Problem Set 6

CSCI E-5a: Programming in R

Let's clear the environment:

```r
rm( list = ls() )
```

Before you begin, load in the R objects for this problem set:

```r
load( "Problem Set 6 R Objects.Rdata" )
```

```r
ls()
```

```
##  [1] "problem.1.part.a.data"
##  [2] "problem.1.part.b.data"
##  [3] "problem.1.part.c.data"
##  [4] "problem.2.data"
##  [5] "problem.3.data"
##  [6] "problem.4.data"
##  [7] "problem.5.item.1.verbal.response.data"
##  [8] "problem.5.item.2.verbal.response.data"
##  [9] "problem.5.item.3.verbal.response.data"
## [10] "problem.6.comprehensive.assessment.raw.score.data"
## [11] "problem.6.midterm.assessment.raw.score.data"
## [12] "problem.6.problem.set.raw.score.data"
## [13] "problem.6.registration.status.data"
```

Problem 1: Recoding Numeric Representations of Categories

We've seen that one strategy for handling categorical data is to encode it as a number. For instance, the four sales offices of WiDgT could be encoded like this:

Office	Numeric Code
Boston	1
London	2
Salt Lake City	3
Shanghai	4

I'm not enthusiastic about this strategy, and if I am working with a dataset using such an approach I convert the numeric codes to a factor.

In this problem we'll explore three different techniques to convert these numeric codes properly.

Note that this data also contains missing data represented by -9, so we'll also have to deal with that.

Part (a): First method

The vector `problem.1.part.a.data` contains the data for the Office categorical variable, represented using the numeric scheme from the problem statement.

In our first approach, we will use three steps:

- First, change all the -9 values to `NA`.
- Second, convert the numeric vector to a factor using the `factor` function.
- Third, change the levels by directly assigning a character vector with the location names to the levels of the factor.

When you're all done, display a frequency count table of the values in the factor.

Solution

First, let's repair the -9 values:

```
minus.9.locations <-
    which( problem.1.part.a.data == -9 )

problem.1.part.a.data[ minus.9.locations ] <- NA
```

Now we can convert this to a factor:

```
problem.1.part.a.factor <-
    factor( problem.1.part.a.data )
```

Now we can convert the levels:

```
levels( problem.1.part.a.factor ) <-
    c( "Boston", "London", "Salt Lake City", "Shanghai" )
```

Let's check this by examining the first few elements of the factor:

```
head( problem.1.part.a.factor )
```

```
## [1] London         Salt Lake City <NA>           Shanghai       Shanghai
## [6] London
## Levels: Boston London Salt Lake City Shanghai
```

Finally, we can make the table:

```
table( problem.1.part.a.factor )
```

```
## problem.1.part.a.factor
##         Boston         London Salt Lake City       Shanghai
##             36             58             19             25
```

Part (b): Second method

The vector `problem.1.part.b.data` contains the data for the Office categorical variable, represented using the numeric scheme from the problem statement.

In this approach, we'll convert the vector to a factor first, and then change the -9 values to an `NA` when we rename the levels.

- First, convert the numeric vector to a factor using the `factor` function.
- Second, rename the levels by directly assigning a character vector with the location names to the levels of the factor. Note that you can use `NA` in the character vector, so this should correspond with the "-9" level.

When you're all done, display a table of the values in the factor.

For part (b), make sure that you always work with the vector `problem.1.part.b.data`.

Solution

We'll start by converting the numeric vector to a factor:

```
problem.1.part.b.factor <-
    factor( problem.1.part.b.data )

problem.1.part.b.factor
```

```
##   [1] 2  3  -9 4  4  2  1  -9 2  2  1  2  1  1  4  3  1  3  2  4  2  2  4  2  2 
##  [26] 1  1  3  2  2  2  -9 -9 1  2  3  4  4  1  1  4  4  4  2  4  2  2  4  2  1 
##  [51] 3  4  -9 2  2  2  3  1  1  2  1  4  2  1  2  2  2  3  2  1  2  4  2  1  1 
##  [76] 4  3  2  3  -9 3  1  3  2  2  2  1  2  1  2  2  2  2  2  1  3  1  2  3  4 
## [101] 3  2  1  2  2  2  -9 2  2  1  2  4  2  -9 2  1  2  3  1  4  3  1  1  1  2 
## [126] 2  1  2  1  1  3  4  4  1  4  2  4  2  2  2  2  1  4  3  4  1 
## Levels: -9 1 2 3 4
```

Let's check the levels of this factor:

```r
levels( problem.1.part.b.factor )
```

```
## [1] "-9" "1"  "2"  "3"  "4"
```

Now we can convert the levels:

```r
levels( problem.1.part.b.factor ) <-
    c( NA, "Boston", "London", "Salt Lake City", "Shanghai" )
```

Let's check this by displaying the levels once again:

```r
levels( problem.1.part.b.factor )
```

```
## [1] "Boston"         "London"         "Salt Lake City" "Shanghai"
```

Finally, we can make the table:

```r
table( problem.1.part.b.factor )
```

```
## problem.1.part.b.factor
##         Boston         London Salt Lake City       Shanghai
##             36             58             19             25
```

Part (c): Third method

The vector `problem.1.part.c.data` contains the data for the Office categorical variable, represented using the numeric scheme from the problem statement.

Now we'll use a third method!

- Pre-specify the factor levels when we convert the vector to a factor, and this will automatically convert the -9 values to `NA`.
- Then rename the factor levels by directly assigning a character vector with the location names.

When you're all done, display a table of the values in the factor.

For part (c), make sure that you always work with the vector `problem.1.part.c.data`.

Solution

```r
problem.1.part.c.factor <-
    factor(
        problem.1.part.c.data,
        levels =
            c( "1", "2", "3", "4" )
    )

levels( problem.1.part.c.factor ) <-
    c( "Boston", "London", "Salt Lake City", "Shanghai" )
```

In fact, we could have also done this:

```
problem.1.part.c.factor <-
    factor(
        problem.1.part.c.data,
        levels =
            c( "1", "2", "3", "4" ),
        labels = c( "Boston", "London", "Salt Lake City", "Shanghai" )
    )
```

Now let’s construct the table:

```
table( problem.1.part.c.factor )
```

```
## problem.1.part.c.factor
##         Boston         London Salt Lake City       Shanghai
##             36             58             19             25
```

End of Problem 1

Problem 2: Fixing Typos

WiDgT has a listing of their customers, stored in the character vector `problem.2.data`. There are 4 types of customers: Individual, Corporate, Academic, and Govenment. Unfortunately, there were some problems with the data entry, and the data contains some mistakes.

Part (a): Tabulating the values

Use the `table()` function to tabulate the values in the vector `problem.2.data`, and display the result.

Solution

```
table( problem.2.data )
```

```
## problem.2.data
##   Academic    acadmic  corporate  Corporate   corprate     govern Government
##         73          1          2         48          1          2         55
## Individual
##         39
```

Part (b): Converting to a factor

Convert the `problem.2.data` vector to a factor, but don't try to repair anything. Then directly display the first 8 values using a `head()` command.

Solution

```
problem.2.factor <-
    factor( problem.2.data )

head( problem.2.factor )
```

```
## [1] Corporate  Government Government govern     Individual Corporate
## 8 Levels: Academic acadmic corporate Corporate corprate govern ... Individual
```

Part (c): Grouping levels

Repair the typos by grouping factor levels. For instance, all the various misspellings of the level "Corporate" should be grouped to the correct name, and likewise with the other levels. When you're done, construct a table and display it.

Solution

```
levels( problem.2.factor ) <-
    c( "Academic", "Academic",
       "Corporate", "Corporate", "Corporate",
       "Government", "Government",
       "Individual"
    )
```

```
table( problem.2.factor )
```

```
## problem.2.factor
##   Academic  Corporate Government Individual
##         74         51         57         39
```

End of Problem 2

Problem 3: Cutting Numeric Data

A depression scale has been developed so that a patient receives a numerical score, ranging from 0 to 15, which then used as the basis for a diagnosis:

- If the score is greater than or equal to 13, then the patient is diagnosed with "Major Depression".
- If the score is greater than or equal to 10 but strictly less than 13, then the patient is diagnosed with "Mild Depression".
- If the score is greater than or equal to 7 but strictly less than 10, then the patient is diagnosed with "Borderline Depression".
- If the score is strictly less than 7, then the patient is diagnosed as "Normal".

The variable `problem.3.data.` contains depression scores for a sample of patients.

Part (a): The `cut()` function

Use the `cut()` function to create a factor with the ranges for the different depression diagnoses. Store this factor in a variable, and directly display the first 8 values using a `head()` statement.

Solution

```
depression.score.factor <-
    cut(
        problem.3.data,
        breaks = c(0, 7, 10, 13, Inf),
        labels = c( "Normal", "Borderline Depression",
                    "Mild Depression", "Major Depression" ),
        right = FALSE
    )

head( depression.score.factor, 8 )
```

```
## [1] Mild Depression       Normal                Mild Depression
## [4] Normal                Borderline Depression Normal
## [7] Normal                Normal
## Levels: Normal Borderline Depression Mild Depression Major Depression
```

Part (b): Frequency count table

Construct a frequency count table of the values in the factor you created in part (a). Save this frequency count table in a variable, and display it directly.

Solution

```
depression.score.frequency.count.table <-
  table( depression.score.factor )

depression.score.frequency.count.table
```

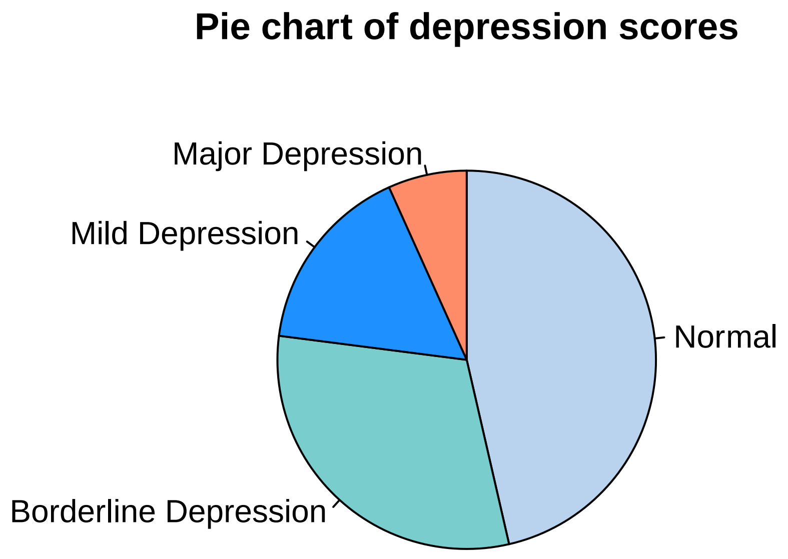
```
## depression.score.factor
##                Normal Borderline Depression       Mild Depression
##                   200                   132                    70
##      Major Depression
##                    29
```

Part (c): Pie chart

Using the table you created in part (b), construct a pie chart to display the relative proportions of the three diagnoses. Be sure to include a main title, set the pie slices to clockwise rotation, and specify a set of colors for the pie chart.

Solution

```
pie(
  x = depression.score.frequency.count.table,
  main = "Pie chart of depression scores",
  clockwise = TRUE,
    col = c( "slategray2", "darkslategray3", "dodgerblue1", "salmon1")
)
```



End of Problem 3

Problem 4: Hank Aaron Career Batting Performance

So far in CSCI S-5a when we've calculated baseball batting statistics I've provided a nice tabulation of all the basic information.

But what if you don't have a nice tabulation, but instead just have a set of raw data values?

The vector `problem.4.data` contains a random sequence of values representing the career singles, doubles, triples, home runs, and outs for Hank Aaron.

Part (a): Tabulating the values

Construct a frequency count table that summarizes the values in `problem.4.data` and display it directly.

Solution

```
table( problem.4.data )
```

```
## problem.4.data
##   Double Home Run      Out   Single   Triple
##      624      755     8593     2294       98
```

Part (b): Reordered table

The problem with this table is that, while all the numbers are correct, they aren't listed in the conventional order. Instead, we would like this order:

- Single
- Double
- Triple
- Home Run
- Out

To resolve this issue, convert the character vector `problem.4.data` into a factor with the preferred ordering of the levels, and save this in a variable. Then once again construct a table and display it; it should still have the same numeric values, but this time they should be listed in the conventional manner.

Solution

```
hank.aaron.at.bats.factor <-
    factor(
        problem.4.data,
        levels =
            c( "Single", "Double", "Triple",
               "Home Run", "Out" )
    )
```

Now let's make the table:

```
hank.aaron.at.bats.frequency.count.table <-
    table( hank.aaron.at.bats.factor )

hank.aaron.at.bats.frequency.count.table
```

```
## hank.aaron.at.bats.factor
##   Single   Double   Triple Home Run      Out
##     2294      624       98      755     8593
```

Part (c): Relative proportions table

Construct a relative proportions table of the values in `problem.4.data`.

In this table, we're not reporting one of the special batting metrics (i.e. batting average, on-base percentage, and slugging percentage) but instead are simply interested in the relative proportions. Thus, we don't have to use the specialized baseball reporting convention of 3 decimal places. Instead, since we're focused on the relative proportions, keep the display simple and just use 2 decimal places.

Solution

```
round(
  proportions( hank.aaron.at.bats.frequency.count.table ),
  digits = 2
)
```

```
## hank.aaron.at.bats.factor
##   Single   Double   Triple Home Run      Out
##     0.19     0.05     0.01     0.06     0.70
```

End of Problem 4

Problem 5: Likert Scales

In a *Likert scale*, study subjects are presented with a set of statements and a range of possible responses for each statement.

For each item, the subject's response is converted to a numerical score, and then the individual scores are summed together to obtain an overall score.

This overall score is then compared to a range of intervals, and the subject is categorized depending on which interval the score falls in.

For instance, suppose we have a simple Likert scale for diagnosing if a subject is upset, consisting of 3 statements:

Item	Statement
1	"I feel sad"
2	"I feel angry"
3	"I feel irritated"

Each statement can be answered with one of 5 verbal responses:

Verbal Response	Numeric Score
"Strongly Disagree"	0
"Disagree"	1
"No Opinion"	2
"Agree"	3
"Strongly Agree"	4

To diagnose a subject, we first convert the subject's verbal responses for each item to the corresponding numeric scores, and then sum the numeric scores to obtain a total score for that subject.

Then we can look up the diagnosis on an interval scale:

Total Score	Diagnosis
0 - 4	"Not Upset"
5 - 8	"Moderately Upset"
9 - 12	"Very Upset"

For instance, suppose a subject has these responses:

Item	Statement	Response	Score
1	"I feel sad"	"Agree"	3
2	"I feel angry"	"No Opinion"	2
3	"I feel irritated"	"Strongly Agree"	4

Thus, the subject has a total score of 3 + 2 + 4 = 9.

Then we can look up the diagnosis on the interval scale, and we find that it falls in the range 9 - 12, which has a diagnosis of "Very Upset".

This is a very simple example of a Likert scale, but don't be deceived – these simple scored questionnaires are the basis for much of mental health research.

Some famous Likert scales that are often used in research are:

- The *Beck's Depression Inventory* (BDI)
- The *Hamilton Depression Rating Scale* (HDRS)
- The *Pittsburgh Sleep Quality Index* (PSQI)

In this problem, we'll score a simple Likert scale, produce a table of the relative proportions for the diagnoses, and construct a barplot visualization.

Part (a): Constructing the verbal response scoring lookup vector

Construct a lookup vector that will convert verbal responses to the corresponding numeric score using the table from the problem statement:

Verbal Response	Numeric Score
"Strongly Disagree"	0
"Disagree"	1
"No Opinion"	2
"Agree"	3
"Strongly Agree"	4

Save this lookup vector in a variable, and display it directly so the TAs can check that you did this properly.

Solution

```r
verbal.response.lookup.vector <-
  c(
    "Strongly Disagree" = 0,
    "Disagree" = 1,
    "No Opinion" = 2,
    "Agree" = 3,
    "Strongly Agree" = 4
  )

verbal.response.lookup.vector
```

```
## Strongly Disagree          Disagree        No Opinion             Agree
##                 0                 1                 2                 3
##    Strongly Agree
##                 4
```

Part (b): Scoring the first response item

The vector `problem.5.item.1.verbal.response.data` contains the verbal responses to the item 1 prompt "I feel sad" for a cohort of study subjects.

Using the lookup vector that you constructed in part (a), create a vector consisting of the numeric scores corresponding to the verbal responses in `problem.5.item.1.verbal.response.data`. Then construct a table of the frequency counts of these numeric scores so that the TAs can be sure you did this correctly.

Solution

```
problem.5.item.1.numeric.scores.vector <-
  verbal.response.lookup.vector[
    problem.5.item.1.verbal.response.data
  ]

problem.5.item.1.numeric.scores.vector[ 1:8 ]
```

```
##    Strongly Agree          Disagree          Disagree          Disagree
##                 4                 1                 1                 1
## Strongly Disagree        No Opinion          Disagree    Strongly Agree
##                 0                 2                 1                 4
```

```
table( problem.5.item.1.numeric.scores.vector )
```

```
## problem.5.item.1.numeric.scores.vector
##   0   1   2   3   4
## 186 228 157 109  73
```

Part (c): Scoring the second response item

The vector `problem.5.item.2.verbal.response.data` contains the verbal responses to the item 2 prompt "I feel angry" for a cohort of study subjects.

Using the lookup vector that you constructed in part (a), create a vector consisting of the numeric scores corresponding to the verbal responses in `problem.5.item.2.verbal.response.data`. Then construct a table of the frequency counts of these numeric scores so that the TAs can be sure you did this correctly.

Solution

```
problem.5.item.2.numeric.scores.vector <-
  verbal.response.lookup.vector[
    problem.5.item.2.verbal.response.data
  ]

table( problem.5.item.2.numeric.scores.vector )
```

```
## problem.5.item.2.numeric.scores.vector
##   0   1   2   3   4
## 207 201 158 114  73
```

Part (d): Scoring the third response item

The vector `problem.5.item.3.verbal.response.data` contains the verbal responses to the item 3 prompt "I feel irritated" for a cohort of study subjects.

Using the lookup vector that you constructed in part (a), create a vector consisting of the numeric scores corresponding to the verbal responses in `problem.5.item.3.verbal.response.data`. Then construct a table of the frequency counts of these numeric scores so that the TAs can be sure you did this correctly.

Solution

```
problem.5.item.3.numeric.scores.vector <-
  verbal.response.lookup.vector[
    problem.5.item.3.verbal.response.data
  ]

table( problem.5.item.3.numeric.scores.vector )
```

```
## problem.5.item.3.numeric.scores.vector
##   0   1   2   3   4
## 190 237 148 103  75
```

Part (e): Calculating the total score

Using vectorized addition with the vectors of numeric scores that you created in parts (b) through (d), construct a vector consisting of the total score on the scale for each participant in the study cohort. Store this vector in a variable, and directly display the first 8 values of it.

Solution

```
problem.5.total.score.vector <-
  problem.5.item.1.numeric.scores.vector +
  problem.5.item.2.numeric.scores.vector +
  problem.5.item.3.numeric.scores.vector

head( problem.5.total.score.vector, n = 8 )
```

```
##    Strongly Agree          Disagree          Disagree          Disagree
##                11                 5                 2                 2
## Strongly Disagree        No Opinion          Disagree    Strongly Agree
##                 3                 9                 3                12
```

Part (f): Categorizing the scores

From the problem statement, we have this table of intervals and their associated verbal diagnosis:

Interval Range	Verbal Diagnosis
0 <= Total Score < 5	"Not Upset"
5 <= Total Score < 9	"Moderately Upset"
9 <= Total Score	"Very Upset"

Using the `cut()` function with the vector of total scores that you constructed in part (e), create a factor with the corresponding verbal diagnoses for the study cohort. Save this factor in a variable, and directly display the first 8 values.

Solution

```
problem.5.verbal.diagnosis.factor <-
  cut(
    x = problem.5.total.score.vector,
    breaks = c( -Inf, 4, 8, 12 ),
    labels = c( "Not Upset", "Moderately Upset", "Very Upset" )
  )

head( problem.5.verbal.diagnosis.factor, n = 8 )
```

```
## [1] Very Upset       Moderately Upset Not Upset        Not Upset       
## [5] Not Upset        Very Upset       Not Upset        Very Upset      
## Levels: Not Upset Moderately Upset Very Upset
```

Part (g): Frequency count table

Using the factor of verbal diagnoses that you constructed in part (f), construct a table of the verbal diagnosis frequency counts. Save this table in a variable, and display it directly for the TAs.

Solution

```
problem.5.verbal.diagnosis.frequency.count.table <-
  table( problem.5.verbal.diagnosis.factor )

problem.5.verbal.diagnosis.frequency.count.table
```

```
## problem.5.verbal.diagnosis.factor
##        Not Upset Moderately Upset       Very Upset 
##              436              195              122
```

Part (h): Relative proportions table

Using the frequency count table that you constructed in part (g), construct a table of the relative proportions of the verbal diagnoses. Save this table in a variable, and display it directly, rounding the values to 2 decimal places.

Solution

```
problem.5.verbal.diagnosis.relative.proportions.table <-
  proportions( problem.5.verbal.diagnosis.frequency.count.table )

round(
  problem.5.verbal.diagnosis.relative.proportions.table,
  digits = 2
)
```

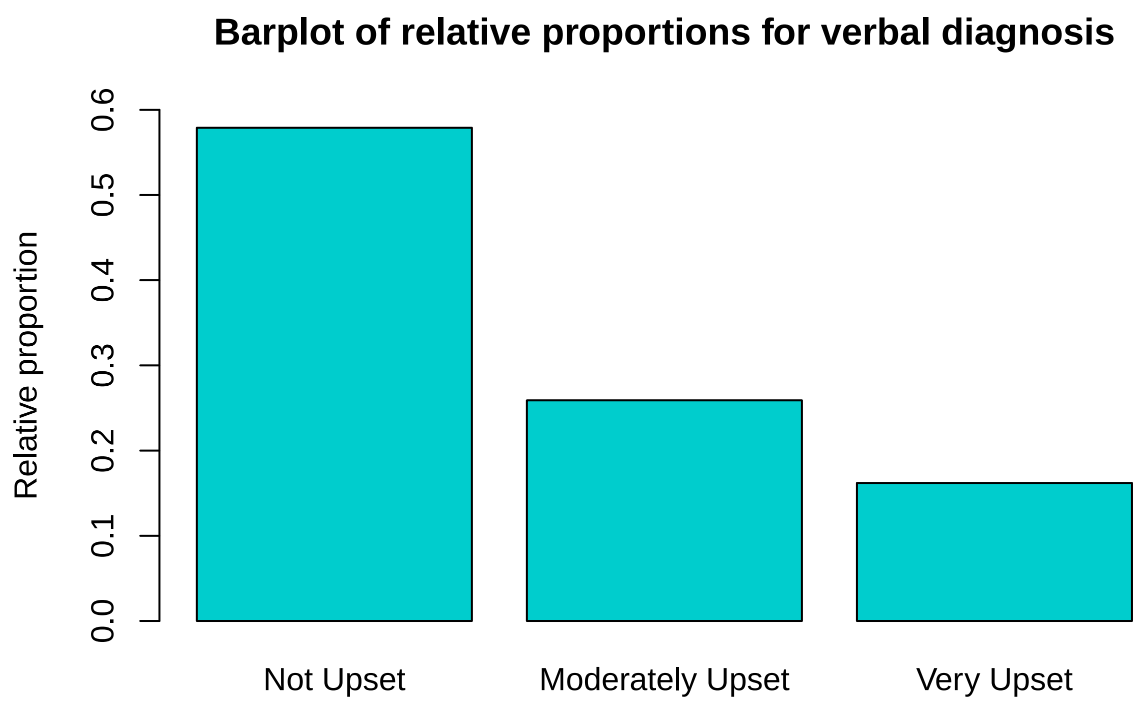
```
## problem.5.verbal.diagnosis.factor
##        Not Upset Moderately Upset       Very Upset 
##             0.58             0.26             0.16
```

Part (i): Barplot

Using the table that you constructed in part (h), draw a barplot that displays the relative proportions of the different diagnoses. Set the range of the y-axis to go from 0 to 0.6, include a main title and titles for the x-axis and y-axis, and explicitly specify a color for the bars.

Solution

```r
barplot(
  height = problem.5.verbal.diagnosis.relative.proportions.table,
  ylim = c(0, 0.6),
  main = "Barplot of relative proportions for verbal diagnosis",
  ylab = "Relative proportion",
  col = "cyan3"
)
```



End of Problem 5

Problem 6: Final Grades – Finally!

This is one of my favorite problems in CSCI E-5a, and I'm excited to present it to you.

The actual calculation itself isn't particularly difficult or amazing, but what makes this problem so appealing is that you'll see how we can perform a sophisticated computation using just the tools that we've developed so far in this course.

Each step is very simple, but when combined together the end result is impressive.

OK, enough blather – let's go!!

We've spend a lot of time in CSCI E-5a learning how to calculate final grades.

That's because this seemingly simple procedure actually involves techniques from most of our lectures so far:

- From Lecture 2, we use the concept of variables to store intermediate results in an extended computation.
- From Lecture 3, we use the concepts of vectors and vectorized operations to automate the calculation of preliminary scores for an entire group of students.
- From Lecture 4, we use the method of iteration to construct a vector consisting of the maximum of the two preliminary scores for each student.
- From Lecture 5, we can perform conditional branching with the `ifelse()` function as an alternative method for constructing a vector consisting of the maximum of the two preliminary scores for each student.
- Back to Lecture 3, we use lookup vectors to pro-rate undergraduate scores.

In Problem 6 from Problem Set 5, we put all these techniques together, and we were able to calculate the final course score, given the raw scores for each student and their registration status.

However, for all the work that we've put into calculating final grades, we *still* aren't able to execute the entire procedure using R.

The problem is that we haven't been able to assign letter grades based on the final course score.

In this problem, we will finally see how to automate this entire process in R, starting with the raw scores for each student and their registration status, and ending with a factor consisting of the final course letter grade for each student.

Before we start: can you think of a tool that we have learned in this lecture that will enable us to perform this final step?

Much of this problem is a repetition of the work that you did in Problem Set 5, Problem 6, and you are encouraged to re-use your code from that problem or to use the code in the Problem Set 5 solutions.

Part (a): Standardizing the Problem Set scores

The vector `problem.6.problem.set.raw.score.data` contains numeric data on problem set raw scores for a group of students.

For this problem, a raw score of 88 points is equivalent to a standardized score of 100 points.

Standardize this numeric data, so that the maximum number of possible points is 100.

Save these standardized values in a variable, and report the first 8 values using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
problem.set.standardized.score.vector <-
  problem.6.problem.set.raw.score.data / 88 * 100

cat(
  "Problem Set standardized score vector:",
  formatC(
    head( problem.set.standardized.score.vector, n = 8),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Problem Set standardized score vector: 86.36 88.64 86.36 92.05 77.27 92.05 86.36 94.32
```

Part (b): Standardizing the Midterm Assessment scores

The vector `problem.6.midterm.assessment.raw.score.data` contains numeric data on Midterm Assessment raw scores for a group of students.

The maximum number of possible points for the Midterm Assessment raw score is 80.

Standardize this numeric data, so that the maximum number of possible points is 100.

Save these standardized values in a variable, and report the first 8 values using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
midterm.assessment.standardized.score.vector <-
  problem.6.midterm.assessment.raw.score.data / 80 * 100

cat(
  "Midterm Assessment standardized score vector:",
  formatC(
    head( midterm.assessment.standardized.score.vector, n = 8),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Midterm Assessment standardized score vector: 90.00 78.75 93.75 93.75 73.75 91.25 91.25 95.00
```

Part (c): Standardizing the Comprehensive Assessment scores

The vector `problem.6.comprehensive.assessment.raw.score.data` contains numeric data on Comprehensive Assessment raw scores for a group of students.

The maximum number of possible points for the Comprehensive Assessment raw score is 80.

Standardize this numeric data, so that the maximum number of possible points is 100.

Save these standardized values in a variable, and report the first 8 values using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
comprehensive.assessment.standardized.score.vector <-
  problem.6.comprehensive.assessment.raw.score.data / 80 * 100

cat(
  "Comprehensive Assessment standardized score vector:",
  formatC(
    head( comprehensive.assessment.standardized.score.vector, n = 8),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Comprehensive Assessment standardized score vector: 90.00 80.00 92.50 91.25 70.00 90.00 87.50 95.00
```

Part (d): Preliminary Score 1

Recall that the Preliminary Score 1 is calculated as a weighted average of 3 components:

Componenent	Weight
Problem Set Standardized Score	0.20
Midterm Assessment Standarized Score	0.30
Comprehensive Assessment Standardized Score	0.50

Using vectorized operations with the standardized score vectors that you created in parts (a) through (c), construct a vector of the Preliminary Score 1 values for the students. Save this vector in a variable, and report the first 8 elements using a `cat()` statement, displaying each value with 2 decimal places.

Solution

```
preliminary.score.1.vector <-
  (0.2 * problem.set.standardized.score.vector) +
  (0.3 * midterm.assessment.standardized.score.vector) +
  (0.5 * comprehensive.assessment.standardized.score.vector)

cat(
  "Preliminary Score 1:",
  formatC(
    head( preliminary.score.1.vector, n = 8 ),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Preliminary Score 1: 89.27 81.35 91.65 92.16 72.58 90.78 88.40 94.86
```

Part (e): Preliminary Score 2

Recall that the Preliminary Score 2 is calculated as a weighted average of 2 components:

Componenent	Weight
Midterm Assessment Standarized Score	0.35
Comprehensive Assessment Standardized Score	0.65

Using vectorized operations with the standardized score vectors that you created in parts (a) through (c), construct a vector of the Preliminary Score 2 values for the students. Save this vector in a variable, and report the first 8 elements using a `cat()` statement, displaying each value with 2 decimal places.

Solution

```
preliminary.score.2.vector <-
  (0.35 * midterm.assessment.standardized.score.vector) +
  (0.65 * comprehensive.assessment.standardized.score.vector)

cat(
  "Preliminary Score 2:",
  formatC(
    head( preliminary.score.2.vector, n = 8 ),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Preliminary Score 2: 90.00 79.56 92.94 92.12 71.31 90.44 88.81 95.00
```

Part (f): Graduate Course Score

The Graduate Course Score is the maximum of the Preliminary Score 1 and Preliminary Score 2 values.

Construct a vector of the Graduate Course Score values for each student in the group. Save this vector in a variable, and display the first 8 elements of your result using a `cat()` statement, displaying each value with 2 decimal places.

Solution

We can do this using iteration:

```
number.of.students <-
  length( preliminary.score.1.vector )

graduate.course.score.vector <-
  numeric( number.of.students )

for( index in 1:number.of.students ) {

  graduate.course.score.vector[ index ] <-
    max(
      preliminary.score.1.vector[ index ],
      preliminary.score.2.vector[ index ]
    )
}

cat(
  "Graduate Course Score vector:",
  formatC(
    head( graduate.course.score.vector, n = 8 ),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Graduate Course Score vector: 90.00 81.35 92.94 92.16 72.58 90.78 88.81 95.00
```

We can also use the `ifelse()` function:

```
graduate.course.score.vector <-
  ifelse(
    preliminary.score.1.vector >= preliminary.score.2.vector,
    preliminary.score.1.vector,
    preliminary.score.2.vector
  )

cat(
  "Graduate Course Score vector:",
  formatC(
    head( graduate.course.score.vector, n = 8 ),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Graduate Course Score vector: 90.00 81.35 92.94 92.16 72.58 90.78 88.81 95.00
```

Part (g): Pro-Rating Undergraduate Scores

Each student has a registration status, which is either "Graduate" or "Undergraduate".

CSCI E-5a is taught as a Master's-level graduate course, and students registered for undergraduate credit are only required to do 75% of the work of students registered for graduate credit.

Thus, I "pro-rate" an undergraduate student's score to reflect this difference in expectations.

If a student is registered for Undergraduate credit, I multiply (or "pro-rate") the Graduate Course Score by 4/3 to obtain the Final Course Score.

If a student is registered for graduate credit, the Final Course Score is just the Graduate Course Score.

You can think of this as multiplying the Graduate Course Score by 1.

Thus, a pro-rating multiplier is associated with each registration status:

Registration Status	Pro-Rating Multiplier
Undergraduate	4/3
Graduate	1

For this problem, the registration status data is contained in the vector `problem.6.registration.status.data`.

First, construct a numeric lookup vector that can take a registration status and return the corresponding pro-rating multiplier.

Next, using this numeric lookup vector and the `problem.6.registration.status.data` vector, construct a vector of pro-rating multipliers for this group of students and save this in a variable.

Finally, use this vector of pro-rating multipliers and the vector of Graduate Course Scores from part (f) to construct a vector of Final Course Scores for each student. Save this vector in a variable, and report the first 8 values of your result using a `cat()` statement, displaying each value with 2 decimal places.

Solution

```r
pro.rating.multiplier.lookup.vector <-
  c(
    "Undergraduate" = 4/3,
    "Graduate" = 1
  )

pro.rating.multiplier.vector <-
  pro.rating.multiplier.lookup.vector[
    problem.6.registration.status.data
  ]

final.course.score.vector <-
  graduate.course.score.vector * pro.rating.multiplier.vector

cat(
  "Final Course Score vector:",
  formatC(
    head( final.course.score.vector, n = 8 ),
    format = "f",
    digits = 2
  ),
  "\n"
)
```

```
## Final Course Score vector: 90.00 81.35 92.94 92.16 72.58 90.78 88.81 95.00
```

Part (h): Assigning letter grades

Now we're going to perform the very last step in our calculation.

Given a student's final score, we determine which range it falls within, and then assign the corresponding letter grade:

Range	Letter Grade
93 <= Final Score	A
90 <= Final Score < 93	A-
87 <= Final Score < 90	B+
83 <= Final Score < 87	B
80 <= Final Score < 83	B-
77 <= Final Score < 80	C+
73 <= Final Score < 77	C
70 <= Final Score < 73	C-
67 <= Final Score < 70	D+
60 <= Final Score < 67	D
Final Score < 60	F

Do you see how we can do this?

We can use the `cut()` function!

Using the `cut()` function along with the vector of Final Course Scores that you calculated in part (g), construct a vector of letter grades for the students in the group. Save this vector in a variable, and report the first 8 elements using a `cat()` statement.

BIG HINT: Look at the Final Course Score for the very first student, and make sure that your code assigns the correct letter grade for this score.

Solution

```r
letter.grades.factor <-
  cut(
    final.course.score.vector,
    breaks = c( -Inf, 60, 67, 70, 73, 77, 80, 83, 87, 90, 93, Inf ),
    labels = c( "F", "D", "D+", "C-", "C", "C+", "B-", "B", "B+", "A-", "A" ),
    right = FALSE
  )

head( letter.grades.factor, n = 8 )
```

```
## [1] A- B- A- A- C- A- B+ A
## Levels: F D D+ C- C C+ B- B B+ A- A
```

Part (i): Grouping the letter grades

Let's group the grades so that they are just the letter itself, without any plus or minus:

Letter Grades	Grouped Letter Grade
A or A-	A
B+, B, or B-	B
C+, C, or C-	C
D+ or D	D
F	F

Create a new factor with just the group levels "A", "B", "C", "D", and "F" by grouping the levels of the letter grades factor you created in part (h). Save this grouped factor in a variable, and directly display the first 8 values.

Solution

```
grouped.letter.grade.factor <-
  letter.grades.factor

levels( grouped.letter.grade.factor ) <-
  c( "F", "D", "D", "C", "C", "C", "B", "B", "B", "A", "A" )

head( grouped.letter.grade.factor )
```

```
## [1] A B A A C A
## Levels: F D C B A
```

Part (j): Constructing a relative proportion table

Construct a table of the relative proportions of the grouped grades. Save this table in a variable, and directly display it, formatting the values with 2 decimal places.

Solution

```
grouped.letter.grade.relative.proportions.table <-
  proportions(
    table( grouped.letter.grade.factor )
  )

round(
  grouped.letter.grade.relative.proportions.table,
  digits = 2
)
```

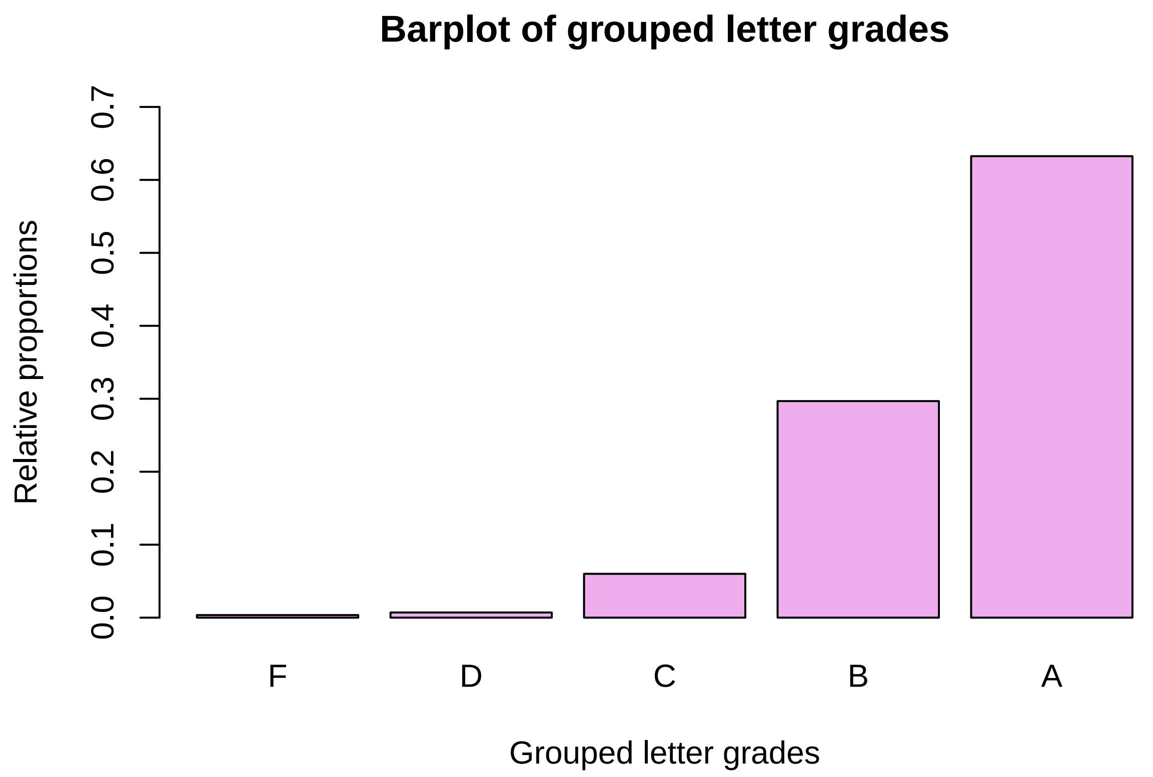
```
## grouped.letter.grade.factor
##    F    D    C    B    A
## 0.00 0.01 0.06 0.30 0.63
```

Part (k): Barplot

Using the table that you constructed in part (j), draw a barplot that displays the relative proportions of the different letter grades. Set the range of the y-axis to go from 0 to 0.7, include a main title and titles for the x-axis and y-axis, and explicitly specify a color for the bars.

Solution

```
barplot(
  height = grouped.letter.grade.relative.proportions.table,
  ylim = c(0, 0.7),
  main = "Barplot of grouped letter grades",
  xlab = "Grouped letter grades",
  ylab = "Relative proportions",
  col = "plum2"
)
```


Barplot of grouped letter grades
Relative proportions
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
F
D
C
B
A
Grouped letter grades