Pressure Point Patterns: A Look Into Tennis Tactics at The French Open

Broderick J. Pinto

Advisor: Dr. Matt Higham

St. Lawrence University

Department of Math, Computer Science, Statistics, and Data Science

April 29, 2025

Table of contents

Abstract

At the highest level of tennis, a player's mental skillset is as – if not, more – important than their physical and technical skillset. Do the elite players of the sport handle pressure situations differently that sets them apart from the rest of the players? The goal of this paper is to tackle this question by leveraging shot placement data (the exact coordinates of the ball using trajectory-tracking technology) collected during the French Open – the most prestigious clay court tournament in the world. The data preparation process involved data scraping, detail-oriented data cleaning and parsing, and joining and filtering functions. Using the processed data, my goal was to create meaningful visualizations that effectively illustrate how the best players in the world serve and return on the most important points. This paper will highlight that for these high-pressure points, the elite players tend to stick to their strengths and even play more aggressively when serving, however, on the return of serve, elite players generally adopt a more conservative approach to their current strategy.

Introduction

In the game of tennis, its unique scoring system sets it apart from sports with more traditional scoring structures. Tennis scoring can be compared to a Chinese nesting doll: a player must win points to win a game, win games to win a set, and win sets to win a match. This layered system creates several natural "reset" points throughout a match (e.g., after the conclusion of a game or set), offering players a chance to regain momentum if they start poorly. However, it also creates specific points that carry significantly more weight – particularly those late in a set when the score is close. These pressure points – referred to as *important points* later in the paper – are relatively rare but have a disproportionately large impact on the trajectory of an entire match.

The importance of handling these pressure moments cannot be overstated. In the words of Roger Federer during his recent commencement address at Dartmouth College, he explained to the audience that he – one of the the greatest champions in the history of tennis – has only won 54% of the points he played, but he ended up winning nearly 80% of all his matches throughout his career. Winning in tennis is not about dominance at every moment; it is about winning the right points at the right time. This statistic – coming from Federer himself – underscores the idea that the difference between the game's legends and their competitors often lies in their ability to excel in the moments that matter most.

While traditional tennis statistics – like first serve percentage or total winners – offer a high-level view of performance, they rarely capture the nuances of tactical adaptations during pressure points. As a solo sport, the gravitational pull of pressure during big moments is intense, and some players have proven they can handle it better than others. While mental toughness is often cited as the key to thriving under pressure, it has historically been difficult to measure objectively. In Stephanie Kovalchik's study of clutch performance and mental toughness among top tennis athletes, she points out that although mental toughness is frequently credited for strong performances in clutch moments, there has been little objective evidence to back up this belief. This gap – the need to identify and visualize the tactical signs of mental resilience – is precisely what this paper aims to explore.

The goal of this project is to move beyond simple outcome-based measures and use shot placement data – capturing the exact coordinates of ball trajectories – to analyze how elite players such as Federer, Nadal, and Djokovic adapt their strategies under pressure. Rather than asking whether players succeed in important moments, this project focuses on how they succeed: by examining whether they alter their serve placement, shot selection, or return tactics during critical points. Identifying these tactical shifts offers insight into the mental and strategic adjustments that underlie elite performance.

By leveraging tennis tracking technology, point importance scoring models, and visual analysis, this paper brings a data-driven perspective to a topic traditionally discussed in broad, subjective terms. In doing so, it aims to provide objective evidence of how the greatest players in tennis consistently manage to win the points that shape matches – and, by extension, careers. Through careful examination of their shot patterns under pressure, this work reveals the subtle yet powerful ways that mental toughness and strategic clarity manifest themselves during the *most important* moments in tennis.

Data and Methods

Data and Variables of Interest

CourtVision Data

The foundation and motivation for this project lies in the powerful data collected at the French Open using Infosys CourtVision tracking technology. We scraped this data from the Roland Garros (French Open) website from the years of 2019 through 2023. Every match recorded is stored as an individual `.csv` file. In each `.csv` file, each row represents an individual point in the match. In this project, there are data from 180 matches which I combined into one main dataset I called `all_matches` which contains 45,672 rows.

`trajectoryData` Variable

This CourtVision data includes a uniquely important variable, aptly named `trajectoryData`, which includes the exact coordinates (`x`, `y`, `z`) and `position` of every ball hit from the majority of stadium-court and some satellite-court matches at the French Open. The `x` coordinate represents the length of the court, the `y` coordinate represents the width of the court, the `z` coordinate represents the height of the ball above the ground, and `position` refers to the location of the ball when the coordinates are tracked: either at contact (`hit`), at the ball's peak (`peak`), when the ball crosses or hits the net (`net`), when the ball bounces (`bounce`), and the last tracked location of the ball (`last`). All coordinates are measured in meters from the center of the court shown in Figure 1.

The primary focus of this project is on the *shot placement* of the serve and return of serve, meaning we want to use the `x` and `y` coordinates at the `bounce` position on the first and second `hit` of each point to get the serve location and return location, respectively.

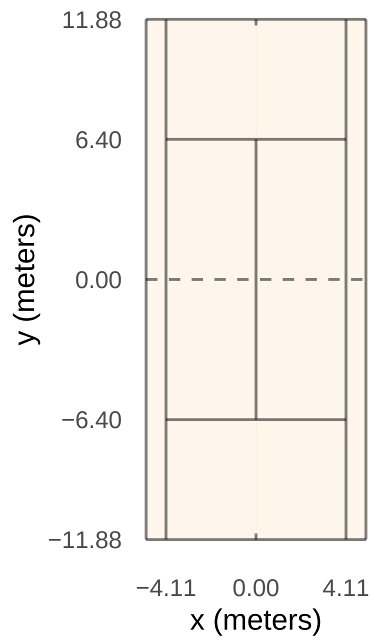

Figure 1: Bird's eye view of a tennis court. The dashed line represents the net. Refer to draw_court() in the Appendix for R code for drawing the scaled court.

Other Variables of Interest

`matchScore` is the score of the match after the point was played. this variable needed a significant amount of processing to get it formatted for the importance joining (see Data Processing section).

`atp_importance` and `wta_importance` are the calculated importance values of the current point based on the current score of the match. These importance values are calculated for every possible tennis score and stored in the `atp_importance` dataset in Kovalchik's `deuce` package. Importance values range from a minimum of 0.0001 and to a maximum of 0.5. Point importance is calculated differently for ATP and WTA matches since ATP French Open matches are best 3 out of 5 sets while WTA French Open matches are best 2 out of 3 sets. Regardless, the calculation for point importance follows the same general probability equation:

$$P(\text{server wins the match} \mid \text{server wins the point}) - P(\text{server wins the match} \mid \text{receiver wins the point}).$$

To characterize points that pass a certain level of importance, I created `atp_is_important` and `wta_is_important` logical variables. I decided – somewhat arbitrarily – that all points with an importance value of 0.1 or higher are *important* (`atp_is_important = TRUE`) and all points with an importance value below 0.1 are *non-important* (`atp_is_important = FALSE`).

`serverId`, `receiverId`, and `scorerId` are the unique identification numbers of the player serving, receiving, and who won the point, respectively. These values are used to join with Jeff Sackmann's player file to get player names instead of identification numbers.

`breakPoint` is another important logical variable I used in this analysis. When the serving player is facing a break point (the server is a point away from getting their serve broken), `breakPoint = TRUE`, otherwise, `breakPoint = FALSE`. Break points typically have a high importance value, especially when the match score is close.

Data Processing

`matchScore` Parsing and Augmentation

The first stage of data processing was parsing the current match score from the `matchScore` variable which initially is stored as a semi-structured data object. The following example represents a score of 6-3, 6-3, 4-0, 40-15:

{‘p1Set1Score’: ‘6’, ‘p1Set2Score’: ‘6’, ‘p1Set3Score’: ‘4’, ‘p1Set4Score’: ‘-1’, ‘p1Set5Score’: ‘-1’, ‘p1Set1TBScore’: ‘-1’, ‘p1Set2TBScore’: ‘-1’, ‘p1Set3TBScore’: ‘-1’, ‘p1Set4TBScore’: ‘-1’, ‘p1Set5TBScore’: ‘-1’, ‘p2Set1Score’: ‘3’, ‘p2Set2Score’: ‘3’, ‘p2Set3Score’: ‘0’, ‘p2Set4Score’: ‘-1’, ‘p2Set5Score’: ‘-1’, ‘p2Set1TBScore’: ‘-1’, ‘p2Set2TBScore’: ‘-1’, ‘p2Set3TBScore’: ‘-1’, ‘p2Set4TBScore’: ‘-1’, ‘p2Set5TBScore’: ‘-1’, ‘p1GameScore’: ‘40’, ‘p2GameScore’: ‘15’}

I parsed this semi-structured data object to obtain the clean form required to join my main `all_matches` dataset with the `atp_importance` dataset. The goal of this step in data processing was to obtain a clean form of the score through a series of steps involving numeric data parsing, filtering and lagging to fix the fact that `matchScore` initially represented the score of the match after the point was played while also handling the issue of first and second serves from the same point being separate rows in `all_matches`, controlling the data types (e.g., having `"GAME"` and `"AD"` as part of the original `matchScore` variable posed several issues), and correcting several small systematic errors in the endings of tiebreak sets (see `clean_point_level()` in the Appendix for the R code). The resulting cleaned score from the above example is illustrated in Table 1.

Table 1: Cleaned and separated match score.

situation	player1_score	player2_score
set1	6	3
set2	6	3
set3	4	0
game	40	15

Importance Joining

Once my `all_matches` dataset contained variables holding the cleaned score of the match, further score processing was required to obtain the score format needed to join with `atp_importance`. The following data processing steps allowed me to create the main `all_matches_importance` dataset with importance values for nearly every point in the `all_matches` dataset – apart from all 2019 matches where the `matchScore` variable was null. I corrected data inconsistencies in player names, augmented the overall match score using the previous set scores of the current match, and combined and reordered the server and receiver scores to follow the tennis scoring convention (i.e., the server's score always comes first). Refer to the Importance Joining section in the Appendix for the R code. The resulting three variables (`game_score`, `set_score`, `match_score`) needed to uniquely identify the current score of the match from the example above are shown in Table 2, and as you might be able to tell from the score, the importance of this particular point is very low.

Table 2: Cleaned match score after joining for importance calculation.

game_score	set_score	match_score	atp_importance	atp_is_important
40-15	4-0	2-0	0.0018507	FALSE

`trajectoryData` Parsing

After cleaning the match score to calculate importance, the `all_matches_importance` dataset is ready for the final step of data processing: `trajectoryData` parsing to obtain shot placement data. The `trajectoryData` variable was initially stored as a JSON object. The following example shows the trajectory data of a missed serve that crossed the net and landed outside of the service box:

[{'x': 11.54, 'y': -1.039, 'z': 2.568, 'position': 'hit'}, {'x': 11.54, 'y': -1.039, 'z': 2.568, 'position': 'peak'}, {'x': 0.0, 'y': 0.915, 'z': 0.946, 'position': 'net'}, {'x': 3.501, 'y': 3.428, 'z': 0.038, 'position': 'bounce'}, {'x': 5.172, 'y': 4.968, 'z': 0.047, 'position': 'last'}]

I parsed this JSON object using the clear delimiters to obtain the cleaner format illustrated in Table 3 (see `clean_shot_level()` in Appendix for the R code).

Table 3: Formatted trajectory data after cleaning, measurements are in meters. For reference, the net height is 0.914 meters in the center (x = 0, y = 0, z = 0.914).

Position	x	y	z
hit	11.540	-1.039	2.568
peak	11.540	-1.039	2.568
net	0.000	0.915	0.946
bounce	3.501	3.428	0.038
last	5.172	4.968	0.047

`filter_matches()` Function

For the purpose of this project, I focused on several case studies of elite champions and other strong competitors. To do so, it was important to have a quick way to access certain matches from the main all_matches_importance dataset. The filter_matches() function takes player and year inputs to select the matches of interest from all_matches_importance, sets the atp_is_important and wta_is_important logical variables, and creates the labels for the visualizations (see filter_matches() section of the Appendix for the R code).

Visualization

The primary method of tactical analysis in this project involves visualizing shot placement on the tennis court. The first step was to obtain the dimensions of a tennis court (Figure 1) to preserve the scale and accurately depict the geometry of the tennis court in two dimensions (see draw_court() for the R code I used to draw the tennis court). I then plotted the shot placement data (for serves and returns) on top of the tennis court ggplot object. To visualize the distribution of serve and return shot placement, I added a 2-dimensional density plot to fill in the tennis court with a color scale corresponding to the concentration of shots.

Finally, I colored the points to compare a player's tactics based on situational importance in two different ways: 1. color points by atp_is_important (or wta_is_important), and 2. color points by breakPoint. Using both of these coloring techniques, I created effective visuals to observe any strategic differences based on the match score and situation. Additionally, I plotted a particular player's matches in a given tournament sequentially, allowing for an opponent-specific strategic analysis as well (see Case Studies section for visuals).

Data Summary Statistics

The data I used for this project are relatively new since CourtVision has only been operational since 2019. In this analysis, I focus on the years of 2020-2023 since these years have fully operational and accurate `matchScore` and `trajectoryData` variables. Table 4 shows some summary statistics by year and in total.

Table 4: Year-by-year data summary statistics.

Year	Total Matches	Total Points	Important Points	% Important Points	Total Shots	Important Shots	% Important Shots
2019	29	5345	0	0.0%	29768	0	0.0%
2020	24	4361	864	19.8%	26168	5578	21.3%
2021	42	7848	1432	18.2%	39957	7389	18.5%
2022	49	9286	1975	21.3%	52393	10960	20.9%
2023	36	6804	1474	21.7%	37416	8145	21.8%
All Years	**180**	**33644**	**5745**	**20.2%**	**185702**	**32072**	**20.6%**

Out of the 180 total matches tracked in this dataset, a total of 33,644 points were recorded. Across those points, 185,702 shots were tracked – yielding an average rally length of approximately 5.5 shots per point. However, not all points carry the same weight. Only 20.2% of all points reached the threshold to be classified as important (i.e., an importance value of 0.1 or higher), meaning just 1 in every 5 points played held greater implications for the outcome of a match. Within those 5,745 important points, players hit a combined 32,072 shots, averaging 5.6 shots per point – a rally length nearly identical to the overall average. Similarly, only 20.6% of all shots hit in the dataset occurred during important points. In short, important points are relatively few and far between, but they represent the critical moments that can ultimately decide the outcome of a match.

Case Studies

Serve Placement Analysis

Rafael Nadal Serves (2022 Title Run)

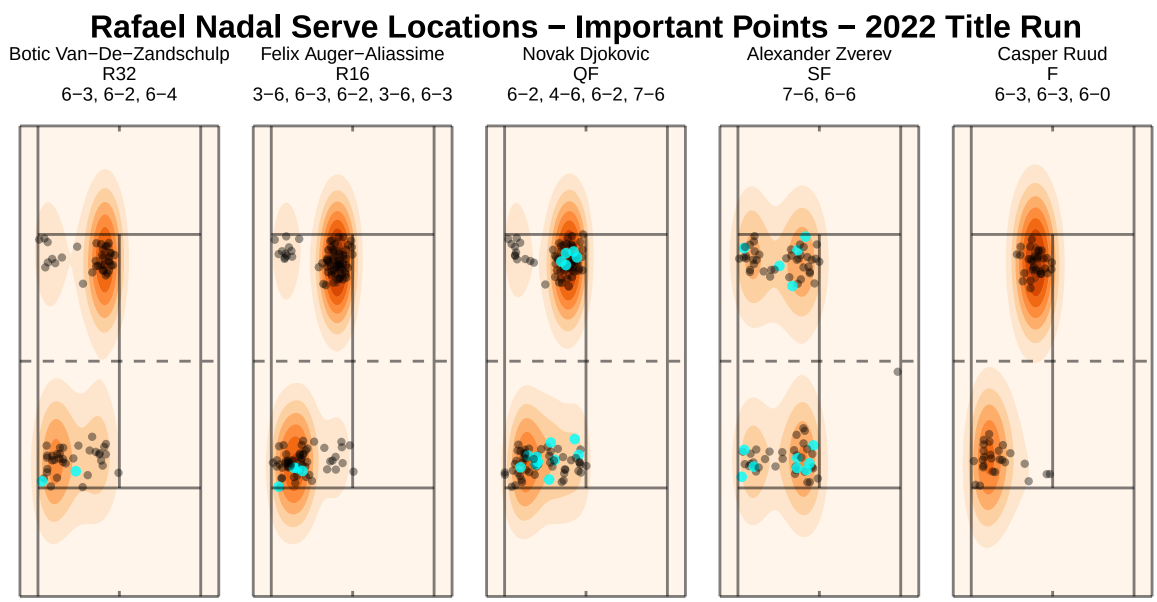

Figure 2: Rafael Nadal's serve placement in his last five rounds to the 2022 French Open title. Blue dots represent serves on important points; black dots represent serves on non-important points. Note that the dots above the net are all serves to the deuce court and the dots below the net are all serves to the ad court.

- Serves on important points: more margin, similar targets

Alexander Zverev Serves (2021 Semifinal Run)

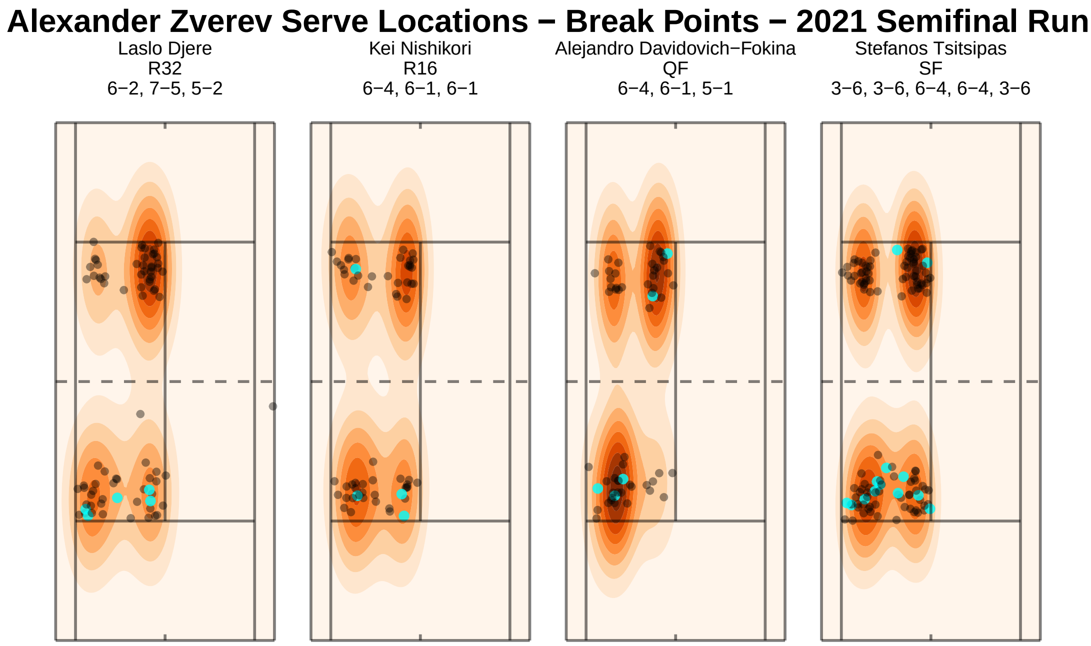

Figure 3: Alexander Zverev's serve placement in his last four rounds to the 2021 French Open semifinal. Blue dots represent serves on break points; black dots represent serves on non-break points.

226 ## Iga Swiatek Serves (2023 Title Run)

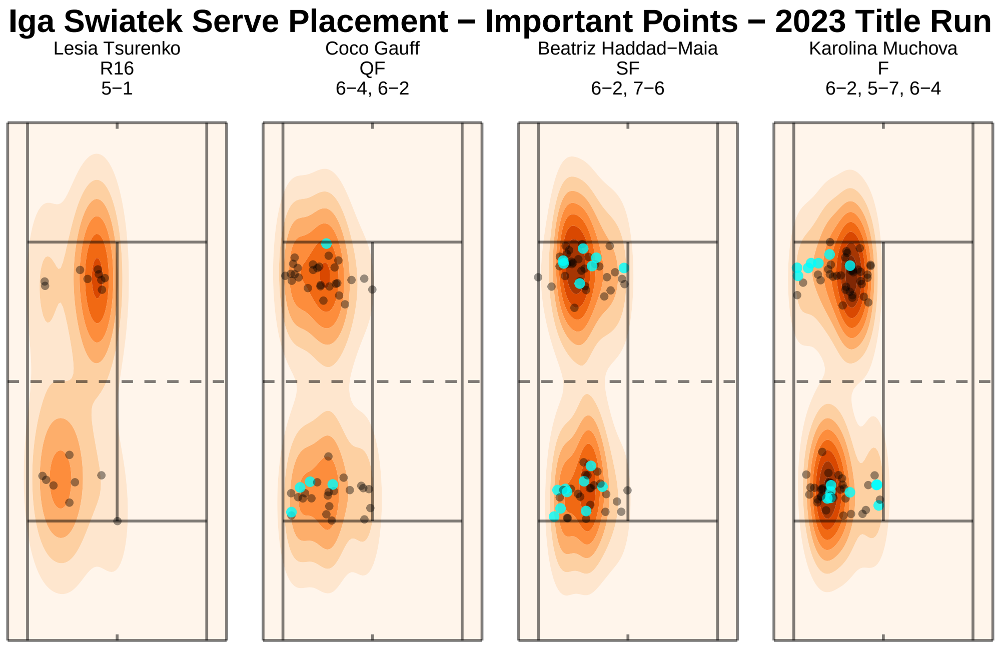

Figure 4: Iga Swiatek's serve placement in her last four rounds to the 2023 French Open title. Blue dots represent serves on important points; black dots represent serves on non-important points.

227 - Consistent center targeting under pressure

Return Placement Analysis

Novak Djokovic Returns (2021 Title Run)

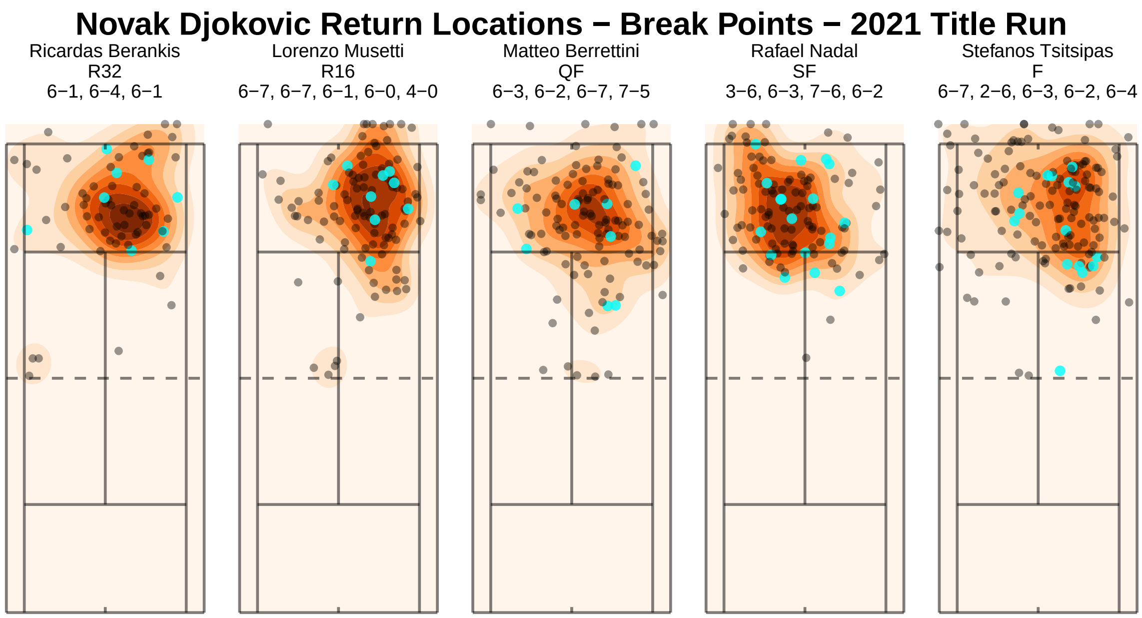

Figure 5: Novak Djokovic's return placement in his last five rounds to the 2021 French Open title. Blue dots represent returns on break points; black dots represent returns on non-break points.

- Elite return depth and accuracy on important points

Stefanos Tsitsipas Returns (2021 Final Run)

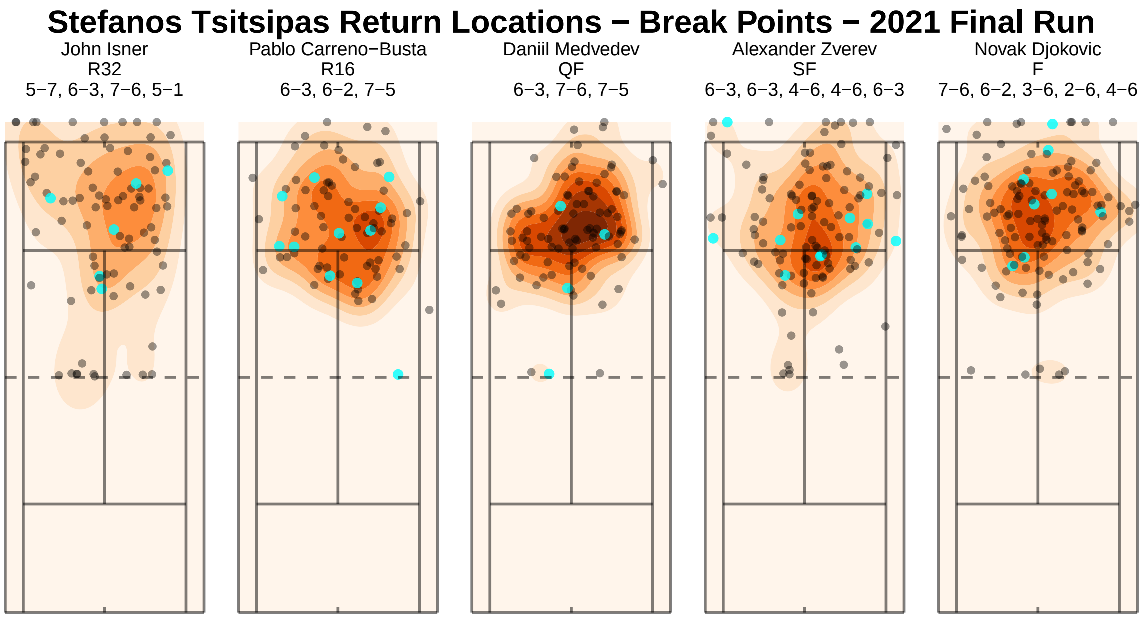

Figure 6: Stefanos Tsitsipas' return placement in his last five rounds to the 2021 French Open final. Blue dots represent returns on break points; black dots represent returns on non-break points.

Coco Gauff Returns (2022 Final Run)

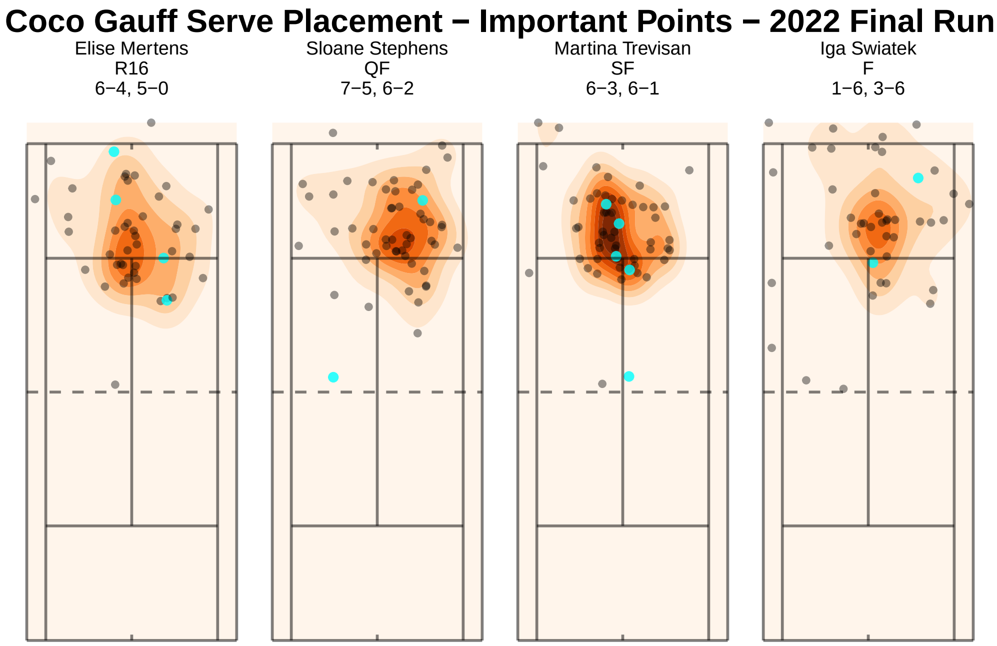

Figure 7: Coco Gauff's return placement in her last four rounds to the 2022 French Open final. Blue dots represent returns on important points; black dots represent returns on non-important points.

Discussion (embed into Case Studies section, general discussion can go into Conclusion)

- Tactical trends: aggression vs. safety, overall tactical change between important / non-important points
- Differences across ATP and WTA examples

Conclusion

- Players adapt differently under pressure:
  - On **serve**, Nadal and Swiatek often maintain their strategy but play *more aggressively* on important points.
  - On **return**, Djokovic prioritizes consistency in high-stakes moments.
- Only ~**20%** of all points are labeled as important, but they have a disproportionately large impact on match outcomes.
- **Shot placement data** allows us to visualize elite players' serve and return patterns in response to pressure, enabling more nuanced performance analysis.
- Future work - rally shots instead of just serve/return

References

- Kovalchik, S. A. (2020). *Measuring Clutch Performance in Professional Tennis.* IJAS.
- deuce R package
- Jeff Sackmann's player and match data
- Infosys Match Centre data

Appendix

`clean_point_level()`

```r
#' Clean Point Level
#'
#' This is a function that cleans the Court Vision data to the
#' *point* level of granularity.
#'
#' @param raw_data is a data frame of a single match of the raw
#' Court Vision data
#' @param player_of_interest is a string of the player's name we want
#' as player1 - first or last name (case insensitive)
#' @return returns a data frame with a row for each point played
#' in the match/matches of interest
#'
#' @examples
#' nadal_final_2022 <- fetch_all_matches(player = "Nadal",
#'                                        year = "2022",
#'                                        round = "F")
#'
#' clean_point_level(nadal_final_2022[[1]])
#'
#' @import tidyverse
#' @export

clean_point_level <- function(raw_data,
                              player_of_interest = "(.|\\s)*\\S(.|\\s)*") {

  all_players <- read_csv("inst/data/all_players.csv")

  second_serve_points <- raw_data |>
    ## must use fetch_all_matches(player, year, round) function to get dataset
    ## with match_id variable
    group_by(match_id) |>
    ## match_info parsing:
    separate(match_info, into = c("label",
                                  "round",
                                  "opponents",
                                  "year",
                                  "court_number",
                                  "file_ending"), sep = "_") |>
    separate(opponents, into = c("player1", "player2"), sep = "-vs-") |>
```

```
  mutate(player1 = sub("-", " ", player1),
         player2 = sub("-", " ", player2)) |>
  mutate(point_index = row_number()) |>
  ## pointId parsing:
  separate(pointId, into = c("set", "game", "point", "serve"), sep = "_") |>
  mutate(set = as.numeric(set),
         game = as.numeric(game),
         point = as.numeric(point),
         serve = as.numeric(serve)) |>
  ## matchScore parsing:
  mutate(matchScore = sub("^.", "", matchScore)) |>
  mutate(matchScore = sub(".$", "", matchScore)) |>
  separate(matchScore, into = c("player1_set1_score",
                                "player1_set2_score",
                                "player1_set3_score",
                                "player1_set4_score",
                                "player1_set5_score",
                                "player1_set1_tbscore",
                                "player1_set2_tbscore",
                                "player1_set3_tbscore",
                                "player1_set4_tbscore",
                                "player1_set5_tbscore",
                                "player2_set1_score",
                                "player2_set2_score",
                                "player2_set3_score",
                                "player2_set4_score",
                                "player2_set5_score",
                                "player2_set1_tbscore",
                                "player2_set2_tbscore",
                                "player2_set3_tbscore",
                                "player2_set4_tbscore",
                                "player2_set5_tbscore",
                                "player1_game_score",
                                "player2_game_score"), sep = ",") |>
  separate(player1_set1_score,
           into = c("label", "player1_set1"),
           sep = ": ") |>
  separate(player1_set2_score,
           into = c("label", "player1_set2"),
           sep = ": ") |>
  separate(player1_set3_score,
           into = c("label", "player1_set3"),
           sep = ": ") |>
  separate(player1_set4_score,
           into = c("label", "player1_set4"),
           sep = ": ") |>
  separate(player1_set5_score,
```

```
           into = c("label", "player1_set5"),
           sep = ": ") |>
  separate(player1_set1_tbscore,
           into = c("label", "player1_set1_tb"),
           sep = ": ") |>
  separate(player1_set2_tbscore,
           into = c("label", "player1_set2_tb"),
           sep = ": ") |>
  separate(player1_set3_tbscore,
           into = c("label", "player1_set3_tb"),
           sep = ": ") |>
  separate(player1_set4_tbscore,
           into = c("label", "player1_set4_tb"),
           sep = ": ") |>
  separate(player1_set5_tbscore,
           into = c("label", "player1_set5_tb"),
           sep = ": ") |>
  separate(player2_set1_score,
           into = c("label", "player2_set1"),
           sep = ": ") |>
  separate(player2_set2_score,
           into = c("label", "player2_set2"),
           sep = ": ") |>
  separate(player2_set3_score,
           into = c("label", "player2_set3"),
           sep = ": ") |>
  separate(player2_set4_score,
           into = c("label", "player2_set4"),
           sep = ": ") |>
  separate(player2_set5_score,
           into = c("label", "player2_set5"),
           sep = ": ") |>
  separate(player2_set1_tbscore,
           into = c("label", "player2_set1_tb"),
           sep = ": ") |>
  separate(player2_set2_tbscore,
           into = c("label", "player2_set2_tb"),
           sep = ": ") |>
  separate(player2_set3_tbscore,
           into = c("label", "player2_set3_tb"),
           sep = ": ") |>
  separate(player2_set4_tbscore,
           into = c("label", "player2_set4_tb"),
           sep = ": ") |>
  separate(player2_set5_tbscore,
           into = c("label", "player2_set5_tb"),
           sep = ": ") |>
```

```
  separate(player1_game_score,
           into = c("label", "player1_game"),
           sep = ": ") |>
  separate(player2_game_score,
           into = c("label", "player2_game"),
           sep = ": ") |>
  mutate(player1_set1 = parse_number(player1_set1),
         player1_set2 = parse_number(player1_set2),
         player1_set3 = parse_number(player1_set3),
         player1_set4 = parse_number(player1_set4),
         player1_set5 = parse_number(player1_set5),
         player2_set1 = parse_number(player2_set1),
         player2_set2 = parse_number(player2_set2),
         player2_set3 = parse_number(player2_set3),
         player2_set4 = parse_number(player2_set4),
         player2_set5 = parse_number(player2_set5),
         player1_game = sub("^.", "", player1_game),
         player1_game = sub(".$", "", player1_game),
         player2_game = sub("^.", "", player2_game),
         player2_game = sub(".$", "", player2_game)) |>
  filter(serve == 2)

formatted_point_level <- raw_data |>
  ## must use fetch_all_matches(player, year, round) function to get dataset
  ## with match_id variable
  group_by(match_id) |>
  ## match_info parsing:
  separate(match_info, into = c("label", "round", "opponents",
                              "year", "court_number", "file_ending"),
           sep = "_") |>
  separate(opponents, into = c("player1", "player2"), sep = "-vs-") |>
  mutate(player1 = sub("-", " ", player1),
         player2 = sub("-", " ", player2)) |>
  mutate(point_index = row_number()) |>
  ## pointId parsing:
  separate(pointId, into = c("set", "game", "point", "serve"), sep = "_") |>
  mutate(set = as.numeric(set),
         game = as.numeric(game),
         point = as.numeric(point),
         serve = as.numeric(serve)) |>
  ## matchScore parsing:
  mutate(matchScore = sub("^.", "", matchScore)) |>
  mutate(matchScore = sub(".$", "", matchScore)) |>
  separate(matchScore, into = c("player1_set1_score",
                              "player1_set2_score",
                              "player1_set3_score",
                              "player1_set4_score",
```

```
                                   "player1_set5_score",
                                   "player1_set1_tbscore",
                                   "player1_set2_tbscore",
                                   "player1_set3_tbscore",
                                   "player1_set4_tbscore",
                                   "player1_set5_tbscore",
                                   "player2_set1_score",
                                   "player2_set2_score",
                                   "player2_set3_score",
                                   "player2_set4_score",
                                   "player2_set5_score",
                                   "player2_set1_tbscore",
                                   "player2_set2_tbscore",
                                   "player2_set3_tbscore",
                                   "player2_set4_tbscore",
                                   "player2_set5_tbscore",
                                   "player1_game_score",
                                   "player2_game_score"), sep = ",") |>
    separate(player1_set1_score, into = c("label", "player1_set1"),
             sep = ": ") |>
    separate(player1_set2_score, into = c("label", "player1_set2"),
             sep = ": ") |>
    separate(player1_set3_score, into = c("label", "player1_set3"),
             sep = ": ") |>
    separate(player1_set4_score, into = c("label", "player1_set4"),
             sep = ": ") |>
    separate(player1_set5_score, into = c("label", "player1_set5"),
             sep = ": ") |>
    separate(player1_set1_tbscore, into = c("label", "player1_set1_tb"),
             sep = ": ") |>
    separate(player1_set2_tbscore, into = c("label", "player1_set2_tb"),
             sep = ": ") |>
    separate(player1_set3_tbscore, into = c("label", "player1_set3_tb"),
             sep = ": ") |>
    separate(player1_set4_tbscore, into = c("label", "player1_set4_tb"),
             sep = ": ") |>
    separate(player1_set5_tbscore, into = c("label", "player1_set5_tb"),
             sep = ": ") |>
    separate(player2_set1_score, into = c("label", "player2_set1"),
             sep = ": ") |>
    separate(player2_set2_score, into = c("label", "player2_set2"),
             sep = ": ") |>
    separate(player2_set3_score, into = c("label", "player2_set3"),
             sep = ": ") |>
    separate(player2_set4_score, into = c("label", "player2_set4"),
             sep = ": ") |>
    separate(player2_set5_score, into = c("label", "player2_set5"),
```

```
           sep = ": ") |>
  separate(player2_set1_tbscore, into = c("label", "player2_set1_tb"),
           sep = ": ") |>
  separate(player2_set2_tbscore, into = c("label", "player2_set2_tb"),
           sep = ": ") |>
  separate(player2_set3_tbscore, into = c("label", "player2_set3_tb"),
           sep = ": ") |>
  separate(player2_set4_tbscore, into = c("label", "player2_set4_tb"),
           sep = ": ") |>
  separate(player2_set5_tbscore, into = c("label", "player2_set5_tb"),
           sep = ": ") |>
  separate(player1_game_score, into = c("label", "player1_game"),
           sep = ": ") |>
  separate(player2_game_score, into = c("label", "player2_game"),
           sep = ": ") |>
  mutate(player1_set1 = parse_number(player1_set1),
         player1_set2 = parse_number(player1_set2),
         player1_set3 = parse_number(player1_set3),
         player1_set4 = parse_number(player1_set4),
         player1_set5 = parse_number(player1_set5),
         player2_set1 = parse_number(player2_set1),
         player2_set2 = parse_number(player2_set2),
         player2_set3 = parse_number(player2_set3),
         player2_set4 = parse_number(player2_set4),
         player2_set5 = parse_number(player2_set5),
         player1_game = sub("^.", "", player1_game),
         player1_game = sub(".$", "", player1_game),
         player2_game = sub("^.", "", player2_game),
         player2_game = sub(".$", "", player2_game)) |>
  filter(serve == 1) |>
  ## lag the game score to get accurate game score:
  mutate(player1_game_lag = lag(player1_game, default = "0"),
         player2_game_lag = lag(player2_game, default = "0")) |>
  ## fill in the second serve points:
  bind_rows(second_serve_points) |>
  arrange(set, game, point, serve) |>
  fill(player1_game_lag, player2_game_lag, .direction = "down") |>
  mutate(player1_game_score = if_else(player1_game_lag == "GAME" |
                                        player2_game_lag == "GAME",
                                      true = "0",
                                      false = player1_game_lag)) |>
  mutate(player2_game_score = if_else(player1_game_lag == "GAME" |
                                        player2_game_lag == "GAME",
                                      true = "0",
                                      false = player2_game_lag)) |>
  ## fix tiebreak ending:
  mutate(
```

```
    # Safely convert to numeric, assigning NA if conversion fails
    player1_score_numeric = suppressWarnings(
      as.numeric(player1_game_score)),
    player2_score_numeric = suppressWarnings(
      as.numeric(player2_game_score)),
    # Create a flag for the condition
    reset_scores = !(player1_game_score %in% c("0", "15", "30", "40")) &
      !(player2_game_score %in% c("0", "15", "30", "40")) &
      !is.na(player1_score_numeric) & !is.na(player2_score_numeric) &
      player1_score_numeric + player2_score_numeric >= 12 &
      abs(player1_score_numeric - player2_score_numeric) == 2,
    # Use the flag to set both scores
    player1_game_score = if_else(reset_scores, "0", player1_game_score),
    player2_game_score = if_else(reset_scores, "0", player2_game_score)) |>

  ## set score lag:
  mutate(player1_set1_lag = ifelse(serve == 2,
                                   lag(player1_set1, 2, default = 0),
                                   lag(player1_set1, default = 0)),
         player1_set2_lag = ifelse(serve == 2,
                                   lag(player1_set2, 2, default = 0),
                                   lag(player1_set2, default = 0)),
         player1_set3_lag = ifelse(serve == 2,
                                   lag(player1_set3, 2, default = 0),
                                   lag(player1_set3, default = 0)),
         player1_set4_lag = ifelse(serve == 2,
                                   lag(player1_set4, 2, default = 0),
                                   lag(player1_set4, default = 0)),
         player1_set5_lag = ifelse(serve == 2,
                                   lag(player1_set5, 2, default = 0),
                                   lag(player1_set5, default = 0)),
         player2_set1_lag = ifelse(serve == 2,
                                   lag(player2_set1, 2, default = 0),
                                   lag(player2_set1, default = 0)),
         player2_set2_lag = ifelse(serve == 2,
                                   lag(player2_set2, 2, default = 0),
                                   lag(player2_set2, default = 0)),
         player2_set3_lag = ifelse(serve == 2,
                                   lag(player2_set3, 2, default = 0),
                                   lag(player2_set3, default = 0)),
         player2_set4_lag = ifelse(serve == 2,
                                   lag(player2_set4, 2, default = 0),
                                   lag(player2_set4, default = 0)),
         player2_set5_lag = ifelse(serve == 2,
                                   lag(player2_set5, 2, default = 0),
                                   lag(player2_set5, default = 0))) |>
  mutate(player1_set_score = case_when(set == 1 ~ player1_set1_lag,
```

```
                                          set == 2 ~ player1_set2_lag,
                                          set == 3 ~ player1_set3_lag,
                                          set == 4 ~ player1_set4_lag,
                                          set == 5 ~ player1_set5_lag)) |>
  mutate(player2_set_score = case_when(set == 1 ~ player2_set1_lag,
                                          set == 2 ~ player2_set2_lag,
                                          set == 3 ~ player2_set3_lag,
                                          set == 4 ~ player2_set4_lag,
                                          set == 5 ~ player2_set5_lag)) |>

  # Replace serverId, scorerId, receiverId with player names instead of ids
  left_join(all_players, by = c("serverId" = "id")) |>
  mutate(serverId = fullName) |>
  select(-fullName) |>
  left_join(all_players, by = c("scorerId" = "id")) |>
  mutate(scorerId = fullName) |>
  select(-fullName) |>
  left_join(all_players, by = c("receiverId" = "id")) |>
  mutate(receiverId = fullName) |>

  # Store original player names and scores
  mutate(
    original_player1 = player1,
    original_player2 = player2,
    original_player1_game_score = player1_game_score,
    original_player2_game_score = player2_game_score,
    original_player1_set_score = player1_set_score,
    original_player2_set_score = player2_set_score) |>

  # Rearrange players and scores based on whether they match
  # the player_of_interest
  mutate(
    player1 = case_when(
      str_detect(str_to_lower(original_player1),
                 str_to_lower(player_of_interest)) ~ original_player1,
      str_detect(str_to_lower(original_player2),
                 str_to_lower(player_of_interest)) ~ original_player2),
    player2 = case_when(
      str_detect(str_to_lower(original_player1),
                 str_to_lower(player_of_interest)) ~ original_player2,
      str_detect(str_to_lower(original_player2),
                 str_to_lower(player_of_interest)) ~ original_player1),
    player1_game_score = case_when(
      str_detect(str_to_lower(original_player1),
                 str_to_lower(player_of_interest))
      ~ original_player1_game_score,
      str_detect(str_to_lower(original_player2),
```

```
                   str_to_lower(player_of_interest))
        ~ original_player2_game_score),
      player2_game_score = case_when(
        str_detect(str_to_lower(original_player1),
                   str_to_lower(player_of_interest))
        ~ original_player2_game_score,
        str_detect(str_to_lower(original_player2),
                   str_to_lower(player_of_interest))
        ~ original_player1_game_score),
      player1_set_score = case_when(
        str_detect(str_to_lower(original_player1),
                   str_to_lower(player_of_interest))
        ~ original_player1_set_score,
        str_detect(str_to_lower(original_player2),
                   str_to_lower(player_of_interest))
        ~ original_player2_set_score),
      player2_set_score = case_when(
        str_detect(str_to_lower(original_player1),
                   str_to_lower(player_of_interest))
        ~ original_player2_set_score,
        str_detect(str_to_lower(original_player2),
                   str_to_lower(player_of_interest))
        ~ original_player1_set_score)) |>
    relocate(set, player1_game_score, player2_game_score,
             player1_set_score, player2_set_score, player1, player2)

  return(formatted_point_level)
}
```

Importance Joining

```
## Join all_matches and atp_importance
join_ready_df <- all_matches |>

  ## Correct Player Names
  mutate(
    serverId = case_when(
      serverId == "Cori Gauff" ~ "Coco Gauff",
      serverId == "Alejandro Davidovich Fokina"
      ~ "Alejandro Davidovich-Fokina",
      serverId == "Tomas Martin Etcheverry" ~ "Tomas Martin-Etcheverry",
      serverId == "Beatriz Haddad Maia" ~ "Beatriz Haddad-Maia",
```

```r
    serverId == "Pablo Carreno Busta" ~ "Pablo Carreno-Busta",
    serverId == "Bernabe Zapata Miralles" ~ "Bernabe Zapata-Miralles",
    serverId == "Anna Karolina Schmiedlova" ~ "Anna Karolina-Schmiedlova",
    serverId == "Jan-Lennard Struff" ~ "Jan Lennard-Struff",
    serverId == "Irina-Camelia Begu" ~ "Irina Camelia-Begu",
    serverId == "Juan Pablo Varillas" ~ "Juan Pablo-Varillas",
    serverId == "Sara Sorribes Tormo" ~ "Sara Sorribes-Tormo",
    serverId == "Botic Van De Zandschulp" ~ "Botic Van-De-Zandschulp",
    serverId == "Genaro Alberto Olivieri" ~ "Genaro Alberto-Olivieri",
    serverId == "Thiago Seyboth Wild" ~ "Thiago Seyboth-Wild",
    TRUE ~ serverId
  ),
  receiverId = case_when(
    receiverId == "Cori Gauff" ~ "Coco Gauff",
    receiverId == "Alejandro Davidovich Fokina"
    ~ "Alejandro Davidovich-Fokina",
    receiverId == "Tomas Martin Etcheverry"
    ~ "Tomas Martin-Etcheverry",
    receiverId == "Beatriz Haddad Maia" ~ "Beatriz Haddad-Maia",
    receiverId == "Pablo Carreno Busta" ~ "Pablo Carreno-Busta",
    receiverId == "Bernabe Zapata Miralles" ~ "Bernabe Zapata-Miralles",
    receiverId == "Anna Karolina Schmiedlova" ~ "Anna Karolina-Schmiedlova",
    receiverId == "Jan-Lennard Struff" ~ "Jan Lennard-Struff",
    receiverId == "Irina-Camelia Begu" ~ "Irina Camelia-Begu",
    receiverId == "Juan Pablo Varillas" ~ "Juan Pablo-Varillas",
    receiverId == "Sara Sorribes Tormo" ~ "Sara Sorribes-Tormo",
    receiverId == "Botic Van De Zandschulp" ~ "Botic Van-De-Zandschulp",
    receiverId == "Genaro Alberto Olivieri" ~ "Genaro Alberto-Olivieri",
    receiverId == "Thiago Seyboth Wild" ~ "Thiago Seyboth-Wild",
    TRUE ~ receiverId
  ),
  scorerId = case_when(
    scorerId == "Cori Gauff" ~ "Coco Gauff",
    scorerId == "Alejandro Davidovich Fokina" ~ "Alejandro Davidovich-Fokina",
    scorerId == "Tomas Martin Etcheverry" ~ "Tomas Martin-Etcheverry",
    scorerId == "Beatriz Haddad Maia" ~ "Beatriz Haddad-Maia",
    scorerId == "Pablo Carreno Busta" ~ "Pablo Carreno-Busta",
    scorerId == "Bernabe Zapata Miralles" ~ "Bernabe Zapata-Miralles",
    scorerId == "Anna Karolina Schmiedlova" ~ "Anna Karolina-Schmiedlova",
    scorerId == "Jan-Lennard Struff" ~ "Jan Lennard-Struff",
    scorerId == "Irina-Camelia Begu" ~ "Irina Camelia-Begu",
    scorerId == "Juan Pablo Varillas" ~ "Juan Pablo-Varillas",
    scorerId == "Sara Sorribes Tormo" ~ "Sara Sorribes-Tormo",
    scorerId == "Botic Van De Zandschulp" ~ "Botic Van-De-Zandschulp",
    scorerId == "Genaro Alberto Olivieri" ~ "Genaro Alberto-Olivieri",
    scorerId == "Thiago Seyboth Wild" ~ "Thiago Seyboth-Wild",
    TRUE ~ scorerId
```

```
  )
) |>
group_by(match_id) |>
mutate(server_game_score = case_when(serverId == player1
                                     ~ player1_game_score,
                                     serverId == player2
                                     ~ player2_game_score),
       receiver_game_score = case_when(receiverId == player1
                                       ~ player1_game_score,
                                       receiverId == player2
                                       ~ player2_game_score),
       server_set_score = case_when(serverId == player1
                                    ~ player1_set_score,
                                    serverId == player2
                                    ~ player2_set_score),
       receiver_set_score = case_when(receiverId == player1
                                      ~ player1_set_score,
                                      receiverId == player2
                                      ~ player2_set_score)) |>
mutate(is_tiebreak = if_else(server_set_score == 6
                             & receiver_set_score == 6,
                             true = TRUE, false = FALSE)) |>
relocate(server_game_score, receiver_game_score,
         server_set_score, receiver_set_score, is_tiebreak) |>

mutate(
  server_game_score2 = case_when(
    (server_game_score == "AD" & receiver_game_score == "40") ~ "40",
    (server_game_score == "40" & receiver_game_score == "AD") ~ "30",
    TRUE ~ server_game_score
  ),
  receiver_game_score2 = case_when(
    (receiver_game_score == "AD" & server_game_score == "40") ~ "40",
    (receiver_game_score == "40" & server_game_score == "AD") ~ "30",
    TRUE ~ receiver_game_score
  )
) |>
mutate(server_game_score = server_game_score2,
       receiver_game_score = receiver_game_score2) |>

mutate(server_game_score = as.numeric(server_game_score),
       receiver_game_score = as.numeric(receiver_game_score)) |>


## Calculate match scores
mutate(player1_match_score = 0,
       player2_match_score = 0) |>
```

```
mutate(player1_match_score = case_when(
  set == 1 ~ 0,
  (player1_set1_lag >= 6 & player1_set1_lag > player2_set1_lag)
  ~ (player1_match_score + 1),
  (player1_set2_lag >= 6 & player1_set2_lag > player2_set2_lag)
  ~ (player1_match_score + 1),
  (player1_set3_lag >= 6 & player1_set3_lag > player2_set3_lag)
  ~ (player1_match_score + 1),
  (player1_set4_lag >= 6 & player1_set4_lag > player2_set4_lag)
  ~ (player1_match_score + 1),
  (player1_set5_lag >= 6 & player1_set5_lag > player2_set5_lag)
  ~ (player1_match_score + 1),
  TRUE ~ player1_match_score)) |>
mutate(player2_match_score = case_when(
  set == 1 ~ 0,
  (player2_set1_lag >= 6 & player2_set1_lag > player1_set1_lag)
  ~ (player2_match_score + 1),
  (player2_set2_lag >= 6 & player2_set2_lag > player1_set2_lag)
  ~ (player2_match_score + 1),
  (player2_set3_lag >= 6 & player2_set3_lag > player1_set3_lag)
  ~ (player2_match_score + 1),
  (player2_set4_lag >= 6 & player2_set4_lag > player1_set4_lag)
  ~ (player2_match_score + 1),
  (player2_set5_lag >= 6 & player2_set5_lag > player1_set5_lag)
  ~ (player2_match_score + 1),
  TRUE ~ player2_match_score)) |>
mutate(server_match_score = case_when(serverId == player1
                                      ~ player1_match_score,
                                      serverId == player2
                                      ~ player2_match_score),
       receiver_match_score = case_when(receiverId == player1
                                        ~ player1_match_score,
                                        receiverId == player2
                                        ~ player2_match_score)) |>

mutate(score_diff = if_else(is_tiebreak == TRUE,
                            if_else(pmax(server_game_score,
                                         receiver_game_score) > 6,
                                    pmax(server_game_score,
                                         receiver_game_score) - 6, 0),
                            0)) |>
mutate(server_game_score = server_game_score - score_diff,
       receiver_game_score = receiver_game_score - score_diff) |>

## Flip server and receiver scores for game_score and set_score
mutate(
  # Preserve original values
```

```r
    server_game_score_og = server_game_score,
    receiver_game_score_og = receiver_game_score,
    server_set_score_og = server_set_score,
    receiver_set_score_og = receiver_set_score,

    # Swap server and receiver scores
    server_game_score = receiver_game_score_og,
    receiver_game_score = server_game_score_og,
    server_set_score = receiver_set_score_og,
    receiver_set_score = server_set_score_og
  ) |>

  ## Combine scores
  mutate(game_score = paste(server_game_score, receiver_game_score,
                            sep = "-"),
         set_score = paste(server_set_score, receiver_set_score, sep = "-"),
         match_score = paste(server_match_score, receiver_match_score,
                             sep = "-")) |>

  ## Handle AD-40 and 40-AD game scores:
  mutate(game_score = case_when(game_score == "AD-40" ~ "40-30",
                                game_score == "40-AD" ~ "30-40",
                                set_score == "0-0" &
                                  !(game_score %in% c("0-0", "0-15",
                                                      "0-30", "0-40",
                                                      "15-0", "15-15",
                                                      "15-30", "15-40",
                                                      "30-0", "30-15",
                                                      "30-30", "30-40",
                                                      "40-0", "40-15",
                                                      "40-30", "40-40"))
                                ~ "0-0",
                                TRUE ~ game_score)) |>
  relocate(server_game_score, receiver_game_score,
           game_score, server_set_score, receiver_set_score,
           set_score, server_match_score, receiver_match_score, match_score)

atp_importance_5 <- atp_importance |>
  filter(bestof == 5) |>
  distinct(point_score, game_score, set_score, .keep_all = TRUE) |>
  mutate(atp_importance = importance) |>
  select(-importance)

wta_importance_3 <- atp_importance |>
  filter(bestof == 3) |>
  distinct(point_score, game_score, set_score, .keep_all = TRUE) |>
  mutate(wta_importance = importance) |>
```

```r
  select(-importance)

all_matches_importance <- join_ready_df |>
  left_join(atp_importance_5, by = c("game_score" = "point_score",
                                     "set_score" = "game_score",
                                     "match_score" = "set_score")) |>
  left_join(wta_importance_3, by = c("game_score" = "point_score",
                                     "set_score" = "game_score",
                                     "match_score" = "set_score")) |>
  relocate(game_score, set_score, match_score, atp_importance, wta_importance)
```

clean_shot_level()

```r
#' Clean Shot Level Function
#'
#' This is a function that parses the trajectory data from the
#' Court Vision data - breaking the match/matches down to the
#' *shot* level of granularity.
#'
#' @param cleaned_data is a data frame of cleaned point-level data
#' @return returns a data frame with several rows (hit, net, peak, bounce) for
#' each shot in the match/matches of interest
#'
#' @examples
#' nadal_2022_cleaned <- clean_and_combine(nadal_2022,
#'                                          player_interest = "Nadal")
#' clean_shot_level(nadal_2022_cleaned)
#'
#' @import tidyverse
#' @export

clean_shot_level <- function(cleaned_data) {
  formatted_shot_level <- cleaned_data |>
    ## trajectoryData parsing:
    mutate(trajectoryData = sub("^..", "", trajectoryData)) |>
    mutate(trajectoryData = sub("..$", "", trajectoryData)) |>
    separate_longer_delim(trajectoryData, delim = "}, {") |>
    group_by(point_index) |>
    mutate(shot_index = row_number()) |>
    separate(trajectoryData, into = c("x", "y", "z", "position"),
             sep = "\\,") |>
    mutate(x = parse_number(x),
```

```r
          y = parse_number(y),
          z = parse_number(z),
          position = sub("^............", "", position)) |>
    mutate(position = gsub("'", "", position)) |>
    ## player_hit variable construction:
    mutate(is_hit = if_else(position == "hit", true = 1, false = 0)) |>
    group_by(point_index) |>
    mutate(hit_count = cumsum(is_hit)) |>
    mutate(player_hit = if_else(hit_count %% 2 == 1, serverId, receiverId)) |>
    ## net_height and net_clearance variables:
    mutate(net_height = 0.00619 * (y^2) + 0.914) |>
    mutate(net_clearance = z - net_height) |>
    relocate(player1_game_score, player2_game_score, player1_set_score,
             player2_set_score, player1, player2, x, y, z, position,
             hit_count, net_clearance)

  return(formatted_shot_level)
}
```

draw_court()

```r
#' Draw Court
#'
#' This is a function that draws the tennis court (dimensions are to scale)
#'
#' @return returns ggplot layers drawing solid lines representing the lines
#' on the tennis court, dashed line represents the net
#'
#' @examples
#' draw_court()
#'
#' @import tidyverse
#' @export

draw_court <- function() {
  list(
    annotate(geom = "segment", x = 5.02, xend = 5.02,
             y = -11.88, yend = 11.88, alpha = 0.5),
    annotate(geom = "segment", x = 4.11, xend = 4.11,
             y = -11.88, yend = 11.88, alpha = 0.5),
    annotate(geom = "segment", x = -5.02, xend = -5.02,
             y = -11.88, yend = 11.88, alpha = 0.5),
```

```r
    annotate(geom = "segment", x = -4.11, xend = -4.11,
             y = -11.88, yend = 11.88, alpha = 0.5),
    annotate(geom = "segment", x = 0, xend = 0, y = -6.4,
             yend = 6.4, alpha = 0.5),
    annotate(geom = "segment", y = 0, yend = 0,
             x = -5.02, xend = 5.02, linetype = 2, alpha = 0.5),
    annotate(geom = "segment", y = -11.88, yend = -11.88,
             x = -5.02, xend = 5.02, alpha = 0.5),
    annotate(geom = "segment", y = 11.88, yend = 11.88,
             x = -5.02, xend = 5.02, alpha = 0.5),
    annotate(geom = "segment", y = -6.4, yend = -6.4,
             x = -4.11, xend = 4.11, alpha = 0.5),
    annotate(geom = "segment", y = 6.4, yend = 6.4,
             x = -4.11, xend = 4.11, alpha = 0.5),
    annotate(geom = "segment", x = 0, xend = 0,
             y = -11.88, yend = -11.6, alpha = 0.5),
    annotate(geom = "segment", x = 0, xend = 0,
             y = 11.88, yend = 11.6, alpha = 0.5),
    theme_void(),
    coord_fixed()
  )
}
```

258 **filter_matches()**

```r
#' Filter Matches Function
#'
#' This is a function that finds all matches of a specified player, year,
#' and/or round using the all_matches_importance data frame
#'
#' @param player is a string of the player's name,
#' first and last name (case sensitive)
#' @param year_of_interest is a string of the year the match was played,
#' between 2019 and 2023
#' @return returns a point-level data frame of all matches given the
#' specified player and year
#'
#' @import tidyverse
#' @export

filter_matches <- function(player = "(.|\\s)*\\S(.|\\s)*",
                           year_of_interest = "(.|\\s)*\\S(.|\\s)*") {
```

```r
filtered_df <- all_matches_importance |>
  # is_important variables
  mutate(atp_is_important = if_else(atp_importance >= 0.1, 1, 0),
         atp_is_important = as.logical(atp_is_important),
         wta_is_important = if_else(wta_importance >= 0.1, 1, 0),
         wta_is_important = as.logical(wta_is_important)) |>

  # Filter based on the parameters of the function
  filter(player1 == player | player2 == player) |>
  filter(year == year_of_interest) |>

  # Parse and combine match_score_overall for plot label
  mutate(
    set1_score = if_else(player == player1,
                         paste(player1_set1, player2_set1, sep = "-"),
                         paste(player2_set1, player1_set1, sep = "-")),
    set2_score = if_else(player == player1,
                         paste(player1_set2, player2_set2, sep = "-"),
                         paste(player2_set2, player1_set2, sep = "-")),
    set3_score = if_else(player == player1,
                         paste(player1_set3, player2_set3, sep = "-"),
                         paste(player2_set3, player1_set3, sep = "-")),
    set4_score = if_else(player == player1,
                         paste(player1_set4, player2_set4, sep = "-"),
                         paste(player2_set4, player1_set4, sep = "-")),
    set5_score = if_else(player == player1,
                         paste(player1_set5, player2_set5, sep = "-"),
                         paste(player2_set5, player1_set5, sep = "-")),
    match_score_overall = pmap_chr(
      list(set1_score, set2_score, set3_score, set4_score, set5_score),
      ~ str_c(
        discard(
          c(...),
          ~ str_count(.x, "-") != 1),
        collapse = ", "
      )
    )
  ) |>

  # Store original player names and scores
  mutate(
    original_player1 = player1,
    original_player2 = player2,
    original_player1_game_score = player1_game_score,
    original_player2_game_score = player2_game_score,
    original_player1_set_score = player1_set_score,
```

```
    original_player2_set_score = player2_set_score) |>

  # Rearrange players and scores based on whether they match the player
  mutate(
    player1 = case_when(
      str_detect(str_to_lower(original_player1), str_to_lower(player))
      ~ original_player1,
      str_detect(str_to_lower(original_player2), str_to_lower(player))
      ~ original_player2),
    player2 = case_when(
      str_detect(str_to_lower(original_player1), str_to_lower(player))
      ~ original_player2,
      str_detect(str_to_lower(original_player2), str_to_lower(player))
      ~ original_player1),
    player1_game_score = case_when(
      str_detect(str_to_lower(original_player1), str_to_lower(player))
      ~ original_player1_game_score,
      str_detect(str_to_lower(original_player2), str_to_lower(player))
      ~ original_player2_game_score),
    player2_game_score = case_when(
      str_detect(str_to_lower(original_player1), str_to_lower(player))
      ~ original_player2_game_score,
      str_detect(str_to_lower(original_player2), str_to_lower(player))
      ~ original_player1_game_score),
    player1_set_score = case_when(
      str_detect(str_to_lower(original_player1), str_to_lower(player))
      ~ original_player1_set_score,
      str_detect(str_to_lower(original_player2), str_to_lower(player))
      ~ original_player2_set_score),
    player2_set_score = case_when(
      str_detect(str_to_lower(original_player1), str_to_lower(player))
      ~ original_player2_set_score,
      str_detect(str_to_lower(original_player2), str_to_lower(player))
      ~ original_player1_set_score)) |>

  # Create plot_label variable
  mutate(plot_label = str_c(player2, round, match_score_overall,
                            sep = "\n")) |>

  # Add numeric round before grouping
  mutate(round = factor(round,
                        levels = c("R64", "R32", "R16", "QF", "SF", "F")),
         round_num = as.numeric(round)) |>

  # Create plot label and final plot label per match
  mutate(plot_label = str_c(player2, round, match_score_overall,
                            sep = "\n")) |>
```

```
    group_by(match_id) |>
    mutate(plot_label_final = last(plot_label[!is.na(plot_label)])) |>
    ungroup() |>
    mutate(
      plot_label_final = as_factor(plot_label_final),
      plot_label_final = fct_reorder(plot_label_final, round_num)
    ) |>

    relocate(plot_label_final, set, player1_game_score, player2_game_score,
             player1_set_score, player2_set_score, player1, player2)

  return(filtered_df)

}
```