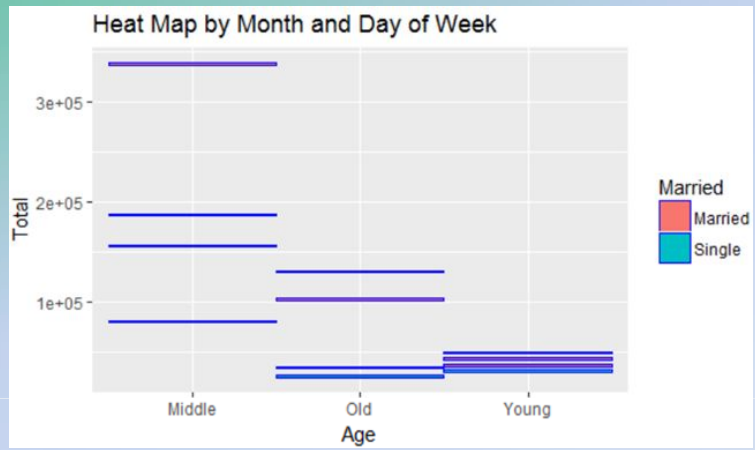
The remaining columns give various information regarding individual customers.

There are 10 variables, the meaning of each is explained as comments in the column headings.

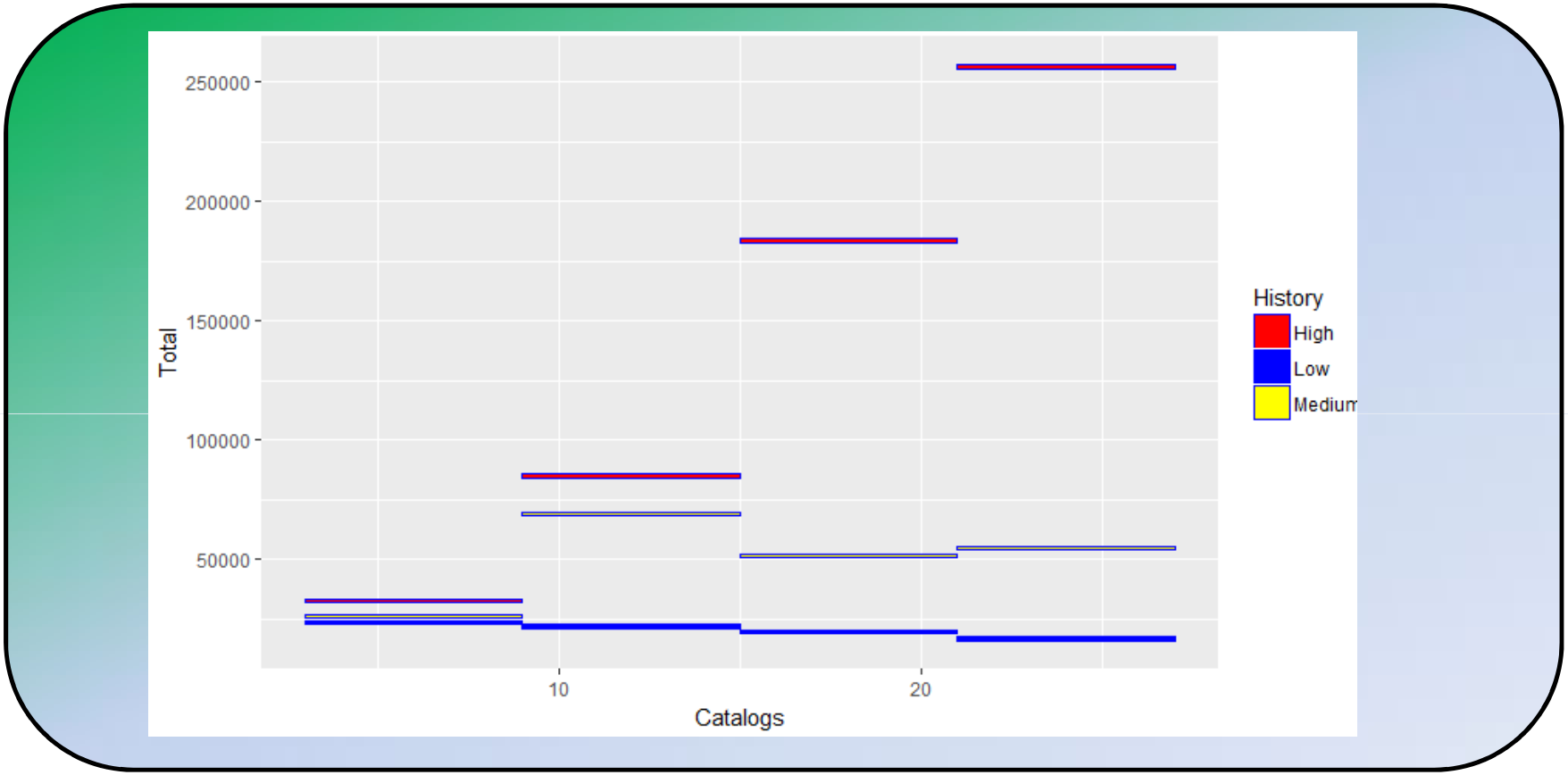
Customer	
Age	Age of a customer
Gender	1 if male, 0 if female
Home Owner	1 if person owns a house 0 otherwise
Married	1 married 0 otherwise
Close	1 if close to stores with merchandise 0 if out in the middle of nowhere
Income	Both person and spouse salary
Children	Number of childrens
PrevCust	1 any prev purchases were made last year , 0 otherwise
PrevSpent	Amount spent last year
Catalogs	Number of catalogs sent to a person this year
BillAmount	Amount spent on total purchases this year

Visualization

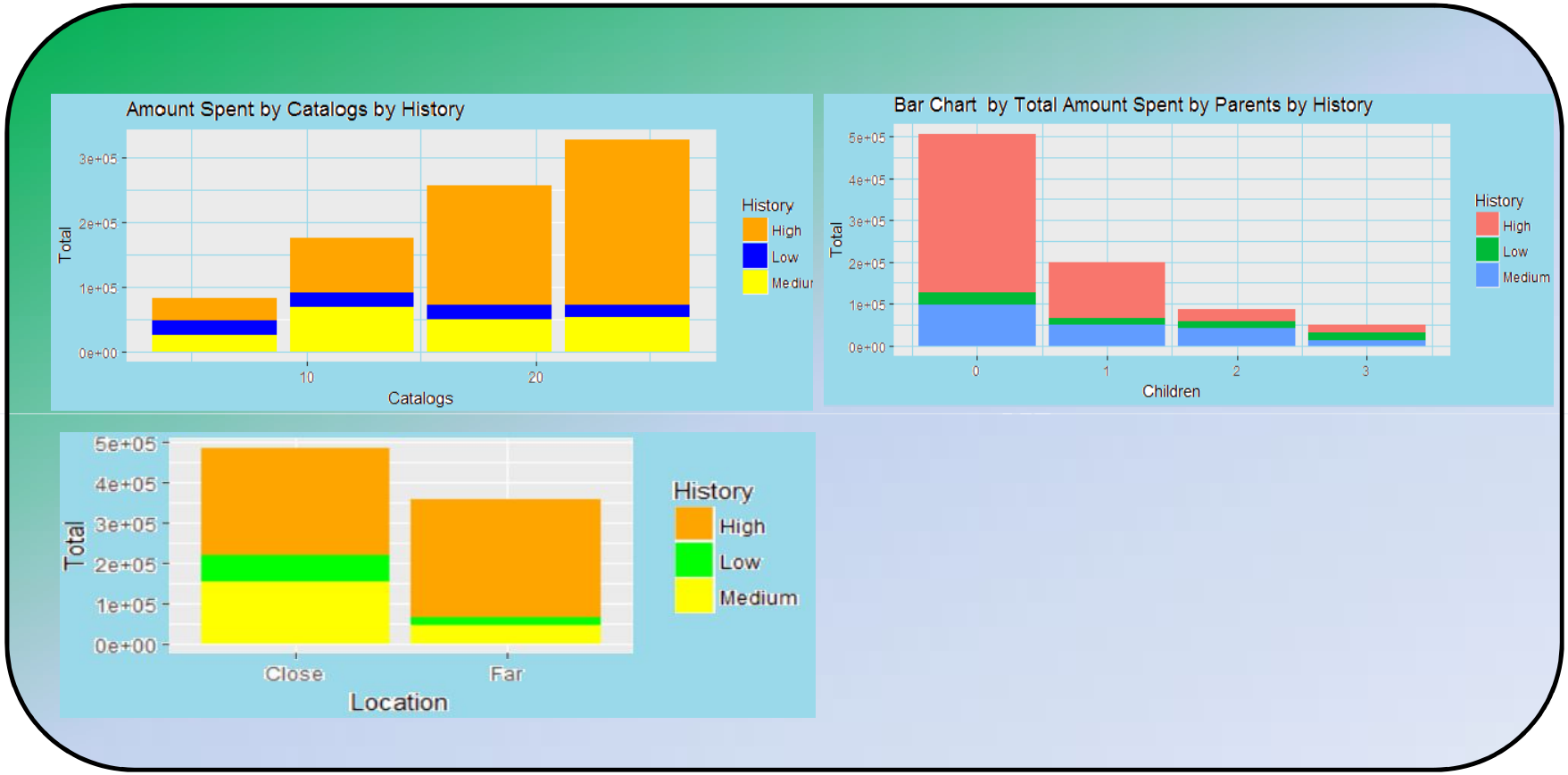




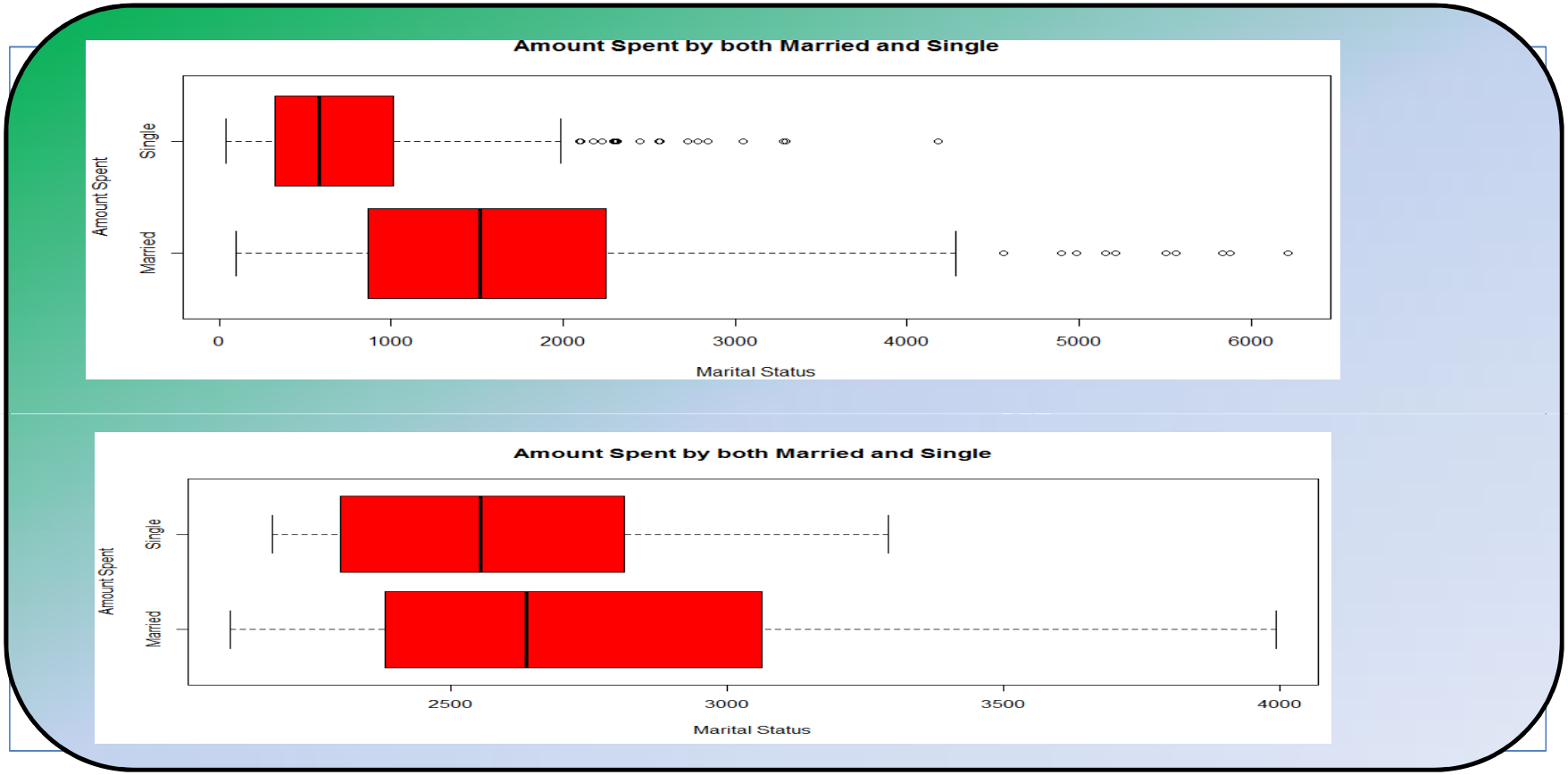
Amount spent by People live close to the mall by History



Amount spent by People live close to the mall by History



Amount spent based on Marital Status



Insights

our male customers, on average, are 23% more married and own 15% more homes than our female customers

Also, our married customers own 84% more homes than single customers, and on average our married customers make up 12.5% more of our previous customer base.

The average age Salary appears to have the strongest correlation with Amount spent out of all of the independent variables

Secondly, that men are our primary target audience. We first begin by establishing the fact that men, on average, spend $388 more on our products than women do.

men have fewer children and received more catalogs, all factors that increase average spending

data.table

What is data.table?

The data.table R package is considered as the fastest package for data manipulation

Analysts generally call R programming not compatible with big datasets (> 10 GB) as it is not memory efficient and loads everything into RAM.

What's special about data.table package ?

The package comprises many functions that perform mostly used data manipulation operations such as applying filter, selecting specific columns, sorting data, adding or deleting columns and aggregating data. Another most important advantage of this package is that it's very easy to learn and use data.table functions.

There are many benchmarks have been conducted in the past to compare dplyr vs data.table. In every benchmark, data.table outperformed .
The efficiency of this package was also compared with python' package (panda). And data.table outperformed.

In CRAN, there are more than 200 packages that are dependent on data.table which makes it listed in the top 5 R's package.

rapply

rapply - *For when you want to apply a function to each element of a **nested list** structure, recursively.*

rapply - *For when you want to apply a function to each element of a **nested list** structure, recursively.*

```
# rapply() recursively apply
x=list(1,list(2,3),4,list(5,list(6,7)))
rapply(x,function(x){x^2})
```

Histograms displays the frequency distribution of a variable

hist() function is used
breaks : The number of bins
label : labels the frequency of each bin
xlim: sets the range for x axis

Working with Diverse Source system data

- Loading data from diverse sources
- Importing plain text files, including the csv , tab-delimited file formats & Examining the data
- Importing excel files
- Import a file set stored in a specific file type
- Work with relative file paths
- Export a data set to a CSV or tab-delimited file

Sasken training, ac

Loading data from csv file

- read.csv comes with utils package , which is automatically loaded in Rsession on startup using which CSV files can be imported into R session.

It uses a comma to separate values within rows

- dataframe 1 <- read.csv("data.csv",sep=",“,head=TRUE)

With stringsAsFactors, you can tell R whether it should convert strings in the flat file to factors.

For all importing functions in the utils package, this argument is TRUE, which means that you import strings as factors. This only makes sense if the strings you import represent categorical variables in R.

If you set stringsAsFactors to FALSE, the data frame columns corresponding to strings in your text file will be character.

Load data from tab delimited file

read.delim('countries.txt', stringsAsFactors = FALSE)

states.txt

```
state    capital pop_mill    area_sqm
South Dakota    Pierre  0.853   77116
New York    Albany  19.746  54555
Oregon  Salem  3.970   98381
Vermont Montpelier  0.627   9616
Hawaii  Honolulu    1.420   10931
```

As you know from the Cleaning Data in R course, the first step when preparing to clean data is to inspect it. Let's refresh your memory on some useful functions that can do that:

dim() returns the dimensions of an object

head() displays the first part of an object

names() returns the names associated with an object

Exporting data sets

- Various options are available with R to export data frames and matrices
- write.csv(dataset, "filename.csv")
  - s <- read.table("D:/Datasets/sales.txt",sep=",",stringsAsFactors=TRUE,head=TRUE)
    write.table(s,"D:/Datasets/saleswriteinR.txt",sep=",")
    write.table(s,"D:/Datasets/saleswriteinR.txt", sep="," , dec=".")
  - Export matrices
    - Write.table (s,"D:/Datasets/saleswriteinR.txt",row.names=FALSE,col.names=FALSE)
- write.csv(dataset, "filename.csv", row.names=F)
- write.table(dataset, "filename.txt", sep="\t", col.names=F)
- The write.csv and write.table functions create an extra column in the file containing the observation numbers. To prevent this, set the row.names argument to F:
  - write.csv(dataset, "filename.csv", row.names=F)

Exporting data from mysql

- Install MySQL
- Install RMySQL
- // Updating the linux package list
- sudo apt-get update
- // Upgrading the updated packages
- sudo apt-get dist-upgrade
- //First, install the MySQL server and client packages:
- sudo apt-get install mysql-server mysql-client
- # to install RMySQL library (refer ex01-04.R)
- install.packages("RMySQL")
- #Loading RMySQL
- library(RMySQL)
- mydb = dbConnect(MySQL(), user='root', password='', dbname='sample_table', host='localhost')

Exporting data from mysql

- To list the tables
  - dbListTables(mydb)
  - To return a list of data fields created under the sample_table table, use the following command:
  - dbListFields(mydb, 'sample_table')
- Importing the data into R
  - rs = dbSendQuery(mydb, "select * from sample_table")
  - dataset = fetch(rs, n=-1)

Exporting data from Excel

- The following are the prerequisites for the xlsx packages:
  - xlsxjars
  - rJava
- Installing xlsxX packages:
  - install.packages("xlsxjars")
  - install.packages("rJava")
  - install.packages("xlsx")
- Importing data into R

```
library(xlsxjars)
library(xlsx)
library(rJava)
```

  - **x <- read.xlsx("D:/Datasets/Ad.xlsx",1)**

Entering data directly

- Chain <- c("Morrisons", "Asda", "Tesco", "Sainsburys")
- Stores <- c(439, NA, 2715, 934)
- Sales.Area <- c(12261, NA, 36722, 19108)
- Market.Share <- c(12.3, 16.9, 30.3, 16.5)
- supermarkets <- data.frame(Chain, Stores, Sales.Area, Market.Share)
- Supermarkets
- To remove the individual vectors ,
  - rm(Chain, Stores, Sales.Area, Market.Share)

Data Exploration - Case Study Assignment

This retail file warranty.csv holds the sales , discount , shipment_cost and other vital data. You need to explore this data to come up with insights. Treat this case study as an open ended and try to come up with a few insights.

Imagine you are responsible for making sure that your marketing stakeholders should be able to come up with the relevant strategies to attract more customers by unique offers with your insights at the optimal cost. Also, you need to perform analysis from the delivery aspect which can drive the logistics to determine how best they can reduce the shipment duration and optimize their shipping cost.

Besides this, solve the below business questions

1. Month wise, Year wise Sales beyond 2012
2. Shipping Cost for the items returned
3. No. of Shipments and the Shipping cost by Demographic
4. Shipment Delivery Duration by Order
5. No. of Items Ordered & Amount Spent by customer by Month
6. Top 3 Sales by Category, Product and State