# Chapter 8
# Data Aggregation

Arthur Li

# Data Tabulation
# The `table` and `ftable` Functions

- The **`table`** function takes the following data type:
  - One or more factors
  - Character vectors that can be coerced into factors
  - A data frame
  - A list whose components can be coerced into factors

- By default, the missing values will *not* be included in the table

```
> a = c(rep("A", 3), rep("B", 2), rep(NA, 3))
> aTable = table(a, exclude = NULL)
> aTable
a
   A    B <NA>
   3    2    3
> class(aTable)
[1] "table"
> mode(aTable)
[1] "numeric"
```

# The `table` and `ftable` Functions

- Each component of the table can be accessed with its name

```
> aTable["A"]
A
3
> as.data.frame(aTable)
     a Freq
1    A    3
2    B    2
3 <NA>    3
```

# The `table` and `ftable` Functions

❖Passing more than one vector to the table function…

```
> race = c(rep("A", 10), rep("W", 6), rep("B", 4))
> gender = c(rep("M", 4), rep("F", 3), rep("M", 6),
      rep("F", 7))
> grade = c(3, 3, rep(c(1, 2, 3), 6))
> table(race, grade, gender)
, , gender = F
    grade
race 1 2 3
   A 1 1 1
   B 1 1 2
   W 1 1 1
, , gender = M
    grade
race 1 2 3
   A 2 2 3
   B 0 0 0
   W 1 1 1
```

# The `table` and `ftable` Functions

```
> as.data.frame(table(grade, race, gender))
   grade race gender Freq
1      1    A      F    1
2      2    A      F    1
3      3    A      F    1
4      1    B      F    1
5      2    B      F    1
6      3    B      F    2
7      1    W      F    1
8      2    W      F    1
9      3    W      F    1
10     1    A      M    2
11     2    A      M    2
12     3    A      M    3
13     1    B      M    0
14     2    B      M    0
15     3    B      M    0
16     1    W      M    1
17     2    W      M    1
18     3    W      M    1
```

**All the possible combinations are included**

# The `table` and `ftable` Functions

- The following three statements will generate the same table:

```
table(data.frame(grade, race, gender))
table(list(grade, race, gender))
table(grade, race, gender)
```

# The `table` and `ftable` Functions

❖The **`ftable`** function: display the table in a *flat* format

```
> ftable(race, grade, gender)
             gender F M
race grade
A    1             1 2
     2             1 2
     3             1 3
B    1             1 0
     2             1 0
     3             2 0
W    1             1 1
     2             1 1
     3             1 1
```

# The `addmargins` Function

- **`addmargins`**: display the margins of a table
  - **`A`**: a table class
  - **`margin`**: 1 refers to the row, 2 refers to the column
  - **`FUN`**: a function that is used for each corresponding margin

```
> tableRaceGender = table(gender, race)
> addmargins(tableRaceGender)
       race
gender   A   B   W Sum
   F     3   4   3  10
   M     7   0   3  10
   Sum  10   4   6  20
```

**Default option:**
- **Both margins are displayed**
- **FUN = sum**

# The addmargins Function

```
> addmargins(tableRaceGender, 2, "mean")

      race
gender        A        B        W   "mean"
     F 3.000000 4.000000 3.000000 3.333333
     M 7.000000 0.000000 3.000000 3.333333
```

```
> addmargins(tableRaceGender, c(1, 2), c("mean", "sum"))

Margins computed over dimensions
in the following order:
1: gender
2: race
      race
gender   A   B   W sum
  F      3   4   3  10
  M      7   0   3  10
mean     5   2   3  10
```

# The `prop.table` Function

❖ **`prop.table`**: create a table of proportions instead of counts

```
> tableRaceGender
       race
gender A B W
     F 3 4 3
     M 7 0 3
```

```
> prop.table(tableRaceGender)
       race
gender    A    B    W
     F 0.15 0.20 0.15
     M 0.35 0.00 0.15
```

**Default: sum of all the cells = 1**

# The `prop.table` Function

❖ **`prop.table`**: create a table of proportions instead of counts

```
> tableRaceGender
      race
gender A B W
     F 3 4 3
     M 7 0 3
```

```
> prop.table(tableRaceGender, margin = 1)
      race
gender   A   B   W
     F 0.3 0.4 0.3
     M 0.7 0.0 0.3
```

**margin = 1: sum of rows = 1**

# The `prop.table` Function

❖ **`prop.table`**: create a table of proportions instead of counts

```
> tableRaceGender
      race
gender A B W
     F 3 4 3
     M 7 0 3
```

```
> prop.table(tableRaceGender, margin = 2)
      race
gender   A   B   W
     F 0.3 1.0 0.5
     M 0.7 0.0 0.5
```

**margin = 2: sum of columns = 1**

# Applying a Function to an Array
## The `apply` Function

- Functions to calculate statistics for rows or columns of a matrix: **`rowSums`**, **`rowMeans`**, **`colSums`**, and **`colMeans`**

- **`apply`**: for a more general operation on successive sections of an array
  - **`X`**: is an array or a matrix (a two-dimensional array)
  - **`MARGIN`**: is an integer vector. E.g. for a matrix, 1 indicates rows, 2 indicates columns, and c(1, 2) indicates rows and columns
  - **`FUN`**: is a function which can be either a built-in or user-defined function that is used to apply separately to each section
  - **`...`**: any additional arguments needed by the **`FUN`**

# The apply Function

```
> (x = matrix(rnorm(12), 3))
           [,1]       [,2]       [,3]       [,4]
[1,] -0.7074952 -0.1123462 -0.6120264  1.4330237
[2,]  0.3645820  0.8811077  0.3411197  1.9803999
[3,]  0.7685329  0.3981059 -1.1293631 -0.3672215
> apply(x, 1, min)
[1] -0.7074952  0.3411197 -1.1293631
> apply(x, 2, min)
[1] -0.7074952 -0.1123462 -1.1293631 -0.3672215
```

```
> apply(x, 2, sd)
[1] 0.7628002 0.4967902 0.7459272 1.2282664
```

```
> x1 = x
> x1[1, 1] = NA
> apply(x1, 1, sd)
[1]        NA 0.7673269 0.8430555
> apply(x1, 1, sd, na.rm = TRUE)
[1] 1.0661518 0.7673269 0.8430555
```

# Applying Functions To an Array With More Than Two Dimensions

- **`iris3:`** 3-dimensional array of size 50 by 4 by 3; it contains the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris
  - 1st dimension: case number within the species subsample
  - 2nd dimension: the variable names Sepal L., Sepal W., Petal L., and Petal W.,
  - 3rd dimension is species (Iris setosa, versicolor, and virginica)

# Applying Functions To an Array With More Than Two Dimensions

```
> iris3
, , Setosa

      Sepal L. Sepal W. Petal L. Petal W.
 [1,]      5.1      3.5      1.4      0.2
 [2,]      4.9      3.0      1.4      0.2
 [3,]      4.7      3.2      1.3      0.2
 ...
[48,]      4.6      3.2      1.4      0.2
[49,]      5.3      3.7      1.5      0.2
[50,]      5.0      3.3      1.4      0.2

, , Versicolor

      Sepal L. Sepal W. Petal L. Petal W.
 [1,]      7.0      3.2      4.7      1.4
 [2,]      6.4      3.2      4.5      1.5
 [3,]      6.9      3.1      4.9      1.5
 ...
[48,]      6.2      2.9      4.3      1.3
[49,]      5.1      2.5      3.0      1.1
[50,]      5.7      2.8      4.1      1.3

, , Virginica
```

# Applying Functions To an Array With More Than Two Dimensions

❖ Calculate the means for each variable by species:

```
> apply(iris3, c(2, 3), mean)
         Setosa Versicolor Virginica
Sepal L.  5.006      5.936     6.588
Sepal W.  3.428      2.770     2.974
Petal L.  1.462      4.260     5.552
Petal W.  0.246      1.326     2.026
```

# Applying Functions To an Array With More Than Two Dimensions

❖ To find the overall means for each variable

```
> apply(iris3, 2, mean)
Sepal L. Sepal W. Petal L. Petal W.
5.843333 3.057333 3.758000 1.199333
```

# Applying a Function to a Vector or a List
## The `lapply` and `sapply` Functions

- Most R functions can operate on each element of a vector but not for a list

- **`lapply`** and **`sapply`**: for operations on the individual components of lists or vectors:
  - **`X`**: is a vector or a list
  - **`FUN`**: is a function to be applied to each element of **`X`**
  - **`...`** : are optional arguments to **`FUN`**

- **`sapply`**: simplifies the result as a vector/an array if possible
- **`lapply`**: returns a list

# The `lapply` and `sapply` Functions

❖ Calculate the mean of each component:

```
> set.seed(1)
> aList = list(a = rnorm(20, 3, 4), b = rnorm(10, 5, 2),
+ c = c(rnorm(15, 2, 1), NA))
> lapply(aList, mean, na.rm = T)
$a
[1] 3.762096

$b
[1] 4.732654

$c
[1] 2.090286
```

```
> sapply(aList, mean, na.rm = T)
       a        b        c
3.762096 4.732654 2.090286
```

# The `lapply` and `sapply` Functions

❖ Calculate the mean and standard deviation:

```
> lapply(aList, function(x) c(MEAN = mean(x, na.rm = T),
+ SD = sd(x, na.rm = T)))
$a
    MEAN       SD
3.762096 3.653015

$b
    MEAN       SD
4.732654 1.911215

$c
    MEAN       SD
2.090286 0.720553
```

```
> sapply(aList, function(x) c(MEAN = mean(x, na.rm = T),
+ SD = sd(x, na.rm = T)))
             a        b        c
MEAN 3.762096 4.732654 2.090286
SD   3.653015 1.911215 0.720553
```

# Example: Extract Information from the Results of Statistical Functions

- Compare the differences for the **Composition**, **Drawing**, **Colour**, and **Expression** variables across different levels in the **School** variable

```
> 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
```

# Example: Extract Information from the Results of Statistical Functions

```
> r = apply(painters[,1:4], 2, function(x){
+       lm(x ~ School, data = painters)
+ })
> mode(r)
[1] "list"
> r[[1]]

Call:
lm(formula = x ~ School, data = painters)

Coefficients:
(Intercept)      SchoolB      SchoolC      SchoolD      SchoolE
     10.400        1.767        2.767       -1.300        3.171
    SchoolF      SchoolG      SchoolH
     -3.150        3.457        3.600
```

# Example: Extract Information from the Results of Statistical Functions

```
> summary(r[[1]])

Call:
lm(formula = x ~ School, data = painters)

Residuals:
    Min      1Q  Median      3Q     Max
-10.400  -2.342   0.750   1.812   6.600

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   10.400      1.171   8.885 1.52e-11 ***
SchoolB        1.767      1.912   0.924   0.3602
SchoolC        2.767      1.912   1.447   0.1546
SchoolD       -1.300      1.655  -0.785   0.4363
SchoolE        3.171      1.824   1.739   0.0888 .
SchoolF       -3.150      2.190  -1.438   0.1571
SchoolG        3.457      1.824   1.895   0.0644 .
SchoolH        3.600      2.190   1.644   0.1070
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.702 on 46 degrees of freedom
Multiple R-squared: 0.2881,     Adjusted R-squared: 0.1797
F-statistic: 2.659 on 7 and 46 DF,  p-value: 0.02137
```

# Example: Extract Information from the Results of Statistical Functions

```
> anova(r[[1]])
Analysis of Variance Table

Response: x
          Df Sum Sq Mean Sq F value  Pr(>F)
School     7 255.05  36.435  2.6591 0.02137 *
Residuals 46 630.29  13.702
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

# Example: Extract Information from the Results of Statistical Functions

- Suppose that you would like to extract the F statistics with its corresponding p-value from the ANOVA table

```
> str(anova(r[[1]]))
Classes ‘anova’ and 'data.frame':       2 obs. of  5 variables:
 $ Df     : int  7 46
 $ Sum Sq : num  255 630
 $ Mean Sq: num  36.4 13.7
 $ F value: num  2.66 NA
 $ Pr(>F) : num  0.0214 NA
 - attr(*, "heading")= chr  "Analysis of Variance Table\n"
"Response: x"
> names(anova(r[[1]]))
[1] "Df"      "Sum Sq"  "Mean Sq" "F value" "Pr(>F)"
```

# Example: Extract Information from the Results of Statistical Functions

❖ Extract the statistics from one component of the list first

```
> foo = anova(r[[1]])
> foo["F value"]
          F value
School     2.6591
Residuals
> foo["F value"][1,1]
[1] 2.65912
> foo["Pr(>F)"]
            Pr(>F)
School    0.02137 *
Residuals
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> foo["Pr(>F)"][1,1]
[1] 0.02136706
```

# Example: Extract Information from the Results of Statistical Functions

```
> sapply(r, function(one){
+     rANOVA = anova(one)
+     c(F = rANOVA["F value"][1,1], p = rANOVA["Pr(>F)"][1,1])
+ })
  Composition     Drawing       Colour Expression
F  2.65911982 3.082908097 8.701385e+00 2.52372862
p  0.02136706 0.009496847 9.152206e-07 0.02771616
```

# Applying a Function for Each Level of a Categorical Variable The `tapply` Function

- **`tapply:`** apply a function to each group of values given by a unique combination of the levels of certain factors
  - **`X`**: is typically a vector
  - **`INDEX`**: is the factor defining the groups
  - **`FUN`**: is the function to be applied
  - **`...`**: are optional arguments to **`FUN`**

- Depending on the number of factors in the **`INDEX`** arguments and the number of values returned by the **`FUN`** function, the class and/or the mode of the object that is returned by the **`tapply`** function differ

# The `tapply` Function

❖ You will see some examples based on the **`quine`** data frame

```
> head(quine)
  Eth Sex Age Lrn Days
1   A   M  F0  SL    2
2   A   M  F0  SL   11
3   A   M  F0  SL   14
4   A   M  F0  AL    5
5   A   M  F0  AL    5
6   A   M  F0  AL   13
```

# The `tapply` Function

- Suppose you would like to calculate the mean **`Days`** for each level of the Age group

```
> (ageTable = tapply(quine$Days, quine$Age, mean))
      F0       F1       F2       F3
14.85185 11.15217 21.05000 19.60606
> class(ageTable)
[1] "array"
> mode(ageTable)
[1] "numeric"
> dim(ageTable)
[1] 4
> dimnames(ageTable)
[[1]]
[1] "F0" "F1" "F2" "F3"
```

**Result: one-dimensional array**

# The `tapply` Function

❖ To convert the result to a data frame, …

```
> as.data.frame(as.table(ageTable))
  Var1     Freq
1   F0 14.85185
2   F1 11.15217
3   F2 21.05000
4   F3 19.60606
```

# The tapply Function

```
> (ageSexTable = tapply(quine$Days, list(quine$Age,
+ quine$Sex), mean))
           F        M
F0 18.70000 12.58824
F1 12.96875  7.00000
F2 18.42105 23.42857
F3 14.00000 27.21429
> class(ageSexTable)
[1] "matrix"
> mode(ageSexTable)
[1] "numeric"
```

# The `split` Functions

- **`split`** + **`sapply`**: apply a function to each group of values given by a unique combination of the levels of certain factors

- **`split`**: split a data vector or a data frame on one or more factors

# The `split` Functions

- One factor with a function returning more than one value

```
> (DaysByAge = split(quine$Days, quine$Age))
$F0
 [1]  2 11 14  5  5 13 20 22  3  5 11 24 45  6 17 67  0  0  2  7 11
[22] 12 25 10 11 20 33

$F1
 [1]  6  6 15  7 14  5  6  6  9 13 23 25 32 53 54  5  5 11 17 19  0
[22]  0  5  5  5 11 17  3  4  5  7  0  1  5  5  5  5  7 11 15  5 14
[43]  6  6  7 28

$F2
 [1]  6 32 53 57 14 16 16 17 40 43 46  8 13 14 20 47 48 60 81  2 22
[22] 30 36  8  0  1  5  7 16 27  0  5 14  2  2  3  8 10 12  1

$F3
 [1]  8 23 23 28 34 36 38  0  2  3  5 10 14 21 36 40  0 30 10 14 27
[22] 41 69  1  9 22  3  3  5 15 18 22 37
```

# The `split` Functions

- One factor with a function returning more than one value

```
> (ageTableNew = sapply(DaysByAge, mean))
      F0       F1       F2       F3
14.85185 11.15217 21.05000 19.60606
> class(ageTableNew)
[1] "numeric"
> mode(ageTableNew)
[1] "numeric"
```

# The `split` Functions

❖ One factor with a function returning more than one value:

```
> (ageTable1New = sapply(split(quine$Days, quine$Age),
+ function(x) return(c(Mean = mean(x), Median = median(x)))))
              F0       F1    F2       F3
Mean   14.85185 11.15217 21.05 19.60606
Median 11.00000  6.00000 14.00 18.00000
> class(ageTable1New)
[1] "matrix"
> mode(ageTable1New)
[1] "numeric"
```

# The `split` Functions

- More than one factor with a function returning one value:

```
> (ageSexTableNew = sapply(split(quine$Days,
+ list(quine$Age, quine$Sex)), mean))
    F0.F     F1.F     F2.F     F3.F     F0.M     F1.M     F2.M     F3.M
18.70000 12.96875 18.42105 14.00000 12.58824  7.00000 23.42857 27.21429
> class(ageSexTableNew)
[1] "numeric"
> mode(ageSexTableNew)
[1] "numeric"
```

# The `split` Functions

- More than one factor with a function returning more than one value:

```
> (ageSexTable1New = sapply(split(quine$Days, list(quine$Age, quine$Sex)),
+     function(x) c(Mean = mean(x), Median = median(x))))
         F0.F     F1.F     F2.F F3.F     F0.M F1.M     F2.M     F3.M
Mean     18.7 12.96875 18.42105   14 12.58824  7.0 23.42857 27.21429
Median 15.5  7.00000 10.00000   10 11.00000  5.5 17.00000 27.50000
> class(ageSexTable1New)
[1] "matrix"
> mode(ageSexTable1New)
[1] "numeric"
```

# The `by` and `aggregate` Functions

- **`by`**, **`aggregate`**: calculate summary statistics for each level of one or more factors for a data frame
- Syntax is similar to **`tapply`** function
- For the **`aggregate`** function, the second argument needs to be a list that contains a variable or a factor

# The by and aggregate Functions

```
> (ageTableBy = by(quine$Days, quine$Age, mean))
quine$Age: F0
[1] 14.85185
------------------------------------------------------------
quine$Age: F1
[1] 11.15217
------------------------------------------------------------
quine$Age: F2
[1] 21.05
------------------------------------------------------------
quine$Age: F3
[1] 19.60606
> class(ageTableBy)
[1] "by"
> mode(ageTableBy)
[1] "numeric"
```

# The by and aggregate Functions

```
> (ageTableAgg = aggregate(quine$Days, list(quine$Age), mean))
  Group.1        x
1      F0 14.85185
2      F1 11.15217
3      F2 21.05000
4      F3 19.60606
> class(ageTableAgg)
[1] "data.frame"
> mode(ageTableAgg)
[1] "list"
```

# The `by` and `aggregate` Functions

❖ The **`iris`** data frame

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

# The by and aggregate Functions

```
> (speciesMeanBy = by(iris[,1:4], iris[,5], colMeans))
iris[, 5]: setosa
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.006        3.428        1.462        0.246
------------------------------------------------------------
iris[, 5]: versicolor
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.936        2.770        4.260        1.326
------------------------------------------------------------
iris[, 5]: virginica
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       6.588        2.974        5.552        2.026
> class(speciesMeanBy)
[1] "by"
> mode(speciesMeanBy)
[1] "list"
```

# The by and aggregate Functions

```
> (speciesMeanAgg = aggregate(iris[,1:4], iris[5], mean))
     Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
> class(speciesMeanAgg)
[1] "data.frame"
> mode(speciesMeanAgg)
[1] "list"
```

# The by and aggregate Functions

```
> by(iris[,1:4], iris[,5], function(x) c(Mean = colMeans(x),
+   SD = apply(x, 2, sd)))
iris[, 5]: setosa
Mean.Sepal.Length  Mean.Sepal.Width Mean.Petal.Length
        5.0060000         3.4280000         1.4620000
 Mean.Petal.Width   SD.Sepal.Length    SD.Sepal.Width
        0.2460000         0.3524897         0.3790644
  SD.Petal.Length    SD.Petal.Width
        0.1736640         0.1053856
------------------------------------------------------
iris[, 5]: versicolor
Mean.Sepal.Length  Mean.Sepal.Width Mean.Petal.Length
        5.9360000         2.7700000         4.2600000
 Mean.Petal.Width   SD.Sepal.Length    SD.Sepal.Width
        1.3260000         0.5161711         0.3137983
  SD.Petal.Length    SD.Petal.Width
        0.4699110         0.1977527
------------------------------------------------------
iris[, 5]: virginica
Mean.Sepal.Length  Mean.Sepal.Width Mean.Petal.Length
        6.5880000         2.9740000         5.5520000
 Mean.Petal.Width   SD.Sepal.Length    SD.Sepal.Width
        2.0260000         0.6358796         0.3224966
  SD.Petal.Length    SD.Petal.Width
        0.5518947         0.2746501
```

# The by and aggregate Functions

```
> aggregate(iris[,1:4], iris[5], function(x) c(Mean = mean(x),SD = sd(x)))
     Species Sepal.Length.Mean Sepal.Length.SD Sepal.Width.Mean
1     setosa          5.0060000       0.3524897        3.4280000
2 versicolor          5.9360000       0.5161711        2.7700000
3  virginica          6.5880000       0.6358796        2.9740000
  Sepal.Width.SD Petal.Length.Mean Petal.Length.SD Petal.Width.Mean
1      0.3790644          1.4620000       0.1736640         0.2460000
2      0.3137983          4.2600000       0.4699110         1.3260000
3      0.3224966          5.5520000       0.5518947         2.0260000
  Petal.Width.SD
1      0.1053856
2      0.1977527
3      0.2746501
```