Student grades case study

STA303/1002 Week 2

2021-01-18

Getting Started

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter* (Mac) or *Ctrl+Shift+Enter*.

```
print("Hello world!")
```

```
## [1] "Hello world!"
```

```
x <- 2 + 2
x
```

```
## [1] 4
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

Markdown and R Markdown

Markdown is a document processor, newer and simpler than LaTeX. It is a lightweight markup language with plain text formatting syntax (i.e. you don't have to click lots of formatting options like in Word, you just write symbols). A potential issue is that it can be hard to debug errors in your .md file.

R Markdown lets you integrate R code and Markdown to create reports with commentary, code, outputs and graphics all in one place with no need to copy and paste things in to a Word doc.

Making a .pdf

When you press *Knit* a .pdf report is written. When you do assignments that is what you will want to submit along with your .Rmd file.

A few R Markdown tips

1. There is a small button at the top left of your source area of R Studio that when pressed will show an outline of the document, based on headings in the file. A heading has the hash sign (you might call it a number sign) in front of it. One hash sign is the biggest heading, followed by two hash, and three hash.
2. A range of keyboard shortcuts can be found here.
3. On the top menu bar, go to Help > Cheatsheets > R Markdown Cheat Sheet to open the main R Markdown cheatsheet.
4. On the top menu bar, go to Help > Markdown Quick Reference to open the *Markdown Quick Reference* in your help pane.

Note: You can use `\newpage` in R Markdown to create a page break, which will be helpful when you're preparing assessments for Crowdmark and want questions to be on separate pages.

Case study

The premise

Suppose you're in grad school at your dream university and are the Head TA for a course called STA101. At the end of semester, the instructor gets sick and needs to have major surgery and so will not be able to do the final admin for the course. The department Chair asks for your help in preparing the final grades. They tell you they'll check over everything for you before the marks are submitted, but you want to make sure you do it right and provide clear and reproducible code so you can impress the Chair.

Thankfully, all the marking is done, and the prof has left you the following files and instructions.

```
list.files("student_data/")
```

```
## [1] "STA101_2-perc-survey.csv"
## [2] "STA101_assessment-scheme-from-syllabus.RDS"
## [3] "STA101_course-roster.csv"
## [4] "STA101_gradebook-from-LMS.xlsx"
```

Your prof's message

Hi,

Thanks for helping out with this, I owe you a 100 coffees next semester when I'm back and recovered.

"STA101_gradebook-from-LMS.xlsx" contains the grades from our learning management system. It covers everything except for the 2% survey we did in Microsoft Forms. For the survey, you have "STA101_2-perc-survey.csv", which has the list of emails of the students that completed the survey. "STA101_course-roster.csv" has each student's name, student ID number, email and tutorial group, it should help you join up the data.

You can find it in the syllabus, but also have given you "STA101_assessment-scheme-from-syllabus.RDS" that has how much each assessment (or set of assessments) is weighted for the final mark. It also has how many points the assessment was out of.

I've already dealt with all the accommodations, so an `NA` in the data from the learning management system means the student didn't submit the assessment and didn't get an accommodation, i.e., they get a 0 on that task.

Also, so many students who are close to a GPA threshold will email once final marks are out asking to be bumped up. I'd never want to be unfair and give a student an advantage just because they asked, where another student in the same situation wouldn't get the advantage because they were too polite to ask. So, I was planning to use the below function to adjust everyone who was 1 percentage point below the threshold up to it. If the Chair asks you to help manage the emails afterwards, tell anyone who asks that the course policy is to make no ad hoc adjustments for individual students as that wouldn't be fair.

```
adjust_GPA_thresh <- function(x){
  thresh <- c(89, 84, 79, 76, 72, 69, 66, 62, 59, 56, 52, 49)
  ifelse(round(x)  %in% thresh, round(x)+1, round(x))
}
```

For submitting the marks, you'll need one file per tutorial group, with just their student id and final grade (rounded to the nearest whole integer). Don't have column titles, it'll mess with the system.

Please also provide the undergraduate administrator a graph of the distribution of the final grades, and if you can, indicate tutorial group too.

Good luck! Your grateful prof

Let's do it

Read in all the data and take a look. There are three different file types used: .xlsx, .csv and .RDS. A file ending .RDS will be a single saved R object, like a data frame.

We can read RDS with `readRDS()` from base R. `read_csv()` from the `readr` package in `tidyverse` (which helps us read rectangular data like .csv and.tsv files) will load the .csv files for us (you can use `read.csv()` too, but I like the behaviour of this one better). Finally, we'll need `read_excel()` from the `readxl` package to read the .xlsx file. All the packages have been installed and loaded for you in the `setup` chunk above.

```
# Read in the grades from the LMS as an object called 'gradebook_raw' using read_excel
gradebook_raw <- read_excel("student_data/STA101_gradebook-from-LMS.xlsx", sheet = 1)

# Read in emails of the students who completed the 2% survey as 'survey_raw'
survey_raw <- read_csv("student_data/STA101_2-perc-survey.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   email = col_character()
## )
```

```
# Read in the course roster as 'roster'
roster <- read_csv("student_data/STA101_course-roster.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   name = col_character(),
##   id = col_double(),
##   email = col_character(),
##   tutorial_group = col_character()
## )
```

```
# Read in the assessment weighting scheme as 'weight'
weight <- readRDS("student_data/STA101_assessment-scheme-from-syllabus.RDS")
```

Explore the data sets using: `glimpse()`, `str()`, `head()` and `View()`.

Check over the files and see if they are what you expect.

You can use the `glimpse()` and/or `head()` and/or `str()` functions to check our your data. If the data isn't too big, I often like to use `View()` to see it in a spreadsheet-like form. This can be done point-and-click way by clicking the name of the object in the Environment pane.

I've also shown some code to very quickly get the names of all the variables in each dataset as an overview, and open each of these files in View mode to take a quick look. That's just for fun, not assessed.

```
# glimpse lets us take a quick look at the data
glimpse(gradebook_raw)
```

```
## Rows: 638
## Columns: 13
## $ `Student Name`                          <chr> "Diab, Amr", "Spears, Britn...
```

```
## $ ID                                    <dbl> 1004211591, 1004407102, 100...
## $ tutorial_group                        <chr> "TUT0301", "TUT0101", "TUT0...
## $ `Problem set 1 (35601)`               <dbl> 7.3, 9.3, 2.9, 4.6, NA, 9.8...
## $ `Problem set 2 (35602)`               <dbl> 5.2, 5.0, 4.3, 4.5, 7.2, 8....
## $ `Problem set 3 (35603)`               <dbl> 8.1, 7.7, 8.0, 8.4, 8.4, 8....
## $ `Problem set 4 (35604)`               <dbl> 6.3, 6.3, 7.2, 8.1, 7.5, 7....
## $ `Problem set 5 (35605)`               <dbl> 7.0, 6.7, 7.1, 6.1, 6.6, 6....
## $ `Assignment 1 (35611)`                <dbl> 37.6, 34.8, 33.1, 32.0, 34....
## $ `Assignment 2 (35612)`                <dbl> 32.6, 34.0, 30.6, 38.6, 30....
## $ `Assignment 3 (35613)`                <dbl> 24.5, 24.6, 42.7, 19.5, 40....
## $ `Professional development task (35621)` <dbl> 14.7, 12.8, 14.2, 16.6, 11....
## $ `Final project (35622)`               <dbl> 5.9, 2.0, 0.0, 17.3, 23.4, ...
```

```
#head lets us look at the top few rows. Can you guess waht tail() does?
head(survey_raw, n = 10)
```

```
## # A tibble: 10 x 1
##    email
##    <chr>
##  1 amr.diab@myuni.edu
##  2 britney.spears@myuni.edu
##  3 benny.andersson@myuni.edu
##  4 bing.crosby@myuni.edu
##  5 buddy.holly@myuni.edu
##  6 jesse.mccartney@myuni.edu
##  7 cyndi.lauper@myuni.edu
##  8 bo.diddley@myuni.edu
##  9 brad.delson@myuni.edu
## 10 miles.davis@myuni.edu
```

```
# str is an even more sophicticated glimpse, that is quite good if
## you know there are some complicated structures with your data. Out of scope for us.
str(roster)
```

```
## tibble [638 x 4] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ name          : chr [1:638] "Diab, Amr" "Spears, Britney" "Andersson, Benny" "Crosby, Bing" ...
##  $ id            : num [1:638] 1.00e+09 1.00e+09 1.00e+09 1.01e+09 1.01e+09 ...
##  $ email         : chr [1:638] "amr.diab@myuni.edu" "britney.spears@myuni.edu" "benny.andersson@myuni
##  $ tutorial_group: chr [1:638] "TUT0301" "TUT0101" "TUT0401" "TUT0101" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   name = col_character(),
##   ..   id = col_double(),
##   ..   email = col_character(),
##   ..   tutorial_group = col_character()
##   .. )
```

```
# View will open a new tab with a spreadsheet like view
# View(weight)
```

Use `janitor` to clean our data

`janitor` is a great package with lots of convenience functions for cleaning up data for use in analysis. The one I use the most is `clean_names()` which will make all the names consistent, specifically, unique, lowercase, spaces replaced with underscores, and special characters simplified or removed.

```
gradebook_clean <- gradebook_raw %>%
  janitor::clean_names()
```

Replace NAs with 0s

There are lots of places you could start with this, but I want to show you this neat trick. `mutate()` as you should know from this week's readings, let's you create a new variable. `mutate_if()` will apply your instructions to every single column that meets your criteria. Looking at the gradebook data, we can see the numeric columns are all our grade columns + the student IDs.

```
# checking which columns are numeric with glimpse
glimpse(gradebook_clean)
```

```
## Rows: 638
## Columns: 13
## $ student_name                          <chr> "Diab, Amr", "Spears, Britney",...
## $ id                                    <dbl> 1004211591, 1004407102, 1004047...
## $ tutorial_group                        <chr> "TUT0301", "TUT0101", "TUT0401"...
## $ problem_set_1_35601                   <dbl> 7.3, 9.3, 2.9, 4.6, NA, 9.8, 4....
## $ problem_set_2_35602                   <dbl> 5.2, 5.0, 4.3, 4.5, 7.2, 8.8, 8...
## $ problem_set_3_35603                   <dbl> 8.1, 7.7, 8.0, 8.4, 8.4, 8.1, 8...
## $ problem_set_4_35604                   <dbl> 6.3, 6.3, 7.2, 8.1, 7.5, 7.4, 6...
## $ problem_set_5_35605                   <dbl> 7.0, 6.7, 7.1, 6.1, 6.6, 6.3, 6...
## $ assignment_1_35611                    <dbl> 37.6, 34.8, 33.1, 32.0, 34.8, 3...
## $ assignment_2_35612                    <dbl> 32.6, 34.0, 30.6, 38.6, 30.6, 3...
## $ assignment_3_35613                    <dbl> 24.5, 24.6, 42.7, 19.5, 40.8, 3...
## $ professional_development_task_35621 <dbl> 14.7, 12.8, 14.2, 16.6, 11.9, 1...
## $ final_project_35622                   <dbl> 5.9, 2.0, 0.0, 17.3, 23.4, 19.1...
```

Let's quickly check if any IDs are NA, because if not, we can just say 'apply to all numeric columns' without worry.

```
# quick check no NA ids
gradebook_clean %>%
  filter(is.na(id))
```

```
## # A tibble: 0 x 13
## # ... with 13 variables: student_name <chr>, id <dbl>, tutorial_group <chr>,
## #   problem_set_1_35601 <dbl>, problem_set_2_35602 <dbl>,
## #   problem_set_3_35603 <dbl>, problem_set_4_35604 <dbl>,
## #   problem_set_5_35605 <dbl>, assignment_1_35611 <dbl>,
## #   assignment_2_35612 <dbl>, assignment_3_35613 <dbl>,
## #   professional_development_task_35621 <dbl>, final_project_35622 <dbl>
```

Use mutate_if to replace missing values in all numeric columns with 0.

```
gradebook <- gradebook_clean %>%
  mutate_if(is.numeric, replace_na, replace = 0)
```

Add the survey to the gradebook

Before you can get to calculating final grades, you'll need make sure the 2% survey is reflected in your gradebook. But for the survey all we have is email, and you don't have email in the gradebook, so you need to use the roster as an intermediary step.

(Employers have told me that they wish students knew more about database joins AND how to check if the behaviour of the join is what they want.

There are 4 types of joins in dplyr. Run `?join` to see the information about them, or search in the help pane.)

```
survey_link <- roster %>%
  full_join(survey_raw, by = "email")

# Can check number of rows by looking at Environment pane or
# checking with nrow(), e.g. nrow(roster)
nrow(survey_link)
```

Check one: do a full join and see if the numbers are right

```
## [1] 658
```

```
nrow(roster)
```

```
## [1] 638
```

```
nrow(gradebook)
```

```
## [1] 638
```

There is a problem! We have gained 20 extra students somehow...

```
# distinct() returns a tibble with only the distinct rows from your original
distinct(survey_raw)
```

Check two: are there duplicates?

```
## # A tibble: 526 x 1
##    email
##    <chr>
##  1 amr.diab@myuni.edu
```

```
##  2 britney.spears@myuni.edu
##  3 benny.andersson@myuni.edu
##  4 bing.crosby@myuni.edu
##  5 buddy.holly@myuni.edu
##  6 jesse.mccartney@myuni.edu
##  7 cyndi.lauper@myuni.edu
##  8 bo.diddley@myuni.edu
##  9 brad.delson@myuni.edu
## 10 miles.davis@myuni.edu
## # ... with 516 more rows
```

```
distinct(survey_link)
```

```
## # A tibble: 658 x 4
##    name                    id email                     tutorial_group
##    <chr>                <dbl> <chr>                     <chr>
##  1 Diab, Amr        1004211591 amr.diab@myuni.edu        TUT0301
##  2 Spears, Britney  1004407102 britney.spears@myuni.edu  TUT0101
##  3 Andersson, Benny 1004047884 benny.andersson@myuni.edu TUT0401
##  4 Crosby, Bing     1006974106 bing.crosby@myuni.edu     TUT0101
##  5 Holly, Buddy     1007854143 buddy.holly@myuni.edu     TUT0101
##  6 McCartney, Jesse 1006143715 jesse.mccartney@myuni.edu TUT0301
##  7 Lauper, Cyndi    1002595647 cyndi.lauper@myuni.edu    TUT0401
##  8 Diddley, Bo      1001598061 bo.diddley@myuni.edu      TUT0301
##  9 Delson, Brad     1005655912 brad.delson@myuni.edu     TUT0101
## 10 Davis, Miles     1006748315 miles.davis@myuni.edu     TUT0201
## # ... with 648 more rows
```

```
distinct(roster)
```

```
## # A tibble: 638 x 4
##    name                    id email                     tutorial_group
##    <chr>                <dbl> <chr>                     <chr>
##  1 Diab, Amr        1004211591 amr.diab@myuni.edu        TUT0301
##  2 Spears, Britney  1004407102 britney.spears@myuni.edu  TUT0101
##  3 Andersson, Benny 1004047884 benny.andersson@myuni.edu TUT0401
##  4 Crosby, Bing     1006974106 bing.crosby@myuni.edu     TUT0101
##  5 Holly, Buddy     1007854143 buddy.holly@myuni.edu     TUT0101
##  6 McCartney, Jesse 1006143715 jesse.mccartney@myuni.edu TUT0301
##  7 Lauper, Cyndi    1002595647 cyndi.lauper@myuni.edu    TUT0401
##  8 Diddley, Bo      1001598061 bo.diddley@myuni.edu      TUT0301
##  9 Delson, Brad     1005655912 brad.delson@myuni.edu     TUT0101
## 10 Davis, Miles     1006748315 miles.davis@myuni.edu     TUT0201
## # ... with 628 more rows
```

Nope, no straight duplicates.

```
# Filter to only rows where name is missing or email is missing
problems <- survey_link %>%
  filter(is.na(name) | is.na(email))
```

Check three: use missings to help guide us We can look at this data and compare it to how the emails are supposed to look, an notice that the problem is because there is a missing full stop in several student emails.

Fix typos The `stringr` package is part of the `tidyverse` and provides lots of options for working with character strings.

`str_replace()` allows to replace parts of strings that match a certain pattern with another pattern.

```
survey <- survey_raw %>%
  mutate(email = str_replace(email, "myuniedu", "myuni.edu"))

#survey_link <- roster %>%
#  full_join(survey, by = "email")

#survey_full <- survey %>%
#  left_join(roster, by="email")

survey_full <- roster %>%
  right_join(survey)
```

```
## Joining, by = "email"
```

Our next step is to link this to the gradebook, but let's make our life easy by creating a dummy variable indicating if the student completed the survey. It is '1' for all the students in this survey_full data.

```
survey_final <- survey_full %>%
  mutate(survey = 1)
```

Merge on to gradebook and replace missing values for `survey`, with a 0.

```
gradebook_wsurv <- gradebook %>%
  full_join(survey_final) %>%
  mutate(survey = replace_na(survey, 0)) %>%
  select(-name, -email)
```

```
## Joining, by = c("id", "tutorial_group")
```

Is the gradebook tidy?

No!*

*Well, no for our purpose of grade calculation.

How we record and store data is often not the best way to actually work with data programmatically. The gradebook as it is currently works if we want students to be the observational unit (one student per row, one observation per row, check), but to do our calculations most easily, we actually want to have **each row represent a particular assessment for a particular student**. In that conception of this data, we actually have two variables: 'assessment' and 'grade' that are currently spread across multiple columns. Our 'one column per variable' rule is broken, if we're thinking about our data this way.

Let's fix that!

```
gradebook_long <- gradebook_wsurv %>%
  pivot_longer(-c(student_name, id, tutorial_group),
               names_to = "assessment", values_to = "grade")
```

Weight the assessments

`str_detect()` will return a TRUE or FALSE depending on if the pattern specified is detected.

`str_to_sentence()` changes a string to 'sentence case', which starts with a capitalized letter and the rest of the letters are lowercase.

`str_replace_all()` replaces all instances of the specified patter, while `str_replace()` only replaces the first instance.

`str_remove()` is a shortcut for `str_replace()` where the replacement is 'nothing', i.e., "".

```
grades_weighted <- gradebook_long %>%
  mutate(assessment_type = case_when(
    str_detect(assessment, "problem") ~ "Problem sets",
    assessment == "survey" ~ "2 percent survey",
    TRUE ~ str_to_sentence(str_replace_all(assessment, "_", " "))
  )) %>%
  mutate(assessment_type = str_remove(assessment_type, "\\s\\d*$")) %>%
  left_join(weight, by = c("assessment_type" = "assessment")) %>%
  group_by(student_name, id, tutorial_group, assessment_type, percentage) %>%
  summarise(grade_by_type = mean(grade/out_of), .groups = "drop") %>%
  group_by(student_name, id, tutorial_group) %>%
  summarise(total = sum(percentage*grade_by_type), .groups = "drop")
```

```
# do a by hand check
# this is for Tom DeLonge
7.2/10*4 + 6.2/10*4 +7.6/10*4+7.2/10*4+7.4/10*4+37.1/50*10+42.2/50*10 + 39.6/50*15 + 49.8/50*40 +17.4/2
```

```
## [1] 86.43
```

Load the adjust_GPA_thresh() function we've been given and apply it

```
adjust_GPA_thresh <- function(x){
  thresh <- c(89, 84, 79, 76, 72, 69, 66, 62, 59, 56, 52, 49)
  ifelse(round(x)  %in% thresh, round(x)+1, round(x))
}

grades_adjusted <- grades_weighted %>%
  mutate(grade_adj = adjust_GPA_thresh(total))
```

(I'm not going to assess you on `group_walk()`, but once again, something that I think is useful and interesting.)

`str_c()` is like `paste()`, if you've used that, and it concatenates strings, meaning you can combine text strings with values from variables and other saved objects. It is also works on vectors, element-wise.

```
grades_adjusted %>%
  select(id, grade_adj, tutorial_group) %>%
  group_by(tutorial_group) %>%
  group_walk(~ write_csv(.x, str_c("student_data/STA101_", .y$tutorial_group, "_grades-for-submission.cs
```

Visualize

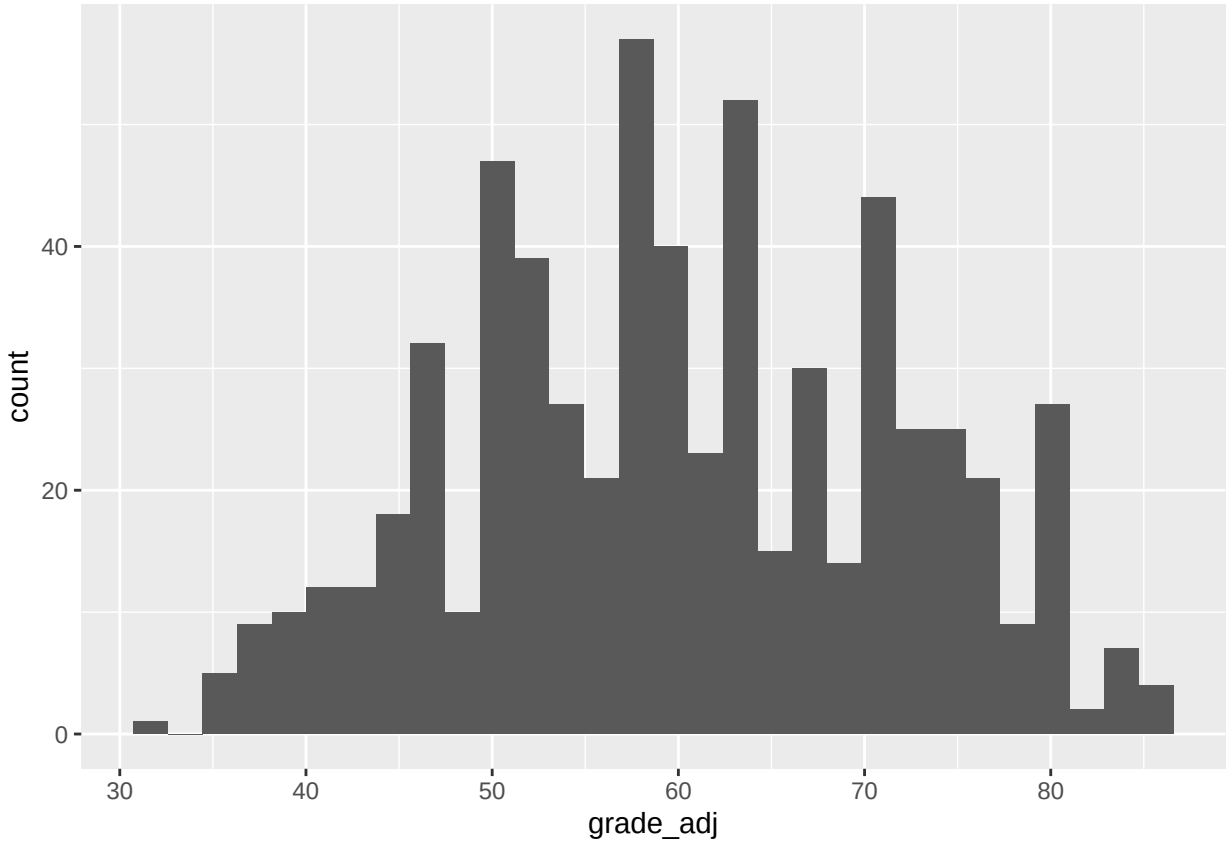
You've been asked to show the distribution of grades, and indicate tutorial group in this visualization. There are a couple things your could try.

```
p <- grades_adjusted %>%
  ggplot(aes(x = grade_adj)) +
  geom_histogram()

p
```

Stacked histogram

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
grades_adjusted %>%
  count(tutorial_group, grade_adj) %>%
  ggplot(aes(x = grade_adj, y = n, fill = tutorial_group)) +
  geom_bar(stat = "identity") +
  scale_x_continuous(limits = c(0, 100)) +
  theme_minimal() +
  labs(title = "STA101 grade distribution by tutorial group",
       subtitle = "Winter 2023",
       x = "Grade (percentage)",
       y = "Number of students",
       caption = str_c("Prepared by the Head TA\n", Sys.Date())) +
  theme(legend.title = element_blank(), legend.position = c(0.15, 0.75)) +
  scale_y_continuous(limits = c(0, 45), breaks = seq(0, 45, by = 5)) +
  scale_fill_brewer(palette = "Dark2")
```

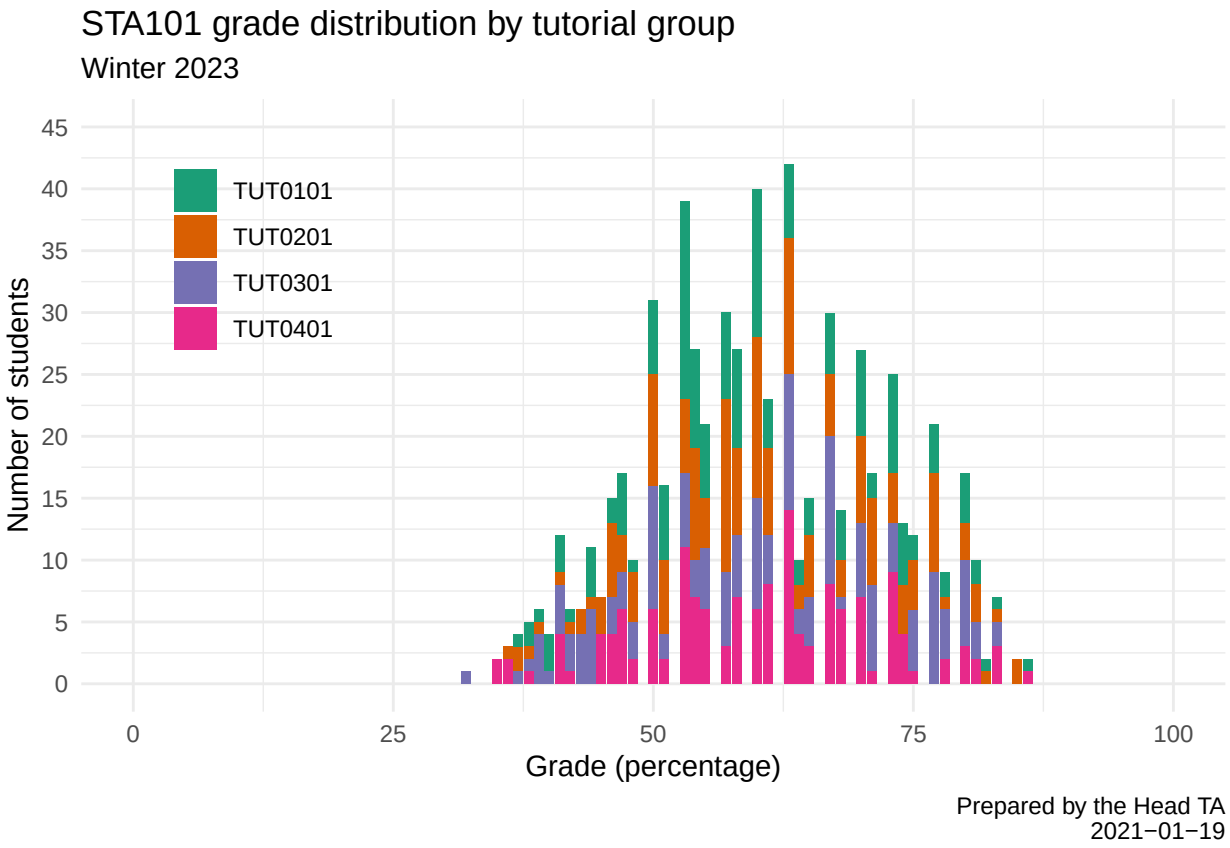


Figure 1: Descriptive caption

```r
# get the mean, median, min, and max scores and n for each tutorial group
```

Check if it seems like the average grades for the tutorial groups are the same. Use what we've learned about ANOVA.

$$H_0 : \mu_{TUT0101} = \mu_{TUT0201} = \mu_{TUT0301} = \mu_{TUT0401}$$

$$H_1 : \text{at least one mean is different}$$

```
# fit lm
summary(lm(grade_adj ~ tutorial_group, data = grades_adjusted))
```

```
##
## Call:
## lm(formula = grade_adj ~ tutorial_group, data = grades_adjusted)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.472  -7.506  -0.472   9.494  26.181
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            59.8194     0.9316  64.208   <2e-16 ***
## tutorial_groupTUT0201   0.6865     1.2846   0.534    0.593
## tutorial_groupTUT0301   0.6527     1.3052   0.500    0.617
## tutorial_groupTUT0401   0.2140     1.3285   0.161    0.872
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.6 on 634 degrees of freedom
## Multiple R-squared:  0.0006372,  Adjusted R-squared:  -0.004092
## F-statistic: 0.1347 on 3 and 634 DF,  p-value: 0.9393
```

This large p-value means we have no evidence against the claim that the tutorial groups all had the same average final grade.

```
grades_adjusted %>%
  count(tutorial_group, grade_adj) %>%
  ggplot(aes(x = grade_adj, y = n, fill = tutorial_group)) +
  geom_bar(stat="identity") +
  scale_x_continuous(limits = c(0, 100)) +
  facet_wrap(~tutorial_group) +
  theme_minimal() +
  theme(legend.position = "none")
```

Faceted bar chart

```
grades_adjusted %>%
  ggplot(aes(x = grade_adj, colour = tutorial_group)) +
  geom_boxplot() +
  scale_x_continuous(limits = c(0, 100)) +
```

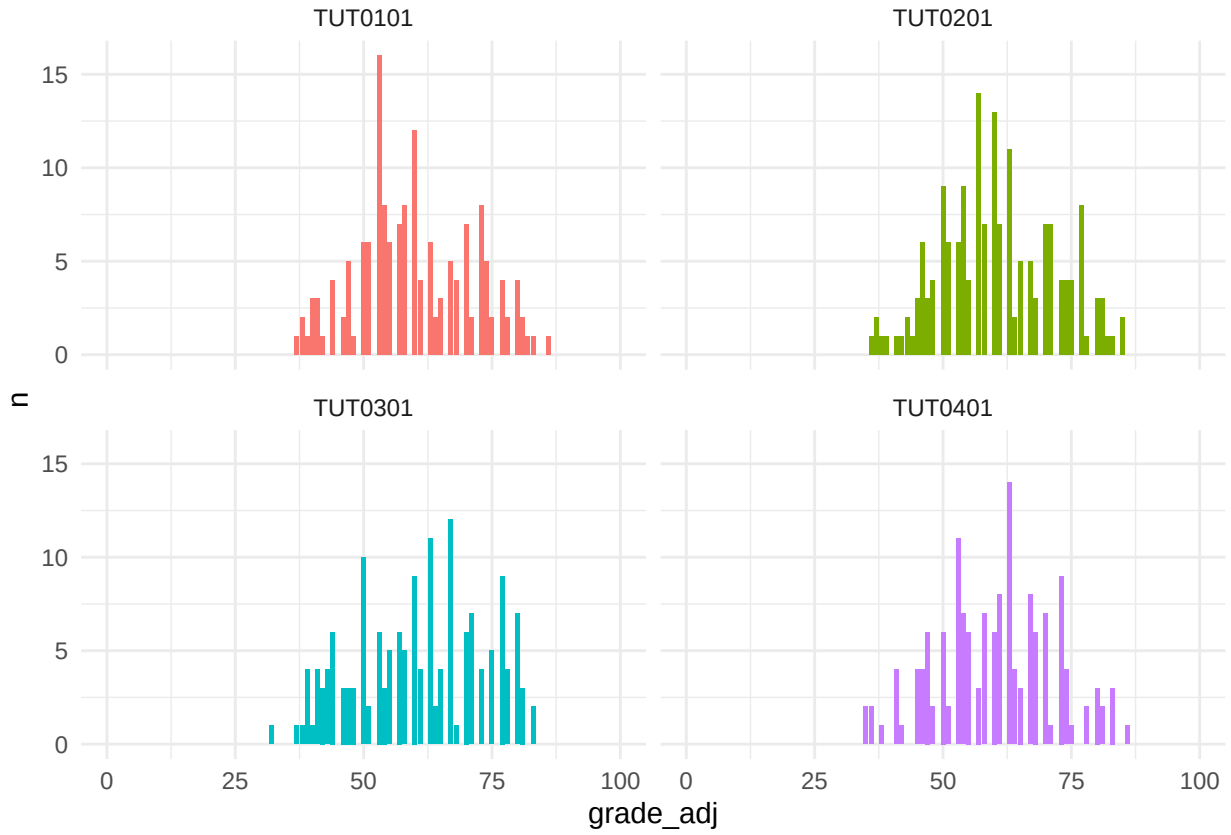
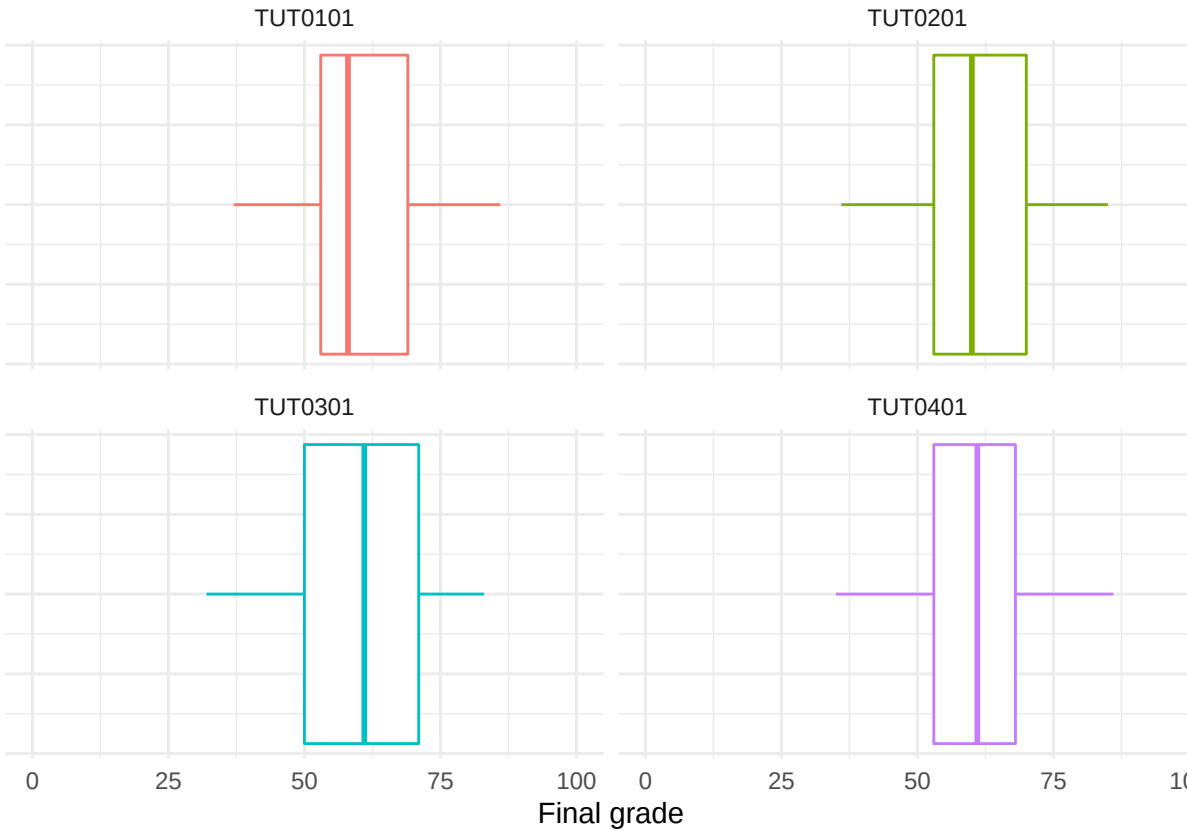


Figure 2: Distribution of final grades for STA101

```
  facet_wrap(~tutorial_group) +
  theme_minimal() +
  theme(legend.position = "none", axis.text.y = element_blank()) +
  labs(x = "Final grade")
```



Faceted boxplot

Forcats (Functions from the `forcats` package to help us manipulate categorical variables, which in turn can help us change and improve the presentation of our plots. In these examples I'll show you the two functions I use the most, `fct_rev()` and `fct_relevel()`.)

Reversing the order of the levels in a categorical variable.

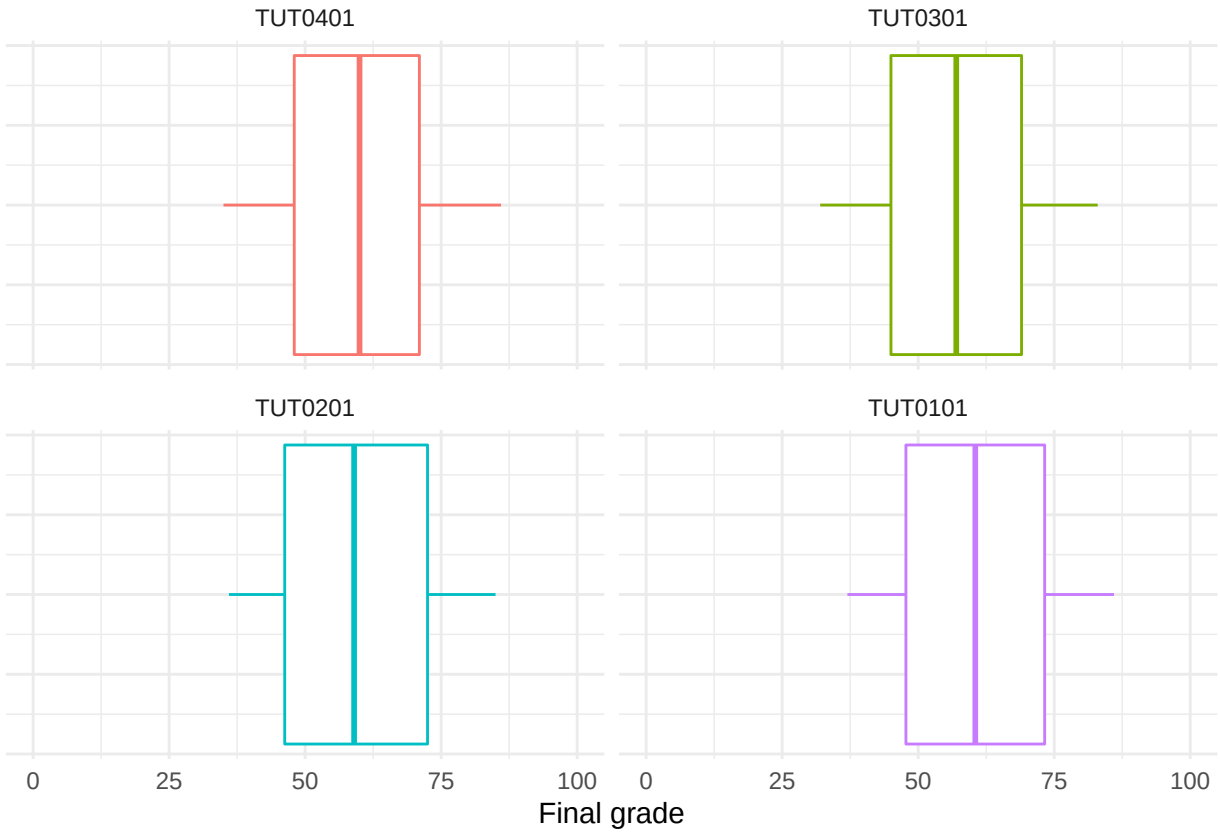
```
reordered <- grades_adjusted %>%
  count(tutorial_group, grade_adj) %>%
  mutate(tutorial_group = fct_rev(tutorial_group))

table(reordered$tutorial_group)
```

```
##
## TUT0401 TUT0301 TUT0201 TUT0101
##      33      35      38      36
```

```
reordered %>%
  ggplot(aes(x = grade_adj, colour = tutorial_group)) +
  geom_boxplot() +
```

```
  scale_x_continuous(limits = c(0, 100)) +
  facet_wrap(~tutorial_group) +
  theme_minimal() +
  theme(legend.position = "none",
        axis.text.y = element_blank()) +
  labs(x = "Final grade")
```



Changing the order of levels in a categorical article.

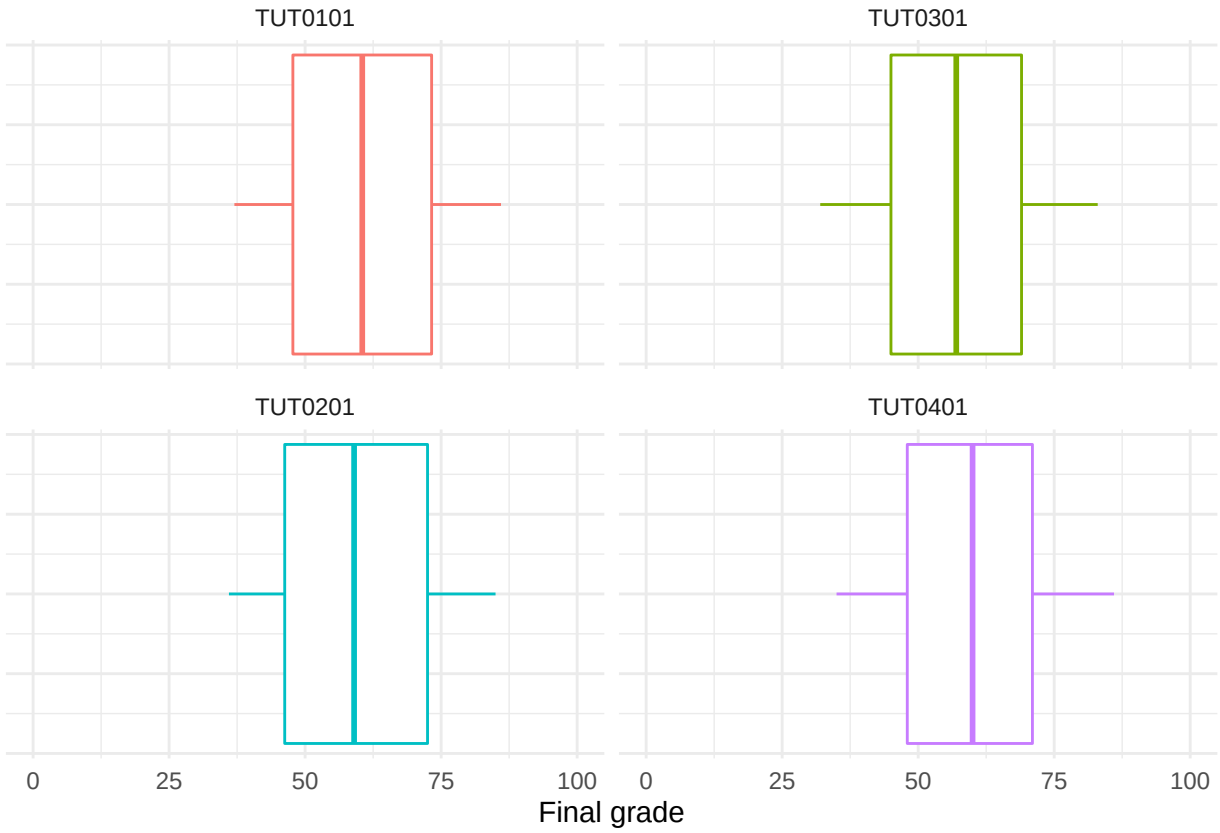
```
reordered <- grades_adjusted %>%
  count(tutorial_group, grade_adj) %>%
  mutate(tutorial_group = fct_relevel(tutorial_group, "TUT0201", after = 2))

table(reordered$tutorial_group)
```

```
##
## TUT0101 TUT0301 TUT0201 TUT0401
##      36      35      38      33
```

```
reordered %>%
  ggplot(aes(x = grade_adj, colour = tutorial_group)) +
  geom_boxplot() +
  scale_x_continuous(limits = c(0, 100)) +
  facet_wrap(~tutorial_group) +
  theme_minimal() +
```

```
  theme(legend.position = "none",
        axis.text.y = element_blank()) +
  labs(x = "Final grade")
```

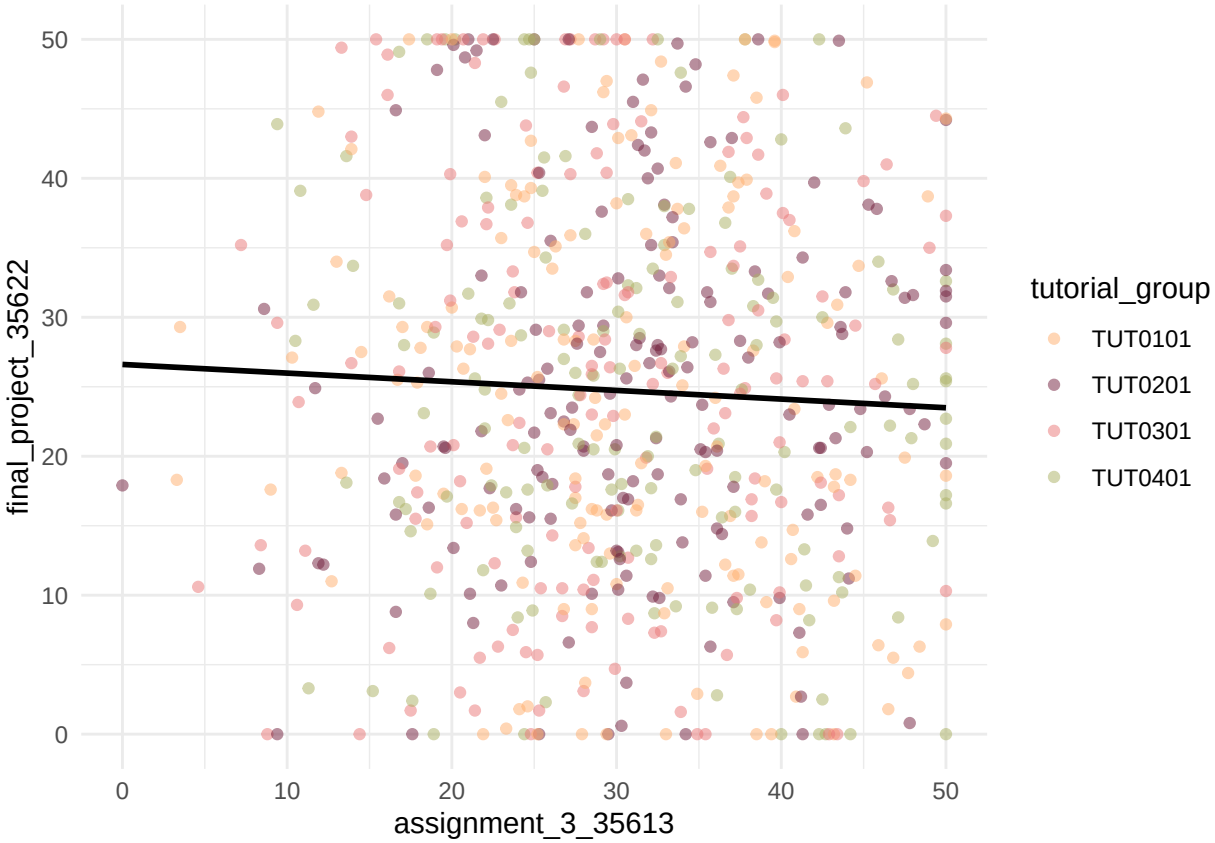


Whip up another quick graph to check the relationship between two of the assessments.

Make a scatter plot with the following requirements:
* use the version of the gradebook with each row representing a student
* put assignment 3 grades on the x-axis
* put final project grades on the y-axis
* set point transparency to 0.5
* colour points by tutorial group (use these hex codes: “#ffaf6d”, “#721f3c”, “#ea7777”, “#abb062”)
* add an overall smooth that is a straight line, the line should be black
* use theme_minimal()

```
gradebook_wsurv %>%
  ggplot(aes(x = assignment_3_35613, y = final_project_35622)) +
  geom_point(alpha = 0.5, aes(colour = tutorial_group)) +
  geom_smooth(method = "lm", se = FALSE, color = "black") +
  theme_minimal() +
  scale_colour_manual(values = c("#ffaf6d", "#721f3c", "#ea7777", "#abb062"))
```

```
## `geom_smooth()` using formula 'y ~ x'
```


final_project_35622
assignment_3_35613
0
10
20
30
40
50
tutorial_group
TUT0101
TUT0201
TUT0301
TUT0401

Credits

- 'Student' names come from: https://dataverse.harvard.edu/file.xhtml?persistentId=doi:10.7910/DVN/28201/VEG34D&version=1.0
- Simulated grades created using this little bounded rnorm variation: https://stackoverflow.com/questions/19343133/setting-upper-and-lower-limits-in-rnorm