score-completion

Linn Zapffe

2025-02-24

The following Rmarkdown document will go over the surveys required for the Sleepin' Deacon challenge at Wake Forest University and score who fulfilled all the requirements for the challenge.

Importing libraries

```
# Read data from Excel files into R
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.3.3
```

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
##  dplyr     1.1.4        readr     2.1.4
##  forcats   1.0.0        stringr   1.5.1
##  ggplot2   3.4.4        tibble    3.2.1
##  lubridate 1.9.3        tidyr     1.3.0
##  purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
##  dplyr::filter() masks stats::filter()
##  dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# To clean up the variable names for data frames
library(janitor)
```

```
## Warning: package 'janitor' was built under R version 4.3.3
```

```
##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
# To glue variable to variable name and call it with glue() and {} inside strings
library(glue)
# Write the data frames as Excel files
library(writexl)
```

```
## Warning: package 'writexl' was built under R version 4.3.3
```

Add a note about downloading libraries

Reading in the Data

Make a vector with the data file names (in a particular order), as well as a vector with the data frame names. Or, make it a dictionary?

Make sure the names for the data files are correct Make sure the files are in the data folder Make sure the file types are correct

To download data from Qualtrics:

- Click Data and Analysis
- Click export and import
- Click Export data…
- Chose Excel
- Click export values
- Click Download
- Wait for it to finish downloading, it will download automatically to your downloads folder
- Extract it from the zip folder
- Move it to the data folder

Cleaning Variable Names

Making a Data Frame Marking Completion

I will use the registration data frame to get a list of all the participants. We can then use this list of email addresses and names to match with each survey and tally up how many of them they have done.

First, doing all of this to prep df_data:

```
# Reading in the data from registration
df_reg <- read_excel("data/Sleepin Deacon 2025 Registration Form (Responses).xlsx")
# Cleaning up the variable names with janitor
df_reg <- df_reg %>%
  clean_names()
df_data <- df_reg
# Adding first and last name to one column
df_data <- df_data %>%
  unite("contact_info", c("first_name", "last_name", "wfu_email_address"), remove = FALSE)
# Converting full_name to all lower case
df_data$contact_info <- df_data$contact_info %>% str_to_lower()
```

Checking for duplicates in df_data:

```
# Create data frame with duplicates from contact_info
df_data_dup <- df_data %>%
  group_by(contact_info) %>%
```

```
  filter(n() > 1) %>%
  ungroup()

# Delete cases if the contact_info is the same
  # NB: they could have answered differently to the other questions, such as whether they want
 the sleep kit. However, that doesn't matter for counting up completed surveys, so it has not
been considered for this duplicate removal
df_data <- df_data %>%
  distinct(contact_info, .keep_all = TRUE)
```

```
# Creating the data frame df_data with only the personal information
df_data <- df_data %>% select("contact_info", "first_name", "last_name", "wfu_email_address")
```

Add in something about what happens if variable names are wrong. Add in error and what to do to change it.

Trying to do everything for day 1

Maybe this can be made into a function

Stuff to do before calling the function:

```
df_check <- tibble(
  contact_info = character(),
  first_name = character(),
  last_name = character(),
  email_address = character(),
  survey = character(),
  reason = character()
  )

# Important that these file paths are in order, from day 1 to day 14
vect_daily_file_paths <- c("data/2025 Sleepin' Deacon Challenge Journal 1 (Responses).xlsx", "
data/2025 Sleepin' Deacon Challenge Journal 2 (Responses).xlsx", "data/2025 Sleepin' Deacon Ch
allenge Journal 3 (Responses).xlsx", "data/2025 Sleepin' Deacon Challenge Journal 4 (Responses
).xlsx", "data/2025 Sleepin' Deacon Challenge Journal 5 (Responses).xlsx", "data/2025 Sleepin'
 Deacon Challenge Journal 6 (Responses).xlsx", "data/2025 Sleepin' Deacon Challenge Journal 7
(Responses).xlsx", "data/2025 Sleepin' Deacon Challenge Journal 8 (Responses).xlsx", "data/202
5 Sleepin' Deacon Challenge Journal 9 (Responses).xlsx", "data/2025 Sleepin' Deacon Challenge
Journal 10 (Responses).xlsx", "data/2025 Sleepin' Deacon Challenge Journal 11 (Responses).xlsx
", "data/2025 Sleepin' Deacon Challenge Journal 12 (Responses).xlsx", "data/2025 Sleepin' Deac
on Challenge Journal 13 (Responses).xlsx", "data/2025 Sleepin' Deacon Challenge Journal 14 (Re
sponses).xlsx")
```

Defining the function for wrangling the daily surveys and adding the information to df_data:

```
# Defining a function that will data wrangle each daily survey and add that information to df_
data, as well as df_check for entries requiring manual processing

process_daily_survey <- function(survey_num, file_path_list, registration_df, manual_check_df)
{
  # survey_num is which daily survey number we are on, the file_path is a list with the file p
aths for the daily surveys (1-14), registration_df is the data_df that we use to match partici
```

```
pants to, and manual_check_df is check_df where we record all the records that have to be look
ed over manually later.

  ### 1. Prepping the daily survey ################################################
  # Reading in the data from the desired survey, based on the file path
  df <- read_excel(vect_daily_file_paths[survey_num])
  # Using janitor to clean the variable names
  df <- df %>%
    clean_names()
  # Adding first and last name to one column
  df <- df %>%
    unite("contact_info", c("first_name", "last_name", "email_address"), remove = FALSE)
  # Converting full_name to all lower case
  df$contact_info <- df$contact_info %>% str_to_lower()


  ### 2. Adding variables to daily survey ##########################################
  # Giving 1 to all responses, to match with data_df later for completion of survey
    # The {} is from the glue package in Tidyverse, it requires := instead of = to work
  df <- df %>%
    mutate("day_{survey_num}" := 1)
  # Making a variable looking for missing values
  df <- df %>%
    mutate(
      "na_{survey_num}" := case_when(
        if_any(everything(), is.na) ~ 1,
        TRUE ~ 0
      )
    )

  # Handling the date variable and formatting
    # For some reason, R reads the time stamp date/time variable in with different formatting,
 either as date-time or as days since origin (a number over 4,000). Therefore, we have to chec
k what format the timestamp variable is in and covert it to just the date as a string
  if (is.double(df$timestamp) == TRUE){
    # If the date is in a date-time format, we cut the time of survey completion from the time
 stamp variable to only see the date
    df <- df %>%
    mutate(
      timestamp = sub(" .*", "", timestamp)
    )
  } else {
    df$timestamp <- df$timestamp %>%
      as.numeric() %>%
      as.Date(origin="1899-12-30") %>%
      as.character()
  }

  # Finding the date that most participants filled out the survey on
  day_mode <- df %>%
      count(timestamp) %>%
      slice(which.max(n))
  # Saving that date as day_mode
  day_mode <- day_mode$timestamp
```

```
  # Creating a new variable for if the survey was filled out at another day than the most comm
on day
  df <- df %>%
    mutate("late_{survey_num}" := if_else(timestamp == day_mode, 0, 1))


  ### 3. Looking for duplicates in df_1 ########################################
  # Create data frame with duplicates from contact_info
  df_dup <- df %>%
    group_by(contact_info) %>%
    filter(n() > 1) %>%
    ungroup()
  # If there are any duplicated rows, which have all the same data, delete the duplicates
  if (nrow(df_dup) != 0){
    df <- df %>%
    select(!(timestamp)) %>%
    distinct(.keep_all = TRUE)
  }
  # Checking whether there are still any duplicated rows left (that has different responses)
  df_dup <- df %>%
    group_by(contact_info) %>%
    filter(n() > 1) %>%
    ungroup()
  # If there are any duplicated rows, which have different data, add these entries to the df_c
heck for manual checking
  if (nrow(df_dup) != 0){
    df_dup <- df_dup %>%
      select(contact_info, first_name, last_name, email_address) %>%
      mutate(survey = as.character(survey_num)) %>%
      mutate(reason = "duplicated")
    df_check <- df_check %>%
      add_row(df_dup)
    # Now, delete these entries from df so that they do not merge in weird ways with df_data
    df <- df %>%
      filter(!contact_info %in% df_dup$contact_info)

  }


  ### 4. Combining the data frames #############################################
  # Join completion and NA variable from df_1 to df_data by contact_info
    # Use right_join so that I can select only certain columns in df_1 to join
  df_data <- df %>%
    select(all_of(c("contact_info", glue("day_{survey_num}"), glue("na_{survey_num}"), glue("l
ate_{survey_num}")))) %>%
    right_join(df_data, join_by(contact_info))
  # Anti_join df_1 and df_data to find data entries that didn't match with the registered part
icipants. For example if they misspelled their contact information
  df_unmatched <- df %>%
    select(c("contact_info", "first_name", "last_name", "email_address")) %>%
    anti_join(df_data, join_by(contact_info))
  # Add survey column to df_unmatch to indicate the day of the survey to look at
  df_unmatched <- df_unmatched %>%
    mutate(survey = as.character(survey_num)) %>%
```

```
    mutate(reason = "unmatched contact information")
  # Add the unmatched responses to df_check for manual walk-through
  df_check <- df_check %>%
    add_row(df_unmatched)
  return(list(df_data, df_check))
}
```

```
# Calling the function we defined earlier to wrangle and combine the information from the diff
erent daily surveys into one df_data and one df_check
  # I tried using lapply here, but it gets kind of messy doing all three things in one lapply
call. I think I might have to write an anonymous function to do this.
for (i in 1:14) {
  obj_list <- process_daily_survey(i, vect_daily_file_paths, df_data, df_check)
  df_data <- obj_list[[1]]
  df_check <- obj_list[[2]]
}
```

```
## Warning in as.Date(., origin = "1899-12-30"): NAs introduced by coercion
```

Counting up the number of completed surveys, surveys with missing values, and late submissions for each participant:

```
# Creating a variable that counts the total number of daily surveys completed
df_data <- df_data %>%
  mutate(
    day_total = rowSums(across(starts_with("day_")), na.rm = TRUE))

# Creating a variable that counts the total number of daily surveys with missing data
df_data <- df_data %>%
  mutate(
    na_total = rowSums(across(starts_with("na_")), na.rm = TRUE))

# Creating a variable that counts the total number of daily surveys that were submitted late
df_data <- df_data %>%
  mutate(
    late_total = rowSums(across(starts_with("late_")), na.rm = TRUE))
```

Now, we data wrangle the pre-survey:

```
# This code will be similar as the above function. However, the pre and post survey have some
differences that makes it so that it is easier to handle it separate from the function, instea
d of creating a lot of if-else statements to switch between daily and pre-post surveys
df_pre <- read_excel("data/Sleepin' Deacon 2025 Pre-Test Survey_February 24, 2025_12.16.xlsx")
# Using janitor to clean the variable names
df_pre <- df_pre %>%
  clean_names()
# Changing the column names for the ones we will use to join later, to match with what we have
 used other places
df_pre <- df_pre %>%
  rename(first_name = q3_1, last_name = q3_2, email_address = q3_3)
# Adding first and last name to one column
df_pre <- df_pre %>%
  unite("contact info", c("first name", "last name", "email address"), remove = FALSE)
```

```r
# Converting full_name to all lower case
df_pre$contact_info <- df_pre$contact_info %>% str_to_lower()
# Removing the line that has the variable names (usually the first line). Indexing it by the f
act that it will be a string, like Start Date instead of a number for start time
df_pre <- df_pre %>%
  filter(start_date != "Start Date")

# Giving 1 to all responses, to match with data_df later for completion of survey
df_pre <- df_pre %>%
  mutate("pre" = 1)
# Make progress numeric so that we can filter it
df_pre$progress <- as.numeric(df_pre$progress)
# Filtering out the responses with no information, by filtering on progress being less than 10
 (%)
df_pre <- df_pre %>%
  filter(progress > 10)

# Filtering out those responses that are only partly finished
df_temp <- df_pre %>%
  filter(progress < 100)
# Select the rows we will need for df_check, renaming them to match the column names of df_che
ck, and adding another column to designate the survey
df_temp <- df_temp %>%
  select(contact_info, first_name, last_name, email_address) %>%
  mutate(survey = "pre") %>%
  mutate(reason = "unfinished pre-survey")
# Adding the rows to df_check
df_check <- df_check %>%
    add_row(df_temp)

# Filtering on unfinished responses, now that the ones we want to look closer at have been add
ed to df_check
df_pre <- df_pre %>%
  filter(progress == 100)

### 3. Looking for duplicates in df_pre ########################################
# Create data frame with duplicates from contact_info
df_dup <- df_pre %>%
  group_by(contact_info) %>%
  filter(n() > 1) %>%
  ungroup()
# If there are any duplicated rows, add these entries to the df_check for manual checking
if (nrow(df_dup) != 0){
  df_dup <- df_dup %>%
    select(contact_info, first_name, last_name, email_address) %>%
    mutate(survey = "pre") %>%
    mutate(reason = "duplcated")

  # Add these rows to df_check
  df_check <- df_check %>%
    add_row(df_dup)

  # Remove these rows from df_pre so that they aren't added to df_data
  df pre <- df pre %>%
```

```
    filter(!contact_info %in% df_dup$contact_info)
}


### 4. Combining the data frames ###############################################
# Join completion and NA variable from df_1 to df_data by contact_info
  # Use right_join so that I can select only certain columns in df_1 to join
df_data <- df_pre %>%
  select(c("contact_info", "pre")) %>%
  right_join(df_data, join_by(contact_info))
# Anti_join df_pre and df_data to find data entries that didn't match with the registered part
icipants. For example if they misspelled their contact information
df_unmatched <- df_pre %>%
  select(c("contact_info", "first_name", "last_name", "email_address")) %>%
  anti_join(df_data, join_by(contact_info))
# Add survey column to df_unmatch to indicate the day of the survey to look at
df_unmatched <- df_unmatched %>%
  mutate(survey = "pre") %>%
    mutate(reason = "unmatched contact information")
# Add the unmatched responses to df_check for manual walk-through
df_check <- df_check %>%
  add_row(df_unmatched)
```

Data wrangling the post-survey:

```
# This code will be similar as the above function. However, the pre and post survey have some
differences that makes it so that it is easier to handle it separate from the function, instea
d of creating a lot of if-else statements to switch between daily and pre-post surveys
df_post <- read_excel("data/Sleepin' Deacon 2025 Post-Test Survey_February 24, 2025_12.14.xlsx
")
# Using janitor to clean the variable names
df_post <- df_post %>%
  clean_names()
# Changing the column names for the ones we will use to join later, to match with what we have
 used other places
df_post <- df_post %>%
  rename(first_name = q2_1, last_name = q2_2, email_address = q2_3)
# Adding first and last name to one column
df_post <- df_post %>%
  unite("contact_info", c("first_name", "last_name", "email_address"), remove = FALSE)
# Converting full_name to all lower case
df_post$contact_info <- df_post$contact_info %>% str_to_lower()
# Removing the line that has the variable names (usually the first line). Indexing it by the f
act that it will be a string, like Start Date instead of a number for start time
df_post <- df_post %>%
  filter(start_date != "Start Date")

# Giving 1 to all responses, to match with data_df later for completion of survey
df_post <- df_post %>%
  mutate("post" = 1)
# Make progress numeric so that we can filter it
df_post$progress <- as.numeric(df_post$progress)
# Filtering out the responses with no information, by filtering on progress being less than 10
```

```
 (%)
df_post <- df_post %>%
  filter(progress > 10)

# Filtering out those responses that are only partly finished
df_temp <- df_post %>%
  filter(progress < 100)
# Select the rows we will need for df_check, renaming them to match the column names of df_che
ck, and adding another column to designate the survey
df_temp <- df_temp %>%
  select(contact_info, first_name, last_name, email_address) %>%
  mutate(survey = "pre") %>%
    mutate(reason = "unfinished post-survey")
# Adding the rows to df_check
df_check <- df_check %>%
    add_row(df_temp)

# Filtering on unfinished responses, now that the ones we want to look closer at have been add
ed to df_check
df_post <- df_post %>%
  filter(progress == 100)

### 3. Looking for duplicates in df_post ########################################
# Create data frame with duplicates from contact_info
df_dup <- df_post %>%
  group_by(contact_info) %>%
  filter(n() > 1) %>%
  ungroup()
# If there are any duplicated rows, add these entries to the df_check for manual checking
if (nrow(df_dup) != 0){
  df_dup <- df_dup %>%
    select(contact_info, first_name, last_name, email_address) %>%
    mutate(survey = "post") %>%
    mutate(reason = "duplicated")

  # Add these rows to df_check
  df_check <- df_check %>%
    add_row(df_dup)

  # Remove these rows from df_post so that they aren't added to df_data
  df_post <- df_post %>%
    filter(!contact_info %in% df_dup$contact_info)
}


### 4. Combining the data frames ################################################
# Join completion and NA variable from df_post to df_data by contact_info
  # Use right_join so that I can select only certain columns in df_post to join
df_data <- df_post %>%
  select(c("contact_info", "post")) %>%
  right_join(df_data, join_by(contact_info))
# Anti_join df_pre and df_data to find data entries that didn't match with the registered part
icipants. For example if they misspelled their contact information
df_unmatched <- df_post %>%
```

```
  select(c("contact_info", "first_name", "last_name", "email_address")) %>%
  anti_join(df_data, join_by(contact_info))
# Add survey column to df_unmatch to indicate the day of the survey to look at
df_unmatched <- df_unmatched %>%
  mutate(survey = "post") %>%
    mutate(reason = "unmatched contact information")
# Add the unmatched responses to df_check for manual walk-through
df_check <- df_check %>%
  add_row(df_unmatched)
```

Creating a variable designating whether they finished the challenge or not:

```
df_data <- df_data %>%
  mutate(completed_challenge = case_when(
    (day_total >= 12 & pre == 1 & post == 1) ~ "Completed",
    TRUE ~ "Not completed"
  ))
```

Saving the results in df_data to an Excel file:

```
# Reordering the columns so that the Excel file will look better
df_data <- df_data %>%
  select(first_name, last_name, wfu_email_address, completed_challenge, day_total, pre, post,
na_total, late_total, everything())
write_xlsx(df_data, "Sleeping Deacon Challenge Tracking From R Script.xlsx")
```

Saving the results from df_check to an Excel file:

```
write_xlsx(df_check, "Sleeping Deacon Challenge - Responses to go over Manually.xlsx")
```

Error checks that have been done

- Checking that the code successfully catches duplicates in df_data and removes them (26/02/2025)
- Checking that the code successfully catches duplicates in df_1 and removes them if all the information(except for the time stamp) is the same or adds them to df_check if the other information differs (26/02/2025)
- Checked that day_1, na_1, and late_1 works correctly (26/02/2025)

Checks

- Flag surveys with one or more missing values and count them to make the user aware of participants that might have skipped a lot of questions
- Check for people signing up twice. If email and full name is the same, delete one row in reg_df
- Check for people with the same name, but different email addresses. Flag them for manual processing?
- The join goes weird if there are two participants with similar information in df_1. Maybe check for duplicates in either email or name in both df_data and df_1 and add these two a list, indicating the number of survey and that they have to be checked manually. Then delete the rows that have duplicates from the automatic calculations and conversions?

Questions

- Does it matter whether they handed in the survey after the due data?

Be aware

- If the total completed number seems unreasonable, look into it, something could have gone wring with the code