Northeastern University

CS6020: Collecting, Storing, and Retrieving Information

Basic Data Shaping

Basic Data Shaping

R OBJECTS

Lesson Objectives

- After completing this lesson, you are able to:
  - create and use vectors, factors, data frames, matrices, arrays, and lists
  - handle missing values

Basic Data Shaping

R OBJECTS: VECTOR

Vector

- The vector is the basic storage type in R.
- A vector holds a collection of values of the same type: numeric, logical, character.
- A vector is treated like an array and its elements can be accessed by indexing.

```
> num.vec <- c(1,3,5,8,13,21)
> num.vec
[1]  1  3  5  8 13 21
> num.vec[3]
[1] 5
> is.vector(num.vec)
[1] TRUE
```

Basic Data Shaping

R OBJECTS: FACTOR

Factor

- The factor type is used to encode categorical data values.
- While a vector can have any number of distinct elements, a factor value is limited to its categories.
- Factors are essential for certain statistical hypothesis tests and models.
- Factors are stored as numbers internally.

Creating Factors

- Factors are created using the `factor()` function which requires a vector of category values as input.

```
> f1 <- factor(c("AA","BA","JB","CO"))
> f1
[1] AA BA JB CO
Levels: AA BA CO JB
> labels(f1)
[1] "1" "2" "3" "4"
> levels(f1)
[1] "AA" "BA" "CO" "JB"
```

Special Considerations

- While factors are stored as unique numeric labels they are not, in fact, of numeric type.
- Therefore, factors cannot be used in numeric operations.

```
> mean(f1)
[1] NA
Warning message:
In mean.default(f1) : argument is not numeric or
logical: returning NA
```

Basic Data Shaping

R OBJECTS: DATA FRAME

Data Frame

- A data frame is a tabular arrangement of rows and columns of vectors and/or factors similar to a spreadsheet.
- The columns represent data attributes while the rows represent records with values for the attributes.
- The vectors making up the data frame must be of the exact same length.

Creating Data Frames

- A data frame is created with the `data.frame()` function requiring the different vectors as input.
- The input vectors represent the columns of the tabular arrangement.

Example: Creating a Data Frame

```
> var1 <- c("BOS","EWR","PBI","CVG")
> var2 <- c(14,19,0,12)
> var3 <- factor(c("Above","Above","Normal","Below"))
> snow.frame <- data.frame(var1,var2,var3)
> snow.frame
  var1 var2   var3
1  BOS   14  Above
2  EWR   19  Above
3  PBI    0 Normal
4  CVG   12  Below

> mode(snow.frame)
[1] "list"
> class(snow.frame)
[1] "data.frame"
```

Accessing Columns

- Columns can be directly accessed using the $ as the column operator: `frame$col`

```
> snow.frame$var1
[1] BOS EWR PBI CVG
Levels: BOS CVG EWR PBI
> mode(snow.frame$var1)
[1] "numeric"
> class(snow.frame$var1)
[1] "factor"
> snow.frame$var2
[1] 14 19  0 12
> mode(snow.frame$var2)
[1] "numeric"
> class(snow.frame$var2)
[1] "numeric"
```

Alternative Column Access

- Suppose you have the data frame "`snow.frame`" with column "`var1`".
- It's columns can be accessed in three ways:
  1. `snow.frame$var1`
  2. `snow.frame[["var1"]]`
  3. `snow.frame[,1]`

```
> snow.frame$var1
[1] BOS EWR PBI CVG
Levels: BOS CVG EWR PBI
> snow.frame[["var1"]]
[1] BOS EWR PBI CVG
Levels: BOS CVG EWR PBI
> snow.frame[,1]
[1] BOS EWR PBI CVG
Levels: BOS CVG EWR PBI
```

Basic Data Shaping

R OBJECTS: MATRIX

Matrices

- A matrix is a two-dimensional arrangement of data similar to a data frame but unlike a data frame its elements must be of the same data type.
- To perform mathematical operations on matrices, its elements must be numeric.

Creating Matrices

- A matrix can be created with:
  - `matrix()`

```
> m <- matrix(rnorm(9),nrow=3,ncol=3)
> m
           [,1]       [,2]      [,3]
[1,] -0.3120017 -1.1159042 -1.266338
[2,]  0.5766845  1.0319907 -0.250442
[3,] -0.2938791  0.6939539 -1.158165
> mode(m)
[1] "numeric"
> class(m)
[1] "matrix"
```

  - `as.matrix()`

```
> m <- as.matrix(snow.frame)
> mode(m)
[1] "character"
> class(m)
[1] "matrix"
```

Transposing a Matrix

- The function `t()` creates a transpose of a matrix.

```
> m
           [,1]       [,2]      [,3]
[1,] -0.3120017 -1.1159042 -1.266338
[2,]  0.5766845  1.0319907 -0.250442
[3,] -0.2938791  0.6939539 -1.158165
> t(m)
           [,1]       [,2]       [,3]
[1,] -0.3120017  0.5766845 -0.2938791
[2,] -1.1159042  1.0319907  0.6939539
[3,] -1.2663379 -0.2504420 -1.1581652
```

Matrix Multiplication

- Two matrices can be multiplied using the `%*%` matrix multiplication operator.

```
> t(m) %*% m
          [,1]      [,2]      [,3]
[1,] 0.5162750 0.7393584 0.5910342
[2,] 0.7393584 2.7918189 0.3509447
[3,] 0.5910342 0.3509447 3.0076795
```

Basic Data Shaping

R OBJECTS: ARRAY

Arrays

- An array is a multi-dimensional data structure, while matrices and data frames are two-dimensional row/column arrangements and vector is a single dimension only.

Example Array

```
> a <- array(dim=c(2,2,3))
> a[,,1] <- rnorm(2)
> a[,,2] <- rnorm(2)
> a[,,3] <- rnorm(2)
> a
, , 1

          [,1]      [,2]
[1,] 0.5370590 0.5370590
[2,] 0.5897154 0.5897154

, , 2

            [,1]        [,2]
[1,]  0.86947620  0.86947620
[2,] -0.05387594 -0.05387594

, , 3

           [,1]       [,2]
[1,] -1.4424228 -1.4424228
[2,] -0.9510549 -0.9510549
```

Accessing Array Elements

- Array elements are accessed through subscripting:

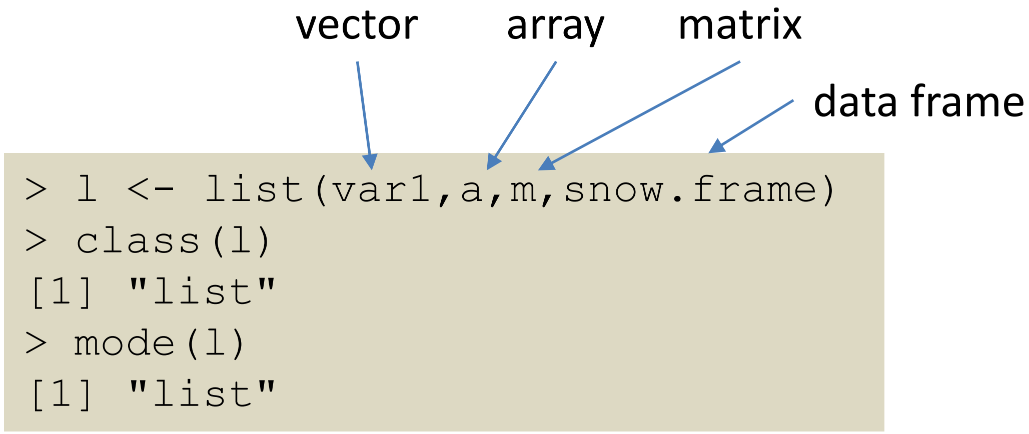
```
> a[1,1,1]
[1] 0.537059
> a[1,1,2]
[1] 0.8694762

> mode(a)
[1] "numeric"
> class(a)
[1] "array"
```

Basic Data Shaping

R OBJECTS: LIST

Lists

- A `list` object is a generic collection that can store objects of any type, including vectors, matrices, arrays, and data frames.
- This is essentially a "bag" data structure.



vector

array

matrix

data frame

```
> l <- list(var1,a,m,snow.frame)
> class(l)
[1] "list"
> mode(l)
[1] "list"
```

Example List

```
> l <- list(var1,a,m,snow.frame)
> l
[[1]]
[1] "BOS" "EWR" "PBI" "CVG"

[[2]]
, , 1

          [,1]      [,2]
[1,] 0.5370590 0.5370590
[2,] 0.5897154 0.5897154

, , 2

            [,1]        [,2]
[1,]  0.86947620  0.86947620
[2,] -0.05387594 -0.05387594

, , 3

           [,1]       [,2]
[1,] -1.4424228 -1.4424228
[2,] -0.9510549 -0.9510549


[[3]]
           [,1]       [,2]      [,3]
[1,] -0.3120017 -1.1159042 -1.266338
[2,]  0.5766845  1.0319907 -0.250442
[3,] -0.2938791  0.6939539 -1.158165

[[4]]
  var1 var2   var3
1  BOS   14  Above
2  EWR   19  Above
3  PBI    0 Normal
4  CVG   12  Below
```

Accessing List Elements

- To access list elements, use indexing:

```
> l[[3]]
           [,1]       [,2]      [,3]
[1,] -0.3120017 -1.1159042 -1.266338
[2,]  0.5766845  1.0319907 -0.250442
[3,] -0.2938791  0.6939539 -1.158165

> l[3]
[[1]]
           [,1]       [,2]      [,3]
[1,] -0.3120017 -1.1159042 -1.266338
[2,]  0.5766845  1.0319907 -0.250442
[3,] -0.2938791  0.6939539 -1.158165
```

Summary

- In this lesson, you learned that:
  - vectors are similar to one dimensional arrays
  - data frames are two-dimensional with a row/column layout in which each column must be of the same data type
  - matrices are two-dimensional numeric data structures that can be operated upon
  - arrays are multi-dimensional data structures where each element is of the same data type
  - lists are similar to a bag data structures

Summary, Review, & Questions…