Vignette for the ePort package: Academic report generation for statistics instructors

Xiaoyue Cheng, Di Cook, Lindsay Rutter, Amy Froelich

`ePort` version 0.0.1 , 2015-11-07

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")`.

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.

Both short and long versions of reports can be generated for any of the aforementioned scenarios. 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 problem in the assignment. Hence, 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.

Introduction

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.

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")
```

```
## [1] "/Users/lindz/ePort/inst/extdata"
```

* 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

The `ePort` package offers several functions that result in useful reports. Below is a brief introduction to some of the available functions.

One section one topic

Short report

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 `ReportFiles` subdirectory of the `extdata` directory. If you have not modified anything in the `extdata` directory, then the `ReportFiles` 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 `OutcomeFiles` (`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 as follows:

```
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.AB.csv", package="ePort")

rewriteData(dataPath)

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

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

makeReport(keyFile=keyPath, dataFile=dataPath, loFile=loPath, reportType=system.file("inst/Rnw/hw-ind
```

We now briefly explain each step of the process. First, we must save the absolute pathway of this 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 `refine_key()` 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 `refine_key()` 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 clean .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.AB.csv", package="ePort")
```

After this, our fifth step is to prime the data file to be compatible with our next steps of generating the reports. We do this by running the `rewrite_data()` 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)
```

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/OutcomeFiles/Topic06.Outcomes.txt", package="ePort")
```

Now that we have primed our three input files (cleaned answer key, data file, and learning outcomes file), our seventh and last step is to generate the report using the `report_routine()` function. Here, we specify that we want to output our report to the `ReportFiles` subdirectory, and that we want a short report.

```
makeReport(keyFile=keypath,dataFile=dataPath,loFile=loPath,reportType=system.file("inst/Rnw/", "hw-in
```

Now that we have successfully run the report generation, we can find our report in the `ReportFiles` subdirectory of the `extdata` example directory. Indeed, we see that we have our short report (`Stat101hwkTopic06ABshort.pdf`) of one topic (`Topic 06`) and one section (`Section AB`).

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

Long report

The short report summarization 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, this type specifying the `reportType` parameter as such

```
makeReport(keyFile=keypath,dataFile=dataPath,loFile=loPath,reportType=system.file("inst/Rnw/", "hw-in
```

We can find our output report in the `ReportFiles` subdirectory of the `extdata` example directory. Indeed, we see that we now have a much longer report (`Stat101hwkTopic06ABlong.pdf`) of one topic (`Topic 06`) and one section (`Section AB`).

```
#isParent("Young","Essex",sbGeneal)
#isParent("Essex","Young",sbGeneal)
```

We see that "Essex" is a parent of "Young", and not vice-versa. In some cases, you may wish to obtain a complete list of all the parents of a given variety. This can be achieved using the `getParent()` function:

```
#getParent("Young",sbGeneal)
#getParent("Tokyo",sbGeneal)
#getYear("Tokyo", sbGeneal)
```

We learn from this that "Essex" is not the only parent of "Young"; "Young" also has a parent "Davis". We also see that "Tokyo" does not have any documented parents in this dataset, and has an older year of introduction than other varieties we have examined thusfar. Likewise, in other cases, you may wish to obtain a complete list of all the children of a given variety. This can be achieved using the `getChild()` function:

```
#getChild("Tokyo",sbGeneal)
#getChild("Ogden",sbGeneal)
```

We find that even though the "Tokyo" variety is a grandparent of the dataset, it only has two children, "Ogden" and "Volstate". However, one of its children, "Ogden", produced Sex-prlength(getChild("Ogden",sbGeneal)) children.

If we want to obtain a list that contains more than just one generation past or previous to a given variety, then we can use the `getAncestors()` and `getDescendants()` functions, where we specify the number of generations we wish to view. This will return a data frame to us with the labels of each ancestor or descendant, along with the number of generations each one is from the given variety.

If we only look at one generation of ancestors of the "Young" variety, we should see the same information we did earlier when we used the `getParent()` function of the Young variety.

Multiple section one topic

Say you have a pair of vertices, and you wish to determine the degree of the shortest path between them, where edges represent parent-child relationships. You can accomplish that with the `getDegree()` function.

```
#getDegree("Tokyo", "Ogden", ig, sbGeneal)
#getDegree("Tokyo", "Holladay", ig, sbGeneal)
```

One section Multiple topics

Until this point, the vignette has introduced functions that return lists, data frames, and statistics about the genealogical dataset. However, the `ePort` package also contains visualization tools for genealogical datasets. Access to various types of visual plots and diagrams of the lineage can allow genealogical researchers to more efficiently and accurately explore an otherwise complicated data structure. Below, we introduce functions in `ePort` that produce visual outputs of the dataset.

Multiple Sections Multiple Topics

One visualization tool, `plotAncDes()`, allows the user to view the ancestors and descendants of a given variety. The inputted variety is highlighted in the center of the plot, ancestors are displayed to the left of the center, and descendants are displayed to the right of the center. The further left or right from the center, the larger the number of generations that particular ancestor/descendant is from the inputted and centered variety.

As such, this plotting command does not provide visual information about specific years associated with each related variety (as is done in some of the visualization tools introduced later), but it does group all varieties from each generation group onto the same position of the horizontal axis. Here, we specify that we want to plot 5 ancestor generations and 4 descendant generations of the variety "Lee":

Short reports

Brief summarizations

Low-scoring students

Long reports

Individual problem summarizations

Clustering analysis

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`.