Lecture 1

IAQS | INSTITUTE OF ACTUARIAL & QUANTITATIVE STUDIES

Class: SY BSc

Subject : Statistical modelling in R

Chapter: Unit 1 Chp 1

Chapter Name: Data Management

Today's Agenda

1. Data Management
2. Learning the need for data management on databases before performing analysis. Performing
3. operations such as splitting data, addition of derived variables, replacing values ,fixing invalid values
4. and removing duplicates in R.

What are Variables?

1. Variables are used to store values in memory.
2. We can store integers, decimals, characters, words or sentences in a variable.
3. In order to assign value to a variable, use the **equal(=)** sign.
4. x = 1
5. y = 2.0
6. name = “Indrani Sen“

Rules for writing identifiers in R

- Identifiers can be a combination of letters ,digits, periods,under scores
- It must start with a letter or period
- If it starts with a period it cant be followed by a digit
- Reserved words in R cannot be used as identifiers

Valid variables

> .x1=100
> p_123=200
> q_=20
> result_mean=20

Invalid variables

```
> .1x=10
Error: unexpected symbol in ".1x"
> results$=10
Error: unexpected '=' in "results$="
> e@mail="indrani@gmail.com"
Error in getClass(cl) : “eigen” is not a defined class
```

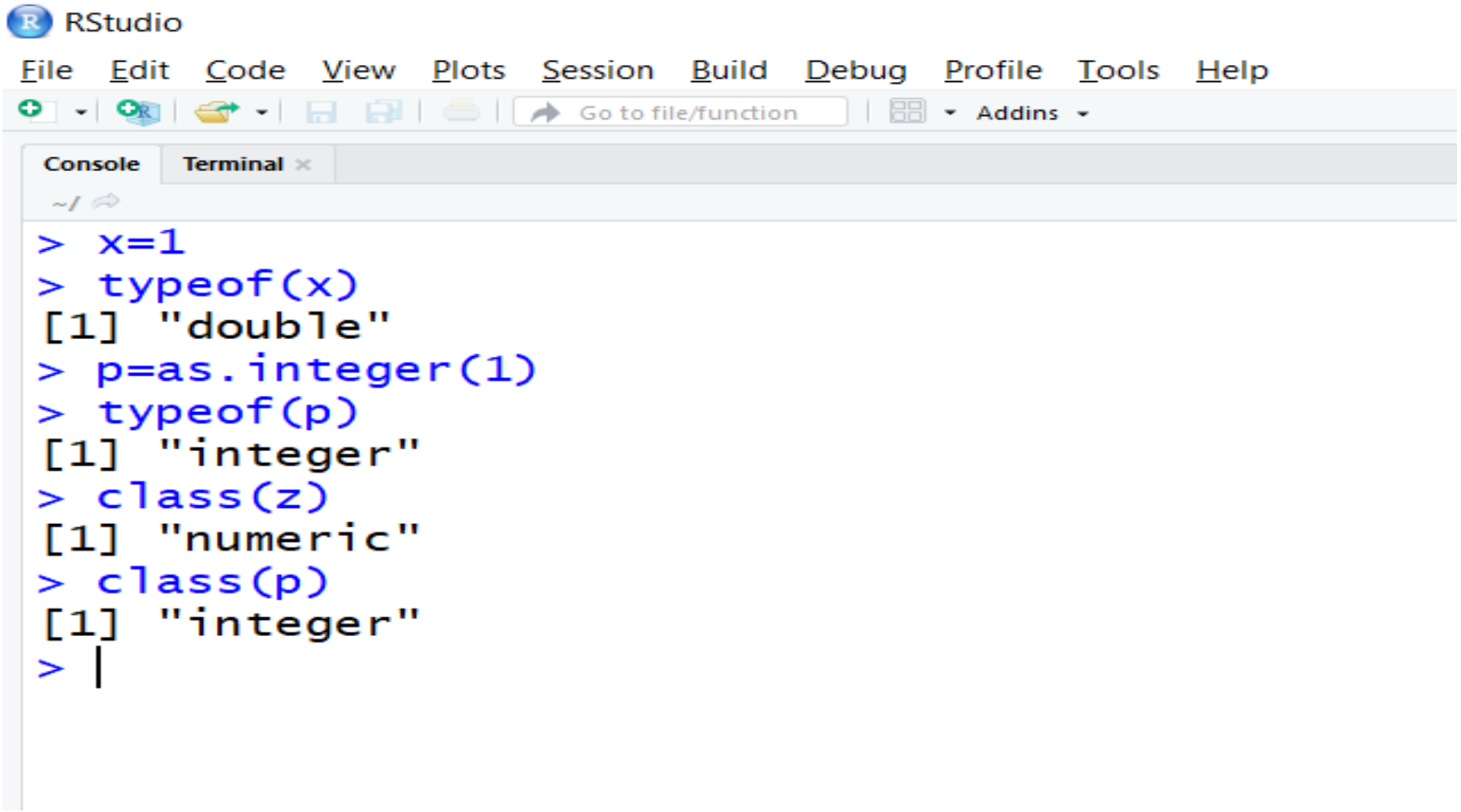
Data types

- character: “cloudy”, “male”
- numeric: 1,0,80
- integer: 2L (the L tells R to store this as an integer)
- logical: TRUE, FALSE
- complex: 1+4i (complex numbers with real and imaginary parts)

Data types

- character: “cloudy”, “male”
- numeric: 1,0,80
- integer: 2L (the L tells R to store this as an integer)
- logical: TRUE, FALSE
- complex: 1+4i (complex numbers with real and imaginary parts)

Data types


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console
Terminal
~/
> x=1
> typeof(x)
[1] "double"
> p=as.integer(1)
> typeof(p)
[1] "integer"
> class(z)
[1] "numeric"
> class(p)
[1] "integer"
>

Data structures

- R's base data structures can be organized by their dimensionality (1d, 2d, or nd) and
- whether they're homogeneous (all contents must be of the same type) or heterogeneous (the contents can be of different types).

Data structures

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

Vector Data structures

- The basic data structure in R is the vector.
- Vectors come in two flavors:
- Atomic vectors and lists.
- They have three common properties:
  - Type, typeof(), what it is.
  - Length, length(), how many elements it contains.
  - Attributes, attributes(), additional arbitrary metadata.

Attributes?

```
> v1=c(10,20,30,40,50)
> attributes(v1)
NULL
> names(v1)=1:5
> v1
 1  2  3  4  5
10 20 30 40 50
```

Some attributes:

class, comment, dim, dimnames, names,

row.names

Attributes?

```
> names(v1)=letters[1:5]

> v1

 a  b  c  d  e

10 20 30 40 50

> comment(v1)="This is a 5 number list
equally spaced at 10"
```

```
> attributes(v1)

$names

[1] "a" "b" "c" "d" "e"


$comment

[1] "This is a 5 number list equally
spaced at 10"
```

Data structures

- There are four common types of atomic vectors :
- Integer
- double (often called numeric)
- Character
- logical
- Complex

Double

```
> dbl_var=c(1.25,2.34,4.56,5.67,2.333)

> print(dbl_var)

[1] 1.250 2.340 4.560 5.670 2.333

> typeof(dbl_var)

[1] "double"

> class(dbl_var)

[1] "numeric"

>
```

Integer

By default numeric variables are double

You have to create Integer variables after suffixing with L

```
> print(int_var)
[1]   1   4   5   6   7   8   9 101
> typeof(int_var)
[1] "double"
> int_var=c(1L,4L,5L,6L,7L,8L,9L,101L)
> typeof(int_var)
[1] "integer"
> class(int_var)
[1] "integer"
>
```

Creating numeric vector with range

```
> var_num_list=c(1:20)

> print(var_num_list)

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

> typeof(var_num_list)

[1] "integer"
```

Seq() method

Generate regular sequences. seq is a standard generic with a default method.

Arguments

from, to: the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument.

By : number: increment of the sequence.

length.out :desired length of the sequence. A non-negative number, which for seq and seq.int will be rounded up if fractional.

along.with: take the length from the length of this argument.

Character vectors

```
v_char=c("Mumbai","Kolkata","Srinagar","Delhi","Lukhnow")
print(v_char)
[1] "Mumbai" "Kolkata" "Srinagar" "Delhi" "Lukhnow"
> typeof(v_char)
[1] "character"
> class(v_char)
[1] "character"
> length(v_char)
[1] 5
> is.character(v_char)
[1] TRUE
> is.integer(v_char)
[1] FALSE
```

Coercion

Coercion. When you call a function with an argument of the wrong type,

R will try to coerce values to a different type so that the function will work.

Values are converted to the simplest type required to represent all information.

The ordering is roughly logical < integer < numeric < complex < character < list.

Example of coercion (numeric to character)

```
> v_str=c("Mumbai",101)
> print(v_str)
[1] "Mumbai" "101"
> v_int=c(100,"1",200,"4")
> print(v_int)
[1] "100" "1"   "200" "4"
> typeof(v_str)
[1] "character"
> typeof(v_int)
[1] "character"
```

Logical to numeric

```
v_logic=c(TRUE,FALSE,T,F,1,0)
> print(v_logic)
[1] 1 0 1 0 1 0
> v_logic=c(TRUE,FALSE,T,F,100,200)
> print(v_logic)
[1] 1 0 1 0 100 200
```

Aggregate functions with numeric vector

```
➢ var_num=c(1L,2L,5.6,7.8,9.78)
➢ > print(var_num)
➢ [1] 1.00 2.00 5.60 7.80 9.78
➢ > typeof(var_num)
➢ [1] "double"
➢ > class(var_num)
➢ [1] "numeric"
```

```
>
```

```
> sum(var_num)

[1] 26.18

>mean(var_num)

[1] 5.236

>median(var_num)

[1] 5.6

> max(var_num)

[1] 9.78

> min(var_num)

[1] 1
```

Aggregate functions with logical vector

When a logical vector is coerced to an integer or double,

TRUE becomes 1 and FALSE becomes 0.

This is very useful in conjunction with sum() and mean()

Sum() gives the total number of TRUE values

Mean() give the % of values which are true

Lists

Lists are different from atomic vectors because their elements can be of any type, including lists.

You construct lists by using list() instead of c():

Example student list

```
> student=list(101,c("Sunil"),c(80,45,90,76,56),c("O+"),5.3,TRUE)
> print(student)
[[1]]
[1] 101
[[2]]
[1] "Sunil"
[[6]]
[1] TRUE
```

Str() shows the structure of data

```
> str(student)
List of 6
 $ : num 101
 $ : chr "Sunil"
 $ : num [1:5] 80 45 90 76 56
 $ : chr "O+"
 $ : num 5.3
 $ : logi TRUE
>
```

Converting a list into atomic vectors

```
> unlist(student)
 [1] "101"   "Sunil" "80"    "45"    "90"    "76"    "56"    "O+"    "5.3"
"TRUE"
```

Factors

- One important use of attributes is to define factors.
- A factor is a vector that can contain only predefined values,
- and is used to store categorical data.
- Factors are built on top of integer vectors using two attributes:
- the class, “factor”, which makes them behave differently from regular integer vectors

Examples

```
> gender=factor(c("male","female","male","female","male","male"))
> print(gender)
[1] male   female male   female male   male
Levels: female male
> gender[7]="female"
> gender[8]="F"
Warning message:
In `[<-.factor`(`*tmp*`, 8, value = "F") :
  invalid factor level, NA generated
```

Matrices and arrays

- A special case of the array is the matrix,
- which has two dimensions.
- Matrices are used commonly as part of the mathematical machinery of statistics.
- Matrices and arrays are created with matrix() and array(),
- or by using the assignment form of dim():

Creating matrix type1

```
> mat_a=matrix(c(2,4,5,2,6,1),ncol=3,nrow=2)

> mat_a

     [,1] [,2] [,3]
[1,]    2    5    6
[2,]    4    2    1
```

Creating matrix type2

```
> mat_b=matrix(2:7,ncol=3,nrow=2)
> mat_b
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    5    7
```

Creating matrix type 3 using rep()

```
> mat_c=matrix(c(rep(4,2),rep(6,4)),ncol=3,nrow=2)

> mat_c

     [,1] [,2] [,3]

[1,]    4    6    6

[2,]    4    6    6
```

Creating matrix type 4 using seq()

```
> mat_d=matrix(seq(from=1,to=10,length.out=6),ncol=3,nrow=2)
> mat_d
     [,1] [,2] [,3]
[1,]  1.0  4.6  8.2
[2,]  2.8  6.4 10.0
```

Reshaping the matrix

```
> dim(mat_d)=c(3,2)
> mat_d
     [,1] [,2]
[1,]  1.0  6.4
[2,]  2.8  8.2
[3,]  4.6 10.0
> str(mat_d)
 num [1:3, 1:2] 1 2.8 4.6 6.4 8.2 10
```