Final_Sanyika

2023-12-07

Python version: https://colab.research.google.com/drive/1H4mIVk9aYLchsxtm73B4AjsVX8lzIpzn?usp=sharing

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
setwd("~/Desktop")
math <- read.csv("student_math_clean.csv")
lang <- read.csv("student_portuguese_clean.csv")
```

QUESTION 1 (a) Using the Portugese language data set for the question. Consider students only if they are at least 18 years or older. For these students, they each received a grade 1 and a grade 2. Create a variable for each student that is the larger of these two scores. Then find the average score of this maximum grouping by the combination of mother job and father job. Of these combinations (i.e. mother job and father job) with more than 5 cases, what combination of jobs had the highest average of the larger of their two test scores?

```r
#see column names in the lang data set
colnames(lang)
```

```
##  [1] "student_id"           "school"               "sex"
##  [4] "age"                  "address_type"         "family_size"
##  [7] "parent_status"        "mother_education"     "father_education"
## [10] "mother_job"           "father_job"           "school_choice_reason"
## [13] "guardian"             "travel_time"          "study_time"
## [16] "class_failures"       "school_support"       "family_support"
## [19] "extra_paid_classes"   "activities"           "nursery_school"
## [22] "higher_ed"            "internet_access"      "romantic_relationship"
## [25] "family_relationship"  "free_time"            "social"
## [28] "weekday_alcohol"      "weekend_alcohol"      "health"
## [31] "absences"             "grade_1"              "grade_2"
## [34] "final_grade"
```

```r
#filter for age > 18
lang %>% filter(age >= 18) %>%
  #add variable that calculates the max between 2 grades
  mutate(max_grade = pmax(grade_1, grade_2)) %>%
  group_by(father_job, mother_job) %>%
  #calculate ave score and count of each group
  summarize(ave_score = mean(max_grade), cases = n()) %>%
  #filter cases>5 and select top average score
  filter(cases>5) %>% arrange(-ave_score) %>% head(1)
```

```
## `summarise()` has grouped output by 'father_job'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 1 × 4
## # Groups:   father_job [1]
##   father_job mother_job ave_score cases
##   <chr>      <chr>          <dbl> <int>
## 1 services   teacher         13.4     7
```

b. Using the math data set, find the mean age by sex and school. Then compute the mean final score by sex and school. Then present the results in a table where each row is a school and each column is an average of the particular variable for one sex (i.e. the first column is mean age females, the second column is mean age males, etc.).

```r
#see names of columns
colnames(math)
```

```
##  [1] "student_id"           "school"               "sex"
##  [4] "age"                  "address_type"         "family_size"
##  [7] "parent_status"        "mother_education"     "father_education"
## [10] "mother_job"           "father_job"           "school_choice_reason"
## [13] "guardian"             "travel_time"          "study_time"
## [16] "class_failures"       "school_support"       "family_support"
## [19] "extra_paid_classes"   "activities"           "nursery_school"
## [22] "higher_ed"            "internet_access"      "romantic_relationship"
## [25] "family_relationship"  "free_time"            "social"
## [28] "weekday_alcohol"      "weekend_alcohol"      "health"
## [31] "absences"             "grade_1"              "grade_2"
## [34] "final_grade"
```

```r
#calculate average age by sex and school
age_mean <- math %>% group_by(sex, school) %>% summarize(mean_age = mean(age))
```

```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups`
## argument.
```

```r
#calculate average final grade by sex and school
score_mean <- math %>% group_by(sex, school) %>% summarize(mean_final = mean(final_gra
de))
```

```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups`
## argument.
```

```r
#join by sex and school and pivot wider
score_mean %>% left_join(age_mean, by = c("sex", "school")) %>%
  pivot_wider(names_from = "sex", values_from = c("mean_age", "mean_final"))
```

```
## # A tibble: 2 × 5
##   school mean_age_F mean_age_M mean_final_F mean_final_M
##   <chr>       <dbl>      <dbl>        <dbl>        <dbl>
## 1 GP           16.6       16.5         9.97        11.1
## 2 MS           17.8       18.2         9.92         9.76
```

c. Reproduce the provided plot.

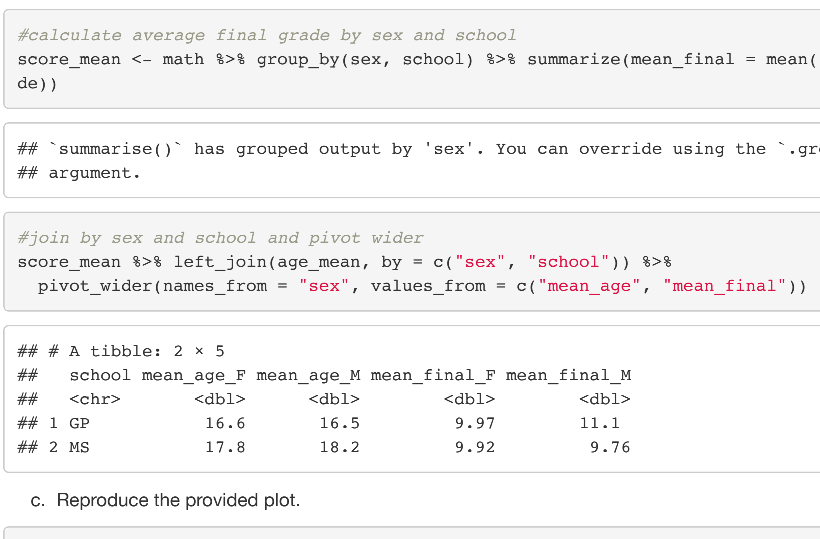
```r
library(ggplot2)

#select relevant columns and add column to state the subject
lang_timeGrade <- lang %>% select(sex, study_time, final_grade) %>%
  mutate(subject = "port")
math_timeGrade <- math %>% select(sex, study_time, final_grade) %>%
  mutate(subject = "math")

#stack the two data frames on top of each other
school <- rbind(lang_timeGrade, math_timeGrade)

#set the order of study_time for it to display on the x axis
order_time <- c("<2 hours", "2 to 5 hours", "5 to 10 hours", ">10 hours")
school$study_time <- factor(school$study_time, levels = order_time)

#create plot with final_grade as a response to study_time and stratify by sex
ggplot(data = school,
       aes(
         x = study_time,
         y = final_grade,
         color = sex
       )
) +
  #create box plot faceted by subject
  geom_boxplot() +
  facet_wrap( ~ subject) +
  xlab("Study Time") +
  ylab("Final Grade")
```

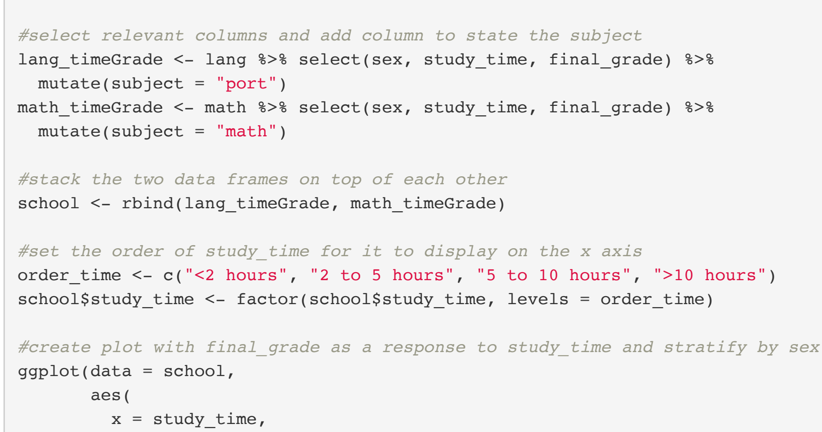


d. Using the math scores data set, for each grade on the first exam, find the average grade on the final exam for all the students who had the same score on exam one. That is, for all students who score, for example, a 10 on the first exam, what was the average final score for this group of students. Do this for all grades on the first exam. Repeat this for the second exam (i.e. For each grade on the second exam, find the average grade on the final exam for all the students who had the same score on exam two). Plot the exam score (first or second) on the x-axis and the average final score on the y-axis using color to indicate whether the point was the first or second exam. The final plot should look like the one provided.

```r
#calculate the mean of grade_1 and add a variable that states the exam number
one <- math %>% group_by(grade_1) %>% summarize(final = mean(final_grade)) %>%
  mutate(type = "one") %>%
  #rename the variable in preparation for stacking
  rename(grade = grade_1)
#calculate the mean of grade_2 and add a variable that states the exam number
two <- math %>% group_by(grade_2) %>% summarize(final = mean(final_grade)) %>%
  mutate(type = "two") %>%
  #rename the variable in preparation for stacking
  rename(grade = grade_2)

#stack the data frames on top of each other
allgrade <- rbind(one, two)

#create plot of final as a response to grade
ggplot(data = allgrade,
       aes(
         x = grade,
         y = final,
         color = type
       )
) + geom_point()
```



QUESTION 2

```r
spot <- read.csv("spotify_songs.csv")
```

a. Plot a line plot of the release year on the x-axis and the number of songs released in this year on the y-axis.

```r
library(lubridate)
```

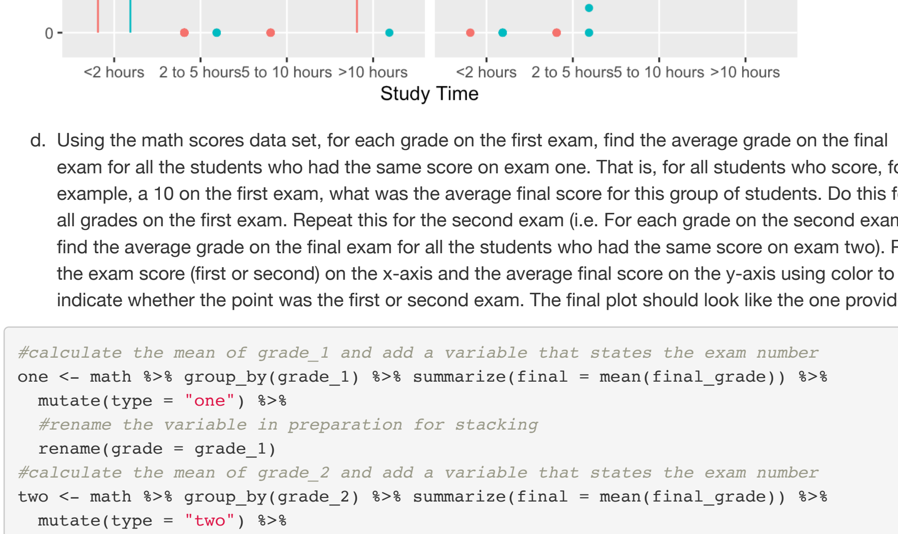
```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
release <- spot %>%
  #add column for date year
  mutate(release_year = year(as.Date(track_album_release_date))) %>%
  #count songs in each year
  group_by(release_year) %>%
  summarize(song_num = n())

ggplot(data = release,
       aes(
         x = release_year,
         y = song_num
         )
       ) + geom_line() +
  xlab("Release Year") +
  ylab("Number of Songs")
```

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```



b. Remove all the songs that only have the release year (as opposed to the full date of release). For each release month, find the 5 most danceable songs (higher scores are better). This will give you a list of 60 total songs (i.e. 12 months times 5 songs each month equals 60). Of these 60 songs, what is the name of the playlist that contains the song with the highest tempo? Of the same 60 songs, what is the name of the playlist that contains the song with the lowest tempo? Finally, what is the most common genre among these 60 songs, and how many songs belong to that genre?

```r
#filter for dates in the correct format
topDance <- spot %>% filter(as.Date(track_album_release_date) == format(as.Date(track_
album_release_date), format = "%Y-%m-%d")) %>%
  #add variable for release month
  mutate(release_month = month(as.Date(track_album_release_date))) %>%
  #remove any repeated track IDs
  distinct(track_id, .keep_all = TRUE) %>%
  #select songs with top 5 danceability scores for each month
  group_by(release_month) %>%
  slice_max(order_by = danceability, n=5) %>%
  #due to tie for 5th in scores for December, remove one of the tracks
  filter(!(track_id %in% c("4ih3rak54fNYhweawmNZcj")))

high_tempo <- topDance %>%
  arrange(-tempo) %>%
  select(playlist_name, tempo) %>%
  head(1)
```

```
## Adding missing grouping variables: `release_month`
```

```r
print(high_tempo)
```

```
## # A tibble: 1 × 3
## # Groups:   release_month [1]
##   release_month playlist_name                              tempo
##           <dbl> <chr>                                      <dbl>
## 1             2 Dirty South Rap Classics by DJ HOTSAUCE  139.
```

```r
low_tempo <- topDance %>%
  #select slowest tempo
  arrange(tempo) %>%
  select(playlist_name, tempo) %>%
  head(1)
```

```
## Adding missing grouping variables: `release_month`
```

```r
print(low_tempo)
```

```
## # A tibble: 1 × 3
## # Groups:   release_month [1]
##   release_month playlist_name tempo
##           <dbl> <chr>         <dbl>
## 1             9 Zona Trap      103.
```

```r
comm_genre <- topDance %>%
  #count songs in each genre
  group_by(playlist_genre) %>%
  summarize(num_songs = n()) %>%
  #select genre with most songs
  arrange(-num_songs) %>%
  head(1)
print(comm_genre)
```

```
## # A tibble: 1 × 2
##   playlist_genre num_songs
##   <chr>              <int>
## 1 rap                   27
```

c. You are a wedding DJ and you are working with a night-mare couple and they have a lot of very specific requests. They want their first dance to be exactly 10 minutes long and consist of two back to back songs (they cannot be the same song!) by the artist Depeche Mode. Find the two Depeche Mode songs in this data set when played back to back the duration is as close as possible to 10 minutes (hint: a cross join may be useful here). (Note1: For loops are not allowed in your solution.)

```r
#filter for artist and select the relevant columns
DM_songs <- spot %>% filter(track_artist == "Depeche Mode") %>% select(track_name, dur
ation_ms)
#create a duplicate
DM_songs2 <- DM_songs

#cross join the two data frames
merge(DM_songs, DM_songs2, by=NULL) %>%
  #sum the durations and convert to minutes by dividing by 60000
  mutate(minutes = (duration_ms.x+duration_ms.y)/60000) %>%
  #filter for row where minutes are closest to 10
  filter(minutes == minutes[which.min(abs(minutes - 10))])
```

```
##                                 track_name.x duration_ms.x
## 1 Enjoy The Silence - 2006 Remastered Version        372813
## 2                     Strangelove - 7" Version        227413
##                                 track_name.y duration_ms.y  minutes
## 1                     Strangelove - 7" Version        227413 10.00377
## 2 Enjoy The Silence - 2006 Remastered Version        372813 10.00377
```

'Enjoy The Silence - 2006 Remastered Version' and 'Strangelove - 7" Version' by Depeche Mode are the songs I'd play.

QUESTION 3 (a) Lets say that you have a population that follows a normal distribution with a mean of 10 and a variance of 20. Generate a simple random sample from this population with sample size n = 25. From this sample, compute the sample mean (i.e. $\bar{x}$). Repeat this process (i.e. generate a sample of size n = 25 and compute $\bar{x}$) a large number of times (say 5000) and store the value of $\bar{x}$ each time. Compute the standard deviation of this collection of $\bar{x}$'s. This is a simulated approximation of the standard error of $\bar{x}$. What is the value that you obtain?

```r
set.seed(1234)
mean <- 10
variance <- 20
sd <- sqrt(variance)
n <- 25
#number of simulations
nsim <- 5000
sample_means <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_means[i] <- mean(sample_data)
}

mean_error25 = sd(sample_means)
mean_error25
```

```
## [1] 0.8899491
```

b. Repeat the previous question but use the sample median instead of the mean. What is the approximate value of the standard error of the sample median? Is this bigger, smaller, or the same as the standard error of the mean?

```r
sample_medians <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_medians[i] <- median(sample_data)
}

median_error25 = sd(sample_medians)
median_error25
```

```
## [1] 1.116712
```

This is bigger than the standard error of the mean.

c. Now repeat both of the previous questions with values of n equal to 10, 25 (you already have this one), 50, 100, 250, 500, 1000. (Note: n is the sample size and it is not the same as the number of simulations!) Now plot n, the sample size on the x-axis versus the standard error on the y-axis as a line plot with points at each of the values of n that were used. There should be one line for median and one line for mean you should use color to distinguish between them.

```r
#n=10
n=10
sample_means <- numeric(nsim)
sample_medians <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_means[i] <- mean(sample_data)
  sample_medians[i] <- median(sample_data)
}

mean_error10 = sd(sample_means)
median_error10 = sd(sample_medians)

#n=50
n=50
sample_means <- numeric(nsim)
sample_medians <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_means[i] <- mean(sample_data)
  sample_medians[i] <- median(sample_data)
}

mean_error50 = sd(sample_means)
median_error50 = sd(sample_medians)

#n=100
n=100
sample_means <- numeric(nsim)
sample_medians <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_means[i] <- mean(sample_data)
  sample_medians[i] <- median(sample_data)
}

mean_error100 = sd(sample_means)
median_error100 = sd(sample_medians)

#n=250
n=250
sample_means <- numeric(nsim)
sample_medians <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_means[i] <- mean(sample_data)
  sample_medians[i] <- median(sample_data)
}

mean_error250 = sd(sample_means)
median_error250 = sd(sample_medians)

#n=500
n=500
sample_means <- numeric(nsim)
sample_medians <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_means[i] <- mean(sample_data)
  sample_medians[i] <- median(sample_data)
}

mean_error500 = sd(sample_means)
median_error500 = sd(sample_medians)

#n=1000
n=1000
sample_means <- numeric(nsim)
sample_medians <- numeric(nsim)

for (i in 1:nsim){
  sample_data <- rnorm(n, mean, sd)
  sample_means[i] <- mean(sample_data)
  sample_medians[i] <- median(sample_data)
}

mean_error1000 = sd(sample_means)
median_error1000 = sd(sample_medians)


#create data frame
errors = data.frame(N = c(10, 25, 50, 100, 250, 500, 1000),
  error_type = c("mean_error", "mean_error", "mean_error",
                 "mean_error", "mean_error", "mean_error",
                 "mean_error", "median_error", "median_error",
                 "median_error","median_error", "median_error",
                 "median_error","median_error"),
  error_value= c(mean_error10, mean_error25, mean_error50,
                 mean_error100, mean_error250, mean_error500,
                 mean_error1000, median_error10, median_error25,
                 median_error50,median_error100, median_error250,
                 median_error500, median_error1000)
)

ggplot(data = errors,
       aes(x=N, y=error_value, color=error_type))+
  geom_line()
```