---

title: "is everything ok?"

exported: 2026-03-01 04:27:38

url: https://gemini.google.com/app/56114e799c65028f

messages: 12

---

is everything ok?

You

read the file

Gemini

I have read through your R Markdown file (`Proj.1.qmd`) and the two assignment screenshots.

Here is a summary of where your project currently stands based on the assignment guidelines:

What You Have Completed

Your R script successfully fulfills the primary requirements outlined in the first "Project 1" screenshot.

- [cite_start]**Data Extraction:** You effectively split the data into odd and even lines to parse the player's name, total points, state, and pre-rating[cite: 7, 8, 9, 10].
- [cite_start]**Averaging Opponent Ratings:** You successfully mapped the opponents' IDs to their pre-ratings to calculate the average pre-tournament rating of each player's opponents[cite: 12, 13].
- [cite_start]**Final Output:** You combined these elements into `final_table` with the exact five requested columns and exported it as a CSV[cite: 14].

What Is Missing

Based on the second screenshot ("Chess ELO calculations"), there is an additional requirement that is not yet implemented in your R script:

- **Expected Score Calculation:** You need to calculate each player's expected score based on the difference in ratings between the player and each of their opponents.
- **Performance Analysis:** You need to calculate the difference between their actual score and expected score to find the **five most overperforming** and **five most underperforming** players.

Would you like me to help you write the R code to implement the expected score formula and find those top 5 overperformers and underperformers?

You

How should I extract WLD and assign scores with this code?

Gemini

You're right on track with your `pivot_longer` approach! Since the `cell` column contains strings like `"W 39"` or `"D 18"`, you can extract the result letter (W, L, D, etc.) using another `str_extract()` and then map those letters to numerical scores using `case_when()`.

Here is how you can update your `mutate` block to pull the match outcome and assign the correct chess scores (1 point for a win, 0.5 for a draw, 0 for a loss, etc.):

```
opp_info <-
  players |>
  select(player_num, starts_with("round")) |>
  pivot_longer(
    cols = starts_with("round"),
    names_to = "round",
    values_to = "cell"
  ) |>
  mutate(
    cell = str_trim(cell),
    # Extract the opponent number
    opp_num = as.integer(str_extract(cell, "\\d+")),

    # Extract the first uppercase letter (W, L, D, U, H, B)
    outcome = str_extract(cell, "^[A-Z]"),

    # Assign numerical scores based on the outcome
    match_score = case_when(
      outcome %in% c("W", "B", "X") ~ 1.0,  # Win, Full-point Bye, Forfeit Win
      outcome %in% c("D", "H") ~ 0.5,       # Draw, Half-point Bye
      outcome %in% c("L", "U", "F") ~ 0.0,  # Loss, Unplayed, Forfeit Loss
      TRUE ~ NA_real_                       # Catch-all for anything else
    )
  )

opp_info |>
  filter(player_num == 1)
```

What's happening here:

- **`outcome = str_extract(cell, "^[A-Z]")`**: The regex `"^[A-Z]"` tells R to look at the very beginning (`^`) of the string and grab the first uppercase letter it sees.
- **`case_when(...)`**: This is dplyr's version of an `if-else` statement. It checks the `outcome` column and assigns `1.0`, `0.5`, or `0.0`. I've included standard USCF tournament codes like `B` (Bye), `H` (Half-point bye), and `U` (Unplayed) just in case those pop up in the dataset!

Would you like me to help you set up the actual Elo expected score formula next using these extracted scores?

You

but BHU has no opp, should not be make it as na?

Gemini

You are absolutely spot on! Since the Elo expected score calculation requires finding the difference between a player's rating and their *opponent's* rating, you cannot calculate an expected score if there is no opponent.

[cite_start]In fact, you already anticipated this perfectly in your original markdown file where you noted that you would process B/H/U matches separately[cite: 3]. For unplayed games, byes, or forfeits, the standard practice in chess rating calculations is to exclude them entirely because no actual rated play occurred.

Since `str_extract(cell, "\\d+")` will already return `NA` for `opp_num` when it sees a solitary "B", "H", or "U", we can just simplify the `case_when` to only score actual played matches (W, L, D) and let everything else default to `NA`.

Here is the revised code:

```
opp_info <-
  players |>
  select(player_num, starts_with("round")) |>
  pivot_longer(
    cols = starts_with("round"),
    names_to = "round",
    values_to = "cell"
  ) |>
  mutate(
    cell = str_trim(cell),
    # Extract the opponent number (will be NA for B, H, U)
    opp_num = as.integer(str_extract(cell, "\\d+")),

    # Extract the first uppercase letter (W, L, D, U, H, B)
    outcome = str_extract(cell, "^[A-Z]"),

    # Only assign scores for games ACTUALLY played against an opponent
    match_score = case_when(
      outcome == "W" ~ 1.0,
      outcome == "D" ~ 0.5,
      outcome == "L" ~ 0.0,
      TRUE ~ NA_real_       # B, H, U, and unplayed games become NA
    )
  ) |>
  # Optional: Filter out the NA rounds right now so they don't mess with your math later
  filter(!is.na(opp_num))

opp_info |>
  filter(player_num == 1)
```

By adding that `filter(!is.na(opp_num))` at the end, you cleanly remove all the B/H/U rounds from this specific dataframe, leaving you with only the valid, rated matches you need for the Elo math.

Would you like to write the Elo formula function next so we can apply it to these filtered matches?

You

match_score = case_when(outcome == "W" ~ 1.0, outcome == "D" ~ 0.5, outcome == "L" ~ 0.0, TRUE ~ NA_real_
expllain the last line

Gemini

Here is a breakdown of exactly what `TRUE ~ NA_real_` is doing in that function:

- **`TRUE` acts as the "else" or "catch-all" condition:** In a `case_when` function, R evaluates each condition line by line. If a match outcome isn't exactly "W", "D", or "L" (for example, if it's a "B", "H", or "U"), it skips down until it hits `TRUE`. Since `TRUE` is always true, it catches absolutely everything that didn't match the rules above it.
- **`NA` means "Not Available" (missing data):** It assigns a blank value to those unplayed or bye rounds. This is exactly what you want, so those non-matches don't skew your calculations later.
- **`_real_` ensures the correct data type:** R's `case_when` is incredibly strict; every single result it produces *must* be of the exact same data type. Since 1.0, 0.5, and 0.0 are all decimals (which R categorizes as "real" or "numeric" numbers), the `NA` must also be specified as a "real" number. If you just typed `NA` (which R defaults to a logical/boolean type), the function would crash and throw a type-mismatch error.

In short, you are safely telling R: *"For any letter that isn't a Win, Draw, or Loss, return a missing numeric value."*

Are you ready to set up the Elo expected score formula next using these newly assigned scores?

You

should I combine all the cols or just do elo rating

Gemini

Looking at your latest progress, you've done a great job restructuring the match data into a clean, long format (`opp_info`). To finish the assignment requirements shown in your screenshots, you should focus on **calculating the Elo expected scores** rather than just combining columns.

The prompt specifically asks for the "expected score" and the "difference from their actual score" to identify overperformers and underperformers.

---

1. The Elo Expected Score Formula

The standard Elo formula for an expected score ($E$) between Player A (rating $R_A$) and Player B (rating $R_B$) is:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

2. How to implement this in your code

Since you have already built `opp_info` and `find_rating`, you can merge them to get both the player's rating and the opponent's rating in one row, then apply the formula.

```
# 1. Join ratings to get Player Rating (R_A) and Opponent Rating (R_B)
elo_calc <- opp_info |>
  left_join(find_rating, by = "player_num") |>               # Get Player's PreRating
  rename(player_rating = opp_pre) |>
  left_join(find_rating, by = c("opp_num" = "player_num")) |> # Get Opponent's PreRating
  rename(opponent_rating = opp_pre) |>
  filter(!is.na(opponent_rating))                             # Only rated games

# 2. Calculate Expected Score for each match
elo_calc <- elo_calc |>
  mutate(
    expected_match_score = 1 / (1 + 10^((opponent_rating - player_rating) / 400))
  )

# 3. Summarize by player to find Over/Under performance
performance_summary <- elo_calc |>
  group_by(player_num) |>
  summarise(
    ActualTotal = sum(game_score, na.rm = TRUE),
    ExpectedTotal = sum(expected_match_score, na.rm = TRUE),
    Difference = ActualTotal - ExpectedTotal
  ) |>
  left_join(players |> select(player_num, name), by = "player_num")
```

---

3. Identifying Over/Underperformers

Once you have the `performance_summary`, you can easily pull the top and bottom five as requested:

- **Top 5 Overperformers:** Use `arrange(desc(Difference)) |> head(5)`.
- **Top 5 Underperformers:** Use `arrange(Difference) |> head(5)`.

Why do it this way?

By performing the calculation at the **match level** (comparing the player to each specific opponent) before summing it up, you are following the accurate Elo method described in your assignment prompt. This is much more precise than just using the "Average Opponent Rating."

Would you like me to help you format these final lists into a table for your report?

You

is everything ok?

Gemini

Your code and logic look solid! [cite_start]You have successfully addressed the "Data Challenges" you anticipated by filtering out rounds with no opponents [cite: 18, 19, 25] [cite_start]and correctly implemented the match-level Elo expected score formula as required by the assignment[cite: 26].

Here is a quick check of your work against the project requirements:

1. Handling Non-Rated Rounds (B/H/U)

[cite_start]Your use of `filter(!is.na(opp_num))` effectively removes the rounds where no opponent exists[cite: 24]. This is critical because the Elo formula, which you correctly defined as: $$E = \frac{1}{1 + 10^{(R_{opp} - R_{self})/400}}$$ [cite_start]requires two ratings to function[cite: 26]. [cite_start]By creating `score_recal`, you ensured that the "Actual Total" (your `new_total`) only includes points earned in those specific rated matches[cite: 25].

2. The Performance Summary

[cite_start]You correctly joined the player's own rating and their opponents' ratings to calculate the `expected_match_score`[cite: 26].

- [cite_start]**Calculation:** You summed these expected scores by player and subtracted them from the `new_total` to get the `Difference`[cite: 27].
- [cite_start]**Identification:** Your final sorting using `desc(Difference)` and `(Difference)` correctly identifies the outliers[cite: 28].

3. Small Suggestions for "Polishing"

While your logic is correct, here are two minor technical "tweaks" to ensure your final report is perfectly accurate:

- [cite_start]**The Difference Calculation:** In your `performance_summary` block, you use `mutate(Difference = score_recal$new_total - Expected_score)`[cite: 27]. It is safer to use the column name directly (just `new_total - Expected_score`) since you already joined `score_recal` into that dataframe. Using `$` inside a `mutate` can sometimes cause issues if the row orders ever get shuffled.
- **Tie-Breaking:** For the "Top 5," if two players have the exact same `Difference`, you might consider including a secondary sort (like by `new_total`) just to keep the list consistent.

Final Results

Your identified players align with the expected behavior of the Elo system:

- [cite_start]**Overperformers:** These players scored significantly more points than their rating (and their opponents' ratings) suggested they would[cite: 17].
- **Underperformers:** These players may have had high ratings but suffered upsets against lower-rated opponents.

Everything looks ready for submission! Would you like me to help you format the final "Top 5" lists into a clean Markdown table for your summary section?