MTH 3220 R Notes 6

7 Factors and Tables

7.1 Creating and Viewing Factors and Their Levels

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

  But they differ from `"character"` vectors in the way R stores them internally, and they contain a bit of extra information called ***levels***.

- To create and look at factors, we use:

```
factor()          # Create a factor from a character vector
length()          # Returns the number of         ents in a factor
levels()          # Examine or set the levels of the factor
nlevels()         # Returns the number of levels of the factor
is.factor()       # Indicates whether or not an object is a factor
str()             # Describes the structure of a factor
```

- `factor()` takes a `"character"` vector and converts it 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***, or unique values:

```
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
```

- For example:

```
as.character(my.fac)

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

- To see how many *elements* a factor has, use `length()`. To see how many *levels* it has, use `nlevels()`. To *look* at the levels, use `levels()`.

```
length(my.fac)

## [1] 9

levels(my.fac)

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

nlevels(my.fac)

## [1] 3
```

Section 7.1 Exercises

Exercise 1 Here's a factor `x.fac`:

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

## [1] a a b b c c d d
## Levels: a b c d
```

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

```
length(x.fac)
```

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

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

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

```
nlevels(x.fac)
```

7.2 The Difference Between Factors and `"character"` Vectors

- Unlike `"character"` vectors, R stores factors internally as *numeric vectors*, with `"integer"` values representing the levels of the factor.

  To see, recalling that `typeof()` indicates the internal storage type of an object, and using `my.fac` and `char.vec` from above:

```
typeof(my.fac)

## [1] "integer"

typeof(char.vec)

## [1] "character"
```

- The integer values used internally to represent the levels of a factor can be viewed using:

```
unclass()          # Remove the class attribute of an object.  Can be
                   # used to view the integer values used internally
                   # to represent the levels of a factor.
```

- For example, here's `my.fac` again:

```
my.fac

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

The integers used to represent the levels of `my.fac` internally can be viewed by typing:

```
unclass(my.fac)

## [1] 1 2 3 1 2 3 1 2 3
## attr(,"levels")
## [1] "ctrl" "trt1" "trt2"
```

The numbering is as follows. An observation made at the first level of the factor (i.e. the level in the first position as returned by `levels()`) is represented by the value 1. An observation made at the second level of the factor (as returned by `levels()`) is represented by the value 2, and so on. Above, because the levels of `my.fac` (as returned by `levels()`) are `"ctrl"`, `"trt1"`, and `"trt2"`, observations made under the `"ctrl"` condition are represented by the value 1, those made under `"trt1"` condition by the value 2, and those made under the `"trt2"` condition by the value 3.

Section 7.2 Exercises

Exercise 2 Here's a factor and its levels:

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

## [1] b b c c a a
## Levels: a b c

levels(y.fac)

## [1] "a" "b" "c"
```

Guess what the integer representation of `y.fac` is, then check your answer by typing:

```
unclass(y.fac)
```

Exercise 3 Here's a factor and its levels:

```
y.fac <- factor(c("high", "med", "low", "high", "med", "low"))
y.fac

## [1] high med  low  high med  low
## Levels: high low med

levels(y.fac)

## [1] "high" "low"  "med"
```

Guess what the integer representation of `y.fac` is, then check your answer by typing:

```
unclass(y.fac)
```

7.3 Creating Tables

- To create a table summarizing categorical (text) data (stored as a factor or `"character"` vector), 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()` counts the number of observations at each level of the factor (or at each unique value in the `"character"` vector), and returns the results in a table.
- For example, here's a `"character"` vector containing responses (Yes/No/NotSure) to a survey question:

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

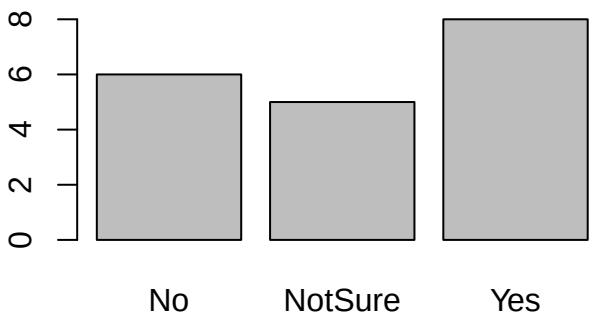
  We can tabulate the responses by typing:

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

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

Here's the bar graph:

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



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

```
prop.table(survey.tab)

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

- We can create a *two-way* table from *two* factors (or `"character"` vectors). For example, here's a data frame in which individuals are cross-classified according to `AgeGroup` and political `Affiliation`:

```
pol.data

##    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, we type:

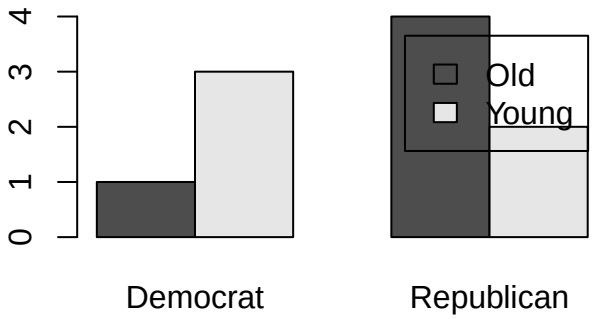
```
pol.tab <- table(pol.data$AgeGroup, pol.data$Affiliation)
pol.tab

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

(Above, we could also have just passed the entire data from to `table()`, i.e. we could've used `table(pol.data)`.)

In this case, the bar graph looks like this:

```
barplot(pol.tab, beside = TRUE, legend.text = TRUE)
```



(Specifying `beside = TRUE` indicates that the columns of the table `pol.tab` should be portrayed as juxtaposed bars, as opposed to stacked bars, and specifying `legend.text = TRUE` indicates that a legend should be included.)

Section 7.3 Exercises

Exercise 4 The built-in R data set `state.region` is a factor indicating the region (Northeast, South, etc.) for each of the 50 states in the U.S. Type:

```
state.region
```

to look at the data set.

a) Use `table()` to count the number of states in each region. Save the table as, say, `states.tab`.

b) Use `prop.table()`, with `states.tab`, to determine the *proportion* of states in each region.

c) Use `barplot()`, with `states.tab`, to create a bar graph of the the number of states in each region. Report your R command(s).