MTH 3240 R Notes 7

6 Data Frames (Cont'd)

6.7 Merging the Columns of Two Data Frames

- To merge (combine) the columns of two data frames, a useful function is:

```
merge()       # Merge two data frames that share one or more variables
              # (columns) in common
```

- For example, we can combine the columns of the following two data frames, whose rows correspond to people:

```
NamesAndAges
```

```
##       Name Age
## 1     John  23
## 2    Karen  27
## 3 Margaret  19
## 4      Ann  36
## 5     Karl  32
## 6     Eric  24
## 7   Justin  22
## 8    Janet  28
```

```
JumbledNamesAndWts
```

```
##       Name Weight
## 1     Eric    184
## 2      Ann    159
## 3    Karen    170
## 4   Justin    161
## 5 Margaret    147
## 6     John    155
## 7    Janet    154
## 8     Karl    201
```

The two data frames contain the *same* eight people, but they're in different orders.

- To merge (combine) `NamesAndAges` with `JumbledNamesAndWts` so that the names get matched, use `merge()`:

```
merge(NamesAndAges, JumbledNamesAndWts, by = "Name")
```

```
##       Name Age Weight
## 1      Ann  36    159
## 2     Eric  24    184
## 3    Janet  28    154
## 4     John  23    155
## 5   Justin  22    161
## 6    Karen  27    170
## 7     Karl  32    201
## 8 Margaret  19    147
```

  Note that `merge()` sorted the rows of both `NamesAndWeights` and `JumbledNamesAndWts` according to the alphabetical ordering of `Name` before merging them.

- Above, we merged `NamesAndAges` with `JumbledNamesAndWts` by the values in *one* column, `Name`.

  We can can also merge two data frames by the values in *two* columns if we need to. For more info, see the help file for `merge()` (type `?merge`).

Section 6.7 Exercises

Exercise 1 Here are two data frames containing responses to two survey questions (on a scale of 1 to 100):

```
dfA <- data.frame(RespondentID = c(1000, 1001, 1002, 1003, 1004, 1005, 1006),
                  Response1 = c(55, 62, 39, 45, 70, 77, 56))
dfA
```

```
##   RespondentID Response1
## 1         1000        55
## 2         1001        62
## 3         1002        39
## 4         1003        45
## 5         1004        70
## 6         1005        77
## 7         1006        56
```

```
dfB <- data.frame(RespondentID = c(1003, 1002, 1000, 1004, 1006, 1001, 1005),
                  Response2 = c(12, 17, 23, 24, 19, 30, 20))
dfB
```

```
##   RespondentID Response2
## 1         1003        12
## 2         1002        17
## 3         1000        23
## 4         1004        24
## 5         1006        19
## 6         1001        30
## 7         1005        20
```

Note that the `RespondentID`s are the same, but in different orders. Write a command involving `merge()`, with `by = "RespondentID"`, that merges the columns of the two data frames. You should end up with this:

```
##   RespondentID Response1 Response2
## 1         1000        55        23
## 2         1001        62        30
## 3         1002        39        17
## 4         1003        45        12
## 5         1004        70        24
## 6         1005        77        20
## 7         1006        56        19
```

6.8 Stacking and Unstacking Columns of a Data Frame

- Sometimes data are arranged in separate columns representing, say, different groups, but we'd prefer them to be in a single column with an adjacent column indicating the group.

  Other times we need to do the opposite (take a single column and turn it into multiple columns).

  The functions below are useful for such tasks.

```
stack()           # "Stack" columns in a data frame
unstack()         # "Unstack" a column in a data frame
```

- For example, the data might be "unstacked" (aka in "wide" format) like this:

```
unstacked.data

##   Grp1 Grp2 Grp3
## 1   23   19   31
## 2   11   26   28
## 3   14   24   34
## 4   16   29   25
```

We "stack" the data (into "long" format) by typing:

```
stacked.data <- stack(unstacked.data)
names(stacked.data) <- c("Response", "Group")
```

```
stacked.data
```

```
##    Response Group
## 1        23  Grp1
## 2        11  Grp1
## 3        14  Grp1
## 4        16  Grp1
## 5        19  Grp2
## 6        26  Grp2
## 7        24  Grp2
## 8        29  Grp2
## 9        31  Grp3
## 10       28  Grp3
## 11       34  Grp3
## 12       25  Grp3
```

- To "unstack" the data, we need indicate which column we want to "unstack" into separate columns and which one is the group indicator for forming column headers.

  We do this by passing a so-called ***formula*** to `unstack()` via its argument `form`. The *formula*'s left side is the variable to be "unstacked" and its right side is the group indicator:

```
unstacked.data <- unstack(stacked.data, form = Response ~ Group)
unstacked.data
```

```
##   Grp1 Grp2 Grp3
## 1   23   19   31
## 2   11   26   28
## 3   14   24   34
## 4   16   29   25
```

  (Above, the *formula* is `Response ~ Group`.)

Section 6.8 Exercises

Exercise 2 Here's a data frame:

```
x <- data.frame(a = c(1, 4, 2), b = c(7, 5, 8), c = c(9, 9, 8))
x
```

```
##   a b c
## 1 1 7 9
## 2 4 5 9
## 3 2 8 8
```

Guess what the following command will do, then check your answer:

```
stack(x)
```

Exercise 3 Here's a data frame containing data from an experiment involving a treatment group and a control group:

```
x <- data.frame(Group = c("Trt", "Trt", "Trt", "Trt", "Trt", "Ctrl",
                          "Ctrl", "Ctrl", "Ctrl", "Ctrl"),
                Y = c(22, 45, 32, 45, 30, 60, 44, 24, 56, 59))
x
```

```
##    Group  Y
## 1    Trt 22
## 2    Trt 45
## 3    Trt 32
## 4    Trt 45
## 5    Trt 30
## 6   Ctrl 60
## 7   Ctrl 44
## 8   Ctrl 24
## 9   Ctrl 56
## 10  Ctrl 59
```

Guess what the following command will do, then check your answer:

```
unstack(x, form = Y ~ Group)
```

7 Factors and Tables

7.1 Creating and Viewing Factors and Their Levels

- ***Factors***, like `"character"` vectors, are used to store categorical (qualitative) data.

  `"character"` vectors are the *preferred* way of storing categorical data in R because they're easier to work with than factors.

Factors are a relic from early versions of R. They differ from `"character"` vectors in the way R stores them internally, and they contain a bit of extra information called ***levels***.

- The following are functions are useful for working with factors:

```
factor()          # Create a factor from a character vector
length()          # Returns the number of elements in a factor
levels()          # Examine the levels of the factor
is.factor()       # Indicates whether or not an object is a factor
```

- Below, we use `factor()` to convert a `"character"` vector to a factor:

```
char.vec <- c("ctrl", "trt1", "trt2", "ctrl", "trt1", "trt2", "ctrl",
              "trt1", "trt2")
```

```
my.fac <- factor(char.vec)
```

```
is.factor(my.fac)

## [1] TRUE
```

- When R prints out a factor, it also indicates its ***levels***, which are the unique values that appear in the factor:

```
my.fac

## [1] ctrl trt1 trt2 ctrl trt1 trt2 ctrl trt1 trt2
## Levels: ctrl trt1 trt2
```

- To convert a factor to a `"character"` vector, we use:

```
as.character()      # Convert a factor to a character vector
```

- `"character"` vectors are easier to work with than factors. Here's an example of converting a factor to a `"character"` vector:

```
as.character(my.fac)

## [1] "ctrl" "trt1" "trt2" "ctrl" "trt1" "trt2" "ctrl" "trt1"
## [9] "trt2"
```

Section 7.1 Exercises

Exercise 4 Here's a factor:

```
x.fac <- factor(c("a", "a", "b", "b", "c", "c", "d", "d"))
```

a) Guess what the result of the following command will be, then check your answer:

```
levels(x.fac)
```

b) Write a command involving `as.character()` that converts `x.fac` to a `"character"` vector.

7.2 Creating Tables

- Categorical (qualitative) data are summarized using tables of *counts* or *proportions*.
- To create a table from a `"character"` vector (or a factor), we use:

```
table()          # Create a table of counts from a factor or
                 # "character" vector
prop.table()     # Create a table of proportions from a table
                 # counts
```

To check whether an object is a table, use:

```
is.table()       # Returns TRUE or FALSE indicating whether an object
                 # is a table
```

To turn a table of counts (or proportions) into a bar graph, we use:

```
barplot()        # Create a bar graph from a table of counts or
                 # proportions
```

- `table()` returns a table of *counts* of the occurrences of each unique value in a `"character"` vector (or level of a factor).
- For example, here's a `"character"` vector containing responses (`"Yes"`, `"No"`, or `"Maybe"`) to a survey question:

```
survey.responses <- c("Yes", "No", "Maybe", "No", "Maybe", "Maybe",
    "No", "Yes", "No", "Yes", "Maybe", "No", "Yes", "Yes", "Yes",
    "Maybe", "Yes", "No", "Yes")
```

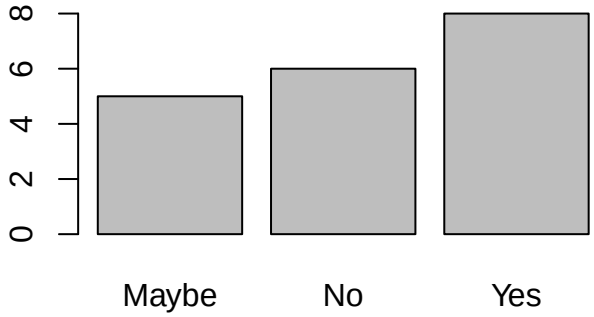
We can tabulate the responses by typing:

```
survey.tab <- table(survey.responses)
survey.tab

## survey.responses
## Maybe    No   Yes
##     5     6     8
```

Here's a bar plot of the *counts*:

```
barplot(survey.tab)
```



- `prop.table()` takes the table of *counts* returned by `table()`, and converts them to *proportions*:

```
prop.table(survey.tab)

## survey.responses
##     Maybe        No       Yes
## 0.2631579 0.3157895 0.4210526
```

- We can create a *two-way* table from *two* `"character"` vectors (or factors).

  For example, here's a data set showing the age group and political affiliation of 10 people:

AgeGroup	Affiliation
Young	Democrat
Young	Republican
Old	Republican
Old	Republican
Young	Democrat
Young	Republican
Old	Democrat
Old	Republican
Old	Republican
Young	Democrat

To put the data in a data frame in R, we could type:

```
age <- c("Young", "Young", "Old", "Old", "Young", "Young", "Old", "Old", "Old",
         "Young")

affil <- c("Democrat", "Republican", "Republican", "Republican", "Democrat",
           "Republican", "Democrat", "Republican", "Republican", "Democrat")

x <- data.frame(AgeGroup = age, Affiliation = affil)

x

##    AgeGroup Affiliation
## 1     Young    Democrat
## 2     Young  Republican
## 3       Old  Republican
## 4       Old  Republican
## 5     Young    Democrat
## 6     Young  Republican
## 7       Old    Democrat
## 8       Old  Republican
## 9       Old  Republican
## 10    Young    Democrat
```

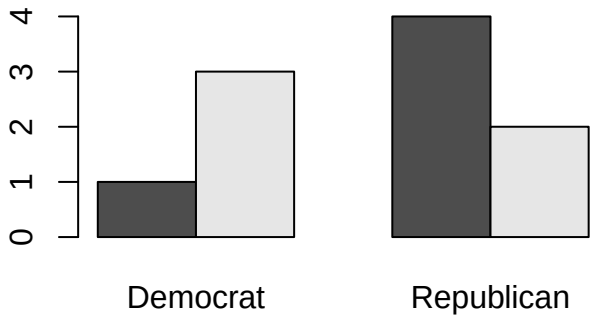
To create a two-way table, in which individuals are cross-classified according to `AgeGroup` and political `Affiliation`, we type:

```
x.tab <- table(x$AgeGroup, x$Affiliation)
x.tab

##
##         Democrat Republican
##   Old          1          4
##   Young        3          2
```

For *two* categorical variables, to make a bar graph, we (usually) specify `beside = TRUE` in `barplot()` so that the bars will be side-by-side (instead of stacked on top of each other). The graph looks like this:

```
barplot(x.tab, beside = TRUE)
```



(The different bar colors represent `AgeGroups`. We could add a legend, if we wanted one, using `legend()`.)

Section 7.2 Exercises

Exercise 5 Here are the `Music` preferences of 11 adults:

```
mus <- c("Rock", "Jazz", "Classical", "Classical", "Rock", "Rock", "Rock",
         "Jazz", "Rock", "Jazz", "Classical")
```

a) Use `table()` to create a table summarizing the music preferences as *counts*. Report your R command(s).

b) Use `prop.table()` to create a table summarizing the music preferences as *proportions*. Report your R command(s).

c) Use `barplot()` and your table from Part *a* to create a bar plot of the *counts*.

Exercise 6 Here are the `Music` *and* `Beverage` preferences of 11 adults:

```r
mus <- c("Rock", "Jazz", "Classical", "Classical", "Rock", "Rock", "Rock",
         "Jazz", "Rock", "Jazz", "Classical")

bev <- c("Beer", "Wine", "Wine", "Wine", "Beer", "Beer", "Wine", "Beer",
         "Beer", "Wine", "Beer")

x <- data.frame(Music = mus, Beverage = bev)
x
```

```
##        Music Beverage
## 1       Rock     Beer
## 2       Jazz     Wine
## 3  Classical     Wine
## 4  Classical     Wine
## 5       Rock     Beer
## 6       Rock     Beer
## 7       Rock     Wine
## 8       Jazz     Beer
## 9       Rock     Beer
## 10      Jazz     Wine
## 11 Classical     Beer
```

a) Use `table()` to create a two-way table, in which individuals are cross-classified according to `Music` and `Beverage` preferences. Report your R command(s).

b) Use `barplot()` (with `beside = TRUE`) and your table from Part *a* to create a bar plot of the *counts*.