Wrangling Chess Tournament Data

Alec McCabe

9/12/2021

Setup

Load libraries

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(stringr)
library("RMySQL")
```

```
## Loading required package: DBI
```

Establish connection to SQL

After processing and formatting the input data as per assignment instructions, we will be saving the output in SQL. I've created two tables:

- players: a table of players
- scores: a table of player scores from tournaments played

```
mydb = dbConnect(MySQL(), user = 'root', dbname='data_607', host='localhost')
```

lapply(dbListConnections(dbDriver(drv = "MySQL")), dbDisconnect)

Import data and inspect

```
data = read.table("https://raw.githubusercontent.com/man-of-moose/masters_607/main/projects/project_1/c
```

```
head(data[,'V1'])
```

```
## [1] "-----------------------------------------------------------------------------------------"
## [2] " Pair "
## [3] " Num  "
## [4] "-----------------------------------------------------------------------------------------"
## [5] "    1 "
## [6] "   ON "
```

Convert [input data] –> [desired assignment format]

This assignment's input data is a "|" delimited .txt file containing information about players in a chess tournament. This project aims to wrangle the input data into a tabular format, with the following columns:

- Player's Name
- Player's State
- Total Number of Points
- Player's Pre-Rating
- Average Pre Chess Rating of Opponents

Change column names

Rename columns to make analysis meaningful

```
column_names <- c("id","name","total","round_1","round_2","round_3","round_4",
                  "round_5","round_6","round_7","delete")

colnames(data) <- column_names
```

Remove "———-" rows

Following data import, there are some rows that serve no purpose for our assignment. Additionally, an unexpected column of NAs was created which we will want to delete. We will use dplyr::filer and dplyr::select to achieve this.

```
data<- data %>%
        filter(str_detect(id, "[a-zA-z\\d]")) %>%
        select(-delete)
```

Looking at our data now

Due to the structure of the input data, and the steps we've taken so far, our current dataframe is structured in an interesting way. Rows with a numeric "id" value contain information about a player's name, their total score, and the rounds they played. While rows with a character "id" contain information about a player's state and pre_score.

```
head(data)
```

Get state_data character vector

We need to capture the state data in a character vector, to later be used to represent the state column of our output. We can do this by filtering for rows that contain 2 capital letters, and saving the "id" column into a variable called state_data

```
state_data <- data$id
state_data <- state_data[grepl("[A-Z]{2}",state_data)]
state_data <- str_trim(state_data, side = c("both"))
```

Get pre_score data

We can extract the pre_score data in the same way we did state_data. Unlike state_data, the pre_score data will require more advanced regex to properly extract.

```
pre_score_data <- data$name

data$id <- data$id %>%
  str_trim(side=c("both"))

pre_score_data <- data %>%
  filter(str_detect(id,"[A-Z]{2}")) %>%
  .$name

pre_score_data <- pre_score_data %>%
              str_extract("R:[^\\d]*\\d*") %>%
              str_replace("R:","") %>%
              str_trim(side=c("both"))
```

Remove rows with non-numeric values

Now that we've extracted the state data and pre_score data, we can remove rows with non-numeric values.

```
data <- data %>%
        filter(str_detect(id,"\\d"))
```

Add state and pre_score columns

And now we can take state_data and pre_score_data and add them as new columns to the recently filtered dataframe.

```
data$state <- state_data
data$pre_score <- as.integer(pre_score_data)
```

Rearrange columns for clarity

Using dply::select and everything() we can easily re-arrange our column values for easier reading.

```
data <- data %>% select(name, state, pre_score, total, everything())
```

```
head(data)
```

Convert total into double

The 'total' value was parsed as a character. since we will be applying math to this later, we need to convert to a double.

```
data$total <- as.double(data$total)
```

Create new column, "oppo_ids"

This new column will include vectors containing the opponent ids for each respective player. As an example, Gary Hua's value here would be:

c(39,21,18,14,7,12,4)

This is achieved by first concatenating each of the "round_" columns. Following this, we use stringr to parse out and collect the opponent ids.

```
data <- data %>% mutate(oppo_ids = str_c(round_1,round_2,round_3,round_4,round_5,round_6,round_7))

data$oppo_ids <- data$oppo_ids %>%
  str_replace_all("[A-Z]","") %>%
  str_trim(side=c("both")) %>%
  str_replace_all("\\s{2,}","|")

data$oppo_ids <- data$oppo_ids %>% str_split("\\|")
```

Create function to calculate average opponent score

This function will use the previously created "oppo_id" column values as input, in order to filter for and average the correct opponent pre_scores.

```
get_avg_oppo_score <- function(id_data) {
  temp_df <- data %>%
              filter(id %in% id_data) %>%
              summarise(pre_score_mean = mean(pre_score, na.rm=TRUE))

  return(temp_df[,1])
}
```

Test it out on the first example

```
get_avg_oppo_score(c(39,21,18,14,7,12,4))
```

```
## [1] 1605.286
```

Apply function to entire dataframe

```
data$avg_oppo_score <- lapply(data$oppo_ids,FUN=get_avg_oppo_score)
```

Calculate total number of games played

We will need this later on for extra credit. Here we are counting how many games each player participated in.

```
data$number_of_games <- as.integer(lapply(lapply(data$oppo_ids, FUN=lengths),FUN=sum))
```

Select only interesting columns

There are a few columns we don't need anymore such as all of the "round_" columns, the "oppo_ids" column, and others. We can use dplyr::select to select only what's interesting.

```
final_data <- data %>%
  select(name, state, total, number_of_games, pre_score, avg_oppo_score)
```

Round avg_oppo_score (Average Opponent Score) and inspect

Based on the description of this project, we will be rounding the values of avg_oppo_score with the round() function.

```
final_data$avg_oppo_score <- as.integer(lapply(final_data$avg_oppo_score,FUN=round))
```

```
final_data
```

Trim names

While you can't tell from the above tibble, many of the player names actually have surrounding white spaces. We can remoe with stringr::str_trim

```
final_data$name <- str_trim(final_data$name, side=c("both"))
```

Calculate expected score for each player

In chess a player's "total score" for a game is determined by whether or not the player wins (+1), loses (+0), or draws (+0.5)

The "expected score" for a player in one game can be represented as a modified probability that they will win, based on their pre_score relative to their opponent's pre_score.

The following function can perform the required calculation:

1/(10^(({oppo_Pre_score}-{pre_score})/400)+1)

Because have already computed averages for our opponent_pre_scores, we can modify the above equation as such:

1/(10^(({oppo_Pre_score}-{pre_score})/400)+1) * {number_of_games}

The function used above was identified from the following sources:

- http://www.uschess.org/index.php/Players-Ratings/Do-NOT-edit-CLOSE-immediately.html
- https://chess.stackexchange.com/questions/18209/how-do-you-calculate-your-tournament-performance-rating

```
final_data <- final_data %>% mutate(expected_total = 1/(10^((avg_oppo_score-pre_score)/400)+1) * number
```

EXTRA CREDIT: which player scored the most points relative to their expected score?

Answer is Aditya Bajaj, who performed very well throughout this tournament. They won 6 out of 7 games, despite the fact that, on average, they were rated nearly 200 points below each of their opponents.

```
final_data %>%
  mutate(score_differential = total - expected_total) %>%
  arrange(desc(score_differential)) %>%
  .[1,c('name','pre_score','avg_oppo_score','total','expected_total','score_differential')]
```

Generate a .CSV file and load values into SQL

Generate a .CSV file

```
write.table(final_data, sep=",", file = "/Users/alecmccabe/Desktop/Masters Program/DATA 607/masters_607
```

Create query to assign ids to players

The reason why I chose to include two tables in my SQL database was to allow for continued use of this script. When new tournaments happen, new players may participate.

This function will work by looking at the total list of tournament participants, and cross-reference that list against the existing SQL table data_607.players

This ensures that if a participant has already been counted in previous tournaments, they will be assigned the same player_id.

Alternatively, if there is a new participant, this function will ensure that their generated player_id does not match any existing ones.

```
assign_player_ids <- function(insert_data, mydb) {
  names <- insert_data$name
  players_string <- str_c('"',str_trim(names,side=c("both")),'"',collapse=",")
  insert_data$id <- NA
  query <- str_interp("SELECT player_name, id FROM data_607.players WHERE player_name in (${players_str

  select_data <- dbGetQuery(mydb, query)

  for (row in 1:nrow(select_data)) {
    select_name <- select_data[row,"player_name"]
    select_id <- select_data[row,"id"]

    insert_data <- within(insert_data, id[name == select_name] <- select_id)
  }

  for (row in 1:nrow(insert_data)){
    if (is.na(insert_data[row,]$id)) {
      if (sum(!is.na(insert_data$id))>0) {
        max_id <- max(insert_data$id, na.rm=TRUE) +1
      } else {
        max_id <- 1
      }
      insert_data[row,]$id <- max_id
    }
  }

  return(insert_data)
}
```

Running id assignment

Because data_607.players is currently empty, each of the participants in this tournament will be provided with incremental ids, starting with 1 and ending at 64.

```
final_data <-assign_player_ids(final_data,mydb)
```

Create insert function to load into data_607.players

This function will load any new players, and their associated player_ids and state information into the data_607.players table.

```
insert_players <- function(data, mydb){

  names <- final_data$name
  players_string <- str_c('"',str_trim(names,side=c("both")),'"',collapse=",")
  query <- str_interp("SELECT player_name, id FROM data_607.players WHERE player_name in (${players_str

  select_data <- dbGetQuery(mydb, query)

  for (row in 1:nrow(final_data)){
    id <- as.integer(final_data[row, "id"])
```

```
    name <- str_trim(final_data[row, "name"], side=c("both"))
    state <- str_trim(final_data[row, "state"], side=c("both"))
    total <- final_data[row, "total"]
    pre_score <- final_data[row, "pre_score"]
    avg_opponent_score <- final_data[row, "avg_oppo_score"]

    insert_query <- str_interp('insert into data_607.players VALUES (${id},"${name}", "${state}")')

    if (name %in% select_data$player_name) {
      next
    } else {
      print(name)
      dbGetQuery(mydb, insert_query)
    }

  }
}
```

Create a function to insert into the data_607.scores table

This function will load a player's performance data and metrics into the data_607.scores table. This function takes 'tournament_id' variable as input in addition to data and db_connection.

```
insert_scores <- function(data, tournament_id, mydb){

  for (row in 1:nrow(final_data)){
    id <- as.integer(final_data[row, "id"])
    name <- str_trim(final_data[row, "name"], side=c("both"))
    total <- final_data[row, "total"]
    pre_score <- final_data[row, "pre_score"]
    avg_opponent_score <- final_data[row, "avg_oppo_score"]
    number_of_games <- final_data[row, "number_of_games"]
    expected_total <- final_data[row, "expected_total"]

    insert_query <- str_interp('insert into scores VALUES (DEFAULT,${tournament_id},${id},${number_of_g

    dbGetQuery(mydb, insert_query)
  }
}
```

Insert into SQL tables The insert_players function prints to the console each player's name that is added to the SQL data ("new players"). As we see below, everyone is added.

```
insert_players(final_data, mydb)
```

```
insert_scores(final_data,1, mydb)
```

Visualize the distribution of 'score_differential' for players

'score_differential' will be defined as the difference between a player's expected total, and their actual total points.

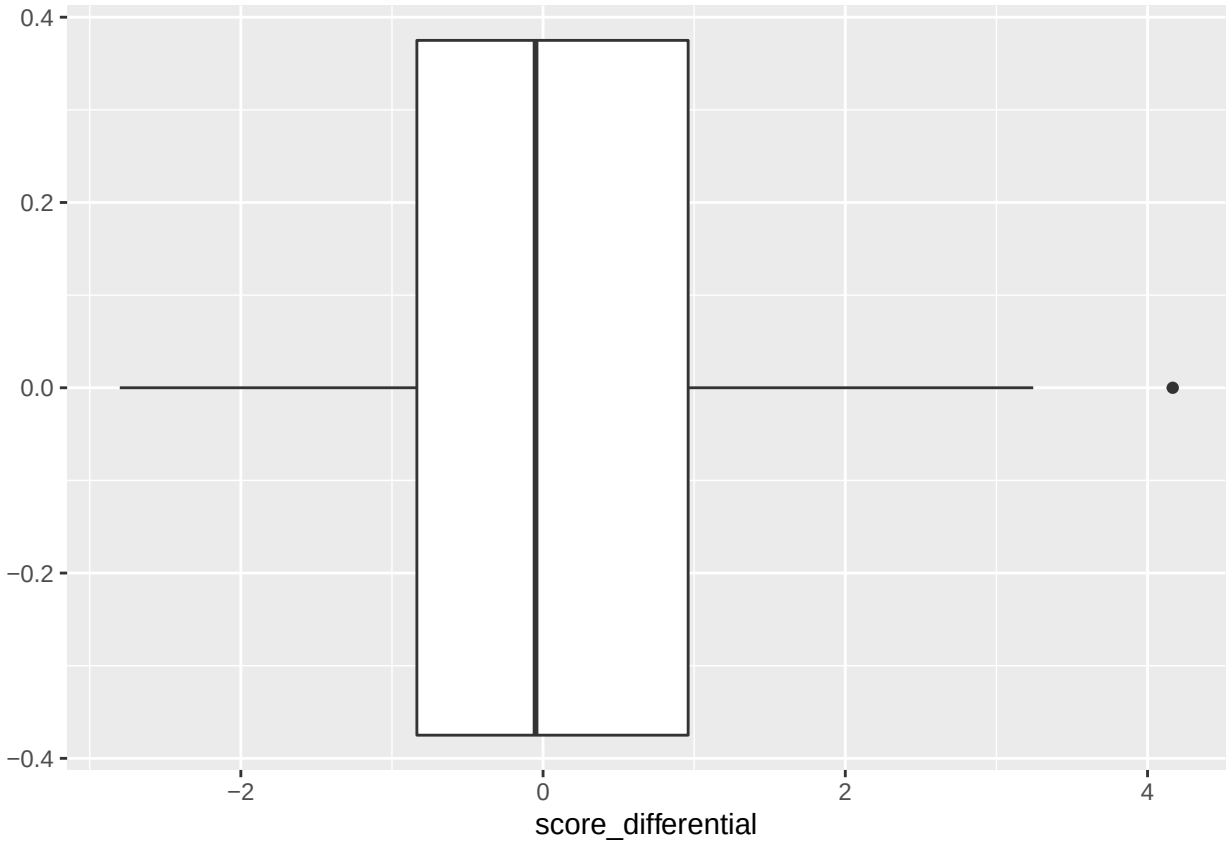
```
final_data <- final_data %>%
  mutate(score_differential = total - expected_total)
```

The below boxplot shows that median score_difference within the population is close to zero, and seemingly normally distributed. Roughly 50% of the population's score_difference is between -0.5 and 0.5. Tail values stretch from -2.5 (performed much worse than expected) all the way to 2.5 (performed much better than expected).

There is also an outlier identified, with a score_differential of 4.166. As we discussed earlier, Aditya performed much better than expected.

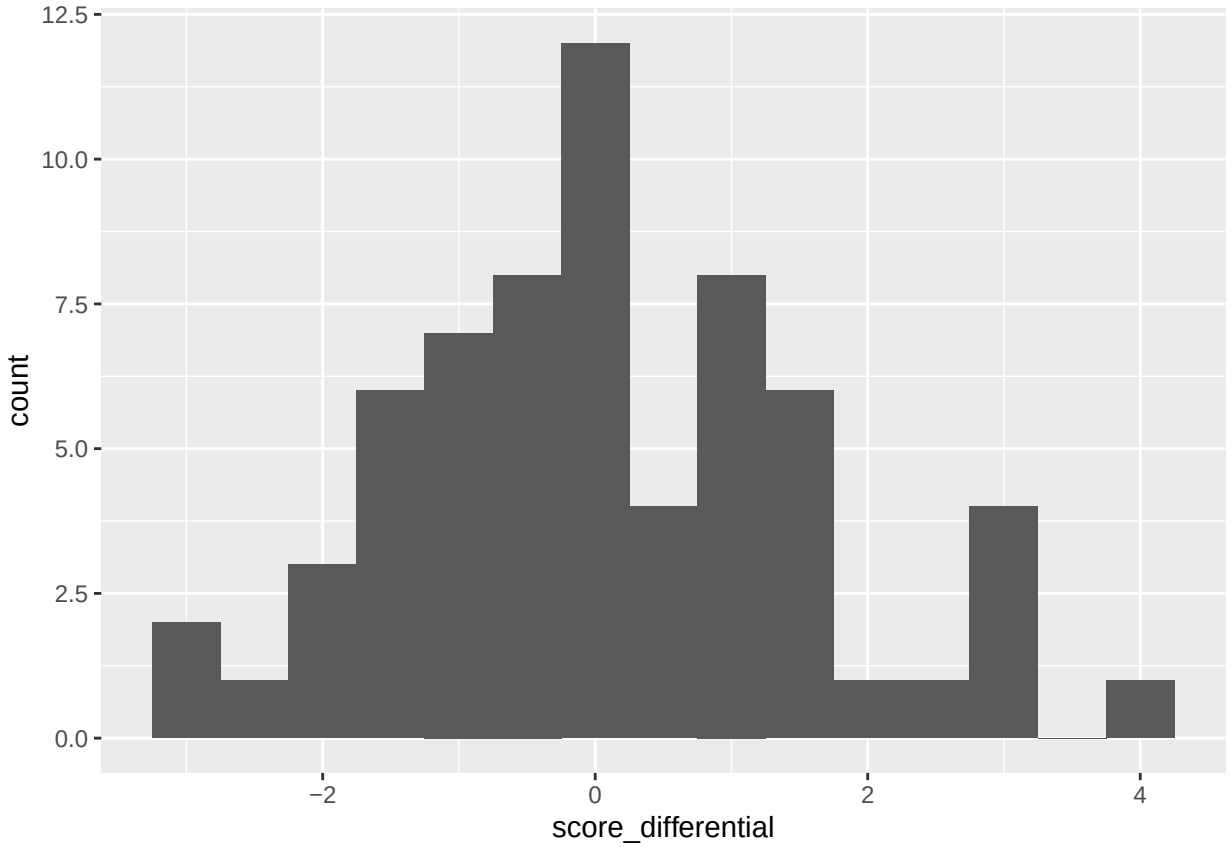
Based on the data, one could make that claim that Aditya's performance was not a fluke, but rather an improper initial pre_score going into the tournament.

```
final_data %>%
  ggplot(aes(x=score_differential)) +
  geom_boxplot()
```



The histogram below confirms our claim that the score_difference distribution is gaussian, with a mean of zero and no apparant skew.

```
final_data %>%
  ggplot(aes(x=score_differential)) +
  geom_histogram(binwidth = .5)
```



Test assign_player_ids inser_players functions

Here we will make sure that the above functions work as expected when presented with new player data.

As an example, imagine that a second tournament includes all of the members of this tournament, plus one new member: Johnny Apple.

If our functions work as expected, then the assign_player_id function will provide Johnny with the id 65, and the insert_players function will only insert Johhny (since the previous players are already contained in the SQL table)

```
final_data <- add_row(final_data,
        name="Johnny Apple",
        state="OH",
        total=5,
        number_of_games = 7,
        pre_score=1111,
        avg_oppo_score=1245,
        expected_total = 4,
        id=NA)
```

```
final_data <-assign_player_ids(final_data,mydb)
```

```
final_data[final_data$name=="Johnny Apple",c('name','id')]
```

```
insert_players(final_data, mydb)
```