# ePort Vignette:
# Academic report generation for statistics instructors

Xiaoyue Cheng, Di Cook, Lindsay Rutter, Amy Froelich

`ePort` version 0.0.1 , 2016-01-12

## Contents

[1]This LaTeX vignette document is created using the R function `Sweave` on the R package `ePort`. It is automatically downloaded with the package and can be accessed with the R command `vignette("ePort")`.

# Introduction

## 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, with output format that allows course coordinators to quickly determine how well and consistently the multiple sections performed on a particular topic. 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, reports can be generated 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 problem set, and each student is assigned a random subset from this set of problems. However, sometimes there may be an unexpected discrepancy in student performance between problems in a given problem set, indicating an unintended discrepancy in the clearness or difficulty level of the problems to which students are randomly assigned. This package will allow users to efficiently find and fix such issues.

## 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:

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

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

```
library(ePort)
```

### Help files

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

```
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:

```
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.

## 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 intersted 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.

* How will others obtain this particular database?????????

### 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 certain number (usually one) from each of these 330 question sets.

### 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 are provided below in Table 1.

Table 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 |

### Learning outcomes

Each topic contains learning outcomes, which are a list of statements that describe what a student is expected to understand after learning 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.

**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 with the use of evaluation after administration to students. 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 enumerated in Table 2.

Table 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 |

Below, an example problem and solution is provided for each of the seven question types 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, 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

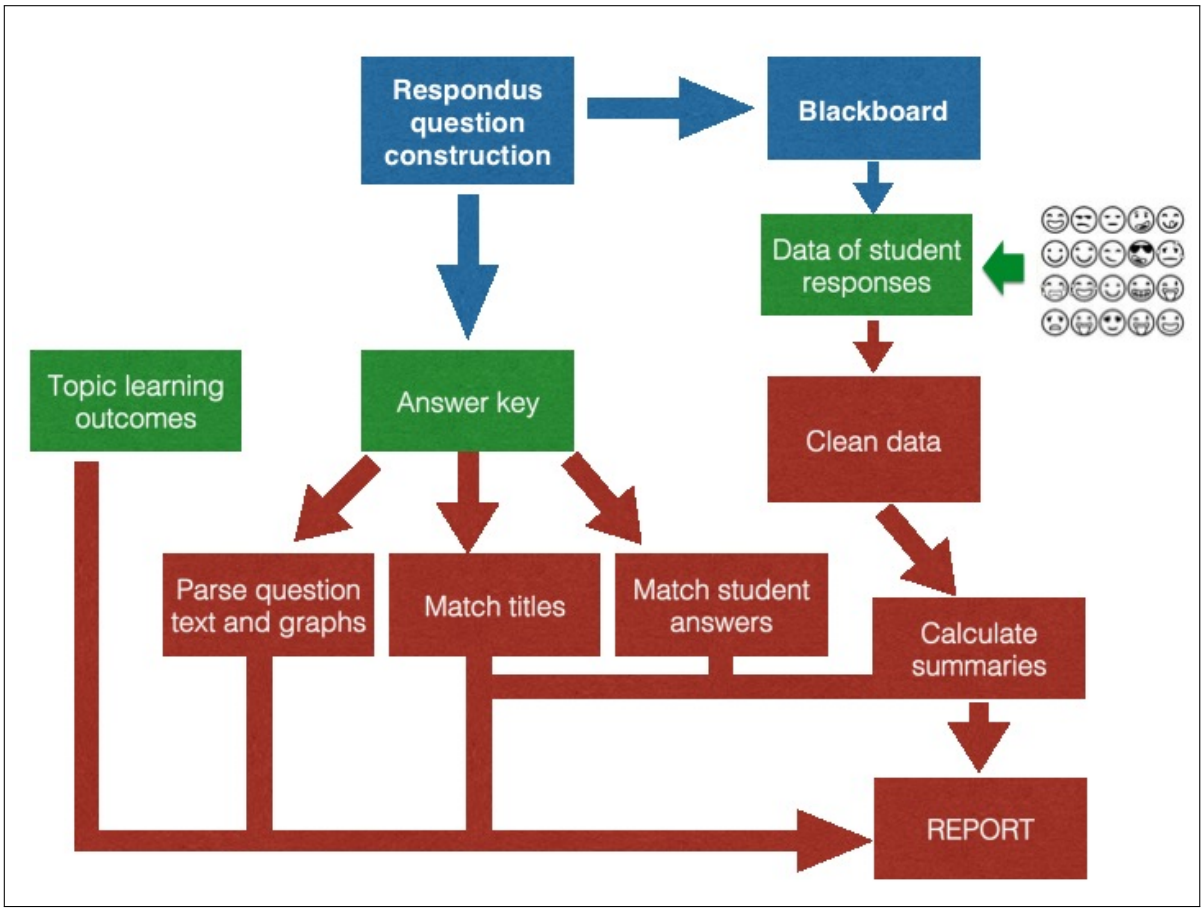
**Question sets**

A question set consists of a set of questions that all have the same format, 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.

## Report generation outline

A diagram of the report generation process can be seen in Figure 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 1: Overview of report generation




### External software - Respondus

External software that must be used for `ePort` to generate the reports are represented in the blue boxes of Figure 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. 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

### External software - Blackboard

As seen in Figure 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 the 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.

### Input files

The three necessary input files for `ePort` to generate reports are represented in green in Figure 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.

### Parsing input files

The functions needed to transform input files to final report files are represented in red in Figure 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 question set a particular student received at random is 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 of the topic. This means that it is not only the text, but also any graphs and equations from the questions that must be parsed in the answer key file. Doing so allows for the verbose report types to republish all components of a given question as was presented to students, such as relevant graphics, equations, and text, alongside the analysis of student performance for that question.

# 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:

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

* Add in picture with layout of the example data file * Describe necessary format of each type of example data file * Warn that real data should not be added to this example data file. To view this file, simply open it in a Web Brower (Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, Apple Safari).

# 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. Since it is most efficient for the user to hard-code in the `reportType` parameter, below is a reference for the parameter options (in quotes) for each of the six report types:

- 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 can also be obtained by running `help(makeReport)`, and will be demonstrated in the next two sections immediately below.

## One topic for one section - short version

### 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 AB`. 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 two example answer key files in the subdirectory `KeyFiles` (`Topic06.Questions.htm` and `Topic03.Questions.htm`), and we will use the `Topic06.Questions.htm` file. ????????????? Additionally, there should be four example data files in the subdirectory `DataFiles` (`Topic03.AB.csv`, `Topic03.CD.csv`, `Topic06.AB.csv`, and `Topic06.CD.csv`), ????????????????????????????????????? and we will use the `Topic06.AB.csv` file. Lastly, there should be two example learning outcome files in the subdirectory `LOFiles` (`Topic03.Outcomes.txt` and `Topic06.Outcomes.txt`), and we will use the `Topic06.Outcomes.txt` file.

The block of code we will use to generate our `Topic 06 Section AB` 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, and is something you can refer back to once you have completed the vignette at least once, should you want the code in one condense location:

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

refineKey(key_htm)

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

dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.AB.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 `key_htm`.

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

Second, we must parse and clean this .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(key_htm)
```

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`.

```
keyPath = gsub("htm$","txt",key_htm)
```

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`.

```
dataPath = system.file("inst/extdata/DataFiles/Topic06/Topic06.AB.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 changes certain non-meaningful character issues that would otherwise cause a problem when running the reports. For more details about the specific process, please run a help command on the function. Below, we rewrite the data file:

```
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 indeed converted it. This 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.AB.csv was already successfully converted to usable format."

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

```
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 path 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 the following message and menu asking for your input:

> 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, upon seeing the menu option, we can also escape out of the command, 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 specifying by hard-coding the `reportType` parameter is especially pertinent for users who wish to run these reports serially for each of many sections. This will be made more clear in a later section of the vignette (Running sequentially on batch of sections).

#### 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 (`Stat101_hwkTopic06_ABshort.pdf`) for `Topic 06` and `Section AB`. At this point, it may be helpful for you to open the short report you just generated.

This short report contains a table that provides an overview of student performance. as seen in Figure 2.

Figure 2: Overview of report generation

| Mean | Std.dev | Min | Q1 | Median | Q3 | Max |
|---|---|---|---|---|---|---|
| 8.33 | 6.56 | 0.00 | 5.00 | 6.00 | 8.00 | 31.00 |
| (24%) | (19%) | (0%) | (14%) | (17%) | (23%) | (89%) |

And a .txt file that contains the list of students who have scored below 80 on this assignment Topic06ABstudentsbelow80.txt.

### One topic for one section - long version

#### Code

The short report we created for one section one topic (`Stat101hwkTopic06ABshort.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 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, all we need to rerun is the `makeReport()` function, only this time specifying the `reportType` parameter with the option of "secTopicLong".

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

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 (`Stat101hwkTopic06AB.pdf`) of one topic (`Topic 06`) for one section (`Section AB`).

**Output**

## One topic comparing multiple sections - short version

Now that we have successfully generated reports for both sections of `Topic 06` separately, we may wish to generate a report that compares the performance between these two sections. In this case, we use the same `makeReport()` function as before. However, for our parameter `dataFile`, we will input a list `dataList` that lists 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:

```
dataFolder = system.file("inst/extdata/DataFiles/Topic06", package = "ePort")
dataList = list.files(path = dataFolder, full.names = TRUE)[1:2]
makeReport(keyFile = keyPath, dataFile = dataList, loFile = loPath, outFile = outPath, reportType = "
```

## One topic comparing multiple sections - long version

We received a brief comparative summary between the sections of Topic 06 in the previous section. 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 = "
```

## One unit (group of topics) for one section

```
dataFolder = system.file("inst/extdata/DataFiles/Topic03_06", package="ePort")
dataList = list.files(path = dataFolder, full.names = TRUE)
for (file in dataList){
  rewriteData(file)
}
dataTable = setDir(dataFolder)
mergedData = mergeData(dataTable)
# Add this to makeReport if we are using this file!!!!!!!!!!!!!!!!!!!
```

```
for (sctn in unique(dataTable$section)) {
  merged = subsetData(mergedData, dataTable, choice = sctn)
  knit("/Users/lindz/ePort/inst/Rnw/hw-topic.Rnw", output = paste0('Stat101hwk_Unit1_Section',sctn, '
}
```

### One unit (group of topics) comparing multiple sections

```
merged = subsetData(mergedData, dataTable)
knit("/Users/lindz/ePort/inst/Rnw/hw-topic-section.Rnw", output = 'Stat101hwk_Unit1_allSections.tex')
```

## Additional tools

### 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. For our example, we will split the Topic 03 data files for both sections AB and CD. !!!!!!!!!!!

```
dataFolder = system.file("inst/extdata/DataFiles/Topic03_Split", package="ePort")
dataList = list.files(path=dataFolder,full.names=TRUE)[1:2]
for(file in dataList){
  rewriteData(file)
  splitFile(file, 9, "ID")
}
```

### Merging files

```
dataPath = "~/Dropbox/NSF Grant 2013-2015/Semesters/Fall 2014/Data Files/Ch9-Ch11"
namelist = list.files(path=dataPath,full.names=TRUE)
for (i in namelist) rewriteData(i) # May not need this rewriting step!
for(i in c('AB','CD','EF','GH','JK','LM')){
  tmp = namelist[grep(i,basename(namelist))]
  combineFiles(tmp[2],tmp[1],paste("Topic11",i,"csv",sep='.'))
}
```

### 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 AB and CD of Topic 03 are located in the `DataFiles/Topic03_Deidentified` folder of the `extdata` folder. If you examine the content of these files, then you will notice that the column called `Username` contains the names of students. Of course, these are not real student names, but instead are fictitious student names generated for example purposes.

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. If you complete this process, you will notice that the original data files have been overwritten.

```
dataFolder = system.file("inst/extdata/DataFiles/Topic03_Deidentified", package="ePort")
getNameList(dataFolder, section=NULL, semester=NULL, secblind=TRUE, save=TRUE)
encodeName(dataFolder, dict=paste(dataFolder, "nameCode.csv", sep='/'))
```

## Running reports sequentially

If a course has multiple sections, and we wish to create an individual report for each of the many sections, then one inconvenient way to accomplish this would be to run the example code above, separately for each section at a time, with new data files each time. However, a more convenient way to accomplish the task would be to run all the reports at once. We are still using the same key and learning outcome files, although we would now need two data files (one with the answers from students in `Section AB` and one with the answers from students in `Section CD`).

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.AB.csv", package="ePort"),
system.file("inst/extdata/DataFiles/Topic06/Topic06.CD.csv", package="ePort"))
```

Next, as demonstrated below, we can generate a report for both sections listed in our `dataListPath` object using a for loop. Our two data files of interest in the for loop (`Topic06.AB.csv` and `Topic06.CD.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 two data files of interest do not share the same data file. We have already executed the `rewriteData()` function for the `Topic06.AB.csv` data file, but we have not yet done so for the `Topic06.CD.csv` file. Hence, we must include the `rewriteData()` function in our for loop to ensure that both data files have this priming step completed.

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

We can also, however, simply

```
dataFolder = system.file("inst/extdata/DataFiles/Topic06", package="ePort")
#namelist = list.files(path=dataFolder, pattern = "^[^.]*\.[^.]*\.[^.]*£", full.names=FALSE)
#http://stackoverflow.com/questions/9949176/match-string-with-exactly-2-of-a-given-character-e-g-2-li

#dataListPath = c(system.file("inst/extdata/DataFiles/Topic06/Topic06.AB.csv", package="ePort"), syst
#topic = gsub('.Questions.txt','',gsub('Topic','',basename(key)))
#namelist = list.files(path=dataPath,full.names=TRUE)
#namelist = namelist[grep(paste('Topic',topic,'\\.',sep=''),basename(namelist))]
#namelist
```

# Report Options

# Conclusions

The `ePort` package offers various plotting tools that can assist those studying genealogical lineages in the data exploration phases. As each plot comes with its advantages and disadvantages, we recommend for users to explore several of the available visualization tools.
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`.