R Programming

UNIT – 1

Introduction

- R is a programming language and software environment for statistical analysis, graphics representation and reporting.
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently managed by the R Development Core Team.
- The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions.
- R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

Features of R

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

Why use R for statistical computing and graphics?

- R is open-source and free!
- R is popular – and increasing in popularity
- R runs on all platforms(Cross-platform interoperability)
- Learning R will increase your chances of getting a job

Company	**Application/Contribution**
Twitter	Monitor user experience
Ford	Analyze social media to support design decisions for their cars
New York Times	Infographics, data journalism
Microsoft	Released Microsoft R Open, an enhanced R distribution and Microsoft R server after acquiring Revolution Analytics in 2015
Human Rights Data Analysis Group	Measure the impact of war
Google	Created the R style guide for the R user community inside Google

Applications of R Programming in Real World

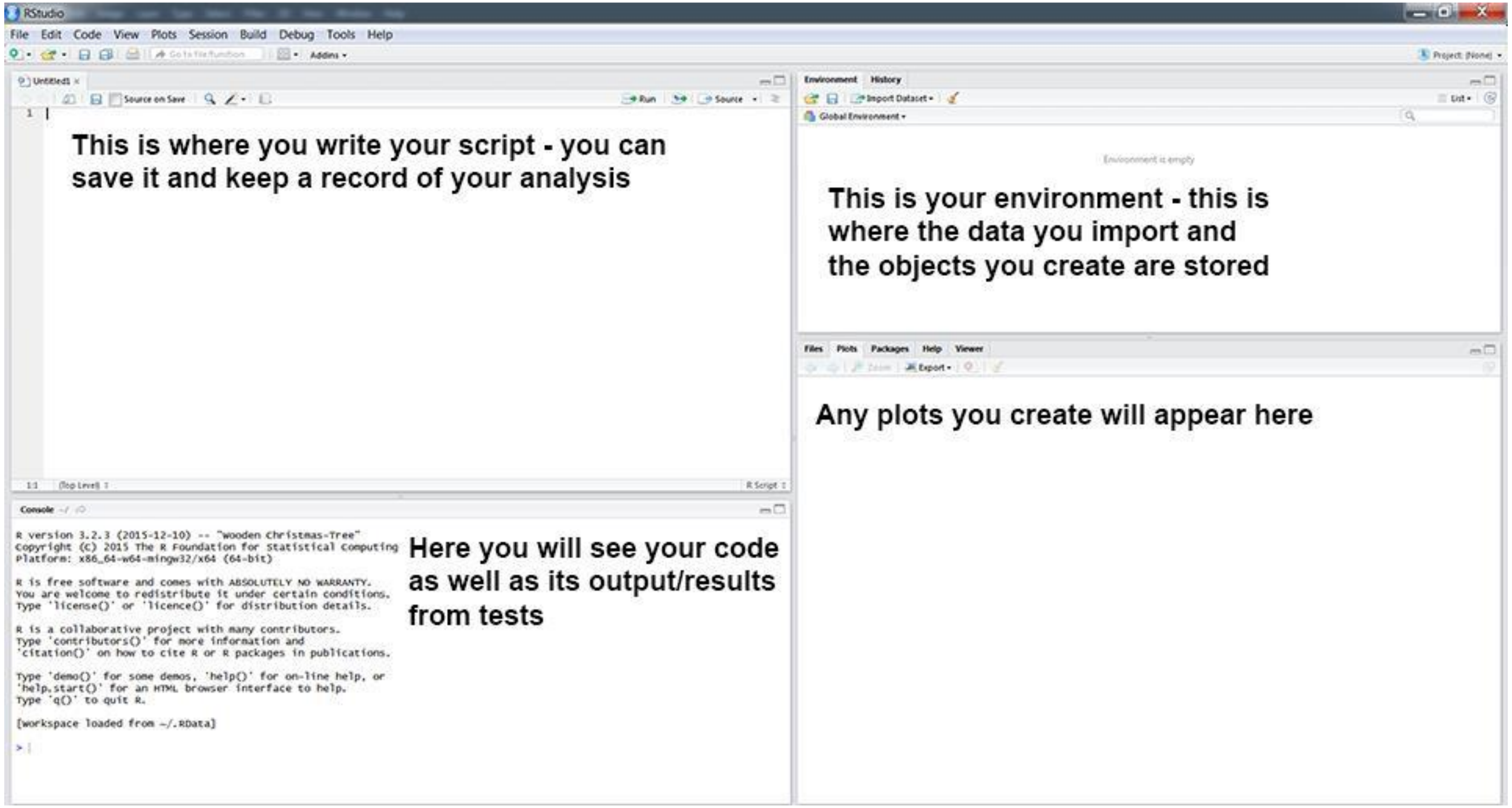
- **Data Science**
- **Statistical computing(9100+ packages)**
- **Machine Learning** (ML tasks like linear and non-linear regression, decision trees, linear and non-linear classification and many more)

Local Environment Setup

- Windows Installation(Comprehensive R Archive Network)
  - https://cran.r-project.org/bin/windows/base/
- You can download the Windows installer version of R from R-4.2.1 for Windows (32/64 bit) and save it in a local directory.
- As it is a Windows installer (.exe) with a name "R-version-win.exe". You can just double click and run the installer accepting the default settings. If your Windows is 32-bit version, it installs the 32-bit version. But if your windows is 64-bit, then it installs both the 32-bit and 64-bit versions.
- After installation you can locate the icon to run the Program in a directory structure "R\R4.2.0\bin\i386\Rgui.exe" under the Windows Program Files. Clicking this icon brings up the R-GUI which is the R console to do R Programming.
- **Rstudio:** An integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

Rstudio

An integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.



This is where you write your script - you can save it and keep a record of your analysis

This is your environment - this is where the data you import and the objects you create are stored

Any plots you create will appear here

Here you will see your code as well as its output/results from tests

```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

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

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

[workspace loaded from ~/.RData]

> 
```

"The only way to learn a new programming language is by writing programs in it." - *Dennis Ritchie*

Reserved words in R

if	else	repeat	while	function
for	in	next	break	TRUE
FALSE	NULL	Inf	NaN	NA
NA_integer_	NA_real_	NA_complex_	NA_character_	…

```
#To get help on reserved words
>help(reserved)
```

or

```
>?reserved
```

Reserved words in R

- TRUE and FALSE are the logical constants in R.
- NULL represents the absence of a value or an undefined value.
- Inf is for "Infinity", for example when 1 is divided by 0 whereas NaN is for "Not a Number", for example when 0 is divided by 0.
- NA stands for "Not Available" and is used to represent missing values.
- R is a case sensitive language. Which mean that **TRUE** and **True** are not the same.
- While the first one is a reserved word denoting a logical constant in R, the latter can be used a variable name.

```
> TRUE <- 1
Error in TRUE <- 1 : invalid (do_set) left-hand side to assignment
> True <- 1
> TRUE
[1] TRUE
> True
[1] 1
```

R Variables and Constants

Variables are used to store data, whose value can be changed according to our need. Unique name given to variable (function and objects as well) is identifier.

Rules for writing Identifiers in R

1. Identifiers can be a combination of letters, digits, period (.) and underscore (_).
2. It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
3. Reserved words in R cannot be used as identifiers.

<u>Valid identifiers in R</u>

total, Sum, .fine.with.dot, this_is_acceptable, Number5

<u>Invalid identifiers in R</u>

tot@l, 5um, _fine, TRUE, .0ne

Note: R support underscore as a valid identifier but it is good practice to use period as word separators

Constants in R

Constants, as the name suggests, are entities whose value cannot be altered

Numeric Constants

- All numbers fall under this category. They can be of type integer, double or complex. It can be checked with the typeof() function.
- Numeric constants followed by L are regarded as integer and those followed by i are regarded as complex.
- Numeric constants preceded by 0x or 0X are interpreted as hexadecimal numbers.
    - Ex:
    - z = 0xAA
    - print(as.hexmode(z))

Numeric Constants - Examples

```
> typeof(5)
[1] "double"

> typeof(5L)
[1] "integer"

> typeof(5i)
[1] "complex"

> 0xff
[1] 255

> 0XF + 1
[1] 16
```

Character Constants - Examples

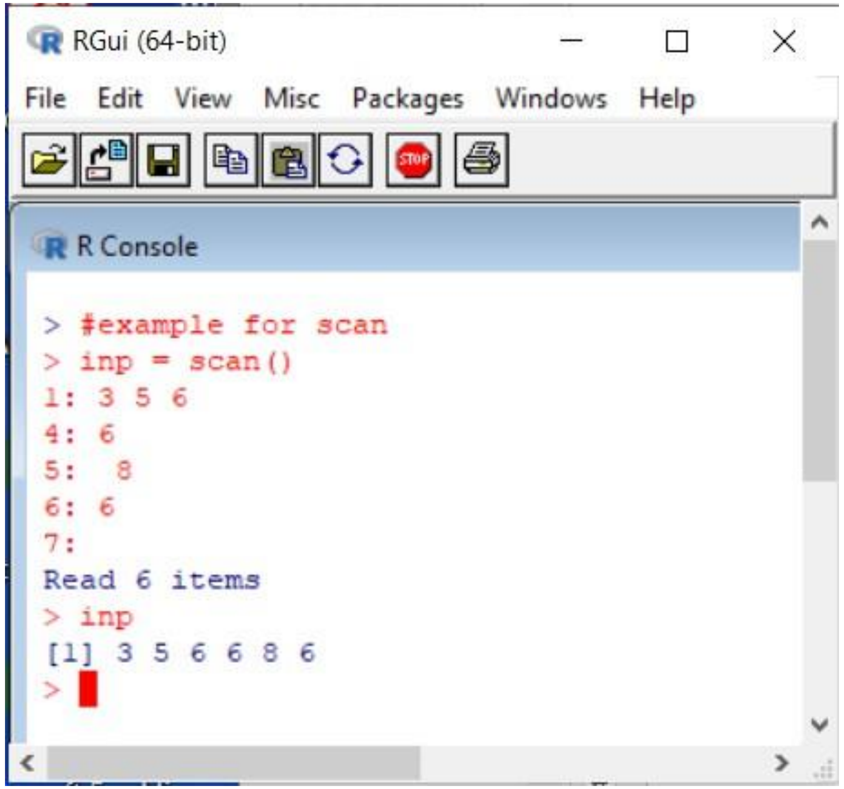
```
> 'example'
[1] "example"

> typeof("5")
[1] "character"
```

Explore class()

Input-Output Features

- **scan()**
- Read data into a vector or list from the console or file

> #example for scan
> inp = scan()
> inp

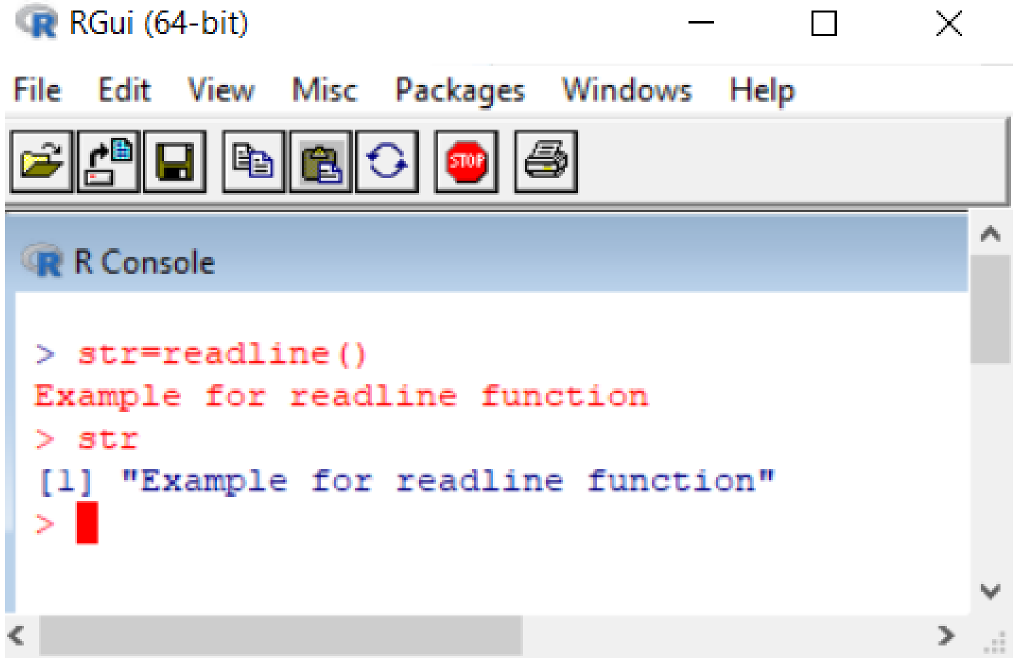


```
> #example for scan
> inp = scan()
1: 3 5 6
4: 6
5:  8
6: 6
7:
Read 6 items
> inp
[1] 3 5 6 6 8 6
> 
```

- **readline()**

Used for inputting a line from the keyboard in the form of a string:

> str = readline()
> str



```
> str=readline()
Example for readline function
> str
[1] "Example for readline function"
> 
```

Output Functions: **print()** and **cat()**

```
print("My first print statement")
[1] "My first print statement"

print("My first print statement", quote = FALSE)
[1] My first print statement

cat("My first print statement")
My first print statement

# Printing quotation marks using cat() keyword
cat("\"My first print statement\"")
"My first print statement"
```

Note:

- cat() is valid only for atomic types (logical, integer, real, complex, character)
- This means that we cannot call cat() on a non-empty list or any type of object.
- As its name implies, it converts arguments to characters and concatenates them.

Print the high-level data type of the given variable.
Print the low-level data type of the given variable.

Input

A *testVariable* containing the variable being tested.

Output

The high-level and low-level data types of that variable.

Sample Input

1.5

Sample Output

numeric
double

```
cat(class(testVariable), "\n") #  high level data type
cat(typeof(testVariable), "\n") # low level of variable
```

R Operators

R has several operators to perform tasks including arithmetic, logical and bitwise operations.

Types of operators in R

Arithmetic operators
Relational operators
Logical operators
Assignment operators

R Arithmetic Operators

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

Examples

```
> x <- 5
> y <- 16
> x+y
[1] 21

> x-y
[1] -11

> x*y
[1] 80
```

```
> y/x
[1] 3.2

> y^x
[1] 1048576

> y%%x
[1] 1

> y%/%x
[1] 3
```

R Relational Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Examples:

```
> x <- 5
> y <- 16

> x<y
[1] TRUE

> x>y
[1] FALSE

> x<=5
[1] TRUE
```

```
> y>=20
[1] FALSE

> y == 16
[1] TRUE

> x != 5
[1] FALSE
```

R Logical Operators

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

- Operators & and | perform element-wise operation producing result having length of the longer operand.
- But && and || examines only the first element of the operands resulting into a single length logical vector.
- Zero is considered FALSE and non-zero numbers are taken as TRUE.

Examples:

```
> x <- c(TRUE,FALSE,0,6)
> y <- c(FALSE,TRUE,FALSE,TRUE)
> !x
[1] FALSE  TRUE  TRUE FALSE

> x&y
[1] FALSE FALSE FALSE  TRUE

> x&&y
[1] FALSE

> x|y
[1]  TRUE  TRUE FALSE  TRUE

> x||y
[1] TRUE
```

R Assignment Operators

Operator	Description
<-, <<-, =	Leftwards assignment
->, ->>	Rightwards assignment

Examples:

```
> x <- 5
> x
[1] 5

> x = 9
> x
[1] 9

> 10 -> x
> x
[1] 10
```

Precedence and Associativity of different operators in R from highest to lowest

Operator	Description	Associativity
^	Exponent	Right to Left
-x, +x	Unary minus, Unary plus	Left to Right
%%	Modulus	Left to Right
*, /	Multiplication, Division	Left to Right
+, –	Addition, Subtraction	Left to Right
<, >, <=, >=, ==, !=	Comparisons	Left to Right
!	Logical NOT	Left to Right
&, &&	Logical AND	Left to Right
|, ||	Logical OR	Left to Right
->, ->>	Rightward assignment	Left to Right
<-, <<-	Leftward assignment	Right to Left
=	Leftward assignment	Right to Left

Flow Control

```
if (test_expression) {
statement
}
```

Example:

```
x <- 5
if(x > 0){
print("Positive number")
}
```

```
[1] "Positive number"
```

```
if (test_expression) {
statement1
} else {
statement2
}
```

Example:

```
x <- -5
if(x > 0){
print("Non-negative number")
} else {
print("Negative number")
}
```

```
[1] "Negative number "
```

```
if ( test_expression1) {
statement1
} else if ( test_expression2) {
statement2
} else if ( test_expression3) {
statement3
} else {
statement4
}
```

Example:

```
x <- 0
if (x < 0) {
print("Negative number")
} else if (x > 0) {
print("Positive number")
} else
print("Zero")
```

```
[1] "Zero"
```

```
(or) if(x > 0) print("Non-negative number") else print("Negative number")
```

ifelse() function

- The ifelse() function is a shorthand vectorized alternative to the standard if...else statement.

Syntax:

```
ifelse(test_expression, x, y)
```

Note:

- test_expression must be a logical vector (or an object that can be coerced to logical).
- The return value is a vector with the same length as test_expression.

Example:

```
a = c(5,7,2,9)
> ifelse(a %% 2 == 0,"even","odd")

[1] "odd"  "odd"  "even" "odd"
```

Syntax of for loop

- A for loop is used to iterate over a vector

Syntax:

```
for (val in sequence)
{
statement
}
```

Example:

```
x <- c(2,5,3,9,8,11,6)
count <- 0
for (val in x) {
if(val %% 2 == 0)  count = count+1
}
print(count)
```

Output:

```
[1] 3
```

while Loop

Syntax:

```
while (test_expression)
{
statement
}
```

Example:

```
i <- 1
while (i < 6) {
print(i)
i = i+1
}
```

Output:

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

break and next Statement

- A break statement is used inside a loop (repeat, for, while) to stop the iterations and flow the control outside of the loop.

Syntax:

```
if (test_expression) {
break
}
```

Example:

```
 x <- 1:5 #create integer sequence
for (val in x) {
if (val == 3){
break
}
print(val)
}
```

Output:

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

break and next Statement

- next statement is useful when we want to skip the current iteration of a loop without terminating it. On encountering next, the R parser skips further evaluation and starts next iteration of the loop.

Syntax:

```
if (test_condition) {
next
}
```

Example:

```
 x <- 1:5  #create integer sequence
for (val in x) {
if (val == 3){
next
}
print(val)
}
```

Output:

```
[1] 1
[1] 2
[1] 4
[1] 5
```

repeat loop

- A repeat loop is used to iterate over a block of code multiple number of times.
- There is no condition check in repeat loop to exit the loop.
- We must ourselves put a condition explicitly inside the body of the loop and use the break statement to exit the loop. Failing to do so will result into an infinite loop.

Syntax:

```
repeat {
statement
}
```

Example:

```
 x <- 1
repeat {
print(x)
x = x+1
if (x == 6){
break
}
}
```

Output:

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

```
x <- c(1:10)
x[(x>8) | (x<5)]
# yields 1 2 3 4 9 10

# How it works
x <- c(1:10)
x
1 2 3 4 5 6 7 8 9 10

x > 8
F F F F F F F F T T

x < 5
T T T T F F F F F F

x > 8 | x < 5
T T T T F F F F T T

x[c(T,T,T,T,F,F,F,F,T,T)]
1 2 3 4 9 10
```