Data Bootcamp Final Project

Chengzhi Wang

May 12, 2025

Link to Code: DataBootcamp_Final Project_Chengzhi Wang.ipynb

Link to Github: https://github.com/ChengzhiWang507/Data-Bootcamp-Final.git

I. Introduction

A. Overview of the data

In the past few years, I have been playing more and more tennis, so for the final project, I was curious what determines whether or not a tennis player would win a match. Thus, it makes sense to analyze the professional game, both because there is data online about it and also that it represents a level where players have more consistency and the data should have less variance. Specifically, I found an online dataset with data from professional matches dating back to 2000 and to the end of 2017. This dataset contained a vast variety of information from in-game statistics to pre-game seeding, ranking, and height of players, as well as much more information like names, location, and tournament level. There will be more information below on the actual specifics of the data in II. Data Description.

B. Predictive Task

Specifically, I want to predict both which player will win and also how many games it will take them to win. In a best of 5 game, the winner could win in 3, 4, or 5 games and that's important as it's an indicator of how close the game was and also has effects on future games, because playing more leads to more fatigue. Thus, this makes the number of games needed to win also something that should be predicted. In III. Models and IV. Results and Implementation, I will go more in depth about how I went about actually predicting both who will win and how many games it takes.

C. Summary Findings

The Logistic Regression seemed to work best when predicting which player will win and the neural network seemed to work best in predicting how many sets a player needed to win.

II. Data Description: Data source and description

A. Data Source

https://www.kaggle.com/datasets/gmadevs/atp-matches-dataset

Contains ATP matches from 2000-2017

B. Description

i. Some of the columns that came with the data (sample of important columns)

Column Name	Type of Data	Description
surface	string	Surface the tournament was played on (eg "hard"/"clay")
draw_size	int	How many people were in the main draw for the tourney
tourney_date	string	Date of the tournament
tourney_level	string	The Level the tournament was("G": Grand Slam, etc)
winner_seed	int	What seed the winner was
winner_entry	string	Did the winner enter through qualifiers, direct admit, etc.
winner_name	string	Name of the winner
loser_hand	string	Did the loser play with left or right hand
loser_ht	int	Height of the loser
loser_ioc	string	Country of the loser
loser_age	float	Age of the loser
winner_rank	int	Rank of the player in the atp rankings
winner_rank_points	int	How many points the winner had in atp points
score	string	The resulting score line of the game(eg "6-3 6-4")
best_of	int	3 or 5
round	string	Round of a tournament(eg "R32" or "QF")
w_ace	int	How many aces the winner scored in the match
w_df	int	How many double faults the winner made
w_svpt	int	How many service points the winner scored

w_bpFaced	int	How many breakpoints the winner faced

ii. Cleaning the data

```
Index: 53571 entries, 0 to ('2017-M-DC-2017-WG-M-SUI-US
Data columns (total 49 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   tourney_id          53508 non-null  object
 1   tourney_name        53494 non-null  object
 2   surface             52250 non-null  object
 3   draw_size           53508 non-null  object
 4   tourney_level       53500 non-null  object
 5   tourney_date        53440 non-null  float64
 6   match_num           53440 non-null  float64
 7   winner_id           53508 non-null  object
 8   winner_seed         24126 non-null  float64
 9   winner_entry        9221 non-null   object
 10  winner_name         53472 non-null  object
 11  winner_hand         53458 non-null  object
 12  winner_ht           50861 non-null  float64
 13  winner_ioc          53468 non-null  object
 14  winner_age          53456 non-null  float64
 15  winner_rank         52421 non-null  float64
 16  winner_rank_points  52421 non-null  float64
 17  loser_id            53468 non-null  float64
 18  loser_seed          14474 non-null  float64
 19  loser_entry         13152 non-null  object
 20  loser_name          53468 non-null  object
 21  loser_hand          53453 non-null  object
 22  loser_ht            48895 non-null  float64
 23  loser_ioc           53468 non-null  object
 24  loser_age           53446 non-null  float64
 25  loser_rank          51744 non-null  float64
 26  loser_rank_points   51744 non-null  float64
 27  score               53467 non-null  object
 28  best_of             53468 non-null  float64
 29  round               50179 non-null  object
 30  minutes             43213 non-null  float64
 31  w_ace               44451 non-null  float64
 32  w_df                44451 non-null  float64
 33  w_svpt              44451 non-null  float64
 34  w_1stIn             44451 non-null  float64
 35  w_1stWon            44451 non-null  float64
 36  w_2ndWon            44451 non-null  float64
 37  w_SvGms             44451 non-null  float64
 38  w_bpSaved           44451 non-null  float64
 39  w_bpFaced           44451 non-null  float64
 40  l_ace               44451 non-null  float64
 41  l_df                44451 non-null  float64
 42  l_svpt              44451 non-null  float64
 43  l_1stIn             44451 non-null  float64
 44  l_1stWon            44451 non-null  float64
 45  l_2ndWon            44451 non-null  float64
 46  l_SvGms             44451 non-null  float64
 47  l_bpSaved           44451 non-null  float64
 48  l_bpFaced           44451 non-null  float64
dtypes: float64(33), object(16)
```

Originally the dataset came with a lot of missing values as can be seen to the left. Many of the rows had missing data, particularly as a result of not having the data or in the case of winner_seed/loser_rank, the players probably just didn't have a rank or seed at the time of playing in the tournament. There was no category for that and upon examination of the data, that was a logical conclusion. Thus, to clean the data, I first removed all the rows where the bottom stats like aces or double faults were not available. All of the bottom stats came together given that there are the same number of non-null values for all of them, so for rows without the stats, there was probably just limited data collection overall for those matches. Afterwards, I added in the Not Seeded and Not Ranked entries into seeding and rank and imputed the mean for any further numerical columns in the data.

iii. Creating the Target Column

The most important column that I added to make the data predictable was the target column. Before doing so, the data was in the format of separating the winners or losers, which means that the model would already know which side had won before predicting, defeating the purpose. To solve this issue, I named the players player 1 and player 2 and then with random 50% probability, swapped the players, labeling in target a 0 if I swapped them and 1 if I kept them the same, thus denoting if the player in the first position actually won.

iv. The columns that I added in

Column Name	Type of Data	Description
target	int(boolean)	1 if the winner is the first player, 0 if second
seeding_difference	int	Difference between the seeds of the players. Given that seeds from 1-35 were possible in the data, denoted not seeded players as 36 when calculating.
rank_group	string	Group of 200 points that the rank falls in (eg "200-399")
player_1_win_streak	int	How many games player 1 had won in a row to that point
player_2_win_streak	int	How many games player 2 had won in a row to that point
streak_diff	int	Player 1's streak - Player 2's streak
num_sets_played	int	Second prediction column, number of sets

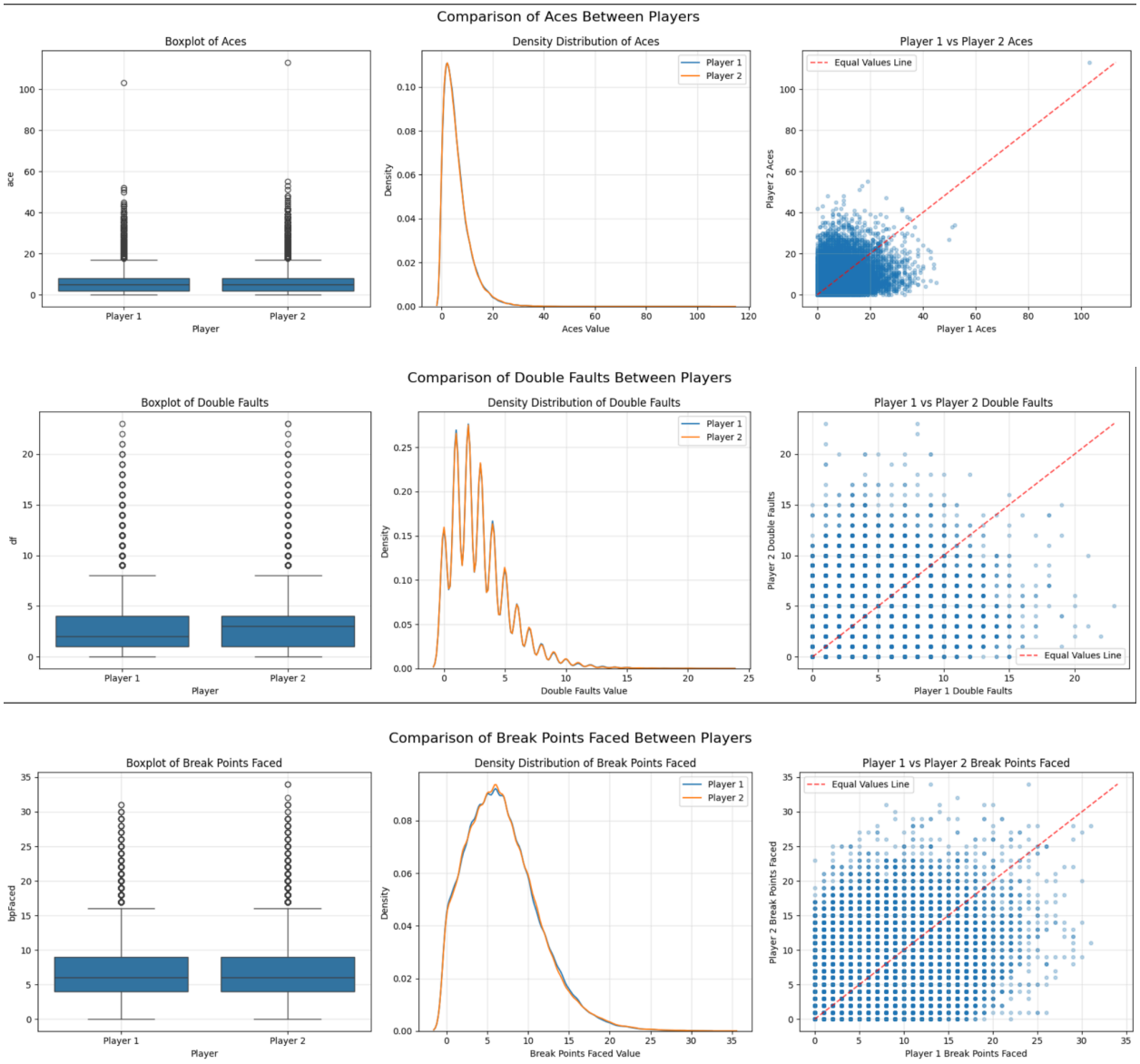
v. Explaining my Decision for New Columns(other than target)

Seeding_difference was created to remove some of the noise that comes with later rounds of a tournament. If a player is seeded 1st and is playing in R128, probably against an unseeded player, that would be a large difference in seeding as compared to in the finals of a grand slam, where the 1st seed might be playing the 2nd seed. To encapsulate the much larger difficulty change that comes with not just being seeded a certain way but also the opponent's seed that a seeded player is playing against, seeding_difference was created.

Rank_group was simply created for plotting purposes, to make the plots more readable by grouping the rank points 200 at a time. The player_1_win_streak / player_2_win_streak were created to follow the momentum of a player up to that moment. Because the data had time series stuff, it was ideal to also analyze how a player had been doing up to that point to determine how they would do in a round. If a player had won 10 games in a row, it would be expected that they would continue the trend. Then, streak_diff was created for the same reason as seeding_difference. It also leads to more noise removal at the final round of tournaments, because both players probably had to win 4-6 games in a row to even get to that point. Finally, the num_sets_played was created to be the second prediction column. It allows for training and testing based on the number of sets that were played and was created by splitting the score data to figure out the number of sets played.

vi. Visualizing the statistics columns

All of the stats columns for players were visualized in the actual code, but for a sample of what they looked like:



In general, most of the stats graphs seemed to look about the same, with a few outliers on the higher end, which occur with longer games and probably just unique situations. Other than those right tail outliers, the rest of the data seems to be generally normally distributed. The stats for one played versus the other in the same match also generally center around being equal, but in some games, some players have more of one stat than the other, probably leading to a difference in ability to win the game.

III. Models and Methods: Overview of models and implementation

A. EDA (in searching for columns that would be effective in the model)

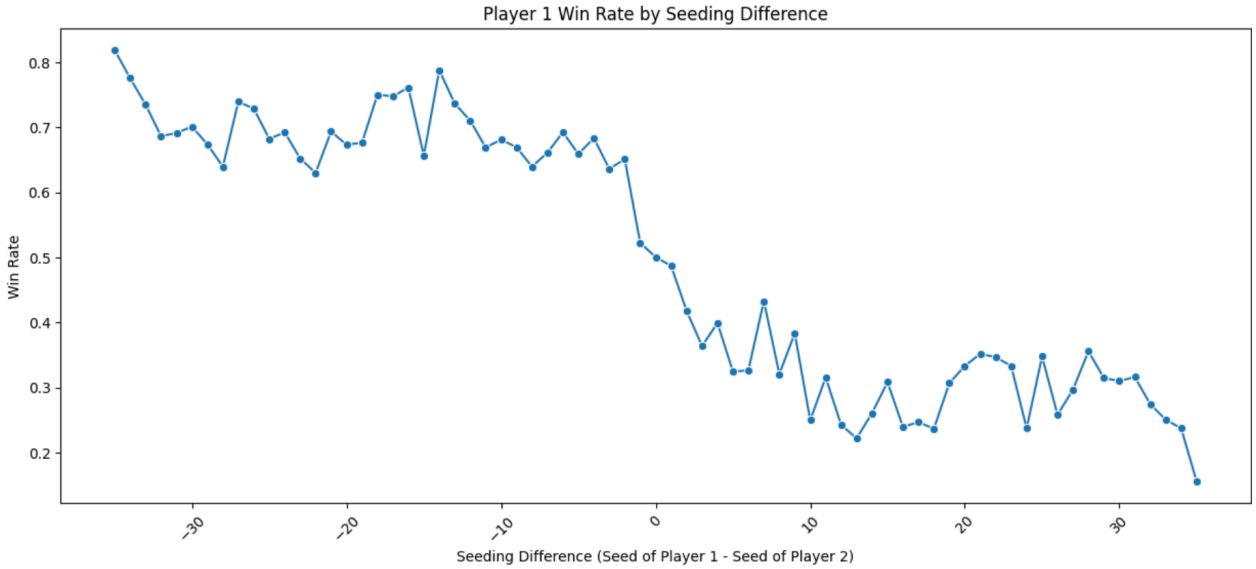
To begin my EDA, I started with a simple heatmap to identify some features that already seem to be important. From there, I arrived at what things I was interested in figuring out from the data, and created guiding questions to lead me through the EDA that I will list below:

- Seeding Questions:
    - Does seeding for a tournament affect how much chance a player has of winning
    - What about the difference in seeding?
- Rank Questions:
    - Does the number of ranking points a player has impact their ability to win?
    - Does the tournament(Grand Slam/Masters/Location) affect how important the rank is?
    - Does having more ranking points improve one's chances as time goes on (with longer games)?
- Height Question:
    - How much does height matter for a player's chance of winning?
- Ace Questions:
    - How important is getting aces in a game?
    - Does the type of court affect how important aces are to win?
- Serve Questions:
    - Does having a high first serve in number correspond to a higher percent win?
    - What about the comparison between 1st serve points won and 2nd serve points won?
- Break Point Questions:
    - Does the number of breakpoints faced correspond to the chance of winning?
    - Does the round in a tournament matter for how important breakpoints are?
- Player Questions
    - Does how many games in a row a player has won contribute to their ability to win the next game?

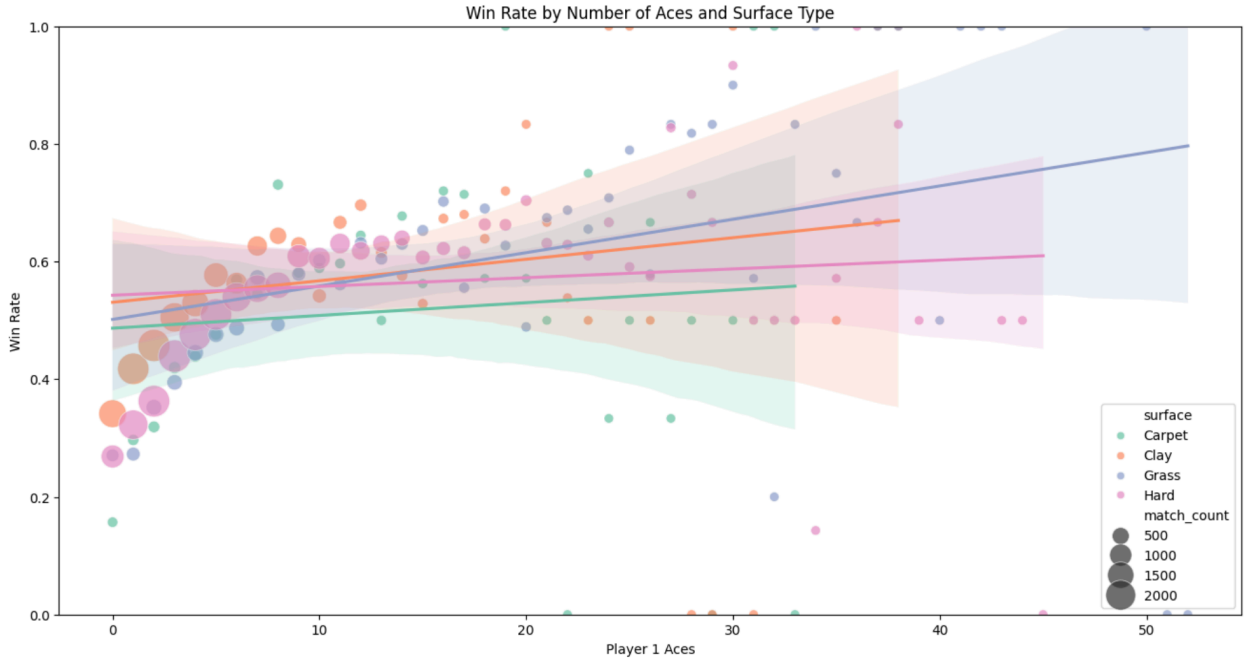
- Do certain Players play better on certain surfaces?

B. Some Results from the EDA

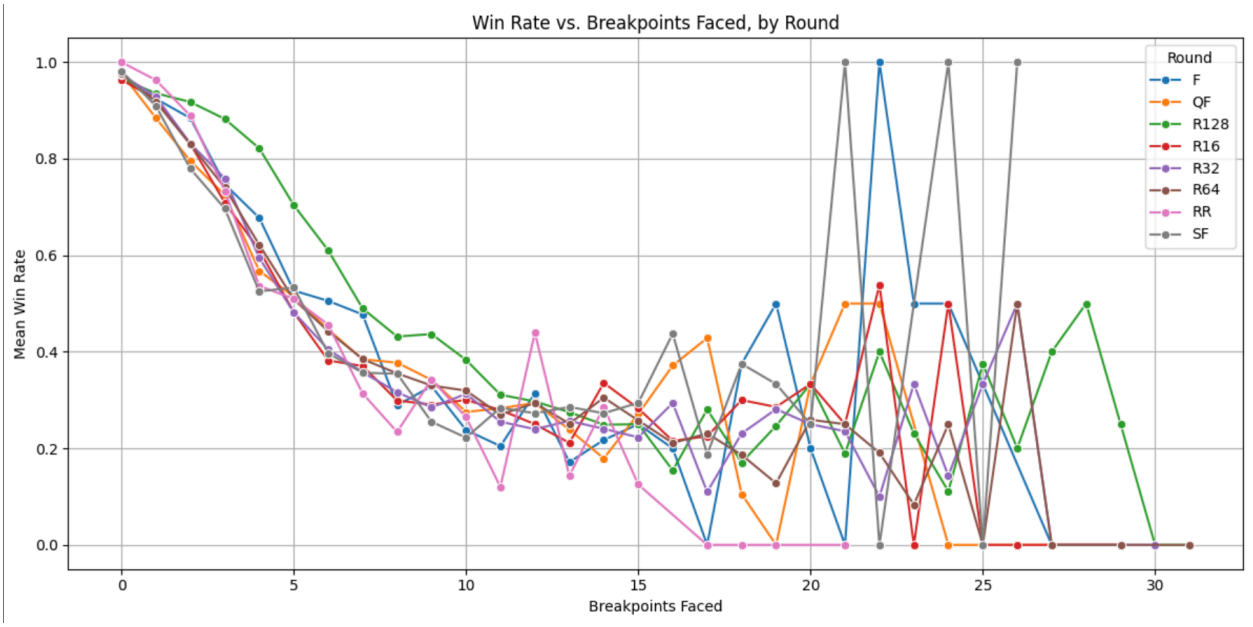
Starting from the seeding and ranking graphs, they showed what was expected. Players that had higher numbers of ranking points coming in and players with higher seeds (closer to 1) were more likely to win, particularly when the difference in seeds was large. As seen in the graph below, for the seeding difference, it almost looks like a reverse cubic function. At a difference of 0 in seeds, the win rate of the first player is approximately 50% and then with a slight shift in difference, there are large shifts in winning percentage, going up/down 10% with just a +/- 3 seed difference. After about a 5 seed difference, it seems to be more flat, implying that players over 5 seeds better than another are much more likely to win and any difference after that is just added.



Looking at more in the match statistics, players who hit more aces also tended to have a higher winning percentage. Players with 0 aces in a game had very low win rates, and with just a few aces that probability grows a lot better. After about 10 or so aces, the winning probability seems to flatten out just like the seeding difference effect had flattened out after a certain significance. Interestingly, aces seemed to matter more on certain surfaces. Particularly, an ace hit on grass had a larger effect on increasing win rate than an ace hit on hard court, which makes sense given that grass courts are the fastest type of court, so aces are significant ways to earn points. A graph showing this is below:



Another important statistic was the number of breakpoints a player faced. Break points are points faced where the opponent has the chance to win a player's service game. Given that service games tend to be in a player's advantage at the professional level, facing many break points is a dire sign of trouble. This turns out to be the case as seen in the graph below where facing more break points leads to a low win rate. This seems to be consistent regardless of round, although facing a few breakpoints seems to be less impactful at the round of 128, which could be a result of larger differences in skill level at the earlier rounds, leading to less impact even if break points are reached.



C. Overview of models used

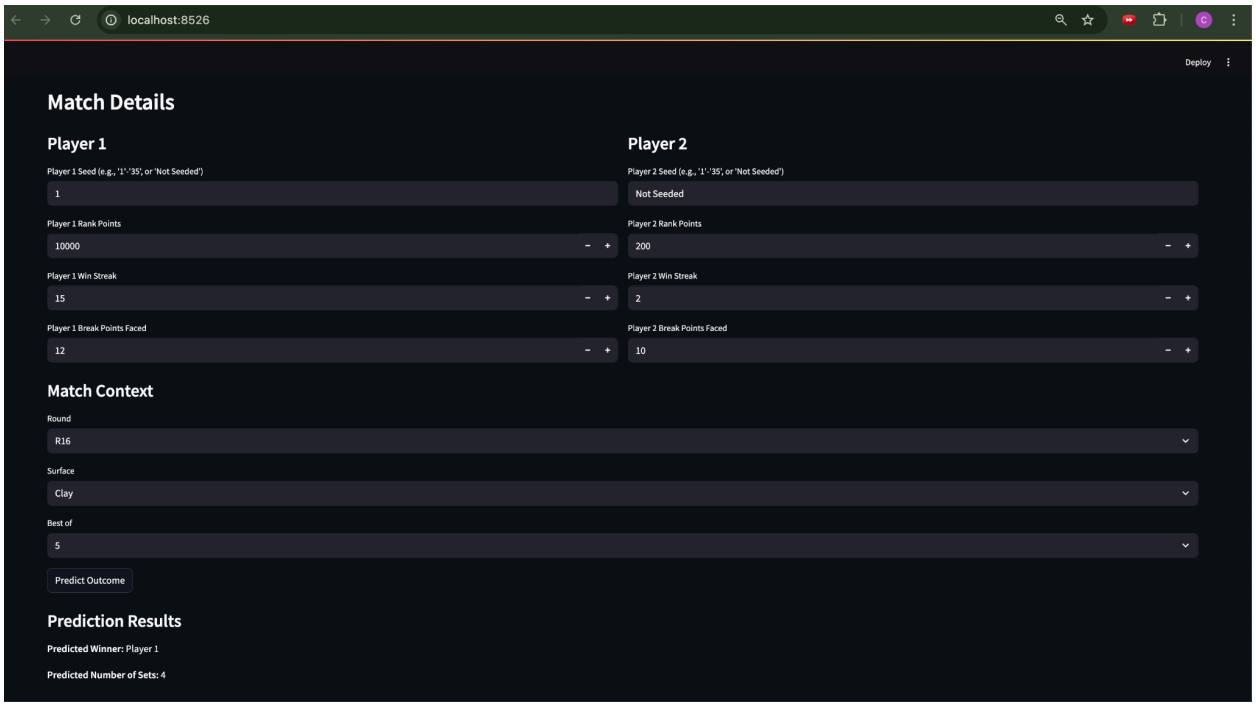
Whether or not a player wins and also how many sets it takes a player to win are both categorical problems given that there is a limited number of possible outcomes for both of them. Thus, it makes sense to use classifiers to determine which category the variables should be when two players face each other in a match. Given that I am using a classification model, I wanted to look at a wide range of classification models. In particular, I decided to start with a logistic regression and then work with a k-nearest neighbors classifier, decision tree model, random forest classifier, and a neural network at the start. After determining which models seemed to work well for determining which player won, I decided to stick with the models performing better to also evaluate the number of sets needed to be played.

For the logistic regression, I used a classic pipe setup for the main part of the model. To encode the data for it to work with the pipe, I used a OneHotEncoder for any categorical variables and a standard scaler for numerical variables. I also split the data into training and testing data, training the Logistic Regression on the training data and later evaluating it on the testing data.

To optimize my other models, I used search, which meant that I was running the models over and over again on a large dataset. Thus, to improve the speed of training, I limited the training data for the models to a random sample of 5000 from within the training data, slightly more than 10% and already a numerically significant amount to capture a large part of the image.

D. Implementation

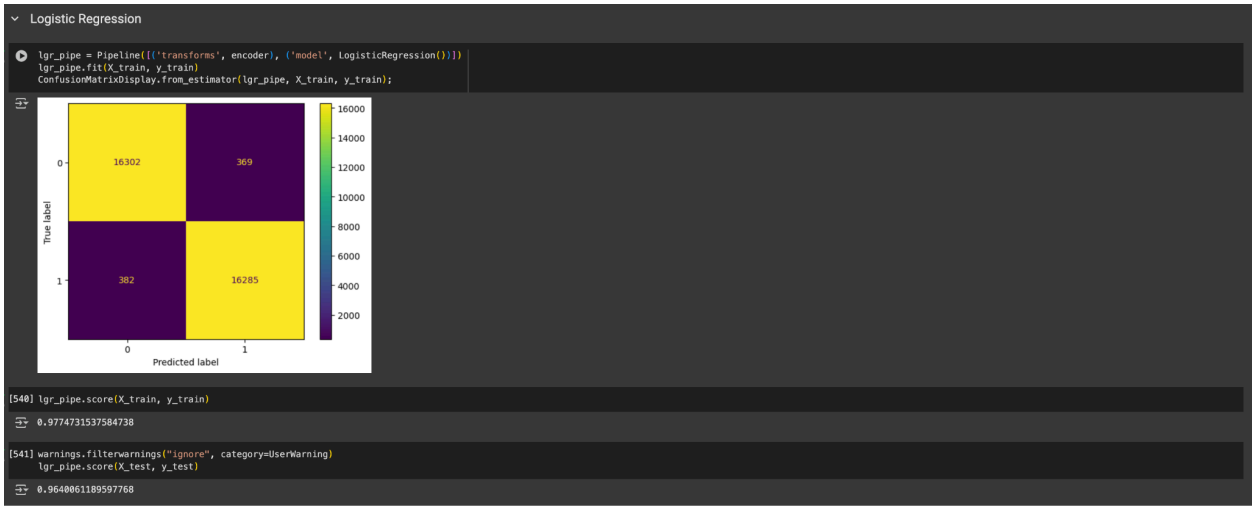
When I had the models built, I implemented it by first creating models for predicting which player would win and then creating models for predicting how many sets it would take them to win. Afterwards, when I had created all of these models (the results of which come in the next section), I decided to implement it further into a Streamlit to add a user interface for it to actually be implemented and used. Because a regular person wouldn't like entering 50+ data points into a model, I limited the streamlit version to a total of 11 features, which would thus be more manageable. In choosing these features, I took a look at the EDA, and saw that some of the specific features that mattered the most included the seeding, the number of rank points, the win streak of the players, break points faced, and so on, and chose to use those in my model. The Streamlit would then have a prediction results section where it would both predict who would win and in how many sets. A brief implementation is shown below:



IV. Results and Interpretation: Review of modeling results and interpretation of performance

A. Review of Modeling Results (which player would win)

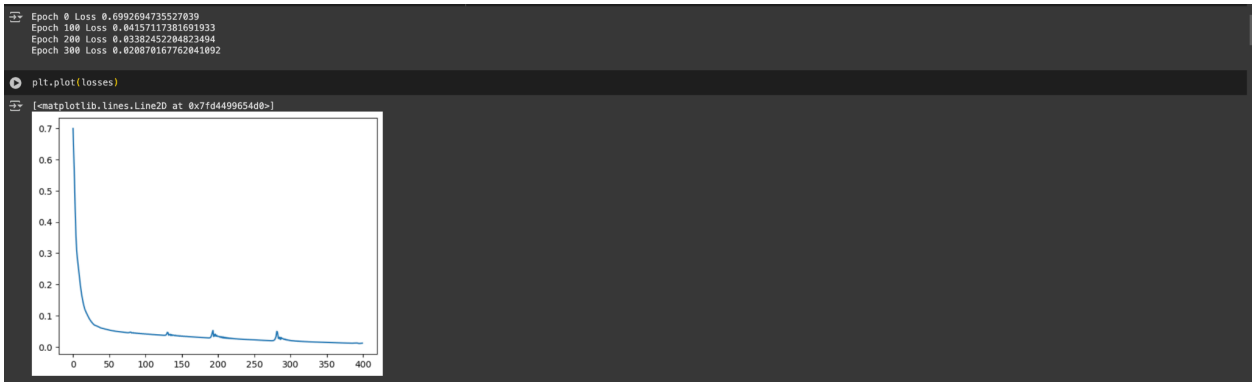
In predicting which player would win, the logistic regression model was really, really accurate. On the training data, it scored 0.9775, or in other words it correctly predicted 97.75% of the outcomes of the matches correctly. This is really impressive given that the baseline is 50% as I had randomly swapped the players 50% of the time, so the chance of the first player being the one that had one was 50% and selecting all of one player would lead to the same random result. Furthermore, the errors seem to be distributed evenly among false negatives and false positives, which indicates a balance in the model. Having either is equally not good for the model as both just indicate that the model thought the other player would win, which is equally important. Thus, accuracy is the most important measure. This makes the 96.4% accuracy on the testing data also really good. Given that the testing data accuracy is very high and also near that of the training data, there seems to be very little if none overfitting going on.



Comparatively, the other models weren't as successful even with grid searching, but still had consistently decent results. For the k-nearest neighbor model, the best neighbor amount was 101, in increments of 100 from 1 to 1000. The sample of 5000 accuracy used to train the model was 89.8%, the entire training data accuracy was 88.8%, and the testing data accuracy was 88.9%. While in comparison to the logistic regression model, this doesn't seem as good, it is still much better than the baseline of 50%.

The decision tree achieved slightly better but generally similar results in predicting which player would win. After grid searching in increments of 2 from 1 to 21, the best max depth was 9. It resulted in 96.3% accuracy on the sample data, 91.8% accuracy on the training data, and 90.9% on the testing data. This does seem to indicate a more overfit model though given the 5% difference in accuracy between the data that it was trained on versus the other data. The random forest classifier seemed to have essentially the same result but better. Using the same grid search params as the k-nearest neighbors, the best param was 201 estimators. It achieved 100% accuracy on the sample data, 94.5% accuracy on the training data, and 93.3% accuracy on the testing data, doing almost as well as the

The neural network model achieved a similar result as the logistic regression, getting to 97.75% accuracy in its prediction of the testing data after 400 epochs, having a pretty consistently decreasing loss graph when looking at the training data except at a few sudden jumps in the graph as shown below.



B. Review of Modeling Results (Number of Sets)

Given how the models did in predicting which player would win and that it would be on the same idea, I chose to focus on the 3 models that had done well for predicting the winning player and implemented them for predicting the number of sets as well. I used mean squared loss to optimize the result because of there being more categories (numerical) possible.

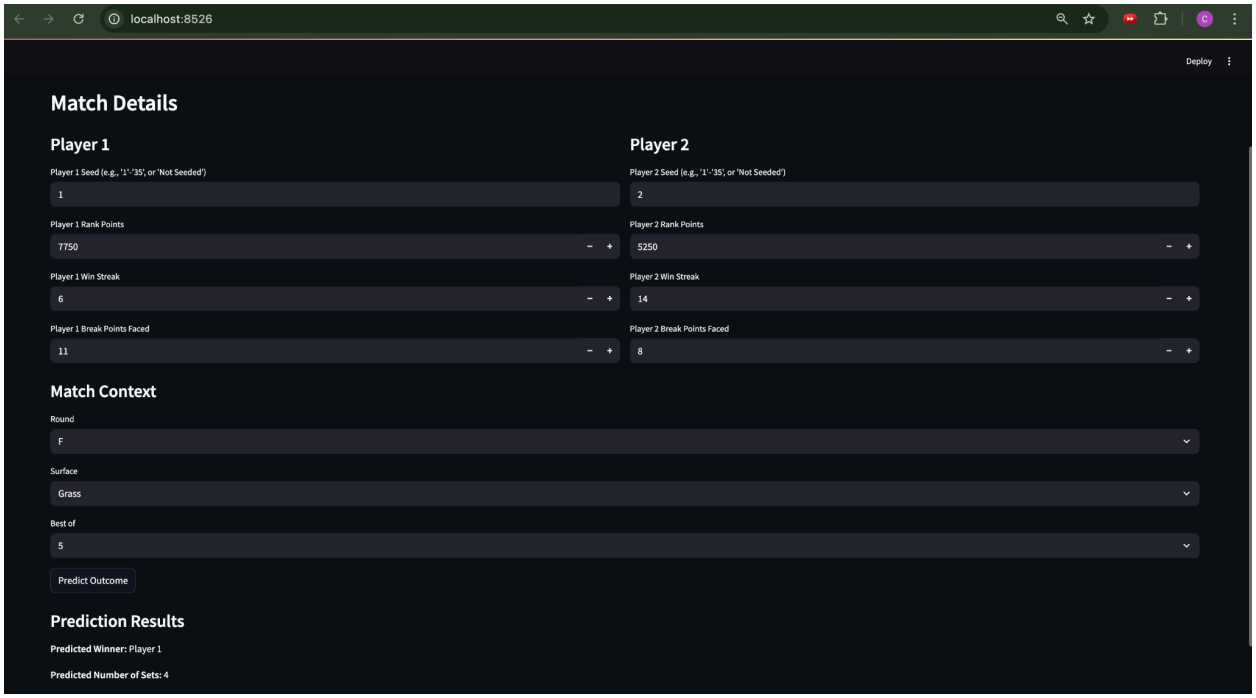
This time, the neural network clearly outperformed the other two models. Both the logistic regression model and the random forest classifier had greater than 5 mean squared error(which still corresponded to an accuracy of over 90%). Comparatively, the neural network had a mean squared error of 0.044, which is much lower. This makes some sense given that the logistic regression model has to deal with multiple possible categories and it's pretty easy for variables to lead to errors in prediction. A neural network might be better to deal with the more complex process.

C. Review of Modeling Results (Streamlit Feature Reduction)

Now, I wanted to combine both predictions for Streamlit, but had to reduce the number of features for the user interface to be interactive. Thus, I chose to proceed with only the best models for predicting both Based on how the models had done previously, I chose the logistic regression for predicting which player would win due to its high accuracy and also low resource and time usage and I chose the neural network for predicting the number of sets. The logistic regression for the testing data using only the 11 features from streamlit was 89.9% accurate, and the neural network was only 78.0% accurate for predicting the number of sets.

Just for testing on a real match purpose, I looked at some stats for the 2007 Wimbledon final between Nadal and Federer. Nadal was the 2nd seed, had approximately 5000 ranking points, faced 8

breakpoints, and had won a tournament going in, leading to approx. a 14 win streak. Federer was the 1st seed, had approximately 8000 ranking points, faced 11 breakpoints and had rested going in with approx. a 7 win streak. My model predicts Federer to win in 4 sets, and the actual result was Federer winning in 5 sets. While my model was off in the number of sets, this was still a pretty close result on an actual match, indicating confidence in the model working.



D. Further Interpretation of Performance

The models overall were all very accurate in their predictions. Across the board in predicting both prediction columns and even with reduced features, the models all achieved over 75% accuracy. This is great given that the baseline model, to predict randomly, would score 50% here as half the target columns is 1 and the other half 0 based on the random swapping. For the set numbers, the baseline is also about 50% because approximately half of all matches ended in 2 sets.

Overall the model did well, given the performance of a real game being tested on my Streamlit and leading to a roughly similar result. There were also fewer features used there when there is easily data online available for other stats like the number of aces hit by both players, also significant for the model as seen through the previous accuracies before reducing features. At least, it does indicate the features used were still fundamental to the probability of a player winning, meaning that break points, win streaks, and more are important.

V. Conclusion and Next Steps: Summary of models and next steps for further analysis

In general, the logistic regression seems to work very well with this tennis dataset, but the neural network, while needing more training, seemed to overall do the best. Specifically, the logistic regression was good at predicting who would win, but seemed to struggle a bit more at predicting the number of sets needed, which the neural network was good at.

For further analysis, I would like to look at specific players. I currently use win streaks as a sort of time series approximator, but I am sure with more time, I could train models for specific players. This would allow me to analyze what parts of certain player's games really contribute to their ability to win.

I would also like to expand on the parameters being grid searched in the random forest model in particular but also in the other models, because the run time started taking over 10 minutes, when I tried running on larger grid sizes of over 10. With more time though, I could see the results of those models out. I would also experiment with grid searching across other parameters for the models.

Finally, I want to do a more in depth user interface that adds a percentage estimator to a player winning as well as a breakdown of the probability of ending up at how many sets needed for a game, as well as a better looking user interface in general.