PROJECT OVERVIEW:
I changed my project idea from my initial proposal because I found that my initial dataset and approach was too computationally costly: I originally had a dataset of 498k rows that computed connections between hockey players across different leagues based on who else they had played with who acted as a bridge between two players, using breadth-first-search to find degrees of separation. However, it ended up being too large and paring it down made some connections impossible. I also struggled to handle duplicate player names and diacritics/accents in my String encoding, so I ended up going back to the dataset from my original search: a record of every Premier League match from 1992-93, when the league was started, to the 2022-23 season.

My initial goal for this project was to use preliminary data analysis to find out if COVID seasons in the Premier League had any effect on home advantage. The effect of home crowd support on match results in the Premier League is well-documented – and considering that during the COVID seasons (part of 2019-20 and the entirety of 2020-21) matches took place behind closed doors, I was curious to see if that affected the home vs away win-rates of teams in the PL. I used the plotters crate to visualize several aspects of my data.

DATA PROCESSING:
I split my code into a main and a mod file (game.rs) which contained all the information to process the csv, create Game and DataFrame structs, and print information about individual games. I started with a read_csv method that took in an empty DataFrame struct to store all the data from the csv (the headers and individual Game structs as a Vector of Games). Indexing by the column number, I created individual vectors for each column and pushed the value from that column-row pairing to the vector, so I could collect each vector into my Game struct. I then pushed each Game struct into the games field of my DataFrame, and returned a Result enum containing a vector of Games matching the one that belonged to my self.games field.

CODE STRUCTURE:
Since I wanted to do a lot of team and season-specific computations, I refactored a lot of my code to create functions that took in each team/season and then used those in my general functions when I wanted to compute an average across teams or seasons. I passed the same vector of Games into all of my functions, and then used nested loops to check if that game was in the correct season and involved the named team. I started by computing the most successful teams using two measures: overall win percentage and number of seasons in the Premier League, which uses a relegation system where the least winning teams in each season are relegated to the next-highest league in the English football system.

For both my top_percent() and top_appearances() functions, I made helper functions that computed the overall win percentage for a given team in a given range of seasons, and the number of seasons the team appeared in that range. In my team_seasons() function, I initially used a HashSet to keep track of the seasons without repeats, but because HashSets are ordered by the hash function buckets and not necessarily the value of each element, I ended up changing this to a vector of seasons, and just used .contains to check if that season was

already in the given vector. To find the top ten most successful teams by both metrics, I looped through every team in the 1993-2023 season range and sorted my vectors by the relevant values.

I created functions to calculate a season's overall home, away, and draw result percentages, and made a test to ensure that they added up to 100. I split this into another module to group the functions by purpose – they are used only to compute result changes.
This was used to find the season with the lowest home win rate and how much of an outlier it was compared to the average home win rate (see output). I found that, consistent with my hypothesis, the 2020-21 season had the lowest home win rate – the only season where the home win rate was lower than the away win rate. This was the only season when games were played entirely without spectators (only a few weeks of the 2019-20 season had no spectators, and in 2021-22, the league welcomed back full stadiums of fans). Remarkably, the home win rate in 2021 was about 8% lower than the overall average, and almost 3.5% lower than the second-lowest season (2016). So while more rigorous statistical analysis would be needed to control for other variables, this shows some speculative association between COVID season protocols and league-wide home advantage.

I then used the plotters crate to graph changes in home, away, and draw rates season-over-season. I color-coded each line and created a corresponding legend for my graph, and chose a scale that would capture the trends accurately. I also added a title and axis labels. This graph visually represents the change that I discussed: it is the only season where the home and away win rates overlap, and the immediate rebound to pre-COVID home advantage levels in 2022 and 23 also demonstrates this dramatic change.

I also calculated the average goals scored per game in each season and found that 2023 had the highest GPG average, which makes sense in context with sports journalists analyzing the increase in offensive tactics over the past few years, while defensive and goalkeeping techniques have not kept pace. I used plotters again to visualize the change in offensive tactics (through goals per game averages) from 1993 to 2023, finding an overall upward trend but with some major fluctuations. For fun, I also looked at a given team's highest game-winning interval across a given season range.

In my main function, I printed most of my information about the top n (10) most successful teams, using the functions I mentioned above. I also described the home win-rate change I had noticed during the COVID season relative to that season's draw and away-win rates. Lastly, I created a user input function that took in a team and a range of seasons, then provided some information about: a) what seasons in that range the team had played in, if any; b) the team's biggest win interval over that range, and information about the game where it took place, using my game print() method; and c) their win rate in that range. At various points in the user input function, I used the Result enum to handle possible errors, such as the season range not falling within the given seasons, the user input team not matching the team names, and the team not playing in the Premier League during that range.

TESTS:

I created two main tests for my program. The first tested the functionality of my home_pct, away_pct, and draw_pct functions. In theory, since these were calculating the proportion of games in a given season that ended in a home win, away win, and draw respectively (which I then multiplied by 100 to get a percentage), they should add up to 100. To accomplish this in my test, I inputted a random season (2023) and added the percentages from each function to a total_rate variable, which I then used the assert_eq! macro to compare to 100.0, and this test passed.

I also tested whether my team_seasons function worked: given a team and a season range, I wanted it to return a tuple containing a Vec<usize> of all the seasons in that range where the team had appeared, and a usize that counted those seasons. I knew that several teams had appeared in all 31 of the Premier League's seasons from 1992-93 to 2022-23, including Manchester United, so I passed their team name into this function to ensure that it matched the overall number of seasons.

My program doesn't have any time-intensive computational processes, so running it is simple. The graphs in plotters output respectively to two PNGs (already included in the Github, along with PNGs of the sample terminal output attached below): 'all-time-rates' and 'goal-averages.'

RESULTS:

```
Over 31 seasons, a total of 50 teams have competed in the Premier League.

Most successful teams by win percentage:
1: Manchester Utd with a win percentage of 70.0840
2: Arsenal with a win percentage of 64.2017
3: Liverpool with a win percentage of 63.1933
4: Chelsea with a win percentage of 60.5042
5: Manchester City with a win percentage of 59.4000
6: Tottenham with a win percentage of 54.2857
7: Newcastle Utd with a win percentage of 49.2537
8: Blackburn with a win percentage of 48.5632
9: Leeds United with a win percentage of 46.3918
10: Everton with a win percentage of 45.8824

Most successful teams by number of seasons:
1: Arsenal with 31 total seasons in the PL
2: Liverpool with 31 total seasons in the PL
3: Manchester Utd with 31 total seasons in the PL
4: Tottenham with 31 total seasons in the PL
5: Chelsea with 31 total seasons in the PL
6: Everton with 31 total seasons in the PL
7: Newcastle Utd with 28 total seasons in the PL
8: Aston Villa with 28 total seasons in the PL
9: West Ham with 27 total seasons in the PL
10: Manchester City with 26 total seasons in the PL

The average home win-rate in the Premier League across all seasons is 45.930%, compared to an away win-rate of 28.391%.
The season with the lowest home-win rate was 2021 with a home-win rate of 37.895%, which is 3.421% worse than the second-worst season of 2016.
In 2021, the draw rate was 21.842% and the away-win rate was 40.263%.
This is a home-away differential of -2.368%. The average home-away differential across all 31 seasons is 17.539%.
The average number of goals scored in a PL game is 2.6682.
The season with the most average goals per game was 2023 with 2.8526 goals per game.
```

```
Enter a team:
Tottenham
Enter a starting season from 1993 to 2023:
2008
Enter an ending season from 1993 to 2023:
2012

Tottenham played in 5 seasons over that interval: [2008, 2009, 2010, 2011, 2012]

The biggest win interval for Tottenham in those seasons was in the below game:
Tottenham beat Wigan Athletic 9-1 in a home win.
This game happened on 2009-11-22 during week 13 of the 2010 season.

The Tottenham win rate for the 2008 to 2012 seasons is 56.8421%.
```

TESTS:

```
running 2 tests
test test_result_rates ... ok
test test_team_appearances ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.20s
```

GRAPHS: