Overview

Individualized exams: Why and how

Among the many logistic challenges of online teaching, administering exams is particularly difficult. This is because in-class exams have no straightforward equivalent in online teaching. Even when the general format of the exam (including the allocated completion time) remains unchanged, two typical rules of in-class exams can hardly be enforced in the virtual world: closed books and no collaboration. While we consider the shift from closed-book to open-book exams less of a problem in quantitative social science classes (as we believe that students should, for instance, focus on understanding when and how to use a given estimator as opposed to memorizing its formula), the lack of mechanisms to monitor compliance with no-collaboration rules is a more pressing concern.

Although fixed-slot exams, taken by all students at the same time for e.g. 90 minutes, minimize the risk of students walking each other through solutions step-by-step, they cannot prevent that students share their answers and workings with one another. This risk is even more apparent in times of online semesters, where students are (rightfully!) well connected through messenger services where they can easily share their answers with fellow students. Unless instructors want to ignore the risk of collaboration (and possibly sanction apparent violations of no-collaboration rules post-hoc) or refrain from administering in-class-style exams altogether, the best available solution is individualizing exams such that sharing answers is no longer a viable option. The obvious problem with this approach is that generating, distributing, and correcting individualized exams will be highly time consuming if done manually.

In this blog post, we therefore present an approach for the automated generation of individualized exam prompts, along with their automated distribution via email, using R and RMarkdown. Individualized exam prompts are generated alongside corresponding solution sheets, which allows for swift grading. Our approach focuses on question types that are widespread in quantitative social science exams. On the one hand, we show how to randomize numerical information (presented in tables or as parts of word problems in the exam prompt), which serve as inputs for examinees' calculations. On the other hand, we show how to randomize the order of choice items and choice alternatives in multiple or single choice problems. This procedure can also be used for randomly extracting a selection of (essay) questions from a larger question pool.

Implementing our approach requires little more than a student roster (i.e., a spreadsheet with students' names and email addresses) and follows three basic steps:

1. Generate an exam template in RMarkdown
2. Generate individualized exam prompts and solution sheets (e.g., in PDF format)
3. Distribute individualized prompts via email

As our approach provides examinees with classical document-based exam prompts, neither online exam platforms nor specialized software skills on the part of students are required. Students can complete their write-up on paper or using word processing or typesetting software and subsequently submit (scans of) their answers through file uploads on university-wide learning platforms such as ILIAS or Moodle.

We have successfully implemented this approach while collaboratively teaching *Quantitative Methods* at the University of Mannheim in Fall 2020. *Quantitative Methods* is a class for first-year Master's and PhD students of political science for whom attendance is mandatory. The course is optional for MMDS (Mannheim Master of Data Science) students and open to MA and PhD students from neighboring disciplines. In 2020, a total of 49 students attended the course.

Prerequisites

There are a few prerequisites for adopting our approach. The first is an installation of R and the installation of the following packages:

Code: R packages used in this tutorial

```
## ---- CRAN Packages ----
## Save package names as a vector of strings
pkgs <-  c("rmarkdown",
           "bookdown",
           "knitr",
           "stargazer",
           "dplyr",
           "xtable")

## Install uninstalled packages
lapply(pkgs[!(pkgs %in% installed.packages())],
       install.packages,
       dependencies = TRUE)

## Load all packages to library
lapply(pkgs,
       library,
       character.only = TRUE)

## ---- Other Packages ----
remotes::install_github("datawookie/emayili")
library(emayili)
```

The second is a *student roster*, e.g., a CSV file or Excel spreadsheet which keeps track of the students enrolled in a given class. This student roster should at least contain the following information on students:

- First name
- Last name
- Email address
- Individualized attachments (file names)
- Optional: A binary indicator filtering students who will not take the exam (e.g., due to sickness) and should thus not receive the prompt

The output below shows a minimal example of what a student roster may look like:

Output: Sample student roster

```
student_roster <- data.frame(
  first_name = c("Denis", "Marcel", "Oliver"),
  last_name = c("Cohen", "Neunhoeffer", "Rittmann"),
  login_id = c("dcohen", "mneunhoe", "orittman"),
  takes_exam = c(FALSE, TRUE, TRUE)
)
student_roster
##   first_name   last_name login_id takes_exam
## 1      Denis       Cohen   dcohen      FALSE
## 2     Marcel Neunhoeffer mneunhoe       TRUE
## 3     Oliver    Rittmann orittman       TRUE
```

Many universities assign students unique login IDs or user names (e.g. `dcohen`) which, in combination with the university domain (e.g., mail.uni-mannheim.de), serve as their default email address. We included these login IDs in our student roster, which allowed us to use them for the generation of both email addresses and attachment file names.

Step 1: Generate an exam template in RMarkdown

The first step involves setting up an `.Rmd` template for the exam. This template includes both fixed and variable components of what will eventually make up students' individualized exam prompts. The *fixed* components include the title page and all other information that are invariant across the individualized prompts (e.g., section headings and most question wordings). The fixed components also include R code chunks containing code for the randomized generation of objects, such as tables and question blocks that one wants to individualize, or individualized solutions. The *variable* components, on the other hand, include any commands that allow for these randomized objects to be printed in-text. This may involve objects embedded via R code chunks or inline R code.

YAML header and setup

The first step in creating the exam template is setting up an `.Rmd` file. We start with the YAML header, which defines the document's meta data, information to be printed on the title page, and the output format for knitting the resulting document. Here, we use `bookdown::pdf_document2` for PDF output and full LaTeX compatibility and specify various commands for document formatting and package use in `header-includes`. You may want to adjust these options to meet the specific requirements of your exam prompt.

Code: YAML header of our QM 2020 Midterm Exam

```
---
title: "QM 2020: Midterm Exam"

subtitle: "(50 points total)"

date: "November 25, 2020"

abstract: "*Please read carefully:* You have **90 minutes** to complete
```

```
the exam, plus **15 minutes** to export your answers as a **single PDF file** and upload the file to [ILIAS](URL). The exam is open book; you can consult any materials you like. You can use a calculator or statistical software to solve mathematical problems. You **must not collborate with or seek help from others**.
 In case of questions or technical difficulties, you can contact us via [Zoom](URL). You can write-up your answers on paper (and scan your pages), in MS Word (using Word's formula editor) or in RMarkdown (using LaTeX math notation). You can of course also input scans or pictures of your mathematical work into a Word or Markdown document, or use a digital pen for the mathematical work if you have a device with a touchscreen. Please make sure to concisely **number your answers** so that they can be matched with the corresponding questions."

fontsize: 12pt
linestretch: 1

output:
  bookdown::pdf_document2:
    fig_caption: yes
    number_sections: yes
    toc: false
    keep_tex: no

header-includes:
  - \usepackage{float}
  - \floatplacement{figure}{ht!}
  - \usepackage{longtable}
  - \usepackage{hyperref}
  - \usepackage{titlesec}
  - \hypersetup{colorlinks = true, linkcolor = blue, urlcolor = blue}
  - \renewcommand{\abstractname}{Instructions}
  - \pagenumbering{gobble}
  - \widowpenalty10000
  - \clubpenalty10000
  - \setlength{\parskip}{6pt}%
  - \setlength{\parindent}{0pt}%
  - \titlespacing\section{0pt}{11pt plus 4pt minus 2pt}{5.5pt plus 2pt minus 2pt}
  - \titlespacing\subsection{0pt}{11pt plus 4pt minus 2pt}{5.5pt plus 2pt minus 2pt}
  - \titlespacing\subsubsection{0pt}{11pt plus 4pt minus 2pt}{5.5pt plus 2pt minus 2pt}
  - \titlespacing\paragraph{0pt}{5.5pt plus 4pt minus 2pt}{11pt plus 2pt minus 2pt}
  - \usepackage{tikz}
  - \usepackage{tkz-tab}
  - \usetikzlibrary{positioning,arrows,calc,fit,arrows.meta}
  - \tikzset{
        modal/.style={>=stealth,shorten >=1pt,shorten <=1pt,auto,node distance=1.5cm,
            semithick},
```

```
        world/.style={circle,draw,minimum size=0.5cm,fill=gray!15},
        globe/.style={circle,draw,minimum size=0.5cm,fill=red!15},
        point/.style={circle,draw,inner sep=0.5mm,fill=black},
        reflexive above/.style={->,loop,looseness=7,in=120,out=60},
        reflexive below/.style={->,loop,looseness=7,in=240,out=300},
        reflexive left/.style={->,loop,looseness=7,in=150,out=210},
        reflexive right/.style={->,loop,looseness=7,in=30,out=330}
    }

link-citations: yes
linkcolor: blue
---
```

Following this, we set up an invisible code chunk to load all R packages (and optionally, define functions) needed for the generation of the exam prompt.

Code: Setup code chunk of RMD template

```
## ---- Setup ----
## Save package names as a vector of strings
pkgs <-  c("bookdown",
           "knitr",
           "stargazer",
           "dplyr",
           "xtable",
           "MASS")

## Install uninstalled packages
lapply(pkgs[!(pkgs %in% installed.packages())],
       install.packages,
       dependencies = TRUE)

## Load all packages to library
lapply(pkgs,
       library,
       character.only = TRUE)


## Global chunk options
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE)
options(scipen=999)

## Functions
inRange <- function(x, a, b) {
  (x - a)  *  (b - x) >= 0
}
```

After this initial setup, one can start with the actual template by typing up the fixed and variable components. In the following, we showcase how this can be done using examples for three

different question types.

Example 1: Randomizing numerical inputs for calculations

In the first set of questions in our exam, students were asked to calculate quantities of interest based on a regression output and descriptive statistics of the variables used in the regression model. We individualized these tasks by randomizing both the output of the regression table (coefficients, standard errors) and the descriptive statistics of the model variables (means, standard deviations). To accomplish this, we defined a placeholder for a student-specific seed (`PLACEHOLDER_SEED`) in our template (to be replaced in the generation of individualized exam prompts further below). We then drew random numbers to define the means and standard deviations of the model variables as well as for the point estimates of the model coefficients using the `sample()` command and generated the standard errors as a deterministic function of the coefficients. These randomly generated quantities were then presented to students in two tables. Below, we show what these tables look like when we set `PLACEHOLDER_SEED <- 20210114`.

Code: Randomizing numerical inputs for calculations

```
## ---- Seed ----
set.seed(PLACEHOLDER_SEED)

## ---- Regression example ----
## Generate numerics
N <- 1000L
mu_X <- sample(45:55, 1)
sd_X <- sample(10:15, 1)
pi_W <- sample(seq(.2, .8, .1), 1)
mu_Z <- sample(9:12, 1)
sd_Z <- sample(2:4, 1)
beta <- c(-sample(12:8, 1),
          sample(seq(0.5, 1.5, 0.5), 1),
          -sample(7:3, 1),
          sample(1:5, 1))
se <- abs(beta) * c(4, 10, 4, 6)^(-1)

## Descriptives table
descriptives <-
  rbind(
    c("Age", mu_X, sd_X),
    c("Female", pi_W, "(dummy)"),
    c("Education", mu_Z, sd_Z)
  )
colnames(descriptives) <-
  c("Variable", "Mean", "Std. Dev.")

## Regression table
regression_table <- cbind(format(beta, nsmall = 1, digits = 0),
                          format(se, nsmall = 1, digits = 1))
rownames(regression_table) <-
  names(beta) <- names(se) <- c("Intercept",
                                "Age",
```

```
                                  "Female",
                                  "Education")
colnames(regression_table) <-
  c("Beta", "Std. Err.")
```

Table 1: Descriptive Statistics.

Variable	Mean	Std. Dev.
Age	48	11
Female	0.8	(dummy)
Education	10	3

Table 2: Estimates from OLS Regression.

	Beta	Std. Err.
Intercept	-8.0	2.00
Age	0.5	0.05
Female	-5.0	1.25
Education	5.0	0.83

Students then had to use this randomized numerical information as inputs for several tasks involving calculations and interpretations, e.g.:

1. Interpret the coefficients for *Age* and *Female*.
2. The author claims that she “cannot reject the null hypothesis that *Education* has no effect on *Income* (\(H_{0}: \beta_{\text{Education}}\)=0)”. Using the coefficient estimate and the standard error for *Education*, construct a (rough) 95 percent confidence interval for the effect of *Education* on *Income* (you may use the rule of thumb for the calculation of the confidence interval). Based on the confidence interval, do you agree with the author? Explain your answer in one sentence.
3. Calculate the first difference in *Income* between low and high values of *Education*, holding all other variables constant at their sample means. Use the mean of *Education* \(\pm\) one standard deviation for low and high values of *Education*, respectively.

Due to the individualized numerical inputs, the correct answers to these questions also varied across students. To allow for swift grading, we set up a solutions template. This was an exact copy of the `.Rmd` exam template with additional blocks for printing the solutions to each individualized problem. For instance, the matching solutions to the three questions using the numerical inputs from the randomly generated tables shown above can be produced as follows:

Code: RMD code for individualized solutions

```
1. Holding all else constant, the prediction of *Income* changes by `r
beta["Age"]` units when *Age* increases by one unit. Holding all else
constant, the prediction of *Income* is `r beta["Female"]` units
higher/lower when $Female=1$ as opposed to $Female=0$.
1. Approximate 95% CI: `r round(beta["Education"] + c(-2, 2) *
se["Education"], 3)`. Exact 95% CI: `r round(beta["Education"] +
qnorm(c(.025, .975)) * se["Education"], 3)`.
1. The first difference in *Income* between low and high values of
*Education* is `r c(0, 40, 0, 0) %*% beta`.
```

Output

1. Holding all else constant, the prediction of *Income* changes by 0.5 units when *Age* increases by one unit. Holding all else constant, the prediction of *Income* is -5 units higher/lower when \(Female=1\) as opposed to \(Female=0\).
2. Approximate 95% CI: 3.333, 6.667. Exact 95% CI: 3.367, 6.633.
3. The first difference in *Income* between low and high values of *Education* is 20.

Example 2: Randomized question prompts

One can, of course, not only randomize numerical input for calculations but also the wording of the corresponding question prompts. The following is an example from our exam in which we first randomized the basic setup of a binomial distribution. We defined four scenarios which differed with respect to the name of a US state and the electoral support for Joseph R. Biden in the 2020 US Presidential Election (and thus, with respect to the probability parameter of the binomial distribution). We then randomly selected one of the four scenarios. The scenario was subsequently printed as the introductory text for the question prompt:

Code: Randomize question prompt

```
## Randomize
set.seed(PLACEHOLDER_SEED)
cat_select <- sample(1:4, 1)

## Define scenarios for introductoty text
cat_intro <- c(
  paste0(
    "Suppose you have sampled four voters from Vermont. ",
    "Each voter has a 2/3 probability of voting for Biden."
  ),
  paste0(
    "Suppose you have sampled four voters from Idaho. ",
    "Each voter has a 1/3 probability of voting for Biden."
  ),
  paste0(
    "Suppose you have sampled four voters from Wyoming. ",
    "Each voter has a 3/10 probability of voting for Biden."
  ),
  paste0(
    "Suppose you have sampled four voters from Washington DC.",
    " Each voter has a 9/10 probability of voting for Biden."
  )
)

cat_prob <- c(2 / 3,
              1 / 3,
              3 / 10,
              9 / 10)

## Print introductory text for prompt
cat(cat_intro[cat_select])
cat('\n')
```

Output

Suppose you have sampled four voters from Wyoming. Each voter has a 3/10 probability of voting for Biden.

In a second step, we also randomized the specific problem, defined by the number of Biden voters \(k\) in a sample of size \(N=4\), and printed the resulting individualized question:

Code: Randomize number of Biden voters

```
## Randomize
set.seed(PLACEHOLDER_SEED)
k_select <- sample(1:3, 1)

## Print questions
cat(paste0("1. What is the probability that exactly ", k_select, " out
of 4 voters vote for Biden?"))
cat('\n')
cat(paste0("2. What is the expected value for the number of Biden
voters in the sample?"))
```

Output

1. What is the probability that exactly 3 out of 4 voters vote for Biden?
2. What is the expected value for the number of Biden voters in the sample?

As before, we made sure to include the solutions to these individualized questions in our solutions template. This resulted in the following output:

Code: RMD code for individualized solutions

```
1. `r sum(dbinom(0:4, size = 4, prob = cat_prob[cat_select])[k_
select:4])`
2. `r 4 * cat_prob[cat_select]`
```

Output

1. 0.3402
2. 1.2

Example 3: Randomizing choice items and alternatives

Lastly, we present an approach for randomizing both items and alternatives for multiple choice and single-choice questions. Our exam included eight single-choice questions, i.e., each question had four to five alternatives of which only one was correct.

To generate a fully randomized single-choice section in our exam, we proceeded as follows. First, we defined a nested list named `single_choice`. This nested list bound together a list of three objects for each of the eight questions:

1. A character `Q`, which contained the question wording,
2. A list of characters `A`, which contained the alternatives, and

3. A character `C`, which contained the correct alternative.

The code below illustrates the definition of this nested list for two questions.

Code: Defining a nested list for items and alternatives

```
## ---- Questions ----
single_choice <- list()
single_choice[[1]] <- list()
single_choice[[1]]$Q <-
  "When we use the term '95% confidence interval' for a sample mean, we
mean that..."
single_choice[[1]]$A <- list(
  "we can reject the null hypothesis with 95 percent confidence.",
  "there is a 95 percent probability that the population mean is within
the interval.",
  "the true population parameter is contained in the interval about 95
times if we were to repeat the experiment one hundred times.",
  "any given 95 percent confidence interval from a random sample will
contain the true population mean.",
  "None of the other answers."
)
single_choice[[1]]$C <- "the true population parameter is contained in
the interval about 95 times if we were to repeat the experiment one
hundred times."

single_choice[[2]] <- list()
single_choice[[2]]$Q <-
  "Let $X_1$ and $X_2$ be two different values of an independent
variable. A first-difference is..."
single_choice[[2]]$A <- list(
  "a quantity of interest from a regression model defined as
$E(Y|X_1)/E(Y|X_2)$.",
  "a quantity of interest  from a regression model defined as $Y|X_1 -
Y|X_2$.",
  "a quantity of interest  from a regression model defined as
$E(Y|X_1)+E(Y|X_2)$.",
  "a quantity of interest  from a regression model defined as $E(Y|1)-
E(Y|0)$.",
  "None of the other answers."
)
single_choice[[2]]$C <- "None of the other answers."
```

In a second step, we randomized the order of question items (`q_order`, which indexes `single-choice` at the highest level) and the order of choice alternatives within question items (`a_order`, which indexes `single_choice` at the second level). Using these indices, we then defined a nested loop to print the randomized question items and choice alternatives. This resulted in the following output:

Code: Randomizing and printing items and choice alternatives

```
## ---- Randomize ----
```

```
set.seed(PLACEHOLDER_SEED)
q_order <- sample(seq_along(single_choice), length(single_choice))
a_order <-
  lapply(single_choice, function(q)
    sample(seq_along(q$A), length(q$A)))

## ---- Print ----
for (q in q_order) {
  cat('\n')
  cat(paste0("1. ", single_choice[[q]]$Q))
  cat('\n')
  for (a in a_order[[q]]) {
    cat(paste0("    a. ", single_choice[[q]]$A[[a]]))
    cat('\n')
  }
}
```

Output

1. When we use the term ‘95% confidence interval’ for a sample mean, we mean that…
    a. any given 95 percent confidence interval from a random sample will contain the true population mean.
    b. None of the other answers.
    c. there is a 95 percent probability that the population mean is within the interval.
    d. the true population parameter is contained in the interval about 95 times if we were to repeat the experiment one hundred times.
    e. we can reject the null hypothesis with 95 percent confidence.
2. Let \(X_1\) and \(X_2\) be two different values of an independent variable. A first-difference is…
    a. None of the other answers.
    b. a quantity of interest from a regression model defined as \(E(Y|1)-E(Y|0)\).
    c. a quantity of interest from a regression model defined as \(E(Y|X_1)/E(Y|X_2)\).
    d. a quantity of interest from a regression model defined as \(E(Y|X_1)+E(Y|X_2)\).
    e. a quantity of interest from a regression model defined as \(Y|X_1 - Y|X_2\).

Lastly, we added the corresponding solutions to our template using the following code:

Code: Printing individualized single-choice solutions

```
## ---- Print ----
for (q in q_order) {
  cat('\n')
  cat(paste0("1. Alternative (",
             letters[which(unlist(single_choice[[q]]$A[a_order[[q]]])
==
                           single_choice[[q]]$C)],
             ")"))
}
```

Output

1. Alternative (d)

2. Alternative (a)

Step 2: Generate individualized exam prompts and solution sheets

Having defined the `.Rmd` exam and solutions templates, the next task is to generate the individualized exam prompts and solutions sheets for *all* students. For this process, we set up a separate R script. Next to loading the required packages `dplyr` and `rmarkdown` packages, we define a function `letter2number()` for a character-to-numeric conversion.

Code: Generating individualized exams (setup)

```
## ---- Setup ----
## Save package names as a vector of strings
pkgs <-  c("dplyr",
           "rmarkdown")

## Install uninstalled packages
lapply(pkgs[!(pkgs %in% installed.packages())],
       install.packages,
       dependencies = TRUE)
## list()
## Load all packages to library
lapply(pkgs,
       library,
       character.only = TRUE)
## [[1]]
##  [1] "emayili"   "xtable"    "dplyr"     "stargazer" "knitr"
"bookdown"
##  [7] "stats"     "graphics"  "grDevices" "utils"     "datasets"
"methods"
## [13] "base"
##
## [[2]]
##  [1] "rmarkdown" "emayili"   "xtable"    "dplyr"     "stargazer"
"knitr"
##  [7] "bookdown"  "stats"     "graphics"  "grDevices" "utils"
"datasets"
## [13] "methods"   "base"
## Function
letter2number <- function(x) {
  (utf8ToInt(x) - utf8ToInt("a") + 1L) %>%
    paste0(collapse = "") %>%
    as.numeric()
}
```

We first need to subset our `student_roster` to those students who take the exam and then generate a student-specific numeric seed. Toward this end, we use the `letter2number()` function for a character-to-numeric conversion of each student's login ID, of which we preserve the first eight digits.

R Code: Generating individualized exams (step 1)

```
## ---- Students list and seeds ----
student_roster <- student_roster %>%
  dplyr::filter(takes_exam) %>%
  group_by(login_id) %>%
  dplyr::mutate(seed = letter2number(login_id)) %>%
  dplyr::mutate(seed = as.integer(substr(seed, 1, 8)))
```

In the next step, we want to use our general RMarkdown exam and solution templates to generate student-specific `.Rmd` files. Toward that end, we first read in the exam and solution templates using the `readChar()` function and store them in `template_exam` and `tempalte_solution`, respectively. We then use a loop through all students in `student_roster` to iteratively replace all occurrences of the placeholder (`PLACEHOLDER_SEED`) with the student-specific seeds and to save the resulting individualized `.Rmd` files under individualized file names (e.g., `midterm_mneunhoe.Rmd` and `solutions_mneunhoe.Rmd`).

R Code: Generating individualized exams (step 2)

```
## ---- Generate exams and solutions from template ----
## Read in templates
template_exam <- readChar(
  "02-midterm-exam/midterm-exam-template.Rmd",
  file.info("02-midterm-exam/midterm-exam-template.Rmd")$size
)
template_solution <- readChar(
  "02-midterm-exam/midterm-solutions-template.Rmd",
  file.info("02-midterm-exam/midterm-solutions-template.Rmd")$size
)


for (s in seq_len(nrow(students))) {
  ## Generate and save individualized exams
  tmp_script <- template_exam
  tmp_script <- gsub("PLACEHOLDER_SEED",
                     as.numeric(students[s, "seed"]),
                     tmp_script)
  cat(tmp_script, file = paste0("02-midterm-exam/",
                                "individualized-exams-rmd/",
                                "midterm_",
                                as.character(students[s, "Login"]),
                                ".Rmd"))

  ## Generate and save individualized solutions
  tmp_script <- template_solution
  tmp_script <- gsub("PLACEHOLDER_SEED",
                     as.numeric(students[s, "seed"]),
                     tmp_script)
  cat(tmp_script, file = paste0("02-midterm-exam/",
                                "individualized-solutions-rmd/",
                                "solutions_",
                                as.character(students[s, "Login"]),
```

```
                                        ".Rmd"))
}
```

In the last step, we then render the individualized RMarkdown scripts by knitting the `.Rmd` files to `.pdf` using the `rmarkdown::render()` function:

R Code: Generating individualized exams (step 3)

```
## ---- Render exams and solutions-----
## Exams
rmd_files <- list.files("02-midterm-exam/individualized-exams-rmd")
rmd_files <- rmd_files[grepl(".Rmd", rmd_files)]
for (rmd_file in rmd_files) {
  rmarkdown::render(
    paste0("02-midterm-exam/individualized-exams-rmd/", rmd_file),
    output_dir = "02-midterm-exam/individualized-exams-pdf")
}

## Solutions
rmd_files <- list.files("02-midterm-exam/individualized-solutions-rmd")
rmd_files <- rmd_files[grepl(".Rmd", rmd_files)]
for (rmd_file in rmd_files) {
  rmarkdown::render(
    paste0("02-midterm-exam/individualized-solutions-rmd/", rmd_file),
    output_dir = "02-midterm-exam/individualized-solutions-pdf")
}
```

This leaves us with individualized PDF files for the exam prompts in the output directory `"02-midterm-exam/individualized-exams-pdf"` (along with individualized PDF files for the corresponding solutions in the directory `"02-midterm-exam/individualized-solutions-pdf"`), whose file names are personalized with students login IDs (e.g., `exam_mneunhoe.pdf`). In the next step, we show how these files can be automatically distributed via email through university mail servers using R.

Step 3: Distribute individualized prompts via email

To send out the exams via email, we use the emayili package. As we want to attach the individualized exam prompts to our emails, we first define an object named `exams` that lists the file names of our attachments. We then specify our email credentials using the `server()` function (here, we obviously specify fake inputs for `host`, `port`, `username`, and `password`).

We then set up the `email_template`, which contains all email meta data that is invariant across students: `from()` and `cc()` addresses as well as the `subject()`. Similarly, we define a `html_body_template` for the text body of our email. We specify a placeholder for student names (`PLACEHOLDER_NAME`), to be replaced with individual names in the next step. Note that the text body is written in HTML. We therefore include line breaks per
and insert hyperlinks via .

R Code: Distributing individualized exams (step 1)

```
## List file names of individualized exam prompts
```

```
exam_path <- "../02-midterm-exam/individualized-exams-pdf"
exams <- list.files(exam_path)

## Define mail server
smtp <- server(host = "smtp.uni.edu",
               port = 465,
               username = "dcohen@uni.edu",
               password = "ThisIsMySecretPassword2020!")

  ## Setup email template
  email_template <- envelope()

  ## Define from, to, and CC addresses
  email_template <- email_template %>%
    from("denis.cohen@uni.edu") %>%
    cc(
      c(
        "denis.cohen@uni.edu",
        "mneunhoe@uni.edu",
        "orittman@uni.edu"
      )
    )

  ## Define subject
  email_template <- email_template %>%
    subject("QM 2020 // Midterm Exam")

  ## Define email body in HTML
  html_body_template <-
    paste0(
      "
",
    "Dear ",
    "PLACEHOLDER_NAME",
    ",",
    "

",
    "attached to this email you find the QM 2020 Midterm Exam.",
    "

",
    "You have 90 minutes to complete the exam, plus 15 minutes to export your",
    " answers as a single PDF file and upload the file to ",
    "ILIAS before 10h15.",
    "

",
    "The exam is open book; you can consult any materials you like. ",
    "You can use a calculator or statistical software to solve mathematical ",
    "problems.",
```

```
    "
",
    "You must not collborate with or seek help from others.",
    "

",
    "In case of questions or technical difficulties, you can contact us via ",
    "Zoom.",
    "

",
    "Good luck!",
    "

",
    "Best,",
    "
",
    "Denis",
    "
"
    )
```

Following this, we can start individualizing and sending out our emails. First, we retrieve the student-specific file path to the individualized exam prompt. As we individualized all file names with students' login ID (e.g., `exam_mneunhoe.pdf`), we can simply paste the path to the directory `exam_path`, the file name prefix `"exam_"`, the idiosyncratic student ID, and the extension `".pdf"`.

Secondly, we need to individualize the `html_body`. For this, we simply replace `"PLACEHOLDER_NAME"` with each student's `first_name` from the `student_roster`.

Thirdly, we compose the individualized `email` by adding the individualized `html_body` to the general `email_template` via the `html()` function, defining the recipient's `to()` address by pasting their unique `login_id` with the university's general email domain, and attaching the `individualized_exam` via the `attachment()` function. The option `name = "QM2020-midterm.pdf"` allows us to assign a general file name for all individualized attachments.

Lastly, we send the email using `smtp()`. This process is repeated iteratively by looping through all students in `student_roster`.

R Code: Distributing individualized exams (step 2)

```
## Start loop through students in student_roster
for (i in seq_len(nrow(student_roster))) {
  ## Retrieve file path for individualized exam
  individualized_exam <- paste0(
    exam_path,
    "/exam_",
    student_roster$login_id[i],
    ".pdf"
    )
```

```
  ## Individualize HTML Body
  html_body <- gsub("PLACEHOLDER_NAME",
                    student_roster$first_name[i],
                    html_body_template)

  ## Compose individualized email and add attachment
  email <- email_template %>%
    to(paste0(student_roster$login_id[i], "@uni.edu")) %>%
    html(html_body) %>%
    attachment(path = individualized_exam, name = "QM2020-midterm.pdf")

  ## Send email
  smtp(email, verbose = TRUE)
}
## End loop through students in student_roster
```

Note that once students' points and grades have been added to the `student_roster`, an equivalent approach can be used for sending out individual grades and feedback.

Conclusion

A central challenge of online teaching is to adapt exams originally designed as closed-book in-class exams such that they can be administered as open-book take-home exams. In this blog post, we have shared our experiences with an approach for generating fully individualized exams at minimal cost. In practice, our approach turned out to be highly workable. It allowed us to generate about 50 distinct exam prompts. The automatic generation of matching solution sheets ensured that the individualization of these exams did not result in an increase in our workload for corrections and grading. We hope that sharing our experiences will help others tackle the new problems and burdens that come with online teaching. If you decide to adopt our approach, we would be excited to hear about your experience.