Chapter 7
Data Manipulations

Arthur Li

Subsetting Data Frames
Subsetting Data Frames By Using Index Vectors

❖ Use, **`painters`**, from the **`MASS`** library

```
> library(MASS)
> head(painters)
               Composition Drawing Colour Expression School
Da Udine                10       8     16          3      A
Da Vinci                15      16      4         14      A
Del Piombo               8      13     16          7      A
Del Sarto               12      16      9          8      A
Fr. Penni                0      15      8          0      A
Guilio Romano           15      16      4         14      A
> dim(painters)
[1] 54  5
```

Subsetting Data Frames By Using Index Vectors

- Selecting the observations from a data frame is similar to selecting rows from a matrix

- To select observations with Colour ≥ 17

```
> painters[painters$Colour >=17,]
          Composition Drawing Colour Expression School
Bassano             6       8     17          0      D
Giorgione           8       9     18          4      D
Pordenone           8      14     17          5      D
Titian             12      15     18          6      D
Rembrandt          15       6     17         12      G
Rubens             18      13     17         17      G
Van Dyck           15      10     17         13      G
```

Subsetting Data Frames By Using Index Vectors

❖ To select ones from school “A” or “D”, use **`is.element`**

```
> painters[is.element(painters$School, c("A", "D")),]
                Composition Drawing Colour Expression School
Da Udine                 10       8     16          3      A
Da Vinci                 15      16      4         14      A
Del Piombo                8      13     16          7      A
...
...
Bassano                   6       8     17          0      D
Bellini                   4       6     14          0      D
Giorgione                 8       9     18          4      D
...
```

❖ **`is.element(painters$School, c("A", "D")) ≠ painters$School == c("A", "D")`**

Subsetting Data Frames By Using Index Vectors

- Selecting variables from a data frame is also similar to selecting columns from a matrix

- To select variables: use the column index or variable names

- The following 4 statements are equivalent

```
> d1 <- painters[, c("School", "Colour")]
> d2 <- painters[c("School", "Colour")]
> d3 <- painters[, c(5, 3)]
> d4 <- painters[c(5, 3)]
```

Subsetting Data Frames By Using Index Vectors

- You can also select observations and variables at the same time

```
> painters[painters$School == "A", c("School", "Colour")]
                School Colour
Da Udine             A     16
Da Vinci             A      4
Del Piombo           A     16
Del Sarto            A      9
Fr. Penni            A      8
Guilio Romano        A      4
Michelangelo         A      4
Perino del Vaga      A      7
Perugino             A     10
Raphael              A     12
```

The `subset` Function

- **`Subset:`** create a data frame by selecting observations and variables

- Two important arguments:
  - **`subset`**: a logical expression that is used to select rows; Missing values are taken as false
  - **`select`**: an expression that is used to select columns

```
> subset(painters, Colour >= 17)
          Composition Drawing Colour Expression School
Bassano             6       8     17          0      D
Giorgione           8       9     18          4      D
Pordenone           8      14     17          5      D
Titian             12      15     18          6      D
Rembrandt          15       6     17         12      G
Rubens             18      13     17         17      G
Van Dyck           15      10     17         13      G
```

The `subset` Function

- The **`select`** argument can be specified as one of the following forms:
  - A vector of integers:

    ```
    select = c(1,2)
    ```
  - A vector of variable names:

    ```
    select = c(Composition, Drawing)
    ```
  - A negative sign can be used

    ```
    select = -c(Composition, Drawing)
    select = -c(1,2)
    ```
  - A ranges of columns can be selected by using (:)

    ```
    select = 1:3
    select = Composition:Colour
    ```

The `subset` Function

```
> subset(painters, Colour >= 17, 1:3)
          Composition Drawing Colour
Bassano             6       8     17
Giorgione           8       9     18
Pordenone           8      14     17
Titian             12      15     18
Rembrandt          15       6     17
Rubens             18      13     17
Van Dyck           15      10     17
```

Creating and Re-coding Variables
Creating Variables By Using the Relational Operators

- ❖ Creating indicator variables based on existing cont./cat. var.
- ❖ The indicator variable can be a logical or an integer vector
- ❖ A “logical” indicator variable is sufficient for modeling
- ❖ A logical vector → an integer vector, use **`as.integer`**

Creating Variables By Using the Relational Operators

❖ Example: create an indicator variable – indicating above or below mean of **`Drawing`**

```
> painters$DrawingInd = painters$Drawing >= mean(painters$Drawing)
> head(painters)
              Composition Drawing Colour Expression School  DrawingInd
Da Udine               10       8     16          3      A       FALSE
Da Vinci               15      16      4         14      A        TRUE
Del Piombo              8      13     16          7      A        TRUE
Del Sarto              12      16      9          8      A        TRUE
Fr. Penni               0      15      8          0      A        TRUE
Guilio Romano          15      16      4         14      A        TRUE
```

Creating Variables By Using the Relational Operators

❖Use the **`transform`** function to create a new variable

```
> paintersNew = transform(painters, DrawingInd = Drawing>mean(Drawing))
> head(paintersNew)
              Composition Drawing Colour Expression School  DrawingInd
Da Udine               10       8     16          3      A       FALSE
Da Vinci               15      16      4         14      A        TRUE
Del Piombo              8      13     16          7      A        TRUE
Del Sarto              12      16      9          8      A        TRUE
Fr. Penni               0      15      8          0      A        TRUE
Guilio Romano          15      16      4         14      A        TRUE
```

Creating Variables By Using the Relational Operators

❖Create an ordinal variable with more than two levels

```
> painters$CompositionOrd = 1 + (painters$Composition >= 8) +
              (painters$Composition >= 12)
> head(painters[c("Composition", "CompositionOrd")])
             Composition CompositionOrd
Da Udine              10              2
Da Vinci              15              3
Del Piombo             8              2
Del Sarto             12              3
Fr. Penni              0              1
Guilio Romano         15              3
```

```
> tapply(painters$Composition, painters$CompositionOrd, min)
1  2  3
0  8 12
> tapply(painters$Composition, painters$CompositionOrd, max)
1  2  3
6 11 18
```

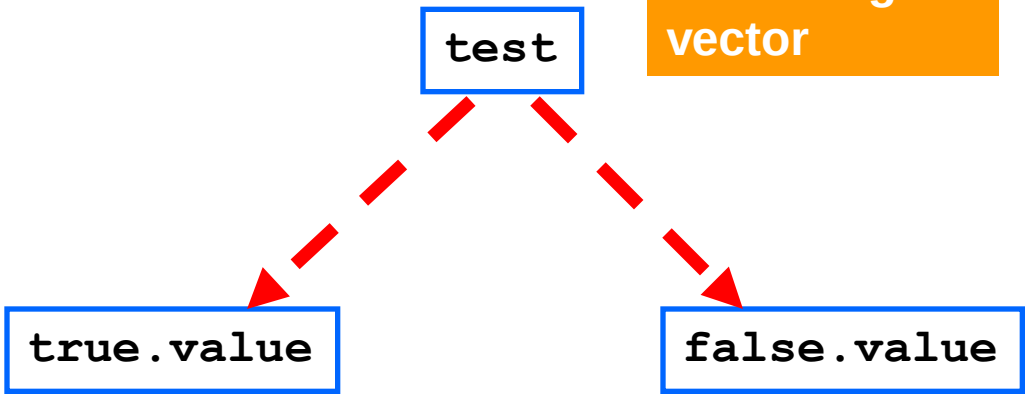
Creating Variables By Using the `ifelse` Function

❖ The **`ifelse`** function has the following form:

```
ifelse (test, true.value, false.value)
```

All the arguments are vectors

test: A logical vector



Creating Variables By Using the `ifelse` Function

```
> table(painters$School)
 A  B  C  D  E  F  G  H
10  6  6 10  7  4  7  4
```

❖ To create an ordinal variable that equals to 1 if the School variable equals to A, B or C, and equals to 2 otherwise

```
> School2 = ifelse(is.element(painters$School, c("A", "B", "C")),
+ 1, 2)
> table(School2, painters$School)
School2  A  B  C  D  E  F  G  H
      1 10  6  6  0  0  0  0  0
      2  0  0  0 10  7  4  7  4
```

Creating Variables By Using the `ifelse` Function

❖ You nest an **`ifelse`** function within another **`ifelse`** function

```
> School3 = ifelse(is.element(painters$School, c("A", "B", "C")),
+ 1, ifelse(is.element(painters$School, c("D", "E")), 2, 3))
> table(School3, painters$School)
School3  A  B  C  D  E  F  G  H
      1 10  6  6  0  0  0  0  0
      2  0  0  0 10  7  0  0  0
      3  0  0  0  0  0  4  7  4
```

Creating a Factor By Using the `cut` Function

❖ Create a factor based on a continuous variable: **`cut`**

```
cut(x, breaks, labels =NULL, include.lowest = FALSE, right = TRUE)
```

The variable that to be divided

Creating a Factor By Using the `cut` Function

❖ Create a factor based on a continuous variable: **`cut`**

```
cut(x, breaks, labels =NULL, include.lowest = FALSE, right = TRUE)
```

A numeric vector of 2 or more cut points.
E.g. breaks = c(n1, n2, n3, n4) →
x is divided into:
(n1, n2], (n2, n3], and (n3, n4]
Note: n1 is not included

Creating a Factor By Using the `cut` Function

❖ Create a factor based on a continuous variable: **`cut`**

```
cut(x, breaks, labels =NULL, include.lowest = FALSE, right = TRUE)
```

labels = NULL:
The resulting factor is labeled as (n1, n2], (n2, n3], and (n3, n4]
It's better provide your own label!

Creating a Factor By Using the `cut` Function

❖ Create a factor based on a continuous variable: **`cut`**

```
cut(x, breaks, labels =NULL, include.lowest = FALSE, right = TRUE)
```

If setting **`include.lowest = TRUE`**,
x is divided into:
[n1, n2], (n2, n3], and (n3, n4]

Creating a Factor By Using the `cut` Function

❖ Create a factor based on a continuous variable: **`cut`**

```
cut(x, breaks, labels =NULL, include.lowest = FALSE, right = TRUE)
```

If setting **`right = FALSE,`** x is divided into:
[n1, n2), [n2, n3), and [n3, n4)

Creating a Factor By Using the `cut` Function

❖ Example: create a 4-level factor based on **`Colour`** variable

```
> qt = quantile(painters$Colour, c(0, 0.25, 0.5, 0.75, 1))
> qt
   0%   25%   50%   75%  100%
 0.00  7.25 10.00 16.00 18.00
> painters$ColourCat = cut(painters$Colour, qt, labels = c("1st",
+ "2nd", "3rd", "4th"), include.lowest=T)
> head(painters)
              Composition Drawing Colour Expression School  ColourCat
Da Udine               10       8     16          3      A        3rd
Da Vinci               15      16      4         14      A        1st
Del Piombo              8      13     16          7      A        3rd
Del Sarto              12      16      9          8      A        2nd
Fr. Penni               0      15      8          0      A        2nd
Guilio Romano          15      16      4         14      A        1st
```

Re-coding Missing Values

❖ Sometimes, we need to recode ’99’ or ‘999’ to NA. E,g.

```
> exampleMissing
   x1 x2
1   1  0
2   2  3
3   3  3
4  99  0
5  10 20
6 999  0
> exampleMissing$x1[is.element(exampleMissing$x1, c(99, 999))] = NA
> exampleMissing
  x1 x2
1  1  0
2  2  3
3  3  3
4 NA  0
5 10 20
6 NA  0
```

Sorting an Object
Sorting a Vector

❖ The **`sort`** function take either numeric or character vector

```
> x = c(5, 2, 1)
> sort(x)
[1] 1 2 5
> sort(x, decreasing = TRUE)
[1] 5 2 1
```

Sorting a Data Frame

- ❖ Create an index vector by using the **`order`** function first
- ❖ Then use this index vector to sort the data frame
- ❖ To illustrate some example, let's create a small data set

```
> paintersABC = painters[is.element(painters$School, c("A", "B", "C")),]
> paintersABC
                Composition Drawing Colour Expression School
Da Udine                 10       8     16          3      A
Da Vinci                 15      16      4         14      A
Del Piombo                8      13     16          7      A
...
F. Zucarro               10      13      8          8      B
Fr. Salviata             13      15      8          8      B
...
Barocci                  14      15      6         10      C
Cortona                  16      14     12          6      C
Josepin                  10      10      6          2      C
...
```

Sorting a Data Frame

❖ Example: sort by row names

```
> nameIndex = order(row.names(paintersABC))
> nameIndex
 [1] 17 18  1  2  3  4 11  5 12  6 19 20  7 13  8  9 14 10 15 21 22 16
> paintersABC[nameIndex, ]
                Composition Drawing Colour Expression School
Barocci                  14      15      6         10      C
Cortona                  16      14     12          6      C
Da Udine                 10       8     16          3      A
Da Vinci                 15      16      4         14      A
Del Piombo                8      13     16          7      A
Del Sarto                12      16      9          8      A
F. Zucarro               10      13      8          8      B
Fr. Penni                 0      15      8          0      A
Fr. Salviata             13      15      8          8      B
Guilio Romano            15      16      4         14      A
Josepin                  10      10      6          2      C
L. Jordaens              13      12      9          6      C
Michelangelo              8      17      4          8      A
Parmigiano               10      15      6          6      B
Perino del Vaga          15      16      7          6      A
Perugino                  4      12     10          4      A
...
```

Sorting a Data Frame

❖ Example: Sort by **`School`** in decreasing order

```
> paintersABC[order(paintersABC$School, decreasing = T), ]
                Composition Drawing Colour Expression School
Barocci                  14      15      6         10      C
Cortona                  16      14     12          6      C
Josepin                  10      10      6          2      C
L. Jordaens              13      12      9          6      C
Testa                    11      15      0          6      C
Vanius                   15      15     12         13      C
F. Zucarro               10      13      8          8      B
Fr. Salviata             13      15      8          8      B
Parmigiano               10      15      6          6      B
Primaticcio              15      14      7         10      B
T. Zucarro               13      14     10          9      B
Volterra                 12      15      5          8      B
Da Udine                 10       8     16          3      A
Da Vinci                 15      16      4         14      A
Del Piombo                8      13     16          7      A
Del Sarto                12      16      9          8      A
Fr. Penni                 0      15      8          0      A
Guilio Romano            15      16      4         14      A
Michelangelo              8      17      4          8      A
Perino del Vaga          15      16      7          6      A
Perugino                  4      12     10          4      A
Raphael                  17      18     12         18      A
```

Sorting a Data Frame

❖ We can sort the data by more than one variable

```
> paintersABC[order(paintersABC$School, paintersABC$Drawing), ]
                Composition Drawing Colour Expression School
Da Udine                 10       8     16          3      A
Perugino                  4      12     10          4      A
Del Piombo                8      13     16          7      A
Fr. Penni                 0      15      8          0      A
Da Vinci                 15      16      4         14      A
Del Sarto                12      16      9          8      A
Guilio Romano            15      16      4         14      A
Perino del Vaga          15      16      7          6      A
Michelangelo              8      17      4          8      A
Raphael                  17      18     12         18      A
F. Zucarro               10      13      8          8      B
Primaticcio              15      14      7         10      B
T. Zucarro               13      14     10          9      B
Fr. Salviata             13      15      8          8      B
Parmigiano               10      15      6          6      B
Volterra                 12      15      5          8      B
Josepin                  10      10      6          2      C
L. Jordaens              13      12      9          6      C
Cortona                  16      14     12          6      C
Barocci                  14      15      6         10      C
Testa                    11      15      0          6      C
Vanius                   15      15     12         13      C
```

Sorting a Data Frame

❖ Decreasing option applies to all the variables

```
> paintersABC[order(paintersABC$School, paintersABC$Colour,
+ paintersABC$Drawing, decreasing = T),]
                Composition Drawing Colour Expression School
Vanius                   15      15     12         13      C
Cortona                  16      14     12          6      C
L. Jordaens              13      12      9          6      C
Barocci                  14      15      6         10      C
Josepin                  10      10      6          2      C
Testa                    11      15      0          6      C
T. Zucarro               13      14     10          9      B
Fr. Salviata             13      15      8          8      B
F. Zucarro               10      13      8          8      B
Primaticcio              15      14      7         10      B
Parmigiano               10      15      6          6      B
Volterra                 12      15      5          8      B
Del Piombo                8      13     16          7      A
Da Udine                 10       8     16          3      A
Raphael                  17      18     12         18      A
Perugino                  4      12     10          4      A
Del Sarto                12      16      9          8      A
Fr. Penni                 0      15      8          0      A
Perino del Vaga          15      16      7          6      A
Michelangelo              8      17      4          8      A
Da Vinci                 15      16      4         14      A
Guilio Romano            15      16      4         14      A
```

Sorting a Data Frame

- For numeric vector, we can use ‘-’ to control which vector needs to be sorted in decreasing order

```
> paintersABC[order(paintersABC$School, -paintersABC$Colour, paintersABC$Drawing), ]
                Composition Drawing Colour Expression School
Da Udine                 10       8     16          3      A
Del Piombo                8      13     16          7      A
Raphael                  17      18     12         18      A
Perugino                  4      12     10          4      A
Del Sarto                12      16      9          8      A
Fr. Penni                 0      15      8          0      A
Perino del Vaga          15      16      7          6      A
Da Vinci                 15      16      4         14      A
Guilio Romano            15      16      4         14      A
Michelangelo              8      17      4          8      A
T. Zucarro               13      14     10          9      B
F. Zucarro               10      13      8          8      B
Fr. Salviata             13      15      8          8      B
Primaticcio              15      14      7         10      B
Parmigiano               10      15      6          6      B
Volterra                 12      15      5          8      B
Cortona                  16      14     12          6      C
Vanius                   15      15     12         13      C
L. Jordaens              13      12      9          6      C
Josepin                  10      10      6          2      C
Barocci                  14      15      6         10      C
Testa                    11      15      0          6      C
```

Sorting a Data Frame

- To use the negative sign to sort a character column, you need to use the **`xtfrm`** function

- The **`xtfrm`** function creates a numeric vector based on its argument, which will sort in the same order as its argument

```
> xtfrm(c("A", "D", "B", "C"))
[1] 1 4 2 3
```

Sorting a Data Frame

```
> painterABC[order(-xtfrm(painterABC$School), painterABC$Drawing), ]
                Composition  Drawing  Colour Expression  School
Josepin                  10       10       6          2       C
L. Jordaens              13       12       9          6       C
Cortona                  16       14      12          6       C
Barocci                  14       15       6         10       C
Testa                    11       15       0          6       C
Vanius                   15       15      12         13       C
F. Zucarro               10       13       8          8       B
Primaticcio              15       14       7         10       B
T. Zucarro               13       14      10          9       B
Fr. Salviata             13       15       8          8       B
Parmigiano               10       15       6          6       B
Volterra                 12       15       5          8       B
Da Udine                 10        8      16          3       A
Perugino                  4       12      10          4       A
...
```

Sorting a Data Frame

- The **`na.last`** argument determines the handing of **`NA`**s
  - **`na.last = NA`**: **`NA`**s are deleted
  - **`na.last = TRUE`** (default): **`NA`**s are placed at the end
  - **`na.last = FALSE`**: **`NA`**s are placed at the beginning

Unique and Duplicated Elements of Vectors

❖**`unique`**: create a vector with only unique components

```
> v = c(letters[1:4], "d", "a")
> v
[1] "a" "b" "c" "d" "d" "a"
> unique(v)
[1] "a" "b" "c" "d"
```

❖If **`fromLast = TRUE`**, the last (or rightmost) of identical elements will be kept

```
> unique(v, fromLast = TRUE)
[1] "b" "c" "d" "a"
```

Unique and Duplicated Elements of Vectors

- **`duplicated`**: determine which elements of a vector are duplicates

```
> v
[1] "a" "b" "c" "d" "d" "a"
> duplicated(v)
[1] FALSE FALSE FALSE FALSE TRUE TRUE
```

```
> v[!duplicated(v)]
[1] "a" "b" "c" "d"
```

```
> v[!duplicated(v, fromLast = T)]
[1] "b" "c" "d" "a"
```

Unique and Duplicated Rows Or Columns of Matrices

- **`unique`**: creates a matrix with unique rows (or columns)

- **`duplicated`**: indicates which rows (or columns) are duplicates

```
> mat = matrix(c(rep("c", 3), "d", "c", rep(c(letters[1:3], "c",
+ "c"), 2), letters[1:3], "d", "c", rep("c", 5), letters[1:3],
+ "c", "c"), ncol = 5, byrow = T)
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,] "c"  "c"  "c"  "d"  "c"
[2,] "a"  "b"  "c"  "c"  "c"
[3,] "a"  "b"  "c"  "c"  "c"
[4,] "a"  "b"  "c"  "d"  "c"
[5,] "c"  "c"  "c"  "c"  "c"
[6,] "a"  "b"  "c"  "c"  "c"
```

Unique and Duplicated Rows Or Columns of Matrices

```
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,] "c"  "c"  "c"  "d"  "c"
[2,] "a"  "b"  "c"  "c"  "c"
[3,] "a"  "b"  "c"  "c"  "c"
[4,] "a"  "b"  "c"  "d"  "c"
[5,] "c"  "c"  "c"  "c"  "c"
[6,] "a"  "b"  "c"  "c"  "c"
```

```
> unique(mat)
     [,1] [,2] [,3] [,4] [,5]
[1,] "c"  "c"  "c"  "d"  "c"
[2,] "a"  "b"  "c"  "c"  "c"
[3,] "a"  "b"  "c"  "d"  "c"
[4,] "c"  "c"  "c"  "c"  "c"
```

Unique and Duplicated Rows Or Columns of Matrices

```
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,] "c"  "c"  "c"  "d"  "c"
[2,] "a"  "b"  "c"  "c"  "c"
[3,] "a"  "b"  "c"  "c"  "c"
[4,] "a"  "b"  "c"  "d"  "c"
[5,] "c"  "c"  "c"  "c"  "c"
[6,] "a"  "b"  "c"  "c"  "c"
```

```
> duplicated(mat)
[1] FALSE FALSE TRUE FALSE FALSE TRUE
```

Unique and Duplicated Rows Or Columns of Matrices

```
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,] "c"  "c"  "c"  "d"  "c"
[2,] "a"  "b"  "c"  "c"  "c"
[3,] "a"  "b"  "c"  "c"  "c"
[4,] "a"  "b"  "c"  "d"  "c"
[5,] "c"  "c"  "c"  "c"  "c"
[6,] "a"  "b"  "c"  "c"  "c"
```

```
> unique(mat, fromLast = TRUE)
     [,1] [,2] [,3] [,4] [,5]
[1,] "c"  "c"  "c"  "d"  "c"
[2,] "a"  "b"  "c"  "d"  "c"
[3,] "c"  "c"  "c"  "c"  "c"
[4,] "a"  "b"  "c"  "c"  "c"
```

Unique and Duplicated Rows Or Columns of Matrices

```
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,] "c"  "c"  "c"  "d"  "c"
[2,] "a"  "b"  "c"  "c"  "c"
[3,] "a"  "b"  "c"  "c"  "c"
[4,] "a"  "b"  "c"  "d"  "c"
[5,] "c"  "c"  "c"  "c"  "c"
[6,] "a"  "b"  "c"  "c"  "c"
```

```
> unique(mat, MARGIN = 2)
     [,1] [,2] [,3] [,4]
[1,] "c"  "c"  "c"  "d"
[2,] "a"  "b"  "c"  "c"
[3,] "a"  "b"  "c"  "c"
[4,] "a"  "b"  "c"  "d"
[5,] "c"  "c"  "c"  "c"
[6,] "a"  "b"  "c"  "c"
```

```
> duplicated(mat, MARGIN = 2)
[1] FALSE FALSE FALSE FALSE TRUE
```

Managing Duplicated Observations of Data Frames

❖ Manipulating a data frame with duplicated observations is similar to the way of manipulating a matrix

❖ You can't use the **MARGIN** option applying to a data frame

```
> set.seed(5)
> dat = data.frame(ID = c(rep("A01", 3), rep("A02", 2), "A03",
+ "A04", rep("A05", 2)), visit = c(3:1, 1, 2, rep(1, 4)),
+ score = round(rnorm(9, 5, 2)))
> dat
   ID visit score
1 A01     3     3
2 A01     2     8
3 A01     1     2
4 A02     1     5
5 A02     2     8
6 A03     1     4
7 A04     1     4
8 A05     1     4
9 A05     1     4
```

Managing Duplicated Observations of Data Frames

```
> dat
  ID visit score
1 A01     3     3
2 A01     2     8
3 A01     1     2
4 A02     1     5
5 A02     2     8
6 A03     1     4
7 A04     1     4
8 A05     1     4
9 A05     1     4
```

```
> unique(dat)
   ID visit score
1 A01     3     3
2 A01     2     8
3 A01     1     2
4 A02     1     5
5 A02     2     8
6 A03     1     4
7 A04     1     4
8 A05     1     4
```

Managing Duplicated Observations of Data Frames

❖ Create a data frame that contains patients' first visit

```
> datSort = dat[order(dat$ID, dat$visit), ]
> datSort
   ID visit score
3 A01     1     2   F   T
2 A01     2     8   T   F
1 A01     3     3   T   F
4 A02     1     5   F   T
5 A02     2     8   T   F
6 A03     1     4   F   T
7 A04     1     4   F   T
8 A05     1     4   F   T
9 A05     1     4   T   F
> firstVisit = datSort[!duplicated(datSort[, 1]), ]
> firstVisit
   ID visit score
3 A01     1     2
4 A02     1     5
6 A03     1     4
7 A04     1     4
8 A05     1     4
```

Managing Duplicated Observations of Data Frames

❖ Create a data frame that contains patients' last visits

```
> lastVisit = datSort[!duplicated(datSort[, 1], fromLast = TRUE),
+ ]
> lastVisit
   ID visit score
1 A01     3     3
5 A02     2     8
6 A03     1     4
7 A04     1     4
9 A05     1     4
```

Managing Duplicated Observations of Data Frames

❖ Create a data frame with one observation per subject and a data frame with multiple observations per subject

```
> dupID = unique(dat$ID[duplicated(dat$ID)])
> dat[is.element(dat$ID, dupID), ]
   ID visit score
1 A01     3     3
2 A01     2     8
3 A01     1     2
4 A02     1     5
5 A02     2     8
8 A05     1     4
9 A05     1     4
```

```
> dat[!is.element(dat$ID, dupID), ]
   ID visit score
6 A03     1     4
7 A04     1     4
```

Combining Data Frames
Combining Data Frames Vertically and Horizontally

- **`cbind:`** combine data frame/matrices/vectors column-wise

- **`rbind:`** combine data frame/matrices/vectors row-wise

```
> data0 = data.frame(x1 = rep(1,6), x2 = rep(2,6))
> newCol = data.frame(x3 = 0:5, x4 = 7:12)
> data0 = cbind(data0, newCol)
> data0
  x1 x2 x3 x4
1  1  2  0  7
2  1  2  1  8
3  1  2  2  9
4  1  2  3 10
5  1  2  4 11
6  1  2  5 12
```

Combining Data Frames Vertically and Horizontally

- **`cbind:`** combine data frame/matrices/vectors column-wise
- **`rbind:`** combine data frame/matrices/vectors row-wise

```
> newRow = data.frame(x1 = c(1,2), x2 = c(3,4), x4 = c(5,6),
+ x3 = c(7,8))
> newRow
  x1 x2 x4 x3
1  1  3  5  7
2  2  4  6  8
> data0 = rbind(data0, newRow)
> data0
  x1 x2 x3 x4
1  1  2  0  7
2  1  2  1  8
3  1  2  2  9
4  1  2  3 10
5  1  2  4 11
6  1  2  5 12
7  1  3  7  5
8  2  4  8  6
```

Merging Data Frames

❖ We don't need to sort before we merge data frames

data1

id	var1
2	A
1	B
3	C

data2

id	var2
3	1
4	3
1	9

Result

id	var1	var2
1	B	9
3	C	1

```
> data1 = data.frame(id= c(2,1,3), var1=c("A","B","C"))
> data2 = data.frame(id= c(3,4,1), var2=c(1,3,9))
> result = merge(data1, data2)
> result
  id var1 var2
1  1    B    9
2  3    C    1
```

Merging Data Frames

❖ Use the **`all.x`** or **`all.y`** arguments

data1

id	var1
2	A
1	B
3	C

data2

id	var2
3	1
4	3
1	9

id	var1	var2
1	B	9
2	A	NA
3	C	1

```
> merge(data1, data2, all.x = TRUE)
  id var1 var2
1  1    B    9
2  2    A   NA
3  3    C    1
```

Merging Data Frames

❖ Use the **`all`** argument

data1

id	var1
2	A
1	B
3	C

data2

id	var2
3	1
4	3
1	9

id	var1	var2
1	B	9
2	A	NA
3	C	1
4	<NA>	3

```
> merge(data1, data2, all = TRUE, by = "id")
  id var1 var2
1  1    B    9
2  2    A   NA
3  3    C    1
4  4 <NA>    3
```

Merging Data Frames

❖ The common columns have different variable names:

id ≠ ID

data2

id	var2
3	1
4	3
1	9

data3

ID	var2
1	1
3	8
4	9

var2 = var2

id	var2.x	var2.y
1	9	1
3	1	8
4	3	9

```
> data3 = data.frame(ID= c(1,3,4), var2=c(1, 8,9))
> merge(data2, data3, by.x="id", by.y="ID")
  id var2.x var2.y
1  1      9      1
2  3      1      8
3  4      3      9
```

Merging Data Frames

❖ Use the **`suffixes`** option

id ≠ ID

data2

id	var2
3	1
4	3
1	9

data3

ID	var2
1	1
3	8
4	9

var2 = var2

id	var2.data2	var2.data3
1	9	1
3	1	8
4	3	9

```
> merge(data2, data3, by.x="id", by.y="ID", suffixes=
+ c(".data2", ".data3"))
  id var2.data2 var2.data3
1  1          9          1
2  3          1          8
3  4          3          9
```

Reshaping Data Frames
The `stack` and `unstack` Functions

- Re-organize a data frame to suit a certain statistical or graphical function

- The following data frame is not suitable for ANOVA

```
> dat1 = data.frame(value1 = round(rnorm(3, 2, 3)),
+ value2 = round(rnorm(3, 4, 2)),
+ value3 = round(rnorm(3, 5, 1)))
> dat1
  value1 value2 value3
1      2      2      5
2      6      4      4
3      0      2      3
```

The `stack` and `unstack` Functions

```
> dat1
  value1 value2 value3
1      2      2      5
2      6      4      4
3      0      2      3
```

```
> dat2 = stack(dat1)
> dat2
  values    ind
1      2 value1
2      6 value1
3      0 value1
4      2 value2
5      4 value2
6      2 value2
7      5 value3
8      4 value3
9      3 value3
```

The `stack` and `unstack` Functions

- **`unstack`**: transform the data frame back to its original form; but it requires a formula

```
> dat3 = cbind(dat2, scores = round(runif(9) * 10))
> dat3
  values    ind scores
1      2 value1      6
2      6 value1      6
3      0 value1      4
4      2 value2      4
5      4 value2      8
6      2 value2      2
7      5 value3      8
8      4 value3      5
9      3 value3      9
```

The `stack` and `unstack` Functions

```
> dat3
  values    ind scores
1      2 value1      6
2      6 value1      6
3      0 value1      4
4      2 value2      4
5      4 value2      8
6      2 value2      2
7      5 value3      8
8      4 value3      5
9      3 value3      9
```

```
> unstack(dat3, values ~ ind)
  value1 value2 value3
1      2      2      5
2      6      4      4
3      0      2      3
```

The `stack` and `unstack` Functions

```
> dat3
  values    ind scores
1      2 value1      6
2      6 value1      6
3      0 value1      4
4      2 value2      4
5      4 value2      8
6      2 value2      2
7      5 value3      8
8      4 value3      5
9      3 value3      9
```

```
> unstack(dat3, scores ~ ind)
  value1 value2 value3
1      6      4      8
2      6      8      5
3      4      2      9
```

Transforming from Wide Format to Long Format

❖ A wide format: many observations per subject

ID	Sex	Score1	Score2	Score3
A	M	3	4	5
B	M	7	8	9
C	M	6	5	4

Data frame

ID	Sex	TIME	Score
A	M	1	3
B	M	1	7
C	F	1	6
A	M	2	4
B	M	2	8
C	F	2	5
A	M	3	5
B	M	3	9
C	F	3	4

```
> long = reshape(wide,
                 varying =list(c("Score1", "Score2", "Score3")),
                 v.names ="Score",
                 timevar="TIME",
                 idvar = "ID",
                 direction="long")
```

Transforming from Wide Format to Long Format

❖ A wide format: many observations per subject

ID	Sex	Score1	Score2	Score3
A	M	3	4	5
B	M	7	8	9
C	M	6	5	4

ID	Sex	TIME	Score
A	M	1	3
B	M	1	7
C	F	1	6
A	M	2	4
B	M	2	8
C	F	2	5
A	M	3	5
B	M	3	9
C	F	3	4

```
> long = reshape(wide,
                 varying =list(c("Score1", "Score2", "Score3")),
                 v.names ="Score",
                 timevar="TIME",
                 idvar = "ID",
                 direction="long")
```

names of the variables in the wide format corresponding to a single variable in the long format

Transforming from Wide Format to Long Format

❖ A wide format: many observations per subject

ID	Sex	Score1	Score2	Score3
A	M	3	4	5
B	M	7	8	9
C	M	6	5	4

ID	Sex	TIME	Score
A	M	1	3
B	M	1	7
C	F	1	6
A	M	2	4
B	M	2	8
C	F	2	5
A	M	3	5
B	M	3	9
C	F	3	4

```
> long = reshape(wide,
                 varying =list(c("Score1", "Score2", "Score3")),
                 v.names ="Score",
                 timevar="TIME",
                 idvar = "ID",
                 direction="long")
```

names of the variables in the long format corresponding to a single variable in the wide format

Transforming from Wide Format to Long Format

❖ A wide format: many observations per subject

ID	Sex	Score1	Score2	Score3
A	M	3	4	5
B	M	7	8	9
C	M	6	5	4

ID	Sex	TIME	Score
A	M	1	3
B	M	1	7
C	F	1	6
A	M	2	4
B	M	2	8
C	F	2	5
A	M	3	5
B	M	3	9
C	F	3	4

```
> long = reshape(wide,
                 varying =list(c("Score1", "Score2", "Score3")),
                 v.names ="Score",
                 timevar="TIME",
                 idvar = "ID",
                 direction="long")
```

variable in the long format differentiating multiple records from the same individual

Transforming from Wide Format to Long Format

❖ A wide format: many observations per subject

ID	Sex	Score1	Score2	Score3
A	M	3	4	5
B	M	7	8	9
C	M	6	5	4

ID	Sex	TIME	Score
A	M	1	3
B	M	1	7
C	F	1	6
A	M	2	4
B	M	2	8
C	F	2	5
A	M	3	5
B	M	3	9
C	F	3	4

```
> long = reshape(wide,
                 varying =list(c("Score1", "Score2", "Score3")),
                 v.names ="Score",
                 timevar="TIME",
                 idvar = "ID",
                 direction="long")
```

Names of the 1 or more variables in the long format that identify multiple records from the same individual

Transforming from Wide Format to Long Format

❖ A wide format: many observations per subject

ID	Sex	Score1	Score2	Score3
A	M	3	4	5
B	M	7	8	9
C	M	6	5	4

ID	Sex	TIME	Score
A	M	1	3
B	M	1	7
C	F	1	6
A	M	2	4
B	M	2	8
C	F	2	5
A	M	3	5
B	M	3	9
C	F	3	4

```
> long = reshape(wide,
                 varying =list(c("Score1", "Score2", "Score3")),
                 v.names ="Score",
                 timevar="TIME",
                 idvar = "ID",
                 direction="long")
```

long: reshape to long format
wide: reshape to wide format

Transforming from Long Format to Wide Format

❖ Similar syntax

ID	Sex	Score1	Score2	Score3
A	M	3	4	5
B	M	7	8	9
C	M	6	5	4

ID	Sex	TIME	Score
A	M	1	3
B	M	1	7
C	F	1	6
A	M	2	4
B	M	2	8
C	F	2	5
A	M	3	5
B	M	3	9
C	F	3	4

```
> wide1 = reshape(long, varying=list(c("Score1", "Score2",
"Score3")), v.names="Score", timevar="TIME", idvar = "ID",
direction="wide")
```