Thesis Proposal

Visualization tools for genealogical and RNA-sequencing datasets

Lindsay Rutter

Program of Study Committee:

Dianne Cook, Major Professor
Amy Toth, Major Professor
Heike Hofmann
Daniel Nettleton
James Reecy

May 16, 2016

Contents

Chapter 1

Introduction

1.1 Summary

The `ePort` package provides tools for course instructors to generate electronic reports regarding student performance. Instructors can produce reports immediately after homework assignment deadlines, and use them to better understand student performance throughout the teaching semester. The goal is to allow instructors to assess and improve upon their teaching approaches in a fast response cycle.

The tools in this package will be especially beneficial for users who supervise large introductory courses. These courses often consist of multiple topics (groups of learning outcomes) that are taught by multiple instructors across multiple sections (groups of students). To accomodate the various ways that student performance can be examined for such courses, the package can generate various reports that can compare within and between topics and sections.

At its simplest, a report can be generated for one topic and one section. This would allow course coordinators to determine how well a particular section performed on a particular topic. Reports can also be generated for one topic across multiple sections, which can allow course coordinators to quickly determine how well and consistently the multiple sections performed on a topic of interest. This could be particularly insightful in cases where discrepancies in student performance are discovered between sections, especially if different instructors and/or teaching methods are being used across the sections.

In addition, we can produce reports for one unit (group of topics), either within one section or between multiple sections. This allows coordinators to assess student performance across all the learning outcomes of the combined topics that form the unit, and to quantify the consistency of how students perform across sections.

In general, both short and long versions of reports can be generated. Short versions of reports provide brief summarizations of student performance without regard to individual problems, whereas long versions of reports provide detailed summarizations of student performance for each individual problem in the assignment. Hence, long reports can also be used to confirm the suitability of assigned problems. For instance, in some courses, problems that assess the same learning outcome and are intended to be of equal difficulty levels are grouped into a question set, and each student is assigned a random subset from this set of problems. However, sometimes, an unexpected discrepancy in student performance between problems in a given quesiton set will be discovered, indicating an unintended discrepancy in the clearness or difficulty level of the problems to which students were randomly assigned. This package will allow users to efficiently find and fix such issues.

1.2 Installation

R is a open source software project for statistical computing, and can be freely downloaded from the Comprehensive **R** Archive Network (`CRAN`) website. The link to contributed documentation on the `CRAN` website offers practical resources for an introduction to **R** in several languages. After downloading and installing **R**, the installation of additional packages is straightforward. To install the `ePort` package from **R**, use the command:

```r
install.packages("ePort")
```

The `ePort` package should now be successfully installed. Next, to render it accessible to the current **R** session, simply type:

```r
library(ePort)
```

1.3 Help files

To access help pages with example syntax and documentation for the available functions of the `ePort` package, please type:

```r
help(package="ePort")
```

To access more detailed information about a specific function in the `ePort` package, use the following help command on that function, such as:

```r
help(mergeSection)
```

The above command will return the help file for the function. Notice that this help file includes freestanding example syntax to illustrate how function commands are executed. This is the case in help files for most functions. The provided example code can be pasted directly into an **R** session.

Chapter 2

Online homework database

Amy Froelich, one of the authors of the `ePort` package, developed an online homework database that has been applied for years in a large multi-section introductory statistics course. The resulting student data from this database has been applied to the `ePort` package, and has been useful in discovering common patterns and/or problems in student learning in this course. In this section of the vignette, we will describe the configuration of this particular database, keeping in mind that a course supervisor who is interested in applying the `ePort` package to their course can do so by constructing their own online homework database in a similar format to the one described below.

2.1 Database structure

The online homework database consists of 184 learning outcomes, which are grouped into 26 topics. In total, the database contains 2000 questions. A given student does not receive all 2000 questions over the course of the semester. Instead, similar questions are grouped together to form 330 question sets, and a given student will receive a specified number of questions (usually one) from each of these 330 question sets.

2.1.1 Topics

The topics in the database cover a broad range of material that includes curriculum from Advanced Placement Statistics and popular introductory statistics textbooks. The topics are not rigidly structured around a specific textbook, and are not self-contained. Hence, course supervisors can tailor their course by selecting a subset of and/or reordering topics from the database. The full list of topic numbers and descriptions from the database is provided below in Table 2.1.

Table 2.1: Topic numbers and descriptions

Number	**Description**
01	Data
02	Descriptive Statistics for a Single Categorical Variable
03	Descriptive Statistics for a Single Quantitative Variable
04	Descriptive Statistics for a Contingency Table
05	Descriptive Statistics for a Single Quantitative Variable between Groups
06	Normal Distribution
07	Descriptive Statistics for a Scatterplot
08	Descriptive Linear Regression
09	Samples and Surveys
10	Experiments
11	Randomness and Probability
12	Introduction to Probability and Events
13	Introduction to Random Variables
14	Binomial and Poisson Distributions
15	Sampling Distribution for the Sample Proportion
16	Confidence Intervals for the Population Proportion
17	Hypothesis Tests for the Population Proportion
18	Sampling Distribution for the Sample Mean
19	Confidence Intervals for the Population Mean
20	Hypothesis Tests for the Population Mean
21	Inference for the Difference in Two Population Proportions
22	Inference for the Difference in Two Population Means
23	Inference for the Mean Difference (Paired Samples)
24	Goodness of Fit Tests
25	Inference for Contingency Tables
26	Inference for Simple Linear Regression

2.1.2 Learning outcomes

Each topic contains learning outcomes, which are a list of statements that describe what a student is expected to understand after completing the topic. Learning outcomes form the main structure of the electronic assessment model of `ePort`. The average topic contains about seven learning outcomes. As an example, the learning outcomes for `Topic 03` are provided in List 1 below.

List 1: Learning outcomes for Topic 03

A. Use standardizing to determine how many standard deviations an observation is away from the mean value.

B. Use z-scores to compare observations for different quantitative variables.

C. Explain how standardizing affects the shape, center, and variability of the distribution of a quantitative variable.

D. Determine which quantitative variables could be modeled using the normal distribution by interpreting graphical representations of the variable.

E. Apply the 68-95-99.7 Rule to any quantitative variable with a normal distribution.

F. Find percentile or area values for any given observation from a normal distribution.

G. Find the value of an observation when given a percentile or area value from the normal distribution.

2.1.3 Question types

The majority of the questions in the database include real data examples that cover diverse application areas, excepting business. The questions are time-tested in that they have continuously been edited and improved upon after evaluating student responses time and again. The majority of these questions also include feedback, which can be correct/incorrect feedback or answer-specific feedback. There are seven types of questions, which are displayed in Table 2.2.

Table 2.2: Seven types of questions in the database

Abbreviation	**Type**	**Possible Points**
TF	True/False	1
MC	Multiple Choice	1
MU	Multiple Answer	1
MA	Matching	Number of Matches
FB	Fill in the Blank	Number of Blanks
JS	Jumbled Sentence	Number of Blanks
CA	Calculation	1

In each of the seven boxes that follow, we provide an example question and example solution. For demonstration purposes, each box covers a different question type in the database.

Example TF:

Does regular exercise lead to higher VO_2 max? VO_2 max is the maximum amount of oxygen in millimeters, one can use in one minute per kilogram of body mass. A random sample of 20 college age women was selected. Each student was asked whether or not they exercised regularly (at least 30 minutes of aerobic exercise 3 times a week). The VO_2 max for each student was also taken. This is an observational study.

a. True
b. False

Correct answer: a

Example MC:

The z-score for a particular observation is z = -3.1. This means the observation is:

a. 3.1 standard deviations above the mean
b. 3.1 standard deviations below the mean
c. 3.1 units above the mean
d. 3.1 units below the mean

Correct answer: b

Example MU:

Which of the following characteristics of pie is/are quantitative variables? Choose ALL that apply.

a. Calorie count
b. Number of cups of flour used
c. Type of pie (pecan, blueberry, etc)
d. Brand of sugar used

Correct answers: a and b

Example MA:

An ultramarathon is a foot race that is longer than 26.2 miles. Doctors have found that people who run an ultramarathon are at increased risk for developing respiratory infections after the race. Doctors beleive that taking vitamin C the 10 days before and the 10 days after the race would reduce the incidence of respiratory infections in the ultramarathon runners. To test their hypothesis, 20 runners were randomly assigned into two groups of 10 runners each. One group was given the same dose of vitamin C, in pill form, for 10 days before and 10 days are the race and the other group was given a sugar pill. Ten days after the race, the two groups were studied to determine how many of the runners in each group developed a respiratory infection. Match the ordered terms (1-4) to the correct description (a-d).

1. Experimental units
2. Response variable
3. Factor
4. Treatments

a. 20 ultramarathon runners
b. The use of vitamin C by ultramarathon runners
c. Whether or not the runner developed a respiratory infection
d. Vitamin C, Sugar pill

Correct answer: a, c, b, d

Example FB:

Fill in the blank with the correct number: Assume the length of female humpback whales can be modeled with a normal distribution with a mean of 13.7 meters and a standard deviation of 0.5 meters. According to the Empirical Rule or 68-95-99.7 Rule, __________ percent of female humpback whales will have a length between 13.2 meters and 14.2 meters.

Correct matches: 68 or 68%

Example JS:

All other things being equal, a __________ confidence interval for a population proportion will be wider than a __________ confidence interval for the same population proportion.

Correct answer: 95%, 90%

Example CA:

The regression line for predicting the value of a variable y from the value of a variable x is as follows:

$y = 1.25 + 0.5x$

Use this equation to predict the value of y when the value of x is 188. Round your final answer to 2 decimal places.

Correct answer: 95.25

2.1.4 Question sets

A question set consists of a set of questions that all have the same format (are all one of the seven possible question types), and all cover the same component of a learning outcome. Question sets allow for different students to be presented with different subsets of similar questions. The number of questions from a given question set that are to be presented to each student is specified, and the questions are selected at random. There is at least one question set per learning outcome, and there is no connection between question sets.

2.2 Report generation outline

A diagram of the report generation process can be seen in Figure 2.1. Blue boxes represent external software (`Respondus` and `Blackboard`) that must be used along with `ePort`. Green boxes represent the three file types that must be provided as input in order for `ePort` to generate reports. Red boxes represent functions within `ePort` that are then used to produce the reports.

Figure 2.1: Overview of report generation. Database questions are constructed in `Respondus`. Students can access and submit their assignments on `Blackboard`. The student responses are saved to the data file, one of the three input files for `ePort`. The other two input files needed for `ePort` are the answer key file and the topic learning outcomes file. The student data file must be parsed/cleaned in `ePort`, and the question titles in the answer key file can then be matched to those in the student data file. Answer texts and graphs alike must be parsed in `ePort` for the report to provide student performance summaries on individual questions. The student topic learning outcome file is also used for analyses on how students performed at the learning outcome level. Student performance summaries using information from these input files is then performed in `ePort`, and collected into reports.

2.2.1 External software - Respondus

External software that must be used for `ePort` to generate the reports are represented in the blue boxes of Figure 2.1. As this figure shows, the database questions are constructed in the software `Respondus`. More information about `Respondus` can be found on their website.

It is important to code question titles in `Respondus` in a format that will allow all database questions to be identified and searched later in the `ePort` assessment model by topic, learning outcome, question type, and question set. That way, summaries of student performance at each of these levels can be performed and published in the reports. An example of a coded question title is shown below:

Example question title: **T16.A.A.04-1.1.MC.1**

T16: Topic 16
A: Learning Outcome A for Topic 16
A: Question Set A for Topic 16
04-1: Question Set A for Topic 16 has 4 questions, and 1 is randomly assigned to each student
1: All questions in Question Set A for Topic 16 are worth 1 point
MC: All questions in Question Set A for Topic 16 are in multiple choice format
1: Question label of this particular question from Question Set A for Topic 16

2.2.2 External software - Blackboard

As seen in Figure 2.1, once the database has been constructed in `Respondus`, we can upload it to `Blackboard`. More information about `Blackboard` can be found on their website.

Typically, assignments on `Blackboard` can be made available to students on the first day the topic is introduced in lecture, and closed a set number of days after the topic is concluded in lecture. Each student can access their assignment over the course of as many sittings as needed, saving their work each time, but they can only submit their assignment once before the deadline. Once the deadline of the assignment has passed, students have access to answers and feedback for their set of questions.

2.2.3 Input files

The three necessary input files for `ePort` to generate reports are represented in green in Figure 2.1. A file that enumerates the learning outcomes is required; an example of this type of file has already been displayed in List 1. An answer key file is also required, and can be generated from the database in `Respondus`. The third required file consists of the answers that students submit when they complete their assignments on `Blackboard`.

2.2.4 Parsing input files

The functions needed to transform input files to final report files are represented in red in Figure 2.1. There are simple cleaning procedures that must be applied to the student response data files from `Blackboard`. One task that must then get done here is matching student answers from the cleaned student response data file to possible answer choices in the answer key from `Respondus`. This step simultaneously requires that the questions that a particular student received at random from each question set are successfully matched to the corresponding question titles in the answer key from `Respondus`.

The final reports consist of data summaries, graphics, and analyses regarding how students performed on the assignments. As will later be explained in more detail, some report types have more verbose versions, which include individual summaries for each question in the topic of interest. This means that it is not only the text, but also any graphs and equations that must be parsed in the answer key file. Doing so allows for the verbose reports to publish all components of a given question, such as relevant graphics, equations, and text, in the same way that it had been presented to students. The analysis of student performance for that question can then also be displayed alongisde the complete question itself.

Chapter 3

Example data

A directory that contains example data is automatically installed with the `ePort` package. The name of this directory is `extdata`. Understanding the location, layout, and content of the `extdata` directory will be necessary to continue with the examples provided in the vignette.

The absolute pathway to the `extdata` directory on your local computer can be determined by typing the following command into the **R** console:

```r
system.file("inst/extdata/", package = "ePort")
```

Chapter 4

Generating Reports

Currently, the `ePort` package offers six report types, depending on what the user is trying to compare and analyze about student performance. The same function `makeReport()` is used to generate each of these six report types; however, the input parameter `reportType` to the function will be different depending on which of the six report types the user wishes to run. It is most efficient for the user to hard-code in the `reportType` parameter. For that reason, below is a reference box for the six types of string options (in quotes) that can be used for the `reportType` parameter, depending on which of the six report types the user will run:

- One topic for one section - short version (“secTopicShort")
- One topic for one section - long version (“secTopicLong")
- One topic comparing multiple sections - short version (“crossSecTopicShort")
- One topic comparing multiple sections - long version (“crossSecTopicLong")
- One unit (group of topics) for one section (“secUnit")
- One unit (group of topics) comparing multiple sections (“crossSecUnit")

This information regarding the parameter options of the six report types can also be obtained by running `help(makeReport)`. In the next parts of the vignette, we will demonstrate how to produce each of the six report types, and, along the way, we will hard code each of the `reportType` parameter options.

4.1 One topic for one section - short version

An instructor may be motivated to generate a short report for one topic and one section if they are seeking answers to the following types of questions:

1. *Overall, how did this section of students do on this homework assignment?*
2. *Which students from this section scored poorly overall on this homework assignment?*
3. *Are there any question sets for this topic that this section of students found easy or difficult?*

4.1.1 Code

We start by demonstrating how to generate the electronic report for one section one topic. This demonstration will use the example input files provided in the previously-described `extdata` directory, and will output the report to the `OutputFiles` subdirectory of the `extdata` directory. If you have not modified anything in the `extdata` directory, then the `OutputFiles` subdirectory should be empty, as we have not generated any example reports yet.

In this demonstration, we will create a report for `Topic 06` and section `A`. Like any individual report, we will require three input files (an answer key file, a data file, and a learning outcome file). There should be three example answer key files in the subdirectory `KeyFiles` (`Topic06.Questions.htm`, `Topic07.Questions.htm`, and (`Topic08.Questions.htm`), and we will use the `Topic06.Questions.htm` file. Additionally, there should be twelve example data files in the subdirectory `DataFiles` (`Topic06.A.csv`, `Topic06.B.csv`, `Topic06.C.csv`, `Topic07.A.csv`, `Topic07.B.csv`, `Topic07.C.csv`, `Topic08.A.csv`, `Topic08.B.csv`, and `Topic08.C.csv`), and we will use the `Topic06.A.csv` file. Lastly, there should be three example learning outcome files in the subdirectory `LOFiles` (`Topic06.Outcomes.txt`, `Topic07.Outcomes.txt`, and `Topic08.Outcomes.txt`), and we will use the `Topic06.Outcomes.txt` file.

The block of code we will use to generate our report for `Topic 06` and section `A` is shown in the code block below. If this is your first time reading through the vignette, it is recommended that you do **not** run this code block all at once just yet. Instead, this code block is designed to provide you with an overview of the procedure; we will discuss each line of the code afterward.

```
keyHTM = system.file("inst/extdata/KeyFiles/Topic06.Questions.htm", package =
  "ePort")
```

```
refineKey(keyHTM)

keyPath = gsub("htm$", "txt", keyHTM)

dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.A.csv", package =
  "ePort")

rewriteData(dataPath)

loPath = system.file("inst/extdata/LOFiles/Topic06.Outcomes.txt", package = "ePort")

outPath = system.file("inst/extdata/OutputFiles", package = "ePort")

makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath)
```

Now that we have seen the entire code required for this procedure, we briefly explain each step of the above code block. Here, we recommend that you follow along by actively running each piece of code, as will be demonstrated below. The first line from the code block is where we save the absolute pathway of the answer key. Here, we save it to a string variable called `keyHTM`.

```
keyHTM = system.file("inst/extdata/KeyFiles/Topic06.Questions.htm", package =
  "ePort")
```

Second, we must parse and clean the .htm answer key file, and convert it to plain text format. We do this by calling the `refineKey()` function of `ePort` on the .htm answer key file. By running the line below, we will create the cleaned .txt file:

```
refineKey(keyHTM)
```

By default, the `refineKey()` function will place the cleaned .txt file into the same directory as the original .html file, and with the same name. Hence, if the you navigate to the `extdata` directory and its `KeyFiles` subdirectory, you should now see the new and cleaned .txt file we just created, `Topic06.Questions.txt`.

Our third step is to save the absolute pathway of this new and cleaned .txt answer key. Below, we save it to a string variable called `keyPath`.

```r
keyPath = gsub("htm$", "txt", keyHTM)
```

Now that we have the absolute path of our cleaned .txt answer key, our fourth step is to define the absolute pathway of our data file. We save this to a variable called `dataPath`.

```r
dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.A.csv", package =
  "ePort")
```

After this, our fifth step is to prime the data file to be compatible with steps later down the pipeline of generating the reports. We do this by running the `rewriteData()` function of `ePort` on the data file. This function parses through the data file, and modifies certain trivial character issues that would otherwise cause a problem when running the reports. For more details about the specific process, please consider running a help command on the function. Below, we rewrite the data file:

```r
rewriteData(dataPath)
```

Upon running the above code snippet, you do not obtain a new file. Instead, the original file is overwritten. For this reason, you might inadvertantly run the function `rewriteData()` on a certain data file more than once, not remembering whether or not you have already converted it. It should not be a problem to run the function any number of times; you would simply receive a message that reads something like: "Note: `Topic06.A.csv` was already successfully converted to usable format."

Our sixth step is to save the absolute pathway of our learning outcome file. Below, we save this path to a variable called `loPath`.

```r
loPath = system.file("inst/extdata/LOFiles/Topic06.Outcomes.txt", package = "ePort")
```

After that, the seventh step is to specify our desired output directory. This is the absolute pathway of where the reports should be saved. Below, we create a variable called `outPath` to specify that we want to output our report to the `OutputFiles` subdirectory.

```
outPath = system.file("inst/extdata/OutputFiles", package = "ePort")
```

Now that we have primed our three input files (cleaned answer key, data file, and learning outcomes file) and specified our output directory, our last step is to generate the report using the `makeReport()` function.

```
makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath)
```

Upon running this code, you will receive a message and menu (as shown below). We received this menu because we did not specify a value for the `reportType` parameter. The menu automatically appears in such circumstances to remind users that, as already mentioned, the `makeReport()` function can produce any one of six reports, and hence we cannot make any report at all until we make that specification.

```
Please enter integer (1-6) corresponding to desired report type below.

Note: If running many reports, it is more efficient to exit now and hard-code the
reportType parameter. See help(makeReport).

1: One topic for one section - short version ("secTopicShort")
2: One topic for one section - long version ("secTopicLong")
3: One topic comparing multiple sections - short version ("crossSecTopicShort")
4: One topic comparing multiple sections - long version ("crossSecTopicLong")
5: One unit (group of topics) for one section ("secUnit")
6: One unit (group of topics) comparing multiple sections ("crossSecUnit")
```

Since we wish to generate the type of report that is the short version of one topic for one section, then we can type the number `1` into the menu console. If we do so, then the report will be generated. However, we can also escape out of the menu, and rerun the last line of code, only this time hard-code specifying the `reportType` parameter as `secTopicShort`. This is shown below:

```
makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath, reportType = "secTopicShort")
```

The advantage to hard-coding the value of the `reportType` parameter is especially pertinent for users who wish to run these reports serially for each of many sections. Otherwise, in this situation, a user would not be able to automate the process, and would need to enter in the same menu selection as each report is generated. This will be made more clear in a later section of the vignette (Running sequentially on batch of sections).

4.1.2 Output

Whether we specify the type of report we wish to generate by hard-coding or selecting from the menu, we should have successfully generated the report. We can find our report in the `OutputFiles` subdirectory of the `extdata` example directory. Indeed, we see that we have our short report (`Stat101hwk_Topic06_A-short.pdf`) for `Topic 06` and section `A`. At this point, it may be helpful for you to open the short report you just generated.

This short report contains tabular and plotting overviews of student performance overall, by learning outcome, and by question set. Next we present a few examples of the types of output that can be found in this report. However, be sure to also view all of the output in the short report you just generated!

A tabular summary of student scores is provided in the report, as seen in Figure 4.1. The report also contains a similar overview of student performance in the form of a histogram, which is not included in this vignette.

Figure 4.1: Summary statistics of student scores. The mean score was 82% with a standard deviation of 15%. Half of the students performed fairly well with a score of at least 88%. However, there were low-scoring students, as the bottom quartile of students scored between 32% and 76% on this assignment.

In addition to understanding how students from this section performed overall for this topic, instructors and course coordinators may wish to determine how students performed across the different learning outcomes. Indeed, the report provides a fluctuation diagram for this purpose, see Figure 4.2.

Figure 4.2: Fluctuation diagram of percentage correct by learning outcome. The area of each square corresponds to the number of students who obtained that score for that learning outcome. Learning outcomes are ordered on the horizontal axis by mean score, with the highest mean score positioned on the left side of the axis. The mean score for each learning outcome is represented with a blue horizontal line. For this section, learning outcome `D` had the highest mean score, and learning outcome `G` had the lowest mean score. For each learning outcome, there remained a sizeable proportion of the 50 students who scored poorly (a score of at most 50%).

Additionally, instructors and course coordinators may wish to obtain an overview of how students from this section performed across the different question sets within the learning outcomes. The report does generate such a graphic, which is shown in Figure 4.3.

Figure 4.3: Summary statistics of the question sets. Rows are sorted by descending mean scores, which are marked red if less than 80%. We see that there were eight question sets for which students of this section had a mean score of less than 80%. Question Set `U` proved particularly difficult for this section of students, with a mean score of 50%.

An instructor may wish to know which students from this section are performing poorly overall on this assignment. To assist this need, the report also includes a list of student names and their corresponding scores, for all cases of students who received a score below 80% on the assignment, as is demonstrated in Figure 4.4.

Figure 4.4: Of the 50 students in this section, 15 of them scored less than 80% on this assignment.

4.1.3 Contacts of low-performing students

Along the same lines, the report automatically generates a .txt file that lists the e-mails of all students who scored below 80% on this assignment. The list is repeated, once in a comma-separated format, and once in a semicolon-separated format. These list formats are intended to provide instructors with a streamlined method to contact students, if needed, to inform them of their low-scoring performance and possibly to recommend seeking additional support and resources to improve their performance in future assignments. You can open the example .txt file we created in this vignette; it should be located in the `OutputFiles` subdirectory of the `extdata` example directory as (`Stat101hwk_Topic06_A_lowScore.txt`).

What an instructor deems an approporiate threshold for a poor performance on a given assignment is arbitrary. For that reason, even though the default value of this threshold is 80, this threshold value can be altered by the user. Discussion and examples of how to alter certain default values in the `ePort` package, including the one just discussed, will occur later in the vignette in the section called Report defaults.

4.2 One topic for one section - long version

An instructor may be motivated to generate a long report for one topic and one section if they are seeking answers to the following types of questions:

1. *Based on how students from this section performed on this topic, are there any noticeable problems where questions in certain question sets not being equally difficult?*
2. *Which students from this section scored poorly on each learning outcome for this homework assignment?*
3. *For each question in the assignment (particularly the difficult ones), what was the distribution of student responses?*

4.2.1 Code

The short report we created for one section one topic (`Stat101hwk_Topic06_A-short.pdf`) may be helpful for users who want to view a brief and overall summarization of student performance. However, we can also generate a more verbose report summarization for one section one topic that would include additional details, such as a separate analysis for each problem in the assignment. We have already defined our three input file pathways and one output file pathway. Hence, if we wish to generate this longer version of the report summarization for one section one topic (sticking with section `A` and `Topic 06`), then all we need to rerun is the `makeReport()` function, only this time specifying the `reportType` parameter with the value of “secTopicLong".

```
makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath, reportType = "secTopicLong")
```

We can find our output report in the `OutputFiles` subdirectory of the `extdata` example directory. Indeed, we see that we now have a much longer report, called `Stat101hwk_Topic06_A-long.pdf`.

4.2.2 Output

The short report featured tools that identified individual students from the section who performed poorly overall. The long report provides a more detailed output that informs instructors on how each individual student did not only overall, but also for each learning outcome. This is shown below in Figure 4.5.

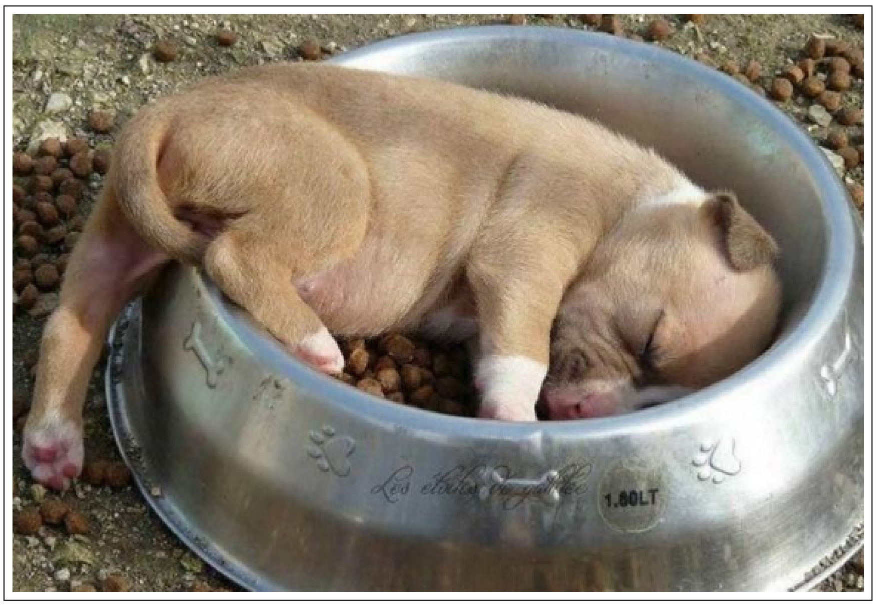

Figure 4.5: Heat map of the student ranks. Blue represents the top ranks, while yellow represents the bottom ranks. The order of students on the vertical axis is by overall rank across all learning outcomes. We see that the five students with the highest overall scores (ales, clarinep, eloish, eveline, and wallazu) performed well across all learning outcomes. Moreover, we see that the student with the lowest overall score (proehld) performed poorly for all learning outcomes, except for learning outcome `D`.

Oftentimes, one of the aims in an online homework assignment is to ensure that any two students in the course are not likely to receive the exact same set of questions on any given assignment, thereby encouraging students to complete their own work. However, it is important to ensure that all possible combinations of question subsets that a student could receive are equally difficult. Hence, it is necessary to confirm that all questions that can be randomly selected from each given question set are equally difficult. The long report does indicate this information to users, as is seen below in Figure 4.6. Please note that this image is abridged for demonstration purposes; if you were to view the same image in the long report you just generated, then you would see that indeed it is a comprehensive table that consecutely lists the pertinent information for all 89 questions in this assignment.

Figure 4.6: Comparison of student performance in this section on each of the 89 possible questions of `Topic 06`, grouped by question set. If any pair of questions in a given question set have a difference in mean scores of more than 15%, then both questions are flagged. Question set `A` contained four questions, with mean scores that ranged from 81.82% to 100.00%. Three of these questions (IDs: 1, 3, and 4) were flagged as having a mean score that was at least 15% different than the mean score of at least one other question in this same question set. In contrast, question set `L` contained nine questions, with mean scores that ranged from 88.89% to 100.00%. Hence, none of these nine questions were flagged as demonstrating a concerningly large difference of at least 15% in difficulty level from within the same question set.

As per the figure above, an instructor can quickly determine if there are questions for which students tend to show poor performance, and if there are question sets for which students tend to show disparate performance between their comprising questions. In either situation, an instructor may wish to examine these questions more closely, and determine how students are answering these questions incorrectly. For this reason, the long report also includes a summary of student responses for each individual question in the assignment, allowing instructors to determine the most common mistakes students in this section are making for these difficult questions.

If you open the long report we generated (`Stat101hwk_Topic06_A-long.pdf`), then you can view the summary for each of the 89 questions in this assignment. Each question has its summary printed on a separate page, beginning on Page 13 and ending on Page 101 of the report. As we saw in Figure 4.6, the first Question ID of the assignment had a mean score of 81.82%. We can ascertain what types of errors students from this section made on this Question ID by viewing its summary (on Page 13 of the report), which is provided below in Figure 4.7.

Figure 4.7: A detailed summary of student performance for the first Question ID in the assignment. We see that 11 of the 50 students from this section received this question at random out of a question set that contains four similar questions. All 11 students responded with an answer: Nine correctly selected answer choice `A`, and two incorrectly selected answer choice `C`. No student incorrectly responded with answer choice `B` or `D`.

4.3 One topic comparing multiple sections - short version

An instructor may be motivated to generate a short report for one topic across multiple sections if they are seeking answers to the following types of questions:

1. *For this topic, is there consistency across the sections in terms of overall performance?*
2. *Combining students from all sections, are there any disparities in student performance between questions from the same question set of this topic?*
3. *Combining students from all sections, are there any problematic questions in this topic, in which students typically perform poorly?*

4.3.1 Code

Now that we know how to successfully generated reports for the three sections (`A`, `B`, and `C`) of `Topic 06` separately, we may wish to generate a report that compares the performance between these three sections. In this case, we use the same `makeReport()` function, just as we did before. However, for our parameter `dataFile`, we will input a list `dataList` that contains the paths to the data files for both sections we wish to compare. Of course, we must also correctly specify the `reportType` parameter to indicate that we wish to generate a short report that compares sections for a given topic:

```r
dataFolder = system.file("inst/extdata/DataFiles/Topic06", package = "ePort")
dataList = list.files(path = dataFolder, full.names = TRUE)[1:3]
makeReport(keyFile = keyPath, dataFile = dataList, loFile = loPath, outFile =
  outPath, reportType = "crossSecTopicShort")
```

Please ensure that the variable `dataList` contains exactly the full pathway to the three data files (`Topic06.A.csv`, `Topic06.B.csv`, and `Topic06.C.csv`). If your variable contains additional pathways, then you have more than just the three necessary input data files in your `DataFiles/Topic06` folder. If that is the case, then you will need to move these extraneous files elsewhere or hardcode the three full pathways into the `dataList` variable.

4.3.2 Output

You should be able to examine the report we just generated (`Stat101hwk_Topic06_crossSectionshort.pdf`) in our example `OutputFiles` folder. A course coordinator may wish to know if student performance for learning outcomes is similar across sections. In the report we just produced, we can indeed quickly obtain this information, as is shown below in Figure 4.8.

Figure 4.8: Average percentages correct for each learning outcome by section. Learning outcomes are sorted by the average percentages correct across all sections, from the highest to the lowest. Scores in red represent cases where a section had an average score for a learning outcome that was less than 80%. We see that the sections showed consistency in that they all did well for learning outcomes `D`, `E`, and `B`, and all did poorly for learning outcomes `F`, `C`, and `G`.

The report also provides tables that, after combining student responses from all sections, displays questions that have a low overall performance score. It additionally includes tables that highlight question sets that seem to have disparate performance scores between the questions from which they are composed.

4.4 One topic comparing multiple sections - long version

An instructor may be motivated to generate a long report for one topic across multiple sections if they are seeking answers to the following types of questions:

1. *For this topic, is there consistency across the sections in terms of overall performance, and in terms of performance by learning outcome and question set?*

2. *Combining students from all sections, for this topic, are there any disparities in student performance between questions from the same question set?*

3. *Combining students from all sections, for this topic, are there any learning outcomes or question sets for which students consistently tend to perform poorly?*

4. *Combining students from all sections, for this topic, for each question in the assignment (particularly the difficult ones), what was the distribution of student responses?*

4.4.1 Code

We received a brief comparative summary between the three sections for `Topic 06` in the previous section of this vignette. If we would like to see a more detailed version of this report that includes comparative information for each question in the topic of interest, then we can simply run the previous line of code, only now specifying the `reportType` parameter to indicate that we wish to generate a long report that compares sections for a given topic:

```
makeReport(keyFile = keyPath, dataFile = dataList, loFile = loPath, outFile =
  outPath, reportType = "crossSecTopicLong")
```

4.4.2 Output

The code above should have generated the long edition of the report (`Stat101hwk_Topic06_crossSectionlong.pdf`). One of the most immediate pieces of information that a course coordinator may wish to glean is whether or not there are discrepancies in student performance for a given topic across multiple sections. This could be especially true if there are different instructors and/or different teaching methods being employed across the different sections. In the report we just produced, we can indeed quickly compare how the students performed overall for `Topic 06` across the three sections in a table that has been copied here in Figure 4.9.

Figure 4.9: The mean, standard deviation, and five number summary for student performance across the three sections. We see that the overall student performance was relatively similar across the three sections, although sections `A` and `C` had overall mean scores that were at least 1.5 points larger than the overall mean score of section `B`.

While the short report allowed us to compare how students performed in `Topic 06` across the sections, the long report will also allow us to make comparisons of student performance across the sections at a more detailed level of learning outcomes and question sets. For instance, the long report includes comparative information at the level of learning outcomes, as is illustrated in Figure 4.10.

Figure 4.10: The three sections were fairly comparable in their performances across the learning outcomes. In general, learning outcomes (such as `B`) that showed little variance between students in a given section also showed little variance between students in the other sections, while some learning outcomes (such as `G`) that showed large variance between students in a given section also showed large variance between students in the other sections. Notably, for learning outcome `C`, all students in section `B` performed less well than all students from sections `A` and `C`; likewise, for learning outcome `B`, all students in section `C` outperformed all students from sections `A` and `B`.

The long edition of the report breaks it down further by comparing the three sections at the level of each question set. This is demonstrated below in Figure 4.11.

Figure 4.11: Mean percent scores for each question set by section. The question sets are sorted by decreasing overall mean scores. Both question sets with the highest overall mean scores across sections are from learning outcome `D`. We see that sections performed fairly consistently across question sets. However, question set `H` had a mean score difference of 24% between sections `A` and `B`.

The long edition of the report also offers tools that do not explicitly compare the sections of interest, but instead, combine data from all the sections of interest to generate more power in any analysis that we could instead do separately using just the data from one section. For instance, we previously demonstrated certain tools in the individual section reports that can allow instructors to confirm that their students will receive equally-challenging randomly-selected subsets of questions; this was shown in Figure 4.6.

A course coordinator could again explore this same potential predicament, only now using the data combined from students across both sections, as is displayed in Figure 4.12. Please note that this image is abridged for demonstration purposes. If you were to view the same image in the long report you just generated, then you would see that indeed it is a comprehensive table that consecutely lists the pertinent information for all 89 possible questions in this assignment.

Figure 4.12: Combined student performance from sections `A`, `B`, and `C` on each question of `Topic 06`, grouped by question set. If any pair of questions in a given question set have a difference in mean scores of more than 15%, then both questions are flagged. Question set `A` contained four problems, and when we only used student performance information from the 50 students in section `A`, three IDs (1, 3, and 4) were flagged as having a mean score that was at least 15% different than the mean score of another question in the same question set (see Figure 4.6). However, now that we have combined student performance information from all 150 students across the three sections, only two IDs (1 and 3) remain flagged. In contrast, question set `I` contained nine problems, and none of these problems were flagged first when we only used the 50 student performances in section `A` (see Figure 4.6) and now when we use all 150 student performances across the three sections.

As per the figure above, an instructor can quickly determine if there are questions for which students tend to perform poorly, and if there question sets for which students tend to show inconsistent performance between the individual questions. In either situation, an instructor may wish to examine these questions more closely, and determine how students are answering these problems incorrectly. For this reason, the long report also includes a summary of student responses for each individual problem in the assignment, allowing instructors to determine the most common mistakes students across the sections are making for these difficult questions.

If you open the long report we generated (`Stat101hwk_Topic06_crossSection-long.pdf`), then you can view the summary for each of the 89 possible questions in this assignment. Each question summary is printed on a separate page, beginning on Page 11 and ending on Page 100 of the report. As we saw in Figure 4.12, Question ID 1 had an overall mean score of 78.26% across the three sections. We can ascertain what types of errors students made on this problem, regardless of section, by viewing the summary for Question ID 1 (on Page 11 of the report), which is provided below in Figure 4.13. Of course, now that we are collecting student responses from the 46 students who received this problem in any of the three sections, we are provided with more power about the distribution of student responses than when only considered the 11 students in section `A`, as was done in Figure 4.7.

Figure 4.13: A detailed summary of student performance for the first Question ID in the assignment. Of the 46 students who received this Question ID, 36 students correctly selected answer choice `A`. The most frequently incorrect response from students was answer choice `C`, followed by answer choice `B`. No students incorrectly answered choice `D`. One student did not respond. In general, this is a similar distribution to what we saw in Figure 4.7, when we only examined the 11 students from section `A` who received this Question ID.

4.5 One unit (group of topics) for one section

An instructor may be motivated to generate a long report for one unit (group of topics) and one section if they are seeking answers to the following types of questions:

1. *Did the students from this section overall perform comparably between the topics of interest?*

2. *Are there are any patterns in how individual students from this section performed between the topics of interest?*

3. *How often are the students from this section missing assignments? Are there any topics for which students from this section in particular miss assignments?*

4.5.1 Code

To generate this type of report, we must be sure to provide a `dataFolder` variable that contains the pathway to a folder that contains our data files of interest. As usual, it is important to ensure that there are no extraneous files in our folder. Since we want to examine three sections (`A`, `B`, and `C`) and three topics (`06`, `07`, and `08`), then we need a folder that contains nothing other than the data files for the nine possible combinations of these sections and topics.

Indeed, the `ePort` package contains such an example folder, which he have not yet used in this vignette. It is located in `extdata/DataFiles/Topic06_07_08`. You can examine this folder to confirm that there is nothing but our nine necessary input files (`Topic06.A.csv`, `Topic06.B.csv`, `Topic06.C.csv`, `Topic07.A.csv`, `Topic07.B.csv`, `Topic07.C.csv`, `Topic08.A.csv`, `Topic08.B.csv`, and `Topic08.C.csv`). We save the pathway to all nine of these files to a variable called `dataList`.

For this type of report, we must also prepare a few variables before running the `makeReport()` function. We prepare these variables using the `setDir()`, `mergeData()`, and `subsetData()` functions. For more information about these functions, please use the `help()` command. The `dataTable` variable we create below is a simple data frame that specifies the topic and section for each input data file of interest. The `mergedData` variable we also create below is a list of data frames that contains the scores that each student from the three sections earned on each topic assignment, the number of missed assignments, and the total scores across all students for each section and topic.

Our goal here is to create a separate report for each section of interest, where each report compares how students from that section performed across topics. We can most efficiently accomplish this by using a for loop, as shown below, where we create one last variable `merged`. This variable is the subset of our `mergedData` variable that only applies for the section of interest.

Lastly, we can run our `makeReport()` function on this section and group of topics. We should specify our unit variable, which contains a default value of 1. In our example, we are indicating that we consider topics 06, 07, and 08 to be from Unit 2.

```
dataFolder = system.file("inst/extdata/DataFiles/Topic06_07_08", package="ePort")
dataList = list.files(path = dataFolder, full.names = TRUE)

for (file in dataList){
  rewriteData(file)
}

dataTable = setDir(dataFolder)
mergedData = mergeData(dataTable)

for (section in unique(dataTable$section)){
  merged = subsetData(mergedData, dataTable, choice = section)
  makeReport(outFile = outPath, unit = 2, section = section, reportType = "secUnit")
}
```

Upon running this code, you should have created three files into your OutputFiles folder, Stat101hwk_Unit2_A.pdf, Stat101hwk_Unit2_B.pdf, and Stat101hwk_Unit2_C.pdf. Each of these files compares how students in that section performed for the three topics that we defined as comprising Unit 2.

4.5.2 Output

An instructor might want to know how their section of students is performing throughout the course duration, and in partiular, if there are any topics within a unit of interest for which their students perform well or poorly. We will examine one of the three reports we just generated (Stat101hwk_Unit2_A.pdf) to discover the types of tools that instructors could use in such situations. The report begins with a quick summary of how students performed for this section (A) across all topics in Unit 2 (topics 06, 07, and 08), as is provided in Figure 4.14.

Figure 4.14: Overall summary of how students in section `A` performed for topics `06`, `07`, and `08`. Students showed the highest mean and median for `Topic 07`, and the lowest mean and median for `Topic 08`.

We can also compare how this section of students performed between these three topics, not only at an overall level, but also at the individual student level, see Figure 4.15.

Figure 4.15: Boxplots with parallel coordinate plot overlaid. Each boxplot represents the distribution of all student scores for a given topic, and each parallel coordinate line represents the scores across all topics for a given student. We see that how a given student performed relative to the rest of the students was, in most cases, consistent across the three topics.

While the figure above can allow us to see if there are trends in how individual students scored between the topics, if we find patterns of interest, we may wish to determine the identity of individual students. This is provided below in Figure 4.16.

Figure 4.16: Heat map of the student scores in this section for the three topics of interest. The students are listed in descending order of their overall average score across the three sections. We can determine that the lowest-performing student (congerf) performed much worse as the topics progressed. In contrast, the highest-performing student (bessbr) performed slightly better as the topics progressed.

The report also produces tables that provide information about missed assignments for this section of students. An instructor could use this tool to determine if there are certain topics for which students from the section of interest tend to fail to submit the assignment, or if there are certain students from this section who tend to fail to submit assignments regardless of topic.

4.6 One unit (group of topics) comparing multiple sections

An instructor may be motivated to generate a report for multiple topics and multiple sections if they are seeking answers to the following types of questions:

1. *Are there certain sections that either perform better or worse consistently across topics?*
2. *What are the easiest and most difficult topics in the course, regardless of section?*
3. *Are there any patterns in missed assignments by section or by topic?*

4.6.1 Code

To accomplish this task, we create a `merged` variable that contains a list of data frames that include the raw scores, percentage scores, number of missing assignments, and average scores for the three sections and three topics of interest. After that, we can simply use our `makeReport()` command, being sure to specify that we are designating the three topics (`06`, `07`, and `08`) as `Unit 2`.

```
merged = subsetData(mergedData, dataTable)
makeReport(outFile = outPath, unit = 2, reportType = "crossSecUnit")
```

This creates four files, `Stat101hwk_Unit2_crossSection.pdf`, `Stat101hwk_Unit2_A_lowScore.txt`, `Stat101hwk_Unit2_B_lowScore.txt`, and `Stat101hwk_Unit2_C_lowScore.txt`. The first file contains our report that compares all combinations of the three sections and three topics of interest. The latter three .txt files contain lists of the e-mail contacts of students from each respective section who performed poorly consistently across the three topics of interest.

4.6.2 Output

From the report we just generated (`Stat101hwk_Unit2_crossSection.pdf`), we can simultaneously examine how students performed across different sections and different topics. As is the case with previous reports, we are provided some basic summaries, as provided in Figure 4.17.

Figure 4.17: Mean scores for all sections and topics of interest. Section `A` had the highest mean score for all three topics.

In this report, we can also compare score distributions for each combination of section and topic (see Figure 4.18).

Figure 4.18: Histogram of student scores for the sections and topics of interest. We see that, for all sections, the score distribution for `Topic 08` appeared more uniform compared to the score distributions for topics `06` and `07`.

One other feature of this report is a tabulation of missing homework for sections and topics of interest (see Figure 4.19). If there are patterns found for this, course coordinators may wish to determine what is causing students to tend to miss homework assignments for a certain section or certain topic.

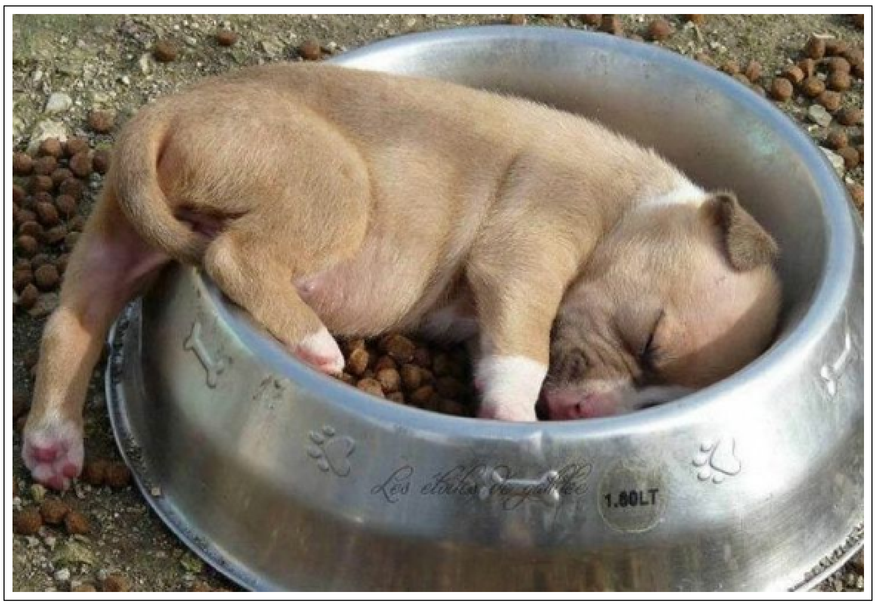

Figure 4.19: Missing homework summary for the three sections and three topics of interest. In each of the three sections, all fifty students submitted homework assignments for all three topics of interest.

4.6.3 Contacts of low-performing students

This report type automatically generates a .txt file for each section that lists the e-mail contacts of all students from that section who scored below a threshold of 20% on at least half of the topic assignments. The format of these .txt files is similar to the one that we already generated for the short version of the report on one section and one topic. We can alter this threshold, as will be described in a later section of the vignette (Report defaults).

You can take a moment to examine these three files (`Stat101hwk_Unit2_A_lowScore.txt`, `Stat101hwk_Unit2_B_lowScore.txt`, and `Stat101hwk_Unit2_C_lowScore.txt`). If you do so, you will notice that Section `A` contained three low-performers, Section `B` contained two low-performers, and Section `C` contained six low-performers.

Chapter 5

Additional tools

5.1 Splitting files

In some cases, one homework assignment may span across two topics. In order to generate separate reports for each of the two topics, we will first need to split the data file. In our example, we will split the `Topic 06` data files for all three sections. We have a special folder in our example data to complete this process; if you examine the folder called `Topic06_Split`, which is located in our example `DataFiles` folder, then you will see that it contains the three files, `Topic06.A.csv`, `Topic06.B.csv`, and `Topic06.C.csv`.

If you open any of these three files, then you will notice there is a “Question ID" column and a “Question" column (which contains Question RsQ). The Question ID refers to the question order that each student completed; in this example assignment, there are 25 Question IDs. Hence, each student received 25 questions, and should have Question IDs for each of these questions consecutively increasing from Question ID 1 to Question ID 25. In contrast, the Question RsQ refers to the absolute ID from Respondus. In this assignment, in total, there were 89 questions in `Respondus`, from which each student received their randomly-selected subset of 25 questions Hence, the Question RsQ should range from 1 to 89.

Because of this, if an instructor wants to split the data files, there are two parameters they can use to designate the cut location. If you run a help() function on the `ePort` function `splitFile()`, then you see that these cut types are called “RsQ" and “ID". Let us say that we need to split these data files into one file that only contains the first 9 ID Questions and another file that only contains the last 16 ID Questions. In that case, we can choose a cut type of “ID" with a value of 9, which corresponds to a cut type of “RsQ" with a value of 22. We show this process below using the cut type of “ID".

```
dataFolder = system.file("inst/extdata/DataFiles/Topic06_Split", package = "ePort")
dataList = list.files(path = dataFolder, full.names = TRUE)[1:3]

for(file in dataList){
  rewriteData(file)
  splitFile(file, 9, "ID")
}
```

Upon running the above code, if you return to the `Topic06_Split` folder, then you will see that six new files have been created, `Topic06.A.part1.csv`, `Topic06.A.part2.csv`, `Topic06.B.part1.csv`, `Topic06.B.part2.csv`, `Topic06.C.part1.csv`, and `Topic06.C.part2.csv`. You can verify that the `part1` files contain the first 9 Question IDs, while the `part2` files contain the last 16 Question IDs.

5.2 Merging files

In some cases, one topic might encompass two homework assignments. In order to generate a report for the overall topic, we will first need to combine the two data files. Such a scenario exists in the online database, in which chapter 9 and 12 homework assignments together form `Topic 11`.

In our example, we will combine the Chapter 9 and 12 homework assignments to form the `Topic 11` data file for sections `A`, `B`, and `C`. We again have a special folder in our example data to complete this process. It is located in our example `DataFiles` folder in a subfolder called `Topic11_Merge`. Upon entering this folder, you should see that it contains six files, `Ch12.A.csv`, `Ch12.B.csv`, `Ch12.C.csv`, `Ch9.A.csv`, `Ch9.B.csv`, and `Ch9.C.csv`. Notice in the code below that we save the topic number to a string variable called `topicNum`.

```
dataFolder = system.file("inst/extdata/DataFiles/Topic11_Merge", package="ePort")
dataList = list.files(path = dataFolder, full.names=TRUE)[1:4]
topicNum = "Topic11"

for (i in dataList) rewriteData(i)

for(i in c('AB', 'CD')){
  tmp = dataList[grep(i, basename(dataList))]
  combineFiles(tmp[2], tmp[1], paste(dirname(tmp[1]), "/", paste(topicNum, i, "csv",
    sep = '.'), sep = ""))
}
```

Running this code should produce two new files (`Topic11.A.csv` and `Topic11.B.csv`) into the `Topic11_Merge` folder. If we explore these files, we see that they have a maximum Question ID value of 15 and a maximum RsQ value of 32. This can verify that this is correct if we explore the original files for the Chapter 9 and 12 homework assignments. The Chapter 9 homework data file had a maximum Question ID value of 10 and a maximum RsQ value of 23, while the Chaper 12 homework data file had a maximum Question ID value of 5, and a maximum RsQ value of 9. Hence, the total number of Question ID and RsQ values in the combined files appear to have been added correctly given the total number of Question ID and RsQ values from the individual files.

5.3 Deidentifying files

It should be noted that data files and the corresponding reports they generate contain student names. At times, it may be necessary to anonymize student names in a given data file, so that it (and any reports that can be generated from it) do not contain confidential information. You can most easily deidentify student names by transferring all data files that you wish to deidentify into a common folder that is otherwise empty. It is important to ensure that no extraneous files are in this common folder.

In this vignette, such an example folder has already been set up for you. The data files for sections `A`, `B`, and `C` of `Topic 06` are located in the `DataFiles/Topic06_Deidentified` folder of the `extdata` folder. In Table 5.1 below, we show lines 20-30 of the file `Topic06.A.csv`. Note that the first three columns (“Username", “Last Name", and “First Name") contains the identifying names of students. Of course, these are not real student names, but instead are placeholder student names generated for example purposes.

Table 5.1: Data files before deidentification

Username	**Last Name**	**First Name**	**Question ID**	**Answer**
adelmanc	Adelman	Candi	Question ID 20	1.07%
adelmanc	Adelman	Candi	Question ID 21	62.47%
adelmanc	Adelman	Candi	Question ID 22	738.3 kg
adelmanc	Adelman	Candi	Question ID 23	765.6 kg
adelmanc	Adelman	Candi	Question ID 24	351.2 days
adelmanc	Adelman	Candi	Question ID 25	50.39 grams
ales	Sester	Alejandra	Question ID 1	0.4 standard deviations above mean
ales	Sester	Alejandra	Question ID 2	0.8 standard deviations below mean
ales	Sester	Alejandra	Question ID 3	1.71
ales	Sester	Alejandra	Question ID 4	-2.57

As is demonstrated below, we first save the pathway to the directory that contains nothing but the files we wish to deidentify. Then, we call the `getNameList()` function of `ePort`. We choose the `save = TRUE` option so that a dictionary file `nameCode.csv` will be generated. This dictionary file contains a list of the original student names and their corresponding deidentified codes. Lastly, we call the `encodeName()` function, which will use the dictionary file we just created to translate the original data files into deidentified data files.

```r
set.seed(1)
dataFolder = system.file("inst/extdata/DataFiles/Topic06_Deidentified", package =
  "ePort")
getNameList(dataFolder, section = NULL, semester = NULL, secblind = TRUE, save =
  TRUE)
encodeName(dataFolder, dict = paste(dataFolder, "nameCode.csv", sep = '/'))
```

If you complete this process, you will notice that the original data files have been overwritten to have anonymized student names. Specifically, if we reexamine the `Topic06.A.csv` file, we see that the column called “Username" now contains deidentified names of students (in the form of something like “ID0034"), and the columns called “First Name" and “Last Name" now contains values of `NA` (see Table 5.2).

Table 5.2: Data files after deidentification

Username	**Last Name**	**First Name**	**Question ID**	**Answer**
ID0040	NA	NA	Question ID 20	1.07%
ID0040	NA	NA	Question ID 21	62.47%
ID0040	NA	NA	Question ID 22	738.3 kg
ID0040	NA	NA	Question ID 23	765.6 kg
ID0040	NA	NA	Question ID 24	351.2 days
ID0040	NA	NA	Question ID 25	50.39 grams
ID0056	NA	NA	Question ID 1	0.4 standard deviations above mean
ID0056	NA	NA	Question ID 2	0.8 standard deviations below mean
ID0056	NA	NA	Question ID 3	1.71
ID0056	NA	NA	Question ID 4	-2.57

We can also examine the `nameCode.csv` dictionary file that was also generated and saved to the directory of interest. This file contains the codes for how the identifying names of the 150 students across the three sections in the directory of interest were deidentified (see Table 5.3).

Table 5.3: Name codes

Username	**Code**
adelmanc	ID0040
ales	ID0056
alinev	ID0085
alisond	ID0134
armadav	ID0030
arnetta	ID0131
augera	ID0137
bardena	ID0095
bessbr	ID0090
bhartz	ID0009

Chapter 6

Running reports sequentially

If we wish to create an individual report for a given topic for each of many sections, then one inconvenient way to accomplish this would be to run the code separately for each section at a time. Since the reports we wish to generate are all from the same topic, then we would be using the same key and learning outcome files. We can demonstrate this approach by building off similar code to what we used at the beginning of the vignette when we covered short version of the report on one section and one topic, where we already defined the key and learning outcome files. Our demonstration of this inefficient method below will use three data files. As we have only ran the `rewriteData()` function on the data file for section `A`, we will now have to run it on the data files for sections `B` and `C`.

```
dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.A.csv", package =
  "ePort")
makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath, reportType = "secTopicShort")

dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.B.csv", package =
  "ePort")
rewriteData(dataPath)
makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath, reportType = "secTopicShort")

dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.C.csv", package =
  "ePort")
rewriteData(dataPath)
makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath, reportType = "secTopicShort")
```

This successfully creates the output reports and text files of low performers for each of our three sections (`Stat101hwk_Topic06_A-short.pdf`, `Stat101hwk_Topic06_B--short.pdf`, `Stat101hwk_Topic06_C-short.pdf`, `Stat101hwk_Topic06_A_lowScore.txt`, `Stat101hwk_Topic06_B_lowScore.txt`, and `Stat101hwk_Topic06_C_lowScore.txt`). However, this method could become burdensome if we were to run the report for many sections. A more convenient way to accomplish the task would be to run all the reports at once. To do this, we need to save a variable `dataListPath` that contains the data files for all sections for this topic of interest.

We can do this one of two ways. We can hard code the two data files needed for the two sections into a vector called `dataListPath` as shown below:

```
dataListPath = c(system.file("inst/extdata/DataFiles/Topic06/Topic06.A.csv",
  package = "ePort"), system.file("inst/extdata/DataFiles/Topic06/Topic06.B.csv",
  package = "ePort"), system.file("inst/extdata/DataFiles/Topic06/Topic06.C.csv",
  package = "ePort"))
```

Or, we can create the same variable using the `list.files()` function of base **R**:

```
dataFolder = system.file("inst/extdata/DataFiles/Topic06", package="ePort")
dataList = list.files(path = dataFolder, full.names=TRUE)[1:3]
```

Next, as demonstrated below, we can generate a report for all three sections listed in our `dataListPath` object using a for loop. Our three data files of interest in the for loop (`Topic06.A.csv`, `Topic06.B.csv`, and `Topic06.C.csv`) are from the same topic, and so they share the same key file and learning outcome file. Hence, we do not have to run the `refineKey()` function on the key file for this topic, as we have already completed this step earlier. However, our three data files of interest do not share the same data file. We have already executed the `rewriteData()` function for the `Topic06.A.csv` data file, but we have not yet done so for the `Topic06.B.csv` and `Topic06.C.csv` files. Hence, we must include the `rewriteData()` function in our for loop to ensure that all data files have this priming step completed (it is not a problem to run the `rewriteData()` function on a file for which it has already been executed).

```
for (i in dataListPath){
  rewriteData(i)
  makeReport(keyFile = keyPath, dataFile = i, loFile = loPath, outFile = outPath,
    reportType = "secTopicShort")
}
```

Chapter 7

Report Defaults

If you have run a help() function on the `ePort` function `makeReport()`, you may have noticed that there are several default values that we have not discussed yet in this vignette. In this section, we will explain six of these default values.

7.1 Parameter options

The six default values we will discuss in this section are as follows:

- **keepFiles**: keep files (.aux, .log, etc) used to generate .tex file, default value is `FALSE`
- **keepTex**: keep .tex file, default value is `FALSE`
- **keepImage**: keep individual image files used to generate .tex file, default value is `FALSE`
- **className**: the name of the class, default value is "Stat101"
- **lowScore**: the value that instructors designate as a minimally acceptable score (out of 100) for a given topic and section, when running the makeReport() routine with a reportType of "secTopicShort". Any student who scores below this value on their assignment will have their e-mail listed in a text file. Default value is 80
- **repeatLowScore**: the proportion that instructors use to determine students who have performed poorly across multiple assignments. If a student has been ranked below this proportion (out of the other students) for at least half of the assignments, then their e-mail will be listed in a text file. This parameter can be defined when running the makeReport() routine with a reportType of "crossSecUnit". Default value is 20

The first three boolean parameters (`keepFiles`, `keepTex`, and `keepImage`) are intended to keep the output directory clutter-free for users. When creating reports with the `ePort` package, the number of files in the output directory could grow exponentially, especially if they are working with multiple topics and multiple sections. For this reason, the default value of the paramters are all set to `FALSE` in order to save the least number of files and images to the output directory. As we have seen in the vignette so far, the only files we generate in our output directory, at default, are the reports and the low-performing text files.

However, these boolean default values can be changed in cases where a user may wish to choose a verbose option. If the user specifies `keepFiles = TRUE` in the `makeReport()` function, then they will output all the files that are generated when running `knitr` on the LaTeX files that produce the reports. This could be useful if, for instance, the corresponding log files can be used to troubleshoot a technical problem. If the user defines `keepTex = TRUE` in the `makeReport()` function, then they will also output not only the final report in its .pdf format, but also the associated .tex file. This could also be used to troubleshoot any technical problems or tailor the reports at this intermediary step. Finally, if the user defines `keepImage = TRUE` in the `makeReport()` function, then they will output each of the figures in the reports into its own .pdf file. This could allow users to save that particular plot.

The fourth parameter (`className`) holds a string value that describes the name of the course for which the user is generating reports. Since the names of the report files and low-performance text files contain the name of the course, then these names will be altered. Additionally, any reference to the class name in the reports will also be changed.

The last two parameters (`lowScore` and `repeatLowScore`) hold integer values that describe the cut-off values used to produce the list of contacts of low-performing students for the "secTopicShort" and "crossSecUnit" types of reports.

7.2 Examples

To best illustrate the default parameters, we will begin by clearing all files in our `OutputFiles` directory.

```
unlink(paste0(system.file("inst/extdata/OutputFiles", package = "ePort"), "/*"))
```

Next, we will rerun the short version of the report on one section and one topic. Here, we are specifying three boolean default parameters to values of `TRUE`, the `className` default parameter to a value of "Math123", and the `lowScore` default parameter to a value of 60.

```r
dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.A.csv", package =
  "ePort")
makeReport(keyFile = keyPath, dataFile = dataPath, loFile = loPath, outFile =
  outPath, reportType = "secTopicShort", keepFiles = TRUE, keepTex = TRUE, keepImage
  = TRUE, className = "Math123", lowScore = 60)
```

If you examine the `OutputFiles` directory, you should see that we successfully generated the report `Math123hwk_Topic06_A-short.pdf`. The name of the report file itself, and the title (and any other mention of the class name within the report) have been changed. Since we set the two boolean parameters `keepText` and `keepFiles` to `TRUE`, then we should also find the corresponding .tex, .aux, .out, and .log files in our output directory. Additionally, since we set the boolean paramater `keepImage` to `TRUE`, then we have a separate copy of all images that were printed in the report, each saved separately as .pdf files. In this case, there were two images in the report, which were saved as `LearningObj_summary-1.pdf` and `histScore-1.pdf`.

We also set the parameter `lowScore` to a value of `60`. Because of this, the number of students who received scores that were deemed “poor performance" was reduced to 5 students. We can see this both from the table in the report that lists low-performing students, and in the .txt file that contains the contacts for these students, `Math123hwk_Topic06_A_lowScore.txt`. Note that previously, when we did not changed the default value of `80` as our threshold for poor performers, we had 15 students who were deemed to have performed poorly (see Figure 4.4).

To prepare for another example, we will again clear all files in our `OutputFiles` directory.

```r
unlink(paste0(system.file("inst/extdata/OutputFiles", package = "ePort"), "/*"))
```

Now, we will run the report for one unit (group of topics) comparing multiple sections , only this time altering the default parameters. We specify the `className` parameter to a value of “Eng444", and the `repeatLowScore` parameter to a value of `70`.

```r
merged = subsetData(mergedData, dataTable)
makeReport(outFile = outPath, unit = 2, reportType = "crossSecUnit", className =
  "Eng444", repeatLowScore = 70)
```

This creates four files, `Eng444hwk_Unit2_crossSection.pdf`, `Eng444hwk_Unit2_A_lowScore.txt`, `Eng444hwk_Unit2_B_lowScore.txt`, and

`Eng444hwk_Unit2_C_lowScore.txt`. Both in our report summaries and in our .txt files of e-mail contacts of low-performing students, we see that there are now 36 students from section `A` who performed poorly, 32 students from section `B` who performed poorly, and 33 students from section `C` who performed poorly. As expected, the number of students who were deemed to have performed poorly has increased from when we ran this same code using the default value of 20 (see Contacts of low-performing students).

Chapter 8

Future avenues

There are several improvements that can be made in future versions of the `ePort` package. This could include plotting and layout updates in the reports, as well as improvements in the user interface in terms of flexibility and efficiency in generating reports. Any user feedback would be taken seriously for these types of improvements, as they may be underlooked in the first version.

There are also aims to expand upon not just the descriptive statistics of student performance, but also in modelling this performance. The beginning of these goals have already been actualized in the unit-level reports. There are generalized linear mixed models that can be applied using assignment score as the response variable and section and topic as the predictor variables. We can also look at mixed effect models, using topic and section as fixed effects and having their interaction included as a term in the model. The random effect in this case would be the student.

Finally, clustering analysis can also be used on the data of student performance: There could be interesting and informative patterns regarding how students acquire the course knowledge throughout the duration of the semester. Some students could be naturally strong in statistics combined with being hard working throughout the semester. Other students could start the semester full of energy only to gradually lose steam. Still other students could underestimate an introductory course, do poorly initially, and then need to take the course more seriously once it becomes evident that the course requires more effort. It would be an interesting approach then to employ clustering methods to discover such behaviors along the semester.

Chapter 9

Conclusions

The ePort package offers various tools that instructors can use to determine for which topics and - more specifically - learning outcomes students tend to excel and struggle. Coordinators can also use the reports to monitor any problems in individual students. If different teaching methods are used, the reports can also assist instructors in determining if certain interventions can promote learning.

This vignette briefly introduced some of the capabilities of the ePort package. Inevitably, new approaches will necessitate new features in subsequent versions and might reveal unforeseen bugs. Please send comments, suggestions, questions, and bug reports to amyf@iastate.edu.