Data Types

1)Vectors

2)Lists

3)Matrices

4)Arrays

5)Factors

6)Data Frames

Vectors

- Vector is a sequence of data elements of the same basic type.
- There are 5 Atomic vectors, also called as five classes of vectors.
  - Character (‘a’, “Hello World!”, ‘False’)
  - Integer (3L, 0L)
  - Numeric (234, 3.1456)
  - Logical (True, False)
  - Complex (3+4i)
- Operator to construct a vector `c(,,,,)`
- Sequence operations (Indexing, replacing, sorting)

Basic Data

- `X <- 1           # numeric type`
- `X <- 2030L       # Integer type`
- `msg <- "hello" # character type`
- `X <- 10:30       # Integer sequence or vector`
- `C(0.5, 0.6)      # numeric vector`
- `c(TRUE, FALSE) # logical vector`
- `c(T, F)          # logical vector`
- `c("a", "b", "c") # character`
- `c(1+0i, 2+4i)      # complex`
- `x <- vector("numeric", length = 10)`

  `[1] 0 0 0 0 0 0 0 0 0 0`

Basic Data: implicit class coercion

- `y <- c(1.7, "a")   #character`
- `y <- c(TRUE, 2)    #numeric`
- `y <- c("a", TRUE) #character`

- **`logical < integer < double < complex < character`**

Basic Data: Explicit class coercion

- `x <- 0:6`
- `class(x)`

  `[1] "integer"`
- `as.numeric(x)`

  `[1] 0 1 2 3 4 5 6`
- `as.logical(x)`

  `[1] FALSE TRUE TRUE TRUE TRUE`
- `as.character(x)`

  `[1] "0" "1" "2" "3" "4" "5" "6"`

Basic Data: Explicit class coercion

- `x <- c("a", "b", "c")`
- `as.numeric(x)`

  ```
  Warning: NAs introduced by coercion
  [1] NA NA NA
  ```
- `as.logical(x)`

  ```
  [1] NA NA NA
  ```
- `as.complex(x)`

  ```
  Warning: NAs introduced by coercion
  [1] NA NA NA
  ```

Exercises

- What type will be output of:
  - x <- 1; x; x+1
  - y <- 2020L
  - 2+i
  - 2+1i
  - 2+0i
  - c1 <- complex(2,2,3); c1
  - “1.7” * 2
  - as.numeric("1.7")*2
  - x <- c(1.2,3L,4.3); x

Vector operations

- Construct a vector : `v=c(1,23,3,4)`
- Sort a vector: `sort(v)`
- Index a vector:
  `v[1], v[2:4], v[-1]` (excludes 1)
- Replace element:
  `v[1]=0, v[6]=34` (`v[5]` will be NA)

Matrices

- Matrices are vectors with a dimension attribute.
- `matrix(data, nrow, ncol, byrow, dimnames)`
- `m <- matrix(nrow = 2, ncol = 3)`
- `m`

```
        [,1]  [,2] [,3]
[1,]      NA    NA   NA
[2,]      NA    NA   NA
```

- `dim(m)`

```
[1] 2 3
```

- `attributes(m)`

```
$dim
[1] 2 3
```

Matrix construction-1

- Matrices are constructed column-wise, so entries can be thought of starting in the "upper left" corner and running down the columns.
- `m <- matrix(1:6, nrow = 2, ncol = 3)`
- `m`

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

Matrix construction-2

- Matrices can also be created directly from vectors by adding a dimension attribute..
- `m <- 1:10`
- `m`

```
[1] 1 2 3 4 5 6 7 8 9 10
```

- `dim(m) <- c(2, 5)`
- `m`

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Matrix construction-3

- Matrices can be created by column-binding or row-binding with the `cbind()` and `rbind()` functions.
- `x <- 1:3`
- `y <- 10:12`
- `cbind(x, y)  #What is the output?`
- `rbind(x, y)  #What is the output?`

Subsetting a Matrix

- `x <- matrix(1:6, 2, 3)`
- `x[1, 2]` #returns one element, dimensions are dropped
- `x[1, 2, drop = FALSE]` #returns matrix
- `x[1, ]` # Extract the first row as vector
- `x[1, , drop = FALSE]` #returns matrix
- `x[, 2]` # Extract the second column
- `x[, 2, drop = FALSE]`
- `X[1,2:3]` # any submatrix
- `rowSums(x)` #row sums
- `colSums(x)` #columns sums

Exercises

-
- Create a matrix with any continuous sequence of numbers of dimension 4x5
- Create a matrix of 2 x 3 with dimension names as r1,r2 and c1, c2,c3
- What is the output of the following:
- m1 <- matrix(3:20,3,8); m1
- m1 <- matrix(3:20,3,5); m1
- CHALLENGE: Try using paste function to construct the row and column names
- Create a matrix with a sequence of numbers arranged row-wise (row-major fashion)

Lists

- Mixing data types permitted in lists
- Mixing data types in vectors converts to lists
- Construct list using `list()` operator
- Merge lists: `lst3=merge(lst1, lst2) or c(lst1, lst2)`
- Many complicated functions produce list objects as their output. Least squares regression, the regression object output is a list.
- eg.
- `lm.xy <- lm(y~x, data = data.frame(x=1:5,y=1:5))`
- `class(lm.xy), mode(lm.xy), str(lm.xy), names(lm.xy)`

List construction-1

- `x <- list(1, "a", TRUE, 1 + 4i)`
- `x`

```
[[1]]
[1] 1
[[2]]
[1] "a"
[[3]]
[1] TRUE
[[4]]
[1] 1+4i
```

Subsetting list

- `x <- list(foo = 1:4, bar = 0.6)`

```
• x
$foo
[1] 1 2 3 4
$bar
[1] 0.6
• x[[1]]
[1] 1 2 3 4
• x[["bar"]]
[1] 0.6
• x$bar
[1] 0.6
```

```
• name <- "foo"

## computed index for "foo"
• x[[name]]
[1] 1 2 3 4

## element "name" doesn't exist!
##(but no error here)
• x$name
NULL

## element "foo" does exist
• x$foo
[1] 1 2 3 4
```

Subsetting nested elements of list

- x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))

Get the 3rd element of the 1st element

- x[[c(1, 3)]]

[1] 14

Same as above

- x[[1]][[3]]

[1] 14

1st element of the 2nd element

- x[[c(2, 1)]]

[1] 3.14

List construction-2

- We can also create an empty list of a pre-specified length with the `vector()` function
- `x <- vector("list", length = 5)`
- `x`

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

Exercises

- Create a list of different datatypes – integer, double,character, boolean, complex and the built-in function mean
- Create 4 vectors of lengths 2,3,4 and 1 and create a list with the two vectors as its two components with names
- Access the 2nd element of the 3rd vector (i.e. 2nd element of the 3rd component of the list)

Factors

- Factors are used to represent categorical data and can be unordered or ordered. One can think of a factor as an integer vector where each integer has a label.
- Factors are important in statistical modeling and are treated specially by modelling functions like `lm()` and `glm()` .
- Using factors with labels is better than using integers because factors are self-describing.
- Having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.
- Factor objects can be created with the `factor()` function.

Factors

- `x <- factor(c("yes", "yes", "no", "yes", "no"))`
- `x`

```
[1] yes yes no yes no
Levels: no yes
```

- `table(x)`

```
x
  no  yes
  2    3
```

- `## See the underlying representation of factor`
- `unclass(x)`

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

- `attr(x,"levels")`

```
[1] "no" "yes"
```

- `levels(x)`
- `[1] "no"  "yes"`

Factors

- Often factors will be automatically created for you when you read a dataset in using a function like `read.table()`
- Those functions often default to creating factors when they encounter data that look like characters or strings

Factors

- `x <- factor(c("yes","yes", "no", "yes", "no"))`
- `x ## Levels are put`

  `[1] yes yes no yes no`

  `Levels: no yes`
- `x <- factor(c("yes","yes", "no", "yes", "no"),`

  `+ levels = c("yes", "no"))`
- `x`

  `[1] yes yes no yes no`

  `Levels: yes no`

Exercises

- Create a vector p <- c(0,3,2,2,1) and convert it into a factor
- Call as.numeric function on the created factor & check output
- Check the levels of the factor using the function levels
- Check the description of the levels function
- CHALLENGE: Assign new levels to the factor using levels function and assignment operation using the vector c('a','d','b','c') & check the contents of the factor (notice any changes?)
- Again assign new levels to the factor of a sequence of numbers in reverse sorted order (e.g. 10,9,8,7)
- Use as.numeric function on the factor created and check output

Removing NA Values

A common task in data analysis is removing missing values (NA s).

- `x <- c(1, 2, NA, 4, NA, 5)`
- `bad <- is.na(x)`
- `print(bad)`

```
[1] FALSE FALSE TRUE FALSE TRUE FALSE
```

- `x[!bad]`

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

Removing NA Values-2

Multiple R objects have NA in the same positions. Subset with no missing values in any of those objects.

- `x <- c(1, 2, NA, 4, NA, 5)`
- `y <- c("a", "b", NA, "d", NA, "f")`
- `good <- complete.cases(x, y)`
- `good`

  `[1] TRUE TRUE FALSE TRUE FALSE TRUE`
- `x[good]`

  `[1] 1 2 4 5`
- `y[good]`

  `[1] "a" "b" "d" "f"`

Vectorized Operations

- `x <- 1:4`
- `y <- 6:9`
- `z <- x + y`
- `z`

  `[1] 7 9 11 13`

Otherwise you need to do this

```
z <- numeric(length(x))

for(i in seq_along(x)) {
        z[i] <- x[i] + y[i]
}

z
[1] 7 9 11 13
```

Vectorized Operations-2

- `x <- 1:4`
- `y <- 6:9`
- `x > 2`

  `[1] FALSE FALSE TRUE TRUE`
- `x >=2, y == 8`
- `x-y, x*y, x/y`

Exercises

- Given v1 containing elements 1, 24, 5, 9, 0
- What is the output of
  - v1 * 2
  - v1 * c(2,2,2,2)   (compare the output with the previous)
  - 1 / v1
  - v1<- v1[2:5]; v1 * c(2,3,2,3)
  - x <-c(1,2,3,4); y <- c(x,0,x); v2 <- 2*x + y + 1; v2
- Explore how the following functions are used with a numeric vector - length, seq, rep, names, min, max, sqrt, sum

Vectorized Matrix Operations-3

- `x <- matrix(1:4, 2, 2)`
- `y <- matrix(rep(10, 4), 2, 2)`

```
## element-wise multiplication
```

- `x * y  # likewise +,-,/`

```
     [,1] [,2]
[1,]   10   30
[2,]   20   40
```

```
## true matrix multiplication
```

- `x %*% y`

```
     [,1] [,2]
[1,]   40   40
[2,]   60   60
```

Index Vectors

- Index vector enables both selection (extraction) and replacement
  - s1<- 0:6
  - s1[1] <- 15; s1 #replacement
  - s1 > 4 | s1 < 2 # index vector created
  - s1[s1 > 4 | s1 < 2] #using index vector to do selection/extraction
  - s1 <- c('ab',1,NA); s1
  - !is.na(s1)
  - v1 <- c(2,5,10,15,23,42,51)
  - v1[v1%%2==0 | v1%%5==0]
- Logical AND and OR operator (element-wise operations): & is AND operator and | is OR operator

Exercises

- What is the output of
  - v1 <-c('1','4','b')
  - as.numeric(v1[!is.na(as.integer(v1))])
  - x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
  - sum((x-mean(x))^2)/(length(x)-1)
  - (x+1)[(!is.na(x)) & x>0] -> z
- Use paste function to create the vector
  - c("X1", "Y2", "X3", "Y4", "X5", "Y6", "X7", "Y8", "X9", "Y10")

Operator Precedence

Operator	Explanation
^	exponentiation (right to left)
- +	unary minus and plus
:	sequence operator
%any%	special operators (including %% and %/%) %*% (mat mul) %/% integer division %% mod oper
* /	multiply, divide
+ -	(binary) add, subtract
< > <= >= == !=	ordering and comparison (relational)
!	negation
& &&	and (&& shortcircuit opertypically used in if statements, & is vectorized operator)
| ||	or
-> ->>	rightwards assignment
<- <<-	assignment (right to left)
=	assignment (right to left)

In descending order (highest on top)

Exercises

- Compute the value and explain the precedence to yourself for the following:
  - 2+3^2
  - 2+3^2^3
  - x <- y <- z <- 0; x; y; z
  - 3+3/2
  - 2:3+2
  - 2:-3
  - 2 > 2:-3