```
In [ ]: import requests
        from bs4 import BeautifulSoup
        import pandas as pd
        import numpy as np
        import time
        import locale
        locale.setlocale(locale.LC_ALL, 'en_US.UTF-8')
        import pandas as pd
        import duckdb
        from selenium import webdriver
        from sklearn.metrics import precision_score,recall_score, f1_score, roc_auc_score
        import os
        import pandas as pd
        from sklearn.linear_model import LogisticRegression
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import KFold
        import scipy.stats
```

Introduction and Research Question

Setting & Significance

The Washington Post has stated that up to $50 billion is spent yearly on tennis betting. Understanding what factors affect the outcome of a tennis match are crucial in these betting areas, in addition to helping tennis fans appreciate the game more. The Association of Tennis Professionals (ATP) compiles a ranking of each male tennis athlete in their league, and also computes metrics about the players skill in serving and returning.

Research Question

Can we use two ATP men's tennis players overall rankings, serve ratings, and return ratings (compiled by the Association of Tennis Professionals) to predict who will win a tennis match between the two players? Specifically, can we use the relative ratings/rankings of each player (ie. rating of player 1 - rating of player 2) to determine who will win the match? Can we use this model to comment on which metrics are the best predictor of tennis performance? (i.e. Is serving more important than returning? Is overall ranking more important than specific skills? etc)

We would like to train a multivariate logistic regression model on all of the 2023 ATP men's tennis match results to predict the winner and loser of a tennis match based on their respective overall rankings and serving/return ratings.

Then, we would like to evaluate the reliability of this model using binary regression metrics like precision, recall, ROC/AUC, and F1. We will perform cross-validation using 8 splits and a 70/10/20 train/validation/test split to determine if our models are statistically better than randomly picking the outcome of the match using a one-sample t-test. Finally, we will also use the cross-validated binary regression metrics to determine if including the serve and return rating in the logistic regression improves the model performance relative to a model that is trained only on the difference between the overall ranking of the men's tennis players. We will use a two-sample t-test to determine this.

In addition to creating a model that can predict the winner of a tennis match, we also intend to explore factors outside of rankings and ELO. Specifically in this project, we explore the effects of the type of courts that the players are playing on, and along with noting that Spain is routinely ranked as the best conutry in tennis, and exploring the history of performance of Spanish players vs. Non-Spanish players in 2023. We will discuss the impact that these have on the outcome of the game. Doing these additional analyses help us to quantify the effect that factors outside of the ones we are considering in our primary model have on the outcome of tennis matches, which can explain why our model is not a perfect predictor of tennis match outcomes. Furthermore, these additional analyses may provide an interesting insight into the game of tennis by commenting on which countries have a more robust tennis program, and which court surfaces benefit different aspects of the game.

Preregistration Questions

Hard vs Clay

Hypothesis: Professional men's tennis players serve better on hard courts than clay courts. We believe that this would possibly be due to the fact that professional tennis players mainly practice and play on hard courts. This answer may suggest that we need to change our model to include what surface the match occurred on.

Analysis: The ATP compiles server ratings for each professional player on each playing surface. We can perform a two sample t-test (alpha = 0.05) between the average server rating on hard courts and clay courts. We predict that there will be a statistically-significant difference between the men's server ratings that indicates that the population mean server rating on hard courts is higher than on clay courts.

Spanish vs. Non-Spanish Players

Hypothesis 2: Spanish players have a better chance of winning than non-Spanish players. Spain is often ranked as the best country in tennis, and Spanish tennis players routinely occupy top ATP rankings. We want to test whether this is true from our data.

Analysis: We scraped the nationalities of all of the lead serve and return players and calculated the probability of every nationality winning a tennis game, using the past 860 games played in 2023. We then analyzed whether the Spanish player probability of winning was significantly higher than the average non-Spanish probability of winning. We calculated the z-score and p-value to determine whether we can reject the null hypothesis: that Spanish players do not perform better than non-Spanish players in a tennis match.

Data Description

```
In [ ]: overall_df = pd.read_csv('overall_df.csv')
```

MOTIVATION

Why was this dataset created?

This dataset was created to create a model that is able to accurately predict the probability that one player will win against another player given data from matches played in 2023. These players are the top 80 players.

Who funded the creation of the dataset?

Association of Tennis Professionals (ATP). This is the official and primary tennis association for all professional games and matches, and as such has the most and most accurate data out of any other website or association. We did not verify the methods in which they calculated the values of rankings of everything. We are assuming that the formulas that the ATP uses to take data and output rankings and ratings accurately measure the real skill of these players.

The specific pages that were taken from ATP are listed as follows:

- https://www.atptour.com/en/rankings/singles
- https://www.atptour.com/en/stats/leaderboard?boardType=serve&timeFrame=52Week&surface=all&versusRank=all&formerNo1=false
- https://www.atptour.com/en/stats/leaderboard?boardType=return&timeFrame=52Week&surface=all&versusRank=all&formerNo1=false.

In addition, information from the specific matches that have taken place in 2023 were found from (https://github.com/JeffSackmann/tennis_atp/blob/master/atp_matches_2023.csv). The source, Jeff Sackman, is experienced in sports statistics, having written and analyzed tennis for many big organizations including the Wall Street Journal and ESPN. In addition, the information contained are completely objective and are not the result of calculations or models, such as when the match took place and the ages of the players.

COMPOSITION

What do the instances that comprise the dataset represent?

The instances that comprise the dataset represent one match between two professional tennis players in the year 2023.

Are there any errors, sources of noise, or redundancies in the dataset?

There are many more factors that affect the probability and outcome of a tennis match, from the surface, to the time of year and day, to the current physical and mental health of each player, age, differing strengths and weaknesses of specific players, and more. We are just focusing on some of the larger factors

Is the dataset self-contained, or does it link to or otherwise rely on external resources?

The dataset was taken from multiple sources listed above and merged together, this model is only accurate currently as the data used will constantly be changed as rankings and ratings of players change

What are the observations (rows) and the attributes (columns)?

Columns:

- winner_name: The name of the winner of the match
- loser_name: The name of the loser of the math
- winner_rank: The current rating/ELO of the winner
- loser_rank: The current rating/ELO of the loser
- winner_serve_rating: The current rating/ELO for the serve of the winner
- loser winner_serve_rating: The current rating/ELO for the serve of the loser
- winner_return_rating: The current rating/ELO for the return of the winner
- loser_return_rating: The current rating/ELO for the return of the loser

Rows:

- Every professional match that has taken place in 2023

COLLECTION PROCESS

What processes might have influenced what data was observed and recorded and what was not?

The ATP website only contains data for a limited number of players, up to 80, which is different for both the serve and return leaders. This means that the overall dataset only contains players that are found in not only the matches in 2023 but also in the serve leaders along with the return leaders. More player stats weren't able to be collected from ATP as it wasn't shown.

Over what time frame was the data collected?

Data for the players were only collected from 2023

How was the data associated with each instance acquired?

The overall, serve, and return ELO's of the players were collected from the ATP website. The matches of 2023 were collected from Jeff Sackman's github.

What preprocessing was done, and how did the data come to be in the form that you are using?

The ATP data all came in multiple tables, which needed to be webscraped, and cleaning and put together. The data from Jeff Sackman's github came as a downloadable csv, from which specific columns were taken from

Data Limitations

The data that we are considering has several limitations.

Limitations of metrics that we collected

- We are using single-number summary statistics generated by the ATP that, broadly speaking, are supposed to assess how "good" a player is overall, at serving specifically, and at returning. We do not know, as it is proprietary, how the ATP is taking "lower level" statistics about a tennis player's performance and turning them into these rankings. If the "magic formula" that ATP uses to rank players and rate them on their serving and return abilities is flawed, that will mean that we are not using reliable or valuable information about their serving ability to predict the winner of a match, which may harm our model. Furthermore, there is an intrinsic assumption that is being made with these metrics that you can boil down all of the important factors about a particular tennis player's serve into one number.
- The rankings and ratings that we downloaded from the internet are extremely current. When the matches occurred, it is possible that one or both of the player's ratings or rankings had significantly changed in the intervening time between the date of the match and when we downloaded the rankings. Because we are considering current rankings and only training the model on matches that occurred in the calendar year 2023, we do not expect that the rankings will have substantially moved.
- The available rankings on the internet are only compiled for the top 80-100 tennis players (depending on the metric) in the ATP. Some of those tennis players have been in the professional circuit for multiple decades and reams of data exist about them on the internet. For some high profile players, like Novak Djokovic, we can trust that the rankings and ratings compiled by ATP are likely to be very reliable because there is a lot of data about those players. However, for some newer players or younger players who may have recently exploded onto the professional tennis circuit, their ratings and rankings may be skewed by insufficient data - perhaps they have over or underperformed relative to their potential in their existing matches, or perhaps the ratings and rankings are biased against newer players by factoring in "cumulative" achievements (like total matches won) into the ranking. ## Limitations of predictions made from the metrics that we collected (and didn't collect)
- In our data cleaning step, we repeatedly filter out matches where we do not have data for both players in the match by doing inner joins on data bases on the names of individual tennis players. If the names are spelled slightly differently - perhaps a typo in the dataset, or perhaps a player has a non-traditional last name with a hyphen or an accent, we may end up removing, systematically and for no good reason, data from our dataset corresponding to matches where players appeared with difficult to spell names.
- We are neglecting a large number of factors that have been proven to affect the outcome of a tennis match and the performance of specific tennis professionals. For example, certain players are expected to perform better on certain surfaces, like clay, relative to others, like grass. Our metrics do not take into account how the surface may affect a tennis player's probability to win a match. If one player is rated better on clay than another player, but worse on grass than another player, our model may make an erroneous prediction depending on which surface the match occured on.
- Our model is missing an interaction term between two players. As any serious tennis player can tell you (we think - none of us are actually serious tennis players), the complicated dynamics in a tennis match can't be boiled down to how "good" a tennis player is relative to his opponent. Some players may play better against players that are ranked far better than them because their specific cocktail of skills (not to mention psychological factors that are far too complicated to be modeled) as well as incidental circumstances on a given day like weather and how much sleep the athlete's got, may lead to an upset that we have no means of predicting from the data that we have. ## Implications of Data Limitations
- Implications of data limitations on people: since this model is only being compiled privately and for an academic audience, we know that our model does not have many implications for people in the "real world". However, if we were selling our model to bettors (who may be deciding what lines to bet) or sports gambling firms (who may be deciding how to set a betting line), the limitations that we have discussed would likely make our model unreliable in predicting the outcomes of matches and we may cause our clients to lose money or embarrass themselves.

Data Analysis

First, we will read in the overall data frame from above. We will drop the vestigial index column. Since we are planning to do regressions not on the ratings/rankings for each player themselves but on the difference between the rankings of the two players, we will make 3 columns corresponding to the difference between the overall rank, the server rating, and the return rating between the two players.

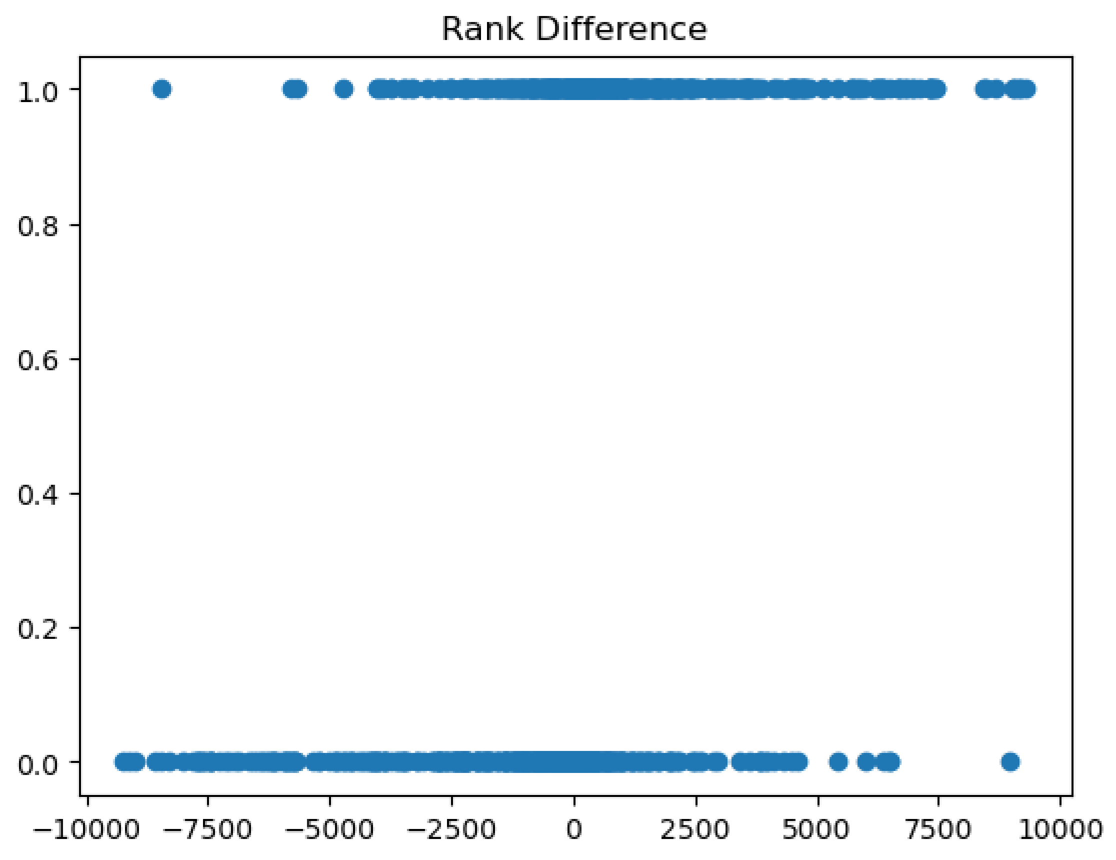
In [ ]:
```
overall_df = pd.read_csv('overall_df.csv')
overall_df = overall_df.drop(['Unnamed: 0'], axis=1)
overall_df['rank_diff'] = overall_df.winner_rank - overall_df.loser_rank
overall_df['serve_diff'] = overall_df.winner_serve_rating - overall_df.loser_serve_rating
overall_df['return_diff'] = overall_df.winner_return_rating - overall_df.loser_return_rating
```

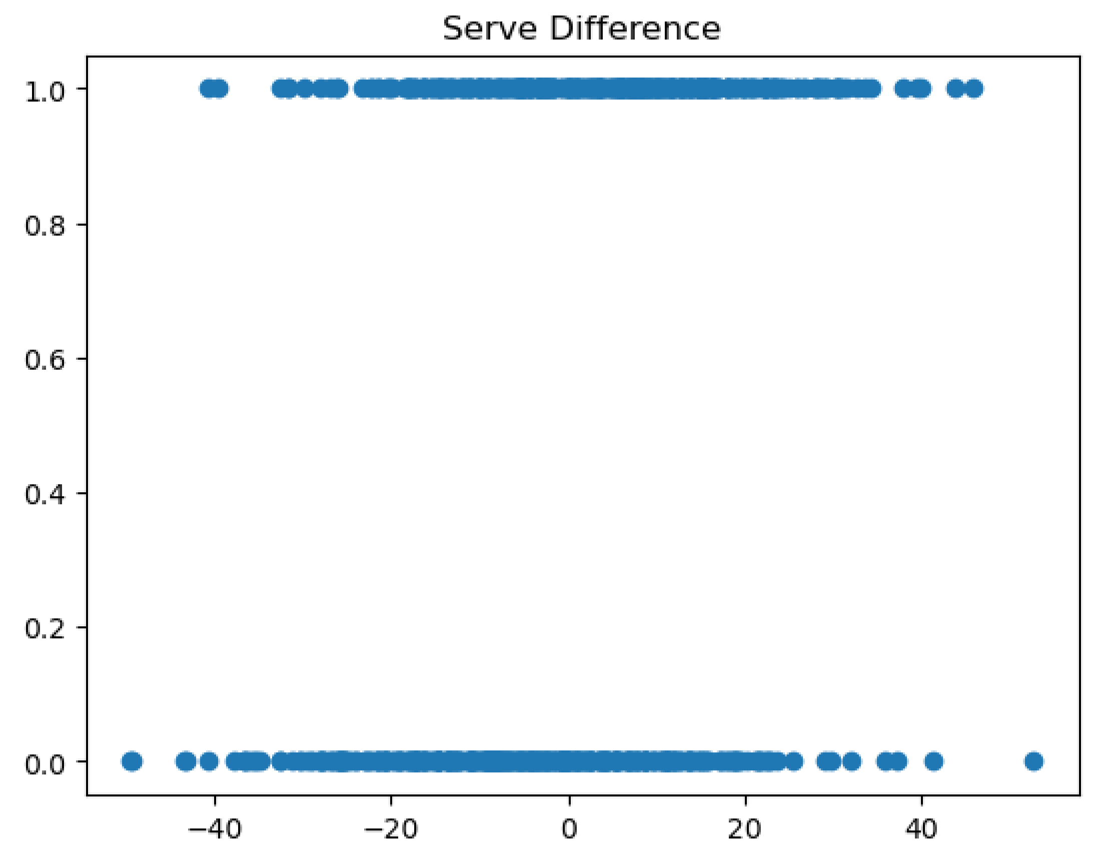
Next, we will make an np array of our inputs, X, formatted in "list of list" form for the purpose of doing a regression on the array. We will also make an np array for our outputs, which are all one (meaning that the ratings are calculated as Player A rating - Player B rating, where Player A is always the winner in our table). We will do a 70/10/20 train/val/test split, with cross-validation of 8 splits. We then plot the outcome of the 800+ matches in our dataset vs the differential ratings and rankings of players in those matches.

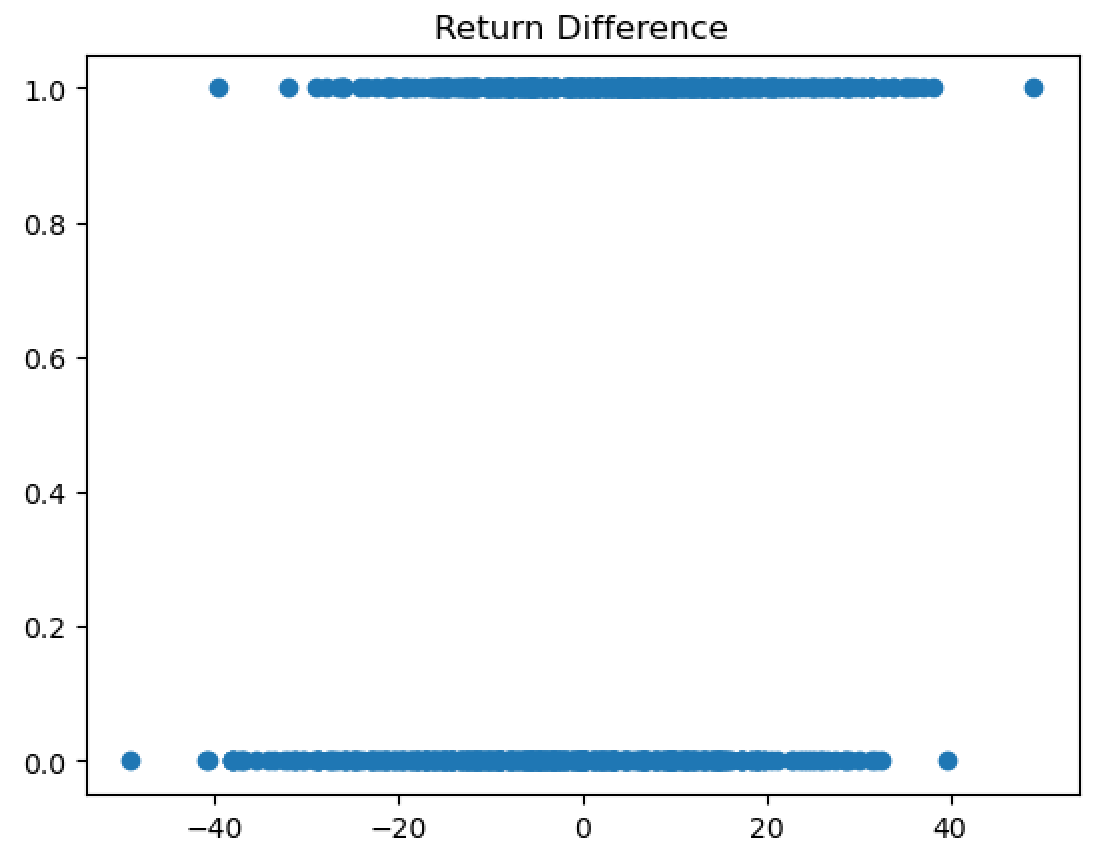
In [ ]:
```
X = np.array(overall_df[['rank_diff', 'serve_diff', 'return_diff']])
y = np.ones((len(X),1))
y = y.flatten()


for i in range(0, len(X), 2):
  X[i] = X[i]*-1
  y[i] = 0
```

```
#split data into 70/10/20 train/validate/test cross-validation splits
num_splits = 8
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
kf = KFold(n_splits = num_splits)
X_train_splits = []
y_train_splits = []
X_train_splits = []
X_val_splits = []
y_val_splits = []

#assign split data into train and validation bins
for train, val in kf.split(X_train):
    X_train_splits.append(X_train[train])
    y_train_splits.append(y_train[train])
    X_val_splits.append(X_train[val])
    y_val_splits.append(y_train[val])


fig1 = plt.figure('Ranking Difference')
plt.title('Rank Difference')
plt.scatter(X[:,0],y)
fig2 = plt.figure('Serve Difference')
plt.title('Serve Difference')
plt.scatter(X[:,1],y)
fig3 = plt.figure('Return Difference')
plt.title('Return Difference')
plt.scatter(X[:,2],y)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7fdc991c68e0>
```







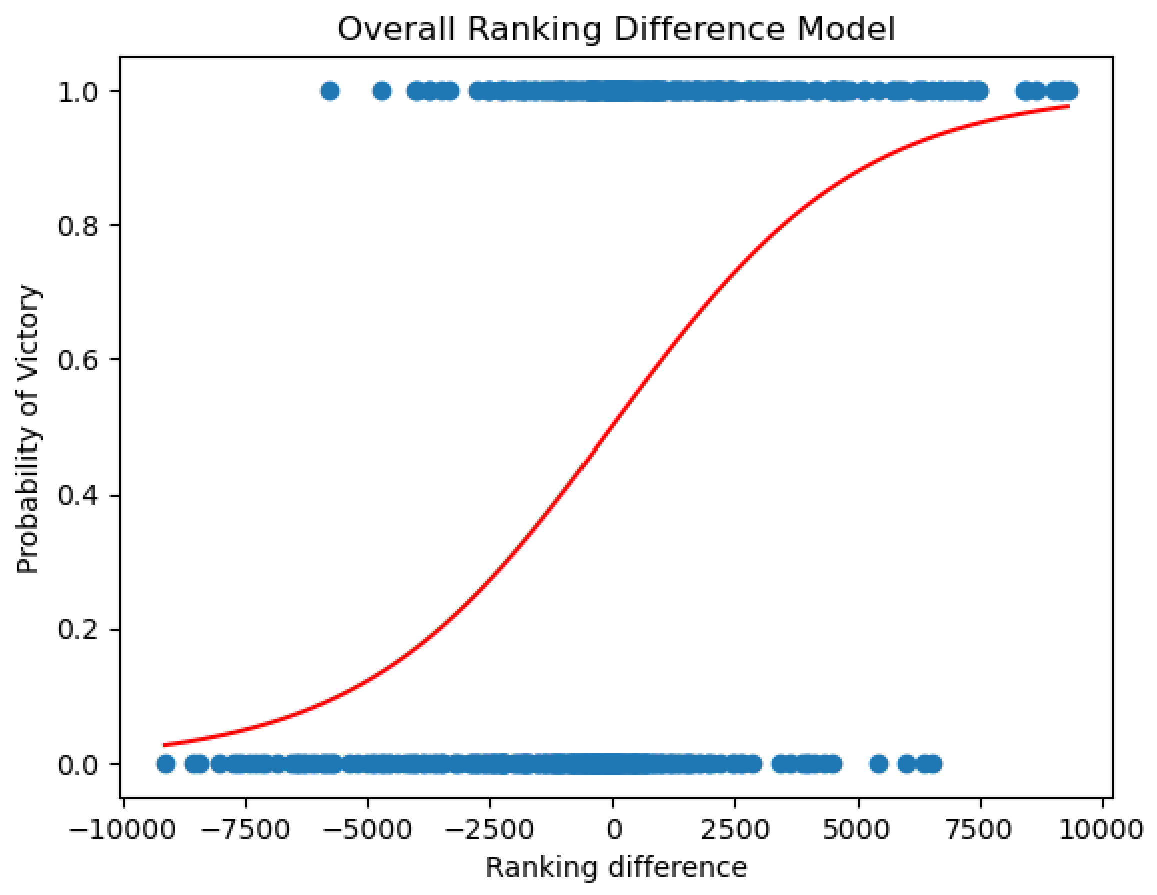
To begin, we will fit a logistic model with the difference in the two tennis player's rankings as the input and the winner of the match between the two players as the output. The red curve is the modeled probability of player A winning the match based on the differential in ranking Player A - Player B. The blue data are the outcomes of the 800+ matches between players we have data on from the 2023 calendar year.

Predicting Winners from Overall Rankings Alone

In the below cell, we fit a logistic regression to the difference between the overall atp rankings for two men's tennis athletes and the outcome of the match for each of the 8 cross-validation splits. We then plot the model for the eighth cross-validation split for the purpose of visualization.

In [ ]:
```
rank_model_splits = []

# fit model to each of the 8 train splits
for i in range(num_splits):
    rank_X =  np.array([[x] for x in X_train_splits[i][:, 0]])
    rank_model_splits.append(LogisticRegression().fit(rank_X, y_train_splits[i]))


fig1 = plt.figure('Overall Ranking Difference Model')
plt.title('Overall Ranking Difference Model')
plt.scatter(X_train_splits[num_splits-1][:,0],y_train_splits[num_splits-1], label = "match_data")
fit_x = np.linspace(np.min(X_train_splits[num_splits-1][:,0].flatten()),
                    np.max(X_train_splits[num_splits-1][:,0].flatten()), 1000)
fit_x = [[x] for x in fit_x]
plt.plot(fit_x, rank_model_splits[num_splits-1].predict_proba(fit_x)[:, 1], color = "red", label = "model fit")
plt.xlabel("Ranking difference")
plt.ylabel("Probability of Victory")
```

Out[ ]:
```
Text(0, 0.5, 'Probability of Victory')
```



In the below cell, for each train split, we are iterating through every threshold between 1 and 100 and calculating binary statistics on the resulting yhats to determine the performance of the model and the optimal place to set the threshold to maximize the reliability of our model's prediction of the outcome of a tennis match.

Specifically, in the below data, I am calculating the precision and recall with each threshold, and also calculating the rocauc score for each threshold.

In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)

precisions_splits_train = []
recalls_splits_train = []
f1s_splits_train = []
```

```
rocaucs_splits_train = []

for i in range(num_splits):
    precisions = []
    recalls = []
    f1s = []
    rocaucs = []
    for threshold in thresholds:
        yhat = []
        for x in np.array([[x] for x in X_train_splits[i][:, 0]]).flatten():
            if rank_model_splits[i].predict_proba([[x]])[0, 1] >= threshold:
                yhat.append(1)
            if rank_model_splits[i].predict_proba([[x]])[0, 1] < threshold:
                yhat.append(0)
        precisions.append(precision_score(y_train_splits[i], yhat))
        recalls.append(recall_score(y_train_splits[i], yhat))
        f1s.append(f1_score(y_train_splits[i], yhat))
        rocaucs.append(roc_auc_score(y_train_splits[i], yhat))
    precisions = np.array(precisions)
    recalls = np.array(recalls)
    precisions_splits_train.append(precisions)
    recalls_splits_train.append(recalls)
    f1s_splits_train.append(f1s)
    rocaucs_splits_train.append(rocaucs)
```

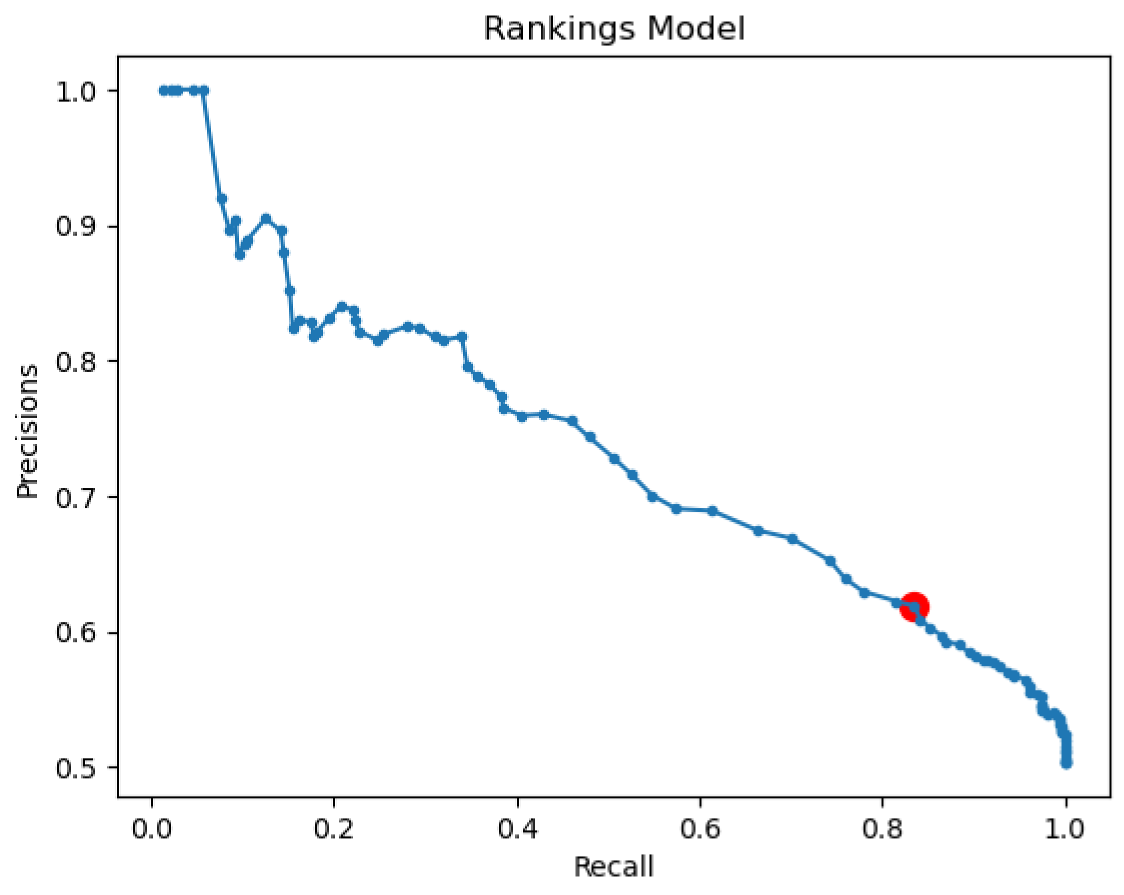
```
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
```

```
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Below, we are plotting the precision and recall from the eigth train split. For the eigth train split, we are using the f1 score to determine the optimal threshold. The optimal threshold from the f1 score is printed below.

```
In [ ]: plt.figure()
        plt.plot(recalls[np.where(recalls != 0)], precisions[:-1][np.where(recalls != 0)],  marker = ".")
        max_index = np.where(f1s == np.max(f1s))

        plt.scatter(recalls[max_index], precisions[max_index], color = "red", marker = "o", s = 100)
        plt.xlabel("Recall")
        plt.ylabel("Precisions")
        plt.title("Rankings Model")
        print('threshold')
        print(thresholds[max_index])
```

```
threshold
[0.44444444]
```

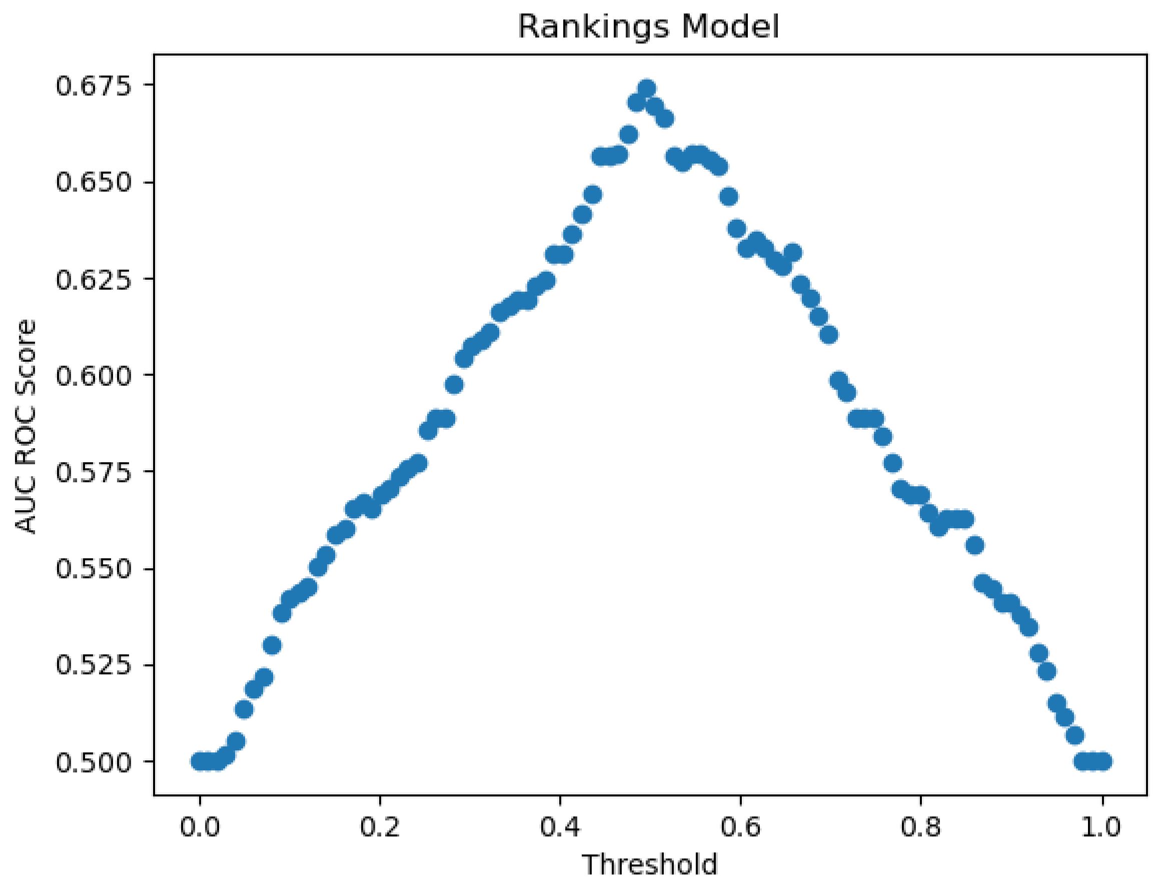


From the above plot, it is clear that the optimization process is essentially optimizing recall at the expense of making precision very close to 0.5, which effectively means that this statistic is prioritizing minimizing false negatives at the expense of tolerating more false positives. At this point on the precision recall curve, the precision is not substantially better than random (corresponding to precision = 0.5). If we were to set the threshold where this model suggests, a little less than half of the players we predict to win the match would actually lose the match.

Instead, we will use ROC-AUC to determine the optimum threshold for our model. In the below plot, we are plotting the rocauc score for the eigth train split. We will calculate the optimal threshold for each train split to determine the optimal threshold for each split. Finally, we will calculate the optimal roc-auc score for each train split, and then find the average and standard deviation in roc-auc score across the splits. This average and stardard deviation will tell us the predictive power of our model on the train set.

In [ ]:
```
optimal_treshold_splits = []
optimal_roc_aucs_train = []
for i in range(num_splits):
    max_index = np.where(rocaucs_splits_train[i] == np.max(rocaucs_splits_train[i]))
    optimal_treshold_splits.append(thresholds[max_index])
    optimal_roc_aucs_train.append(rocaucs_splits_train[i][max_index[0][0]])
plt.figure()
plt.scatter(thresholds, rocaucs)
plt.xlabel("Threshold")
plt.ylabel("AUC ROC Score")
plt.title("Rankings Model")
print("optimal_treshold: %.3f +/- %.3f" %(np.mean(optimal_treshold_splits), np.std(optimal_treshold_splits)))
print("auc roc score for test set: %.3f +/- %.3f" % (np.mean(optimal_roc_aucs_train),
                                                     np.std(optimal_roc_aucs_train)))
```

```
optimal_treshold: 0.499 +/- 0.005
auc roc score for test set: 0.675 +/- 0.005
```



Discussion of Significance

From above, we found that the optimal threshold occurs at 0.495 +/- 0.005, with a maximum auc roc score of 0.680 +/- 0.009. An auc roc score of 0.5 corresponds to essentially random classification, and an auc roc score of 1 is a perfect classifier. So this model seems to suggest that our model, when evaluated on the train set, is actually predictive, but is closer to randomness than to perfect classification. Also, a threshold value of around 0.5 stands to reason, since we expect that the player with the higher ranking should almost certainly have an advantage. We will accept that our model is better than randomly guessing if we achieve a p-value of below 5%.

First, we must assess if the optimal roc-aucs on the train set are normally distributed. To do that, we will use the normaltest function from scipy. The null hypothesis is that the distribution is normal.

```
In [ ]: test = scipy.stats.normaltest(optimal_roc_aucs_train)
        print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.822
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n
=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

With a p-value far in excess of 0.05, we can conclude that the distribution is normal. We can now perform a one-sided one sample t-test on our rocauc score to check whether it is distinguishably larger than 0.5 (randomly guessing the outcome of a match.

```
In [ ]: t_stat, p_value = scipy.stats.ttest_1samp(optimal_roc_aucs_train, .50, alternative = 'greater')
        print("From the test data, there is a %.11f probability that the true roc-auc "
              +"score of the model is less than or equal to 0.5." %p_value)
```

```
From the test data, there is a 0.00000000011 probability that the true roc-auc score of the model is less than or equal to 0.5.
```

Therefore, our model is distinguishable from randomly picking winners of matches, when evaluated on the train set, at the 5% level.

Next, we will test the model on the validation sets using essentially the same procedure but fixing the threshold based on the optimal value that we found on the train set.

```
In [ ]: thresholds = np.linspace(0, 1.0, 100)

        precisions_splits_val = []
        recalls_splits_val = []
        f1s_splits_val = []
        rocaucs_splits_val = []

        for i in range(num_splits):
            threshold = optimal_treshold_splits[i]
            yhat = []
            for x in np.array([[x] for x in X_val_splits[i][:, 0]]).flatten():
                if rank_model_splits[i].predict_proba([[x]])[0, 1] >= threshold:
                    yhat.append(1)
                if rank_model_splits[i].predict_proba([[x]])[0, 1] < threshold:
                    yhat.append(0)
            precisions_splits_val.append(precision_score(y_val_splits[i], yhat))
            recalls_splits_val.append(recall_score(y_val_splits[i], yhat))
            f1s_splits_val.append(f1_score(y_val_splits[i], yhat))
            rocaucs_splits_val.append(roc_auc_score(y_val_splits[i], yhat))
```

```
In [ ]: print("The ROC-AUCs score on the validation sets: %.3f +/- %.3f" %(np.mean(rocaucs_splits_val),
                                                                            np.std(rocaucs_splits_val)))
```

```
The ROC-AUCs score on the validation sets: 0.665 +/- 0.041
```

Discussion of Significance

To determine if our model is statistically distinguishable from randomly guessing when tested on the validation sets, we will perform the same one-sample t test. First we will assess the normality of the optimal roc-aucs scores from the validation sets.

```
In [ ]: test = scipy.stats.normaltest(rocaucs_splits_val)
        print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.657
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n
=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

With a p value far in excess of p = 0.05, we can conclude that the cross-validated validation roc-aucs come from a normal distribution.

In [ ]:
```
t_stat, p_value = scipy.stats.ttest_1samp(rocaucs_splits_val, .50, alternative = 'greater')
print("From the test data, there is a %.6f probability that the true roc-auc score of "
      +"the model is less than or equal to 0.5." %p_value)
```

```
From the test data, there is a 0.000007 probability that the true roc-auc score of the model is less than or equal to 0.5.
```

This score is slightly worse than the train set, but still distinguishably better than randomly guessing at the 5% level. Therefore, we can conclude that the model generalizes relatively well to the validation set, with the overall conclusions about its predictive power being the same as for the train set. Finally, we will calculate the ROC-AUC score for the test set.

In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)

precisions_splits_test = []
recalls_splits_test = []
f1s_splits_test = []
rocaucs_splits_test = []

for i in range(num_splits):
    threshold = optimal_treshold_splits[i]
    yhat = []
    for x in np.array([[x] for x in X_test[:, 0]]).flatten():
        if rank_model_splits[i].predict_proba([[x]])[0, 1] >= threshold:
            yhat.append(1)
        if rank_model_splits[i].predict_proba([[x]])[0, 1] < threshold:
            yhat.append(0)
    precisions_splits_test.append(precision_score(y_test, yhat))
    recalls_splits_test.append(recall_score(y_test, yhat))
    f1s_splits_test.append(f1_score(y_test, yhat))
    rocaucs_splits_test.append(roc_auc_score(y_test, yhat))
```

In [ ]:
```
print("The ROC-AUCs score on the test set: %.3f +/- %.4f" %(np.mean(rocaucs_splits_test),
                                                            np.std(rocaucs_splits_test)))
```

```
The ROC-AUCs score on the test set: 0.715 +/- 0.0002
```

Evaluation of Significance

We will perform the same 1-sided, 1-sample t test to determine if the model performs better on the test set than randomly guessing winners of matches. First we will assess the normality of the rocauc scores from the test set.

In [ ]:
```
test = scipy.stats.normaltest(rocaucs_splits_test)
print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.085
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n
=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

With a p value in excess of 0.05, we can conclude that the rocaucs from the test set come from a normal distribution and can proceed with a t-test.

In [ ]:
```
t_stat, p_value = scipy.stats.ttest_1samp(rocaucs_splits_test, .50, alternative = 'greater')
print("From the test data, there is a %.22f probability that the true roc-auc score of "
      +"the model is less than or equal to 0.5." %p_value)
```

```
From the test data, there is a 0.0000000000000000000001 probability that the true roc-auc score of the model is less than or equal to 0.5.
```

Finally, based on the exact same reasoning as for the train and test sets, we can claim that the model is statistically distinguishable from guessing at the 5% level. This seems to indicate that our model generalizes well to data outside of the set that it was trained on.

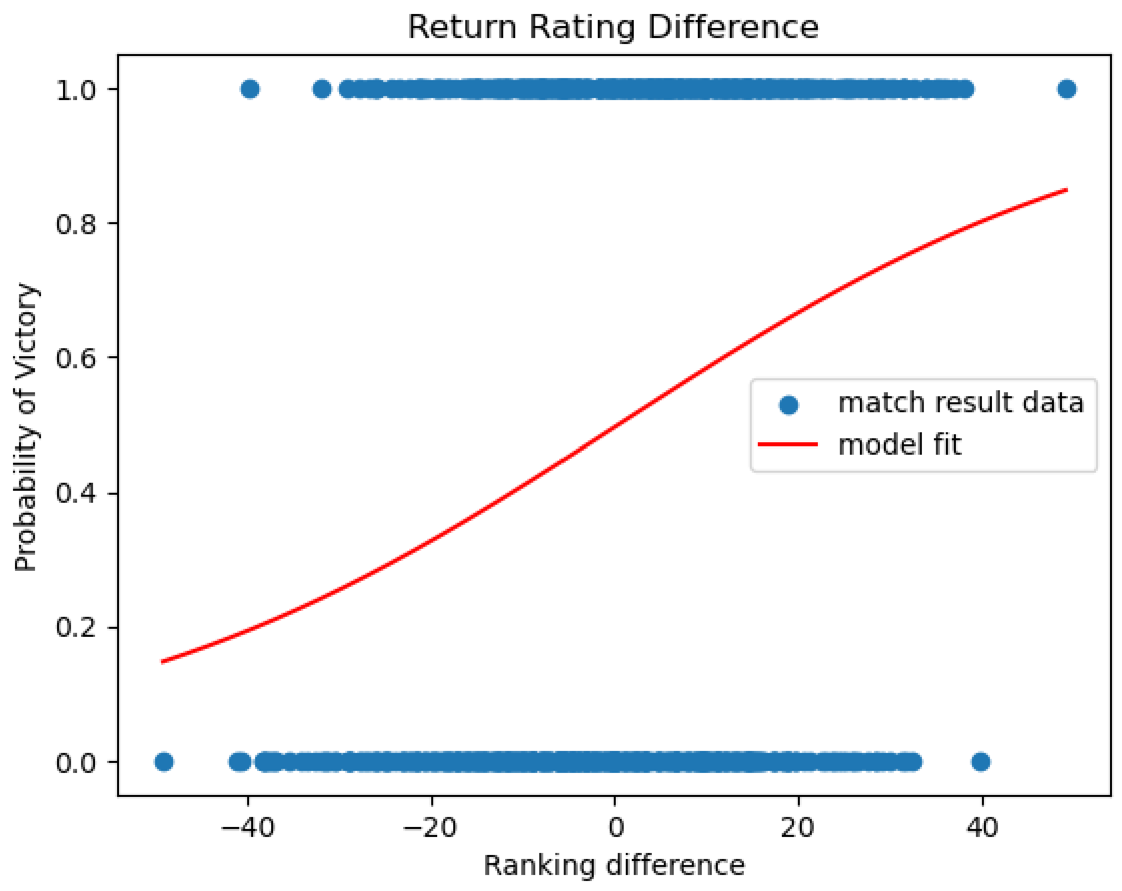
In the below cells, I will repeat the same steps but making the predictions from server ratings and return ratings, without cross-validation, just to assess their relevance. Once their relevance has been assessed, we will include them in a model that features the overall ranking, return rating, and server rating.

Predicting Winners from Return Ratings

Below, I have plotted a linear regression to the 800+ matches in our dataset based on the return rating differential, exactly as we did in the corresponding above cell from the overall rankings.

In [ ]:
```
return_X =  np.array([[x] for x in X[:, 2]])
return_model = LogisticRegression().fit(return_X, y)
fig1 = plt.figure('Return Rating Difference')
plt.title('Return Rating Difference')
plt.scatter(X[:,2],y, label = "match result data")
fit_x = np.linspace(np.min(return_X.flatten()), np.max(return_X.flatten()), 1000)
fit_x = [[x] for x in fit_x]
plt.plot(fit_x, return_model.predict_proba(fit_x)[:, 1], color = "red", label = "model fit")
plt.legend()
plt.xlabel("Ranking difference")
plt.ylabel("Probability of Victory")
```

Out[ ]:
```
Text(0, 0.5, 'Probability of Victory')
```



In the below cell, I am calculating precision, recall, f1 score, rocauc score at thresholds spaced evenly by 1% from 0 to 1.

In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)
precisions = []
recalls = []
f1s = []
rocaucs = []
```

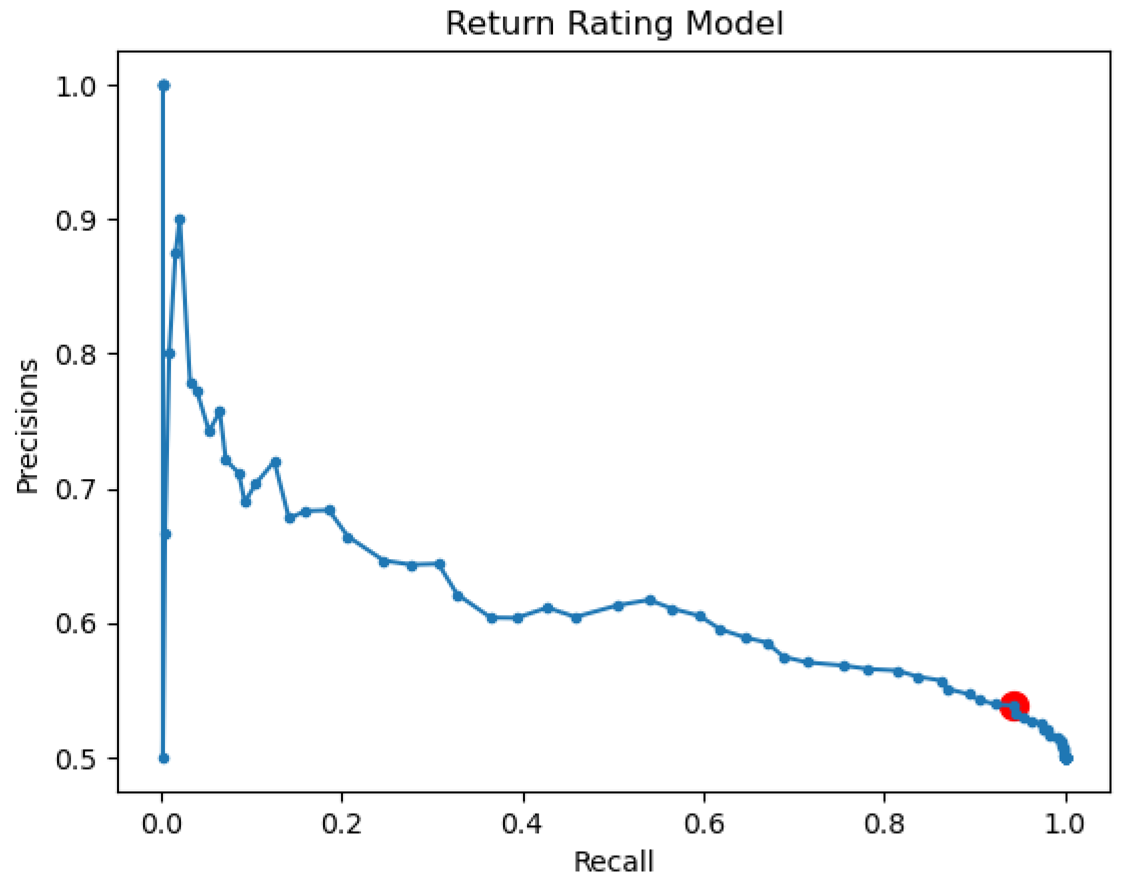
```
for threshold in thresholds:
    yhat = []
    for x in return_X.flatten():
        if return_model.predict_proba([[x]])[0, 1] >= threshold:
            yhat.append(1)
        if return_model.predict_proba([[x]])[0, 1] < threshold:
            yhat.append(0)
    precisions.append(precision_score(y, yhat))
    recalls.append(recall_score(y, yhat))
    f1s.append(f1_score(y, yhat))
    rocaucs.append(roc_auc_score(y, yhat))
precisions = np.array(precisions)
recalls = np.array(recalls)
```

```
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Below, I am plotting the precision vs recall plot for the return rating model, exactly as in the corresponding plot for the overall rating model.

In [ ]:
```
plt.figure()
plt.plot(recalls[np.where(recalls != 0)], precisions[:-1][np.where(recalls != 0)],  marker = ".")
max_index = np.where(f1s == np.max(f1s))

plt.scatter(recalls[max_index], precisions[max_index], color = "red", marker = "o", s = 100)
plt.xlabel("Recall")
plt.ylabel("Precisions")
plt.title("Return Rating Model")
print('threshold')
print(thresholds[max_index])
```

```
threshold
[0.35353535]
```

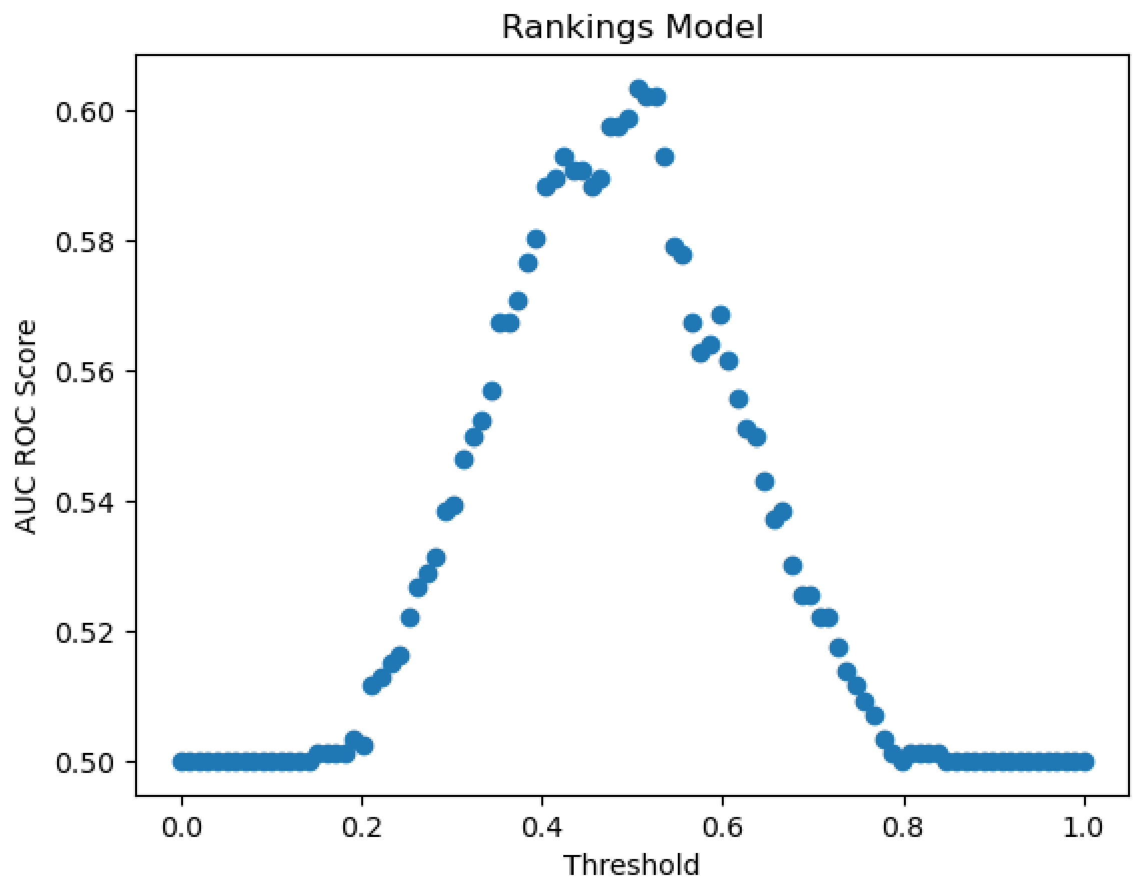


Based on the above plot, we can conclude that a very similar trend is occuring as above - prioritizing recall, and therefore minimizing false negatives at the expense of tolerating more false positives. If we were to set the threshold based on minimization of the F1 score, a little less than half of the players we predict to win the match would actually lose the match.

In the below plot, we plot the rocauc score curve vs the threshold from the rocauc scores calculated above and use that to obtain the optimal threshold and predictive power of our model.

In [ ]:
```
max_index = np.where(rocaucs == np.max(rocaucs))
plt.figure()
plt.scatter(thresholds, rocaucs)
plt.xlabel("Threshold")
plt.ylabel("AUC ROC Score")
plt.title("Rankings Model")
print("threshold")
print(thresholds[max_index])
print("max roc auc score")
print(rocaucs[max_index[0][0]])
```

```
threshold
[0.50505051]
max roc auc score
0.6034883720930232
```



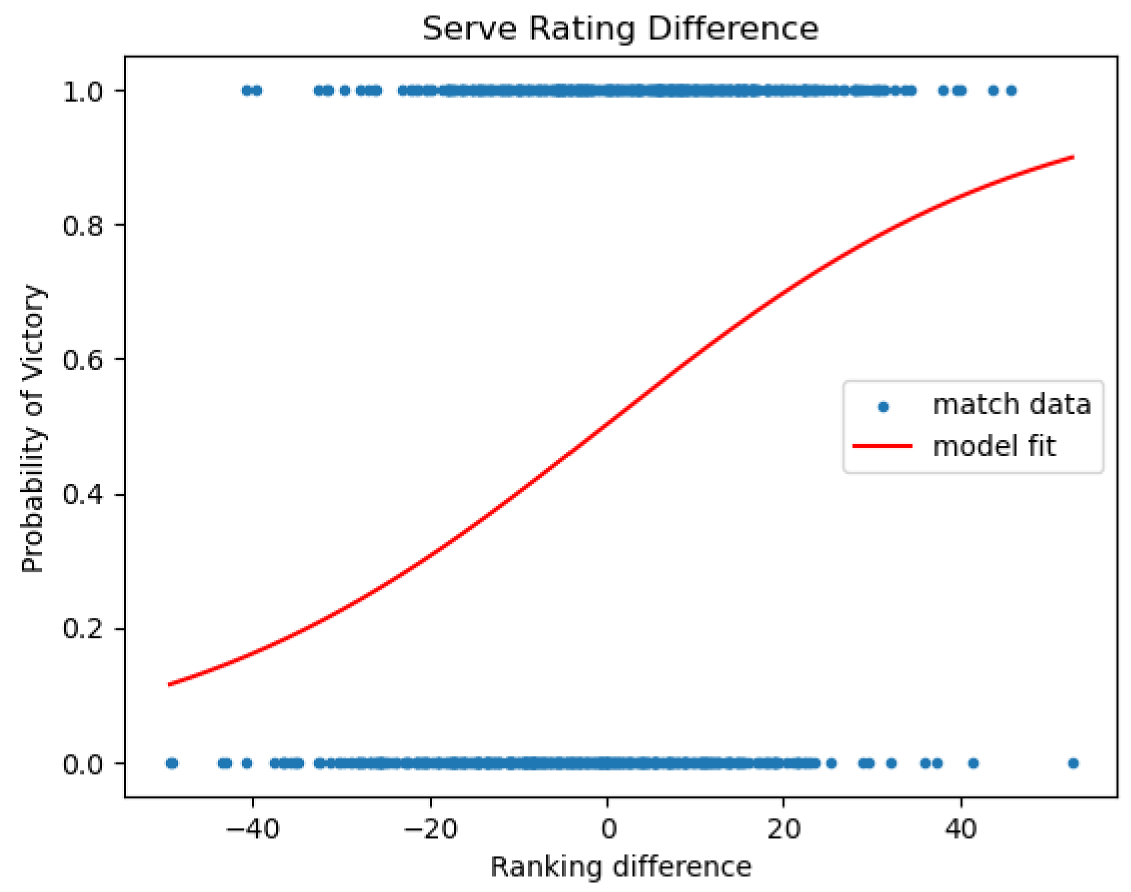
Since the max rocauc score is lower for the model based on return rating rather than the cross-validated rocauc score from the overall ranking, we can tentatively say that overall ranking is a better predictor, head to head, of who will win a tennis match than return rating. That being said, the return rating will probably be worthwhile to include in the model.

Predicting Winners from Serve Ratings

Below, we are fitting and plotting a 1-dimensional logistic regression to serve rating vs the outcomes of our match dataset. This procedure is almost identical to the previous two sections.

In [ ]:
```
serve_X =  np.array([[x] for x in X[:, 1]])
serving_model = LogisticRegression().fit(serve_X, y)
fig1 = plt.figure('Serve Rating Difference')
plt.title('Serve Rating Difference')
plt.scatter(X[:,1],y, label = "match data", marker = '.')
fit_x = np.linspace(np.min(serve_X.flatten()), np.max(serve_X.flatten()), 1000)
fit_x = [[x] for x in fit_x]
plt.plot(fit_x, serving_model.predict_proba(fit_x)[:, 1], color = "red", label = "model fit")
plt.legend()
plt.xlabel("Ranking difference")
plt.ylabel("Probability of Victory")
```

Out[ ]:
```
Text(0, 0.5, 'Probability of Victory')
```



In the below block, we are calculating the same binary statistics for the same 100 thresholds as in the previous two sections using an identical procedure.

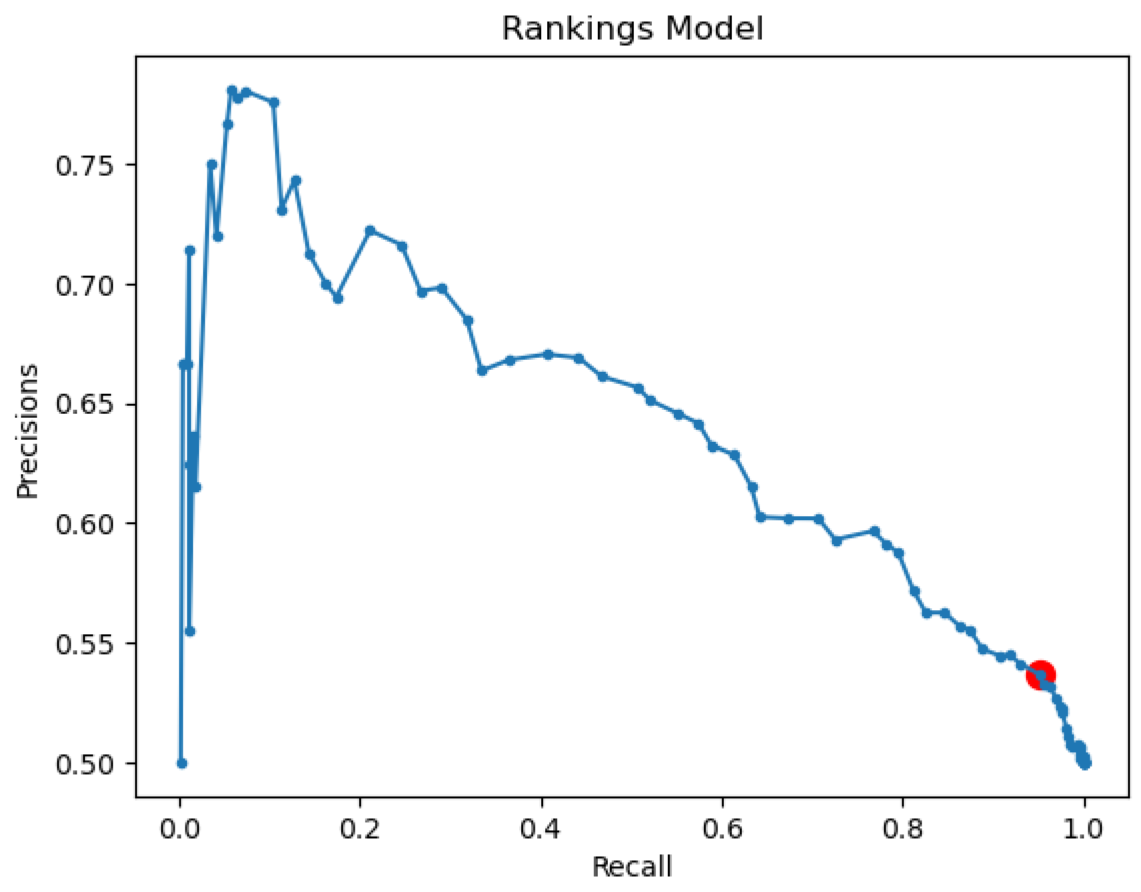
```
In [ ]: thresholds = np.linspace(0, 1.0, 100)
        precisions = []
        recalls = []
        f1s = []
        rocaucs = []
        for threshold in thresholds:
            yhat = []
            for x in serve_X.flatten():
                if serving_model.predict_proba([[x]])[0, 1] >= threshold:
                    yhat.append(1)
                if serving_model.predict_proba([[x]])[0, 1] < threshold:
                    yhat.append(0)
            precisions.append(precision_score(y, yhat))
            recalls.append(recall_score(y, yhat))
            f1s.append(f1_score(y, yhat))
            rocaucs.append(roc_auc_score(y, yhat))
        precisions = np.array(precisions)
        recalls = np.array(recalls)
```

```
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In the below plot, we are generating a precision vs recall plot using an identical procedure as the above two sections.

```
In [ ]: plt.figure()
        plt.plot(recalls[np.where(recalls != 0)], precisions[:-1][np.where(recalls != 0)],  marker = ".")
        max_index = np.where(f1s == np.max(f1s))

        plt.scatter(recalls[max_index], precisions[max_index], color = "red", marker = "o", s = 100)
        plt.xlabel("Recall")
        plt.ylabel("Precisions")
        plt.title("Rankings Model")
        print('threshold')
        print(thresholds[max_index])
```

```
threshold
[0.32323232]
```

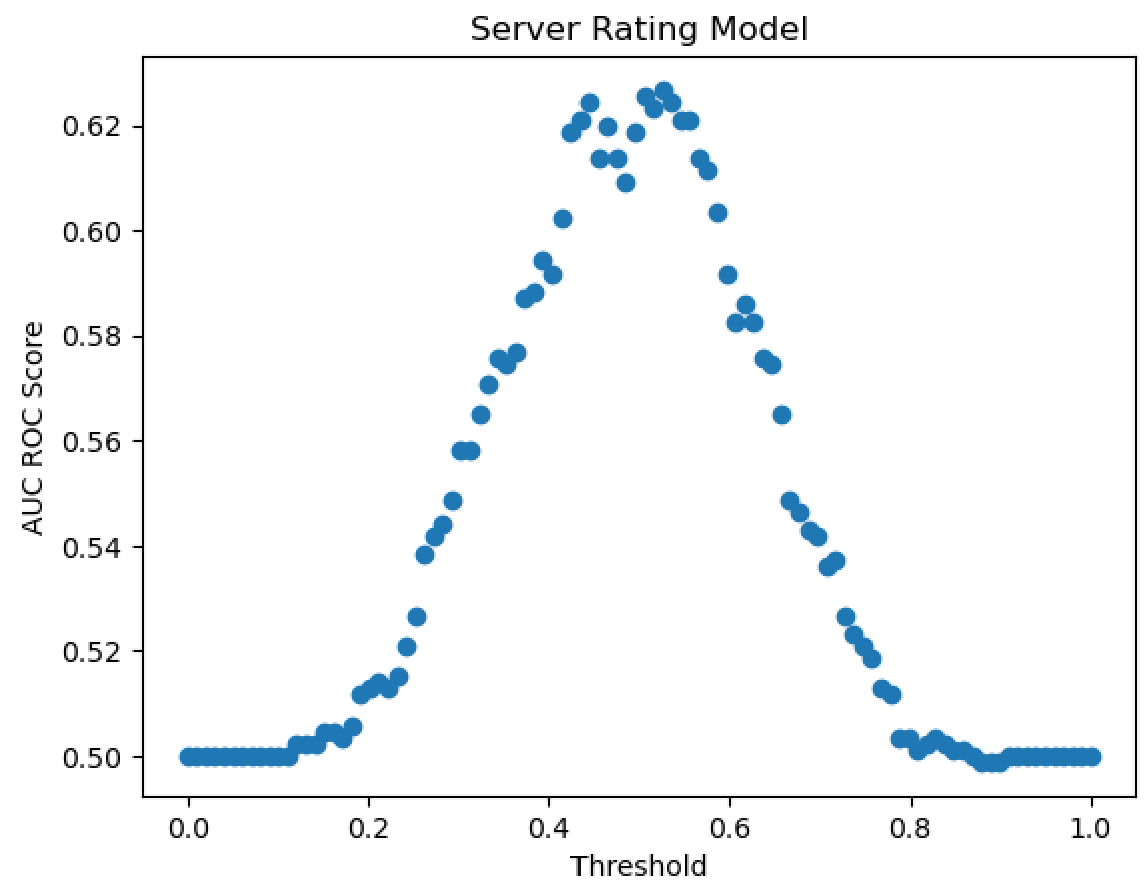


We notice the same trend of a low optimal threshold and the same "hardly better than random" precision with the benefit of a very high recall. This is probably not where we want to optimally set the threshold as false negatives are about as important as false positives in our model.

Below, we are generating an rocauc score vs threshold plot using an identical procedure as in the proceeding two sections.

In [ ]:
```
max_index = np.where(rocaucs == np.max(rocaucs))
plt.figure()
plt.scatter(thresholds, rocaucs)
plt.xlabel("Threshold")
plt.ylabel("AUC ROC Score")
plt.title("Server Rating Model")
print("threshold")
print(thresholds[max_index])
print("max roc auc score")
print(rocaucs[max_index[0][0]])
```

```
threshold
[0.52525253]
max roc auc score
0.6267441860465115
```



Since the max rocauc score is lower for the model based on serve rating rather than the cross-validated rocauc score from the overall ranking, we can tentatively say that overall ranking is a better predictor, head to head, of who will win a tennis match than serve rating. That being said, the serve rating will probably be worthwhile to include in the model.

Before proceeding with a multivariate model, let's assess the colinearity of the serve rating, return rating, and overall ranking.

In [ ]:
```
print(overall_df[["rank_diff", "serve_diff", "return_diff"]].corr())
```

```
             rank_diff  serve_diff  return_diff
rank_diff     1.000000    0.418351     0.411180
serve_diff    0.418351    1.000000    -0.243635
return_diff   0.411180   -0.243635     1.000000
```

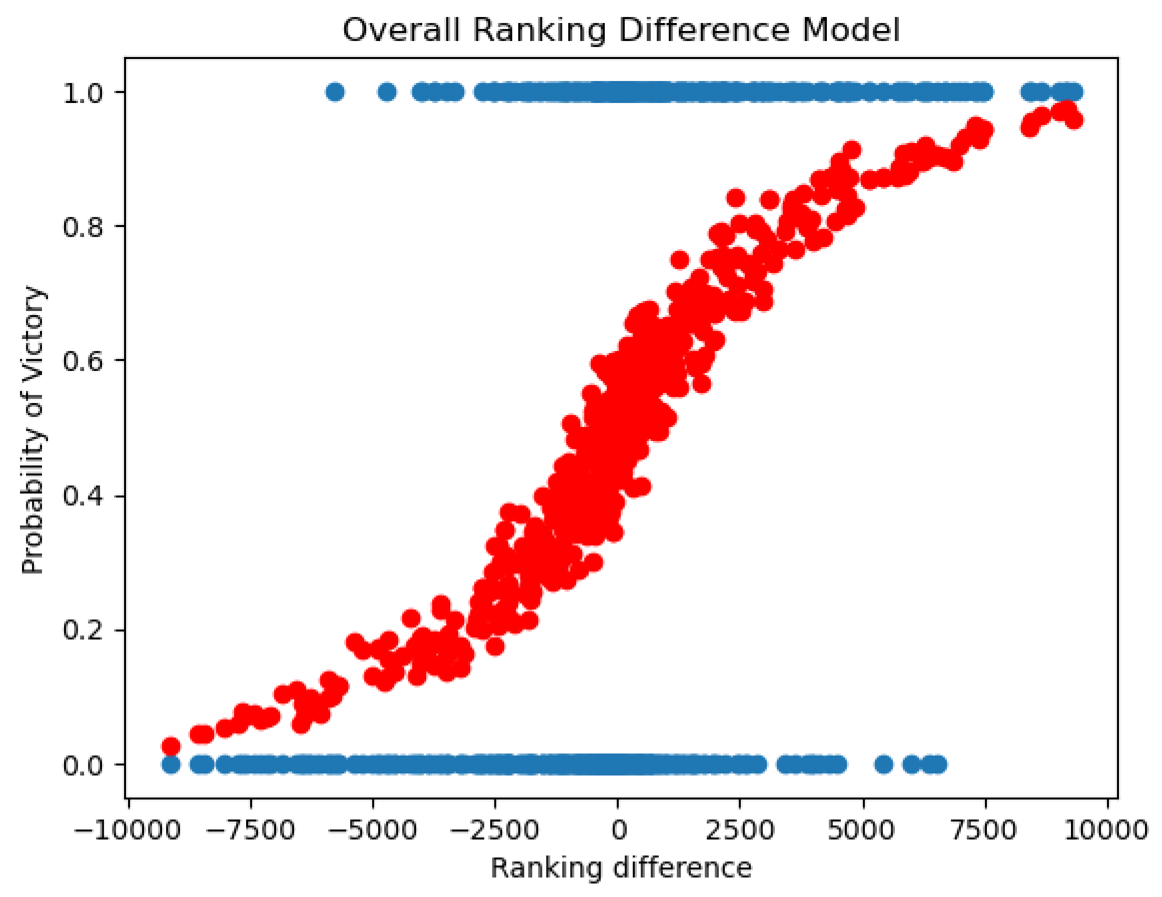
Our domain knowledge suggests that the difference in overall rankings, return ratings, and serve ratings are all separately relevant for the outcome of a tennis match between two players. It appears that they have a dagnerous amount of multicollinearity, particuarly between return_diff and rank_diff. We will lean on our domain knowledge here and take a risk with the multicollinearity. We will attempt to cross-validate our results with 8 splits of data using the same 70/10/20 train/validation/test split as we used for the model based on the overall rankings alone.

In [ ]:
```
num_splits = 8
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
kf = KFold(n_splits = num_splits)
X_train_splits = []
y_train_splits = []
X_train_splits = []
X_val_splits = []
y_val_splits = []

for train, val in kf.split(X_train):
    X_train_splits.append(X_train[train])
    y_train_splits.append(y_train[train])
    X_val_splits.append(X_train[val])
    y_val_splits.append(y_train[val])
```

In the below cell, I am training a logistic regression to the 8 splits of training data that I generated. I will then plot the data and fit from the eighth split, taking the projection onto the "overall ranking difference" axis. We immediately notice that the logistic regression curve has gotten "fuzzier", which matches our intuition since it is now taking into account two additional variables. We further notice that it seems like these variables are making a "band" around the rough shape of the logistic curve that we would expect from fitting onto the overall ranking difference alone, which would suggest that the most heavily-weighted variable in the model is the ranking difference.

In [ ]:
```
multivar_model_splits = []
for i in range(num_splits):
    #format the inputs
    multivar_X = X_train_splits[i]
    multivar_model_splits.append(LogisticRegression().fit(multivar_X, y_train_splits[i]))

fig1 = plt.figure('Overall Ranking Difference Model')
plt.title('Overall Ranking Difference Model')
plt.scatter(X_train_splits[num_splits-1][:,0],y_train_splits[num_splits-1], label = "match_data")
fit_x = np.linspace(np.min(X_train_splits[num_splits-1].flatten()),
                    np.max(X_train_splits[num_splits-1].flatten()), 1000)
plt.scatter(multivar_X[:, 0], multivar_model_splits[num_splits-1].predict_proba(multivar_X)[:, 1],
            color = "red", label = "model fit")
plt.xlabel("Ranking difference")
plt.ylabel("Probability of Victory")
```

Out[ ]:
```
Text(0, 0.5, 'Probability of Victory')
```



In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)

precisions_splits_train = []
recalls_splits_train = []
f1s_splits_train = []
rocaucs_splits_train = []


for i in range(num_splits):
    precisions = []
    recalls = []
    f1s = []
    rocaucs = []
    for threshold in thresholds:
        yhat = []
        for x in X_train_splits[i]:
            if multivar_model_splits[i].predict_proba([x])[0, 1] >= threshold:
                yhat.append(1)
            if multivar_model_splits[i].predict_proba([x])[0,1] < threshold:
                yhat.append(0)
        precisions.append(precision_score(y_train_splits[i], yhat))
        recalls.append(recall_score(y_train_splits[i], yhat))
        f1s.append(f1_score(y_train_splits[i], yhat))
        rocaucs.append(roc_auc_score(y_train_splits[i], yhat))
    precisions = np.array(precisions)
    recalls = np.array(recalls)
    precisions_splits_train.append(precisions)
    recalls_splits_train.append(recalls)
    f1s_splits_train.append(f1s)
    rocaucs_splits_train.append(rocaucs)
```

```
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
```

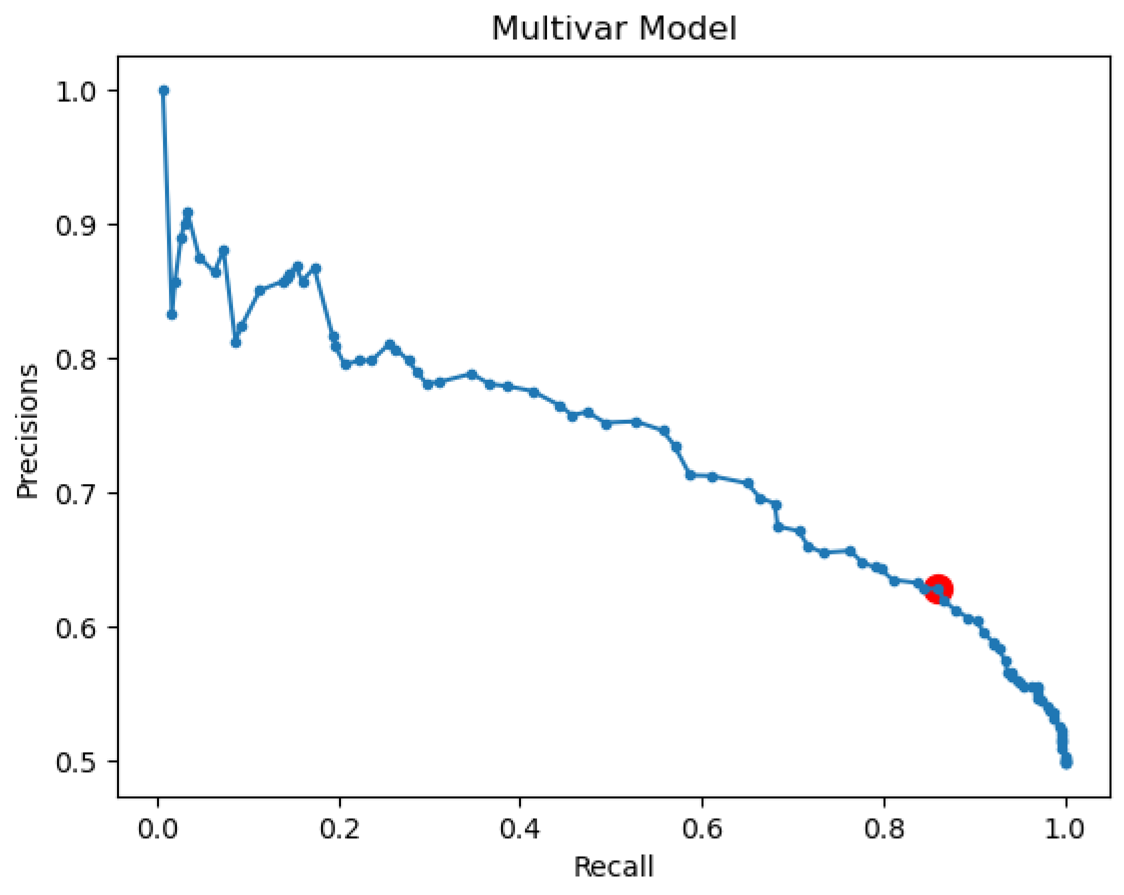
```
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In the below block, we calculate precision, recall, F1 scores, and aucroc scores for 100 thresholds spaced evenly between 0 and 1, mimicking almost exactly the procedure that we used for the 1D model except this time feeding the model 3 inputs of 1 input. We do this for each of the 8 train sets from the cross-validation splits. The optimal thresholds, precisions, recalls, f1s, and rocaucs are calculated first on the train data.

Next, we generate a precision recall curve using the exact same methodology as for the 1D models. We find that we are able to strike a better balance between precision and recall with a threshold much closer to 0.5 than in the 1 dimensional models.

In [ ]:
```
plt.figure()
plt.plot(recalls[np.where(recalls != 0)], precisions[:-1][np.where(recalls != 0)],  marker = ".")
max_index = np.where(f1s == np.max(f1s))

plt.scatter(recalls[max_index], precisions[max_index], color = "red", marker = "o", s = 100)
plt.xlabel("Recall")
plt.ylabel("Precisions")
plt.title("Multivar Model")
print('threshold')
print(thresholds[max_index])
```

```
threshold
[0.39393939]
```

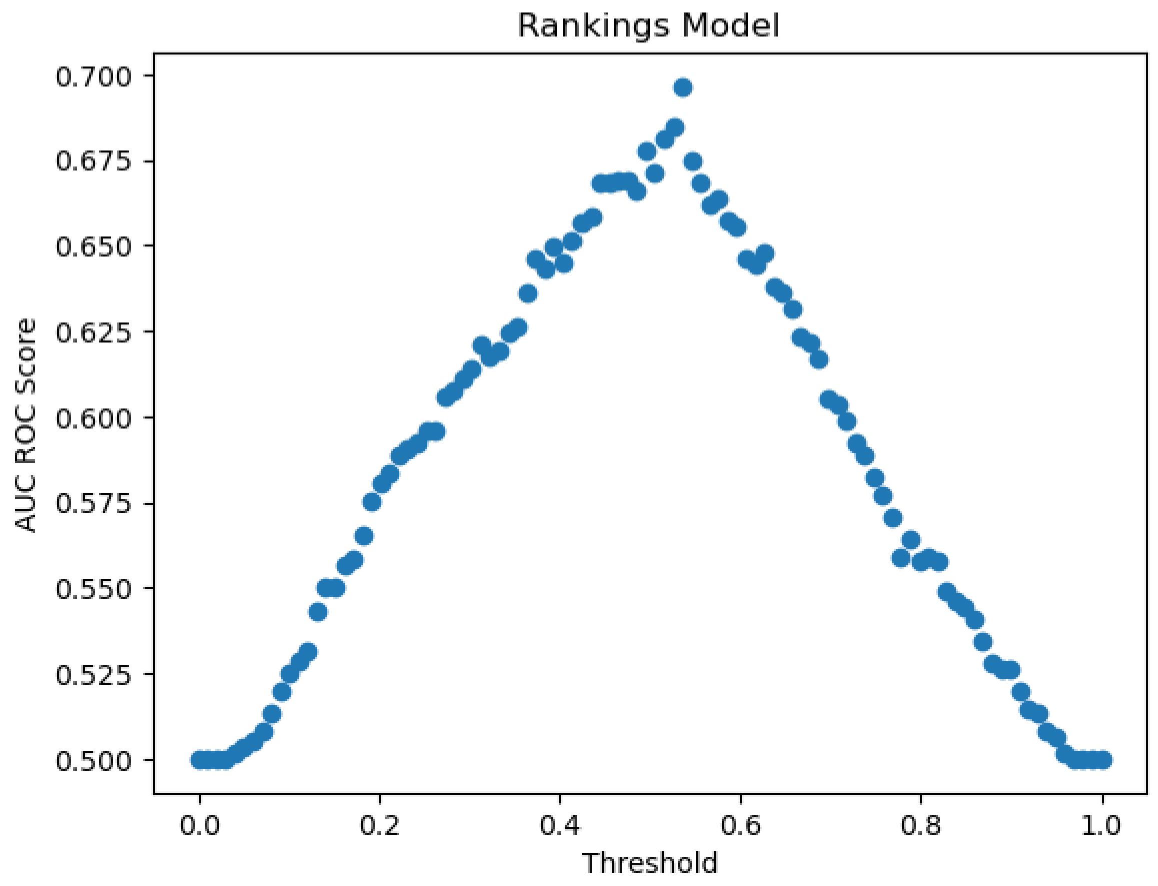


Taking into account the three ratings, we are able to achieve a recall in excess of 80% and a precision in excess of 0.60% simultaneously, meaning that of the players we predict to win their match, around 60% of them actually did win their match, and of the players who actually did win their match, we were able to identify 80% of them. The F1 score still shows a slight preference for minimizing false negatives at the expense of a larger number of false positives, which there is no reason to prefer in our particular case.

Next I will generate a roc-auc threshold plot using the same methodology as in the 1 dimensional cases for the 8th split. I will also calculate the optimal roc-auc score for each split, average them, and calculate their standard deviations.

In [ ]:
```
optimal_treshold_splits = []
optimal_roc_aucs_train = []
for i in range(num_splits):
    max_index = np.where(rocaucs_splits_train[i] == np.max(rocaucs_splits_train[i]))
    optimal_treshold_splits.append(thresholds[max_index])
    optimal_roc_aucs_train.append(rocaucs_splits_train[i][max_index[0][0]])
plt.figure()
plt.scatter(thresholds, rocaucs)
plt.xlabel("Threshold")
plt.ylabel("AUC ROC Score")
plt.title("Rankings Model")
print("optimal_treshold: %.3f +/- %.3f" %(np.mean(optimal_treshold_splits), np.std(optimal_treshold_splits)))
print("auc roc score for test set: %.3f +/- %.3f" % (np.mean(optimal_roc_aucs_train),
                                                      np.std(optimal_roc_aucs_train)))
```

```
optimal_treshold: 0.514 +/- 0.038
auc roc score for test set: 0.688 +/- 0.008
```



The total auc-roc is better than any of the three single-variable models that we developed. However, it is only very barely better than our model based on the overall ranking and nothing else.

Evaluation of Significance

Next, we will perform a one-sample one-sided t test to assess whether the model is distinguishable from randomly guessing the outcome of tennis matches for the train data. First, we will assess whether the roc-auc scores from the train data come from a normal distribution.

In [ ]:
```
test = scipy.stats.normaltest(optimal_roc_aucs_train)
print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.822
```

```
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n
=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Since the null hypothesis is that the distribution is normally distributed, and our p-value is far in excess of 0.05, we fail to reject the null hypothesis and can proceed with the t-test.

In [ ]:
```
t_stat, p_value = scipy.stats.ttest_1samp(optimal_roc_aucs_train, .50, alternative = 'greater')
print("From the test data, there is a %.12f probability that the true roc-auc score of "
      +"the model is less than or equal to 0.5." %p_value)
```

```
From the test data, there is a 0.000000000004 probability that the true roc-auc score of the model is less than or equal to 0.5.
```

This suggests that our model is distinguishable from randomly guessing the outcome of tennis matches on the train set far in excess of the 5% level. Next, we will evaluate the performance of the model on the validation set.

In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)

precisions_splits_val = []
recalls_splits_val = []
f1s_splits_val = []
rocaucs_splits_val = []
```

```
for i in range(num_splits):
    threshold = optimal_treshold_splits[i]
    yhat = []
    for x in X_val_splits[i]:
        if multivar_model_splits[i].predict_proba([x])[0, 1] >= threshold:
            yhat.append(1)
        if multivar_model_splits[i].predict_proba([x])[0, 1] < threshold:
            yhat.append(0)
    precisions_splits_val.append(precision_score(y_val_splits[i], yhat))
    recalls_splits_val.append(recall_score(y_val_splits[i], yhat))
    f1s_splits_val.append(f1_score(y_val_splits[i], yhat))
    rocaucs_splits_val.append(roc_auc_score(y_val_splits[i], yhat))
```

In [ ]:
```
print("The ROC-AUCs score on the validation sets: %.3f +/- %.3f" %(np.mean(rocaucs_splits_val),
                                                                    np.std(rocaucs_splits_val)))
```

```
The ROC-AUCs score on the validation sets: 0.669 +/- 0.056
```

Evaluation of Significance

We will continue with the usual procedure of assessing normality and then performing the t-test.

In [ ]:
```
test = scipy.stats.normaltest(optimal_roc_aucs_train)
print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.822
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n
=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

In [ ]:
```
t_stat, p_value = scipy.stats.ttest_1samp(rocaucs_splits_val, .50, alternative = 'greater')
print("From the test data, there is a %.5f probability that the true roc-auc score of "
      +"the model is less than or equal to 0.5." %p_value)
```

```
From the test data, there is a 0.00005 probability that the true roc-auc score of the model is less than or equal to 0.5.
```

This suggests that our model is distinguishable from randomly guessing the outcome of tennis matches on the validation set far in excess of the 5% level. Next, we will evaluate the performance of the model on the test set.

In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)

precisions_splits_test = []
recalls_splits_test = []
f1s_splits_test = []
rocaucs_splits_test_multivar = []

for i in range(num_splits):
    threshold = optimal_treshold_splits[i]
    yhat = []
    for x in X_test:
        if multivar_model_splits[i].predict_proba([x])[0, 1] >= threshold:
            yhat.append(1)
        if multivar_model_splits[i].predict_proba([x])[0, 1] < threshold:
            yhat.append(0)
    precisions_splits_test.append(precision_score(y_test, yhat))
    recalls_splits_test.append(recall_score(y_test, yhat))
    f1s_splits_test.append(f1_score(y_test, yhat))
    rocaucs_splits_test_multivar.append(roc_auc_score(y_test, yhat))
```

In [ ]:
```
print("The ROC-AUCs score on the test sets: %.3f +/- %.3f" %(np.mean(rocaucs_splits_test_multivar),
                                                             np.std(rocaucs_splits_test_multivar)))
```

```
The ROC-AUCs score on the test sets: 0.704 +/- 0.004
```

Evaluation of Significance

We will now perform the exact same one-sided one-sample t test for significance to determine if the model we developed is better than randomly deciding the winner of a match on the test set.

In [ ]:
```
test = scipy.stats.normaltest(rocaucs_splits_test_multivar)
print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.567
/Users/colinmurphy/opt/anaconda3/envs/info2950/lib/python3.9/site-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n
=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

In [ ]:
```
t_stat, p_value = scipy.stats.ttest_1samp(rocaucs_splits_test_multivar, .50, alternative = 'greater')
print("From the test data, there is a %.13f probability that the true roc-auc score of "
      +"the model is less than or equal to 0.5." %p_value)
```

```
From the test data, there is a 0.0000000000004 probability that the true roc-auc score of the model is less than or equal to 0.5.
```

Clearly, our model is distinguishable from randomly guessing the outcome of the tennis match on the test set far in excess of the 5% level.

Comparing the Single Variable (Overall Ranking Only) and Multivariable Model

Now, we want to determine if including the serve ratings and return ratings improved the predictive power of our model as measured by the roc-auc score on the test set. To do this, we will perform a one-sided two-sample t test from the optimal roc-auc scores that we calculated for each model.

In [ ]:
```
t_stat, p_value = scipy.stats.ttest_ind(rocaucs_splits_test_multivar,
                                        rocaucs_splits_test, alternative = "greater")
print("The probability that roc-auc score on the test set from the multivariate model "
      +"is less than or equal to the roc-auc score from the single-variable overall ranking only model is %.4f." % p_value)
```

```
0.7146213657876943
0.7038624070317783
THe probability that roc-auc score on the test set from the multivariate model is less than or equal to the roc-auc score from the single-variable overall ranking only model is
1.0000.
```

This suggests that our multivariable model performs does not perform any better on its respective test set, as measured by auc-roc, than the single variable model at the 5% level. Therefore, we can conclude that including additional variables does not improve the predictive power of our model.

Interpreting Regressions

To get a sense of the dependence of our predictions of the probability for outcomes of tennis matches by our model on the variables that we considered in the model, we will train two models to the complete data set of roughly 800 matches. The methodology that these models are being trained with has been cross-validated and shown to be significantly better than randomly picking the outcome of matches. While we will not cross-validate these models specifically, we feel that, for the sake of interpretation, these models will demonstrate the dependence on variables that we desire to show.

In [ ]:
```
model = LogisticRegression().fit(X, y)
rank_X =  np.array([[x] for x in X[:, 0]])

rank_model = LogisticRegression().fit(rank_X, y)


print('\n for rank model')
print('coefficient: '+str(rank_model.coef_[0][0]))
print('intercept: '+str(rank_model.intercept_[0]))

print('\n for serve model')
print('coefficient: '+str(serving_model.coef_[0][0]))
print('intercept: '+str(serving_model.intercept_[0]))

print('\n for rank model')
print('coefficient: '+str(return_model.coef_[0][0]))
print('intercept: '+str(return_model.intercept_[0]))

print('\n For multivariate model')
print("rank, serve, and return coefficients: "+str(model.coef_[0][0])+", "+str(model.coef_[0][1])
      +", "+str(model.coef_[0][2]))
print("intercept: "+str(model.intercept_[0]))
```

```
 for rank model
coefficient: 0.0003858758373498945
intercept: -7.710486809948004e-09

 for serve model
coefficient: 0.041356223738606694
intercept: 0.008834884559995378

 for rank model
coefficient: 0.03536415876779259
intercept: -0.017220181949380194

 For multivariate model
rank, serve, and return coefficients: 0.00023530977411153288, 0.027268731613258202, 0.020392671865657064
intercept: -0.0001101355422372207
```

For the rank model, for a 1 unit increase in the difference in rank between the players, the probability that the first player wins increases by a factor of e^(0.000386)

For the serve model, for a 1 unit increase in the difference in serve ratings between the players, the probability that the first player wins increases by a factor of e^(0.0413)

For the return model, for a 1 unit increase in the difference in return ratings between the players, the probability that the first player wins increases by a factor of e^(0.0354)

For the multivariable model, for a 1 unit increase in the difference in rank points between the two players, all else remaining the same, the probability that the first player player wins increases by a factor of e^(0.0002). For a 1 unit increase in serving points between the two players, all else remaining the same, the probability that the first player wins increases by a factor of e^(0.0272). For a 1 unit increase in the difference in return points between the two players, all else remaining the same, the probability that the first player wins increases by a factor of e^(0.0204)

The coefficients for the models are low, which supports the figures having very gradual slopes. Even the highest differences only provide a 90 percent chance of winning in the models. This shows that the rating differences do not have a very large impact on the outcome of matches. There are a lot of instances where a player with lower ratings still wins. There are likely other parameters that have higher impacts on the score. In addition, the players that we are collecting and analyzing data for are the top players in the world. Every player has an extremely high level of skill and experience, and as such at this level the differences in skill are minimal, therefore making the differences in ratings even less impactful.

The intercepts for the models are extremely low, with the rank and return models having an essentially negligible intercept. This shows that when the rating difference between two players is 0, there is a 50/50 percent chance of winning. This makes sense as if their ratings are 0, they should have an equal chance of winning.

Hard vs Clay

This next section covers our analysis of whether male professional tennis players serve differently on different court types.

Hypothesis 1: Professional men's tennis players serve better on hard courts than clay courts. We believe that this would possibly be due to the fact that professional tennis players mainly practice and play on hard courts. This answer may suggest that we need to change our model to include what surface the match occurred on. Analysis: The ATP compiles server ratings for each professional player on each playing surface.

We can perform a two sample t-test between the average server rating on hard courts and clay courts. We predict that there will be a statistically-significant difference between the men;s server ratings that indicates that the population mean server rating on hard courts is higher than on clay courts.

To start, we'll define a short function to return the serve rating for all the players

In [ ]:
```
def GetServe(id, row):
    t = row[id].text
    first_space = t.find(' ')
    y = t[first_space+1:]
    stat = y[y.find('\n')+1:y.find('\n') +6]
    first_slash = t.index('\n')
    name = t[first_slash+1:t.rfind('\n')]
    return {'Name': name, 'Serve Rating': float(stat)}
```

Now, we scrape the data from the ATP website for serve rankings for both hard and clay courts. First, hard courts.

In [ ]:
```
hard_link = 'https://www.atptour.com/en/stats/leaderboard?boardType=serve&'
+'timeFrame=52Week&surface=hard&versusRank=all&formerNo1=false'
clay_link = 'https://www.atptour.com/en/stats/leaderboard?boardType=serve&'
+'timeFrame=52Week&surface=clay&versusRank=all&formerNo1=false'
```

In [ ]:
```
hard_browser = webdriver.Firefox(executable_path=str(str(os.getcwd())+'/geckodriver'))
hard_browser.get(hard_link)
```

In [ ]:
```
hard_table = hard_browser.find_element_by_id('leaderboardTable')
r = hard_table.find_elements_by_tag_name('tr')
hard_df = pd.DataFrame(GetServe(0,r), index = [0])
for x in range(len(r))[1:]:
    record = pd.Series(GetServe(x,r))
    hard_df = pd.concat([hard_df, record.to_frame().T], ignore_index = True)
```

In [ ]:
```
hard_df.head()
```

Out[ ]:

	Name	Serve Rating
0	Novak Djokovic	296.8
1	Hubert Hurkacz	296.5
2	Stefanos Tsitsipas	295.3
3	Nicolas Jarry	294.0
4	Thanasi Kokkinakis	293.4

Now we repeat the process for clay courts.

In [ ]:
```
clay_browser = webdriver.Firefox(executable_path=str(str(os.getcwd())+'/geckodriver'))
clay_browser.get(clay_link)
```

In [ ]:
```
clay_table = clay_browser.find_element_by_id('leaderboardTable')
r = clay_table.find_elements_by_tag_name('tr')
clay_df = pd.DataFrame(GetServe(0,r), index = [0])
for x in range(len(r))[1:]:
    record = pd.Series(GetServe(x,r))
    clay_df = pd.concat([clay_df, record.to_frame().T], ignore_index = True)
```

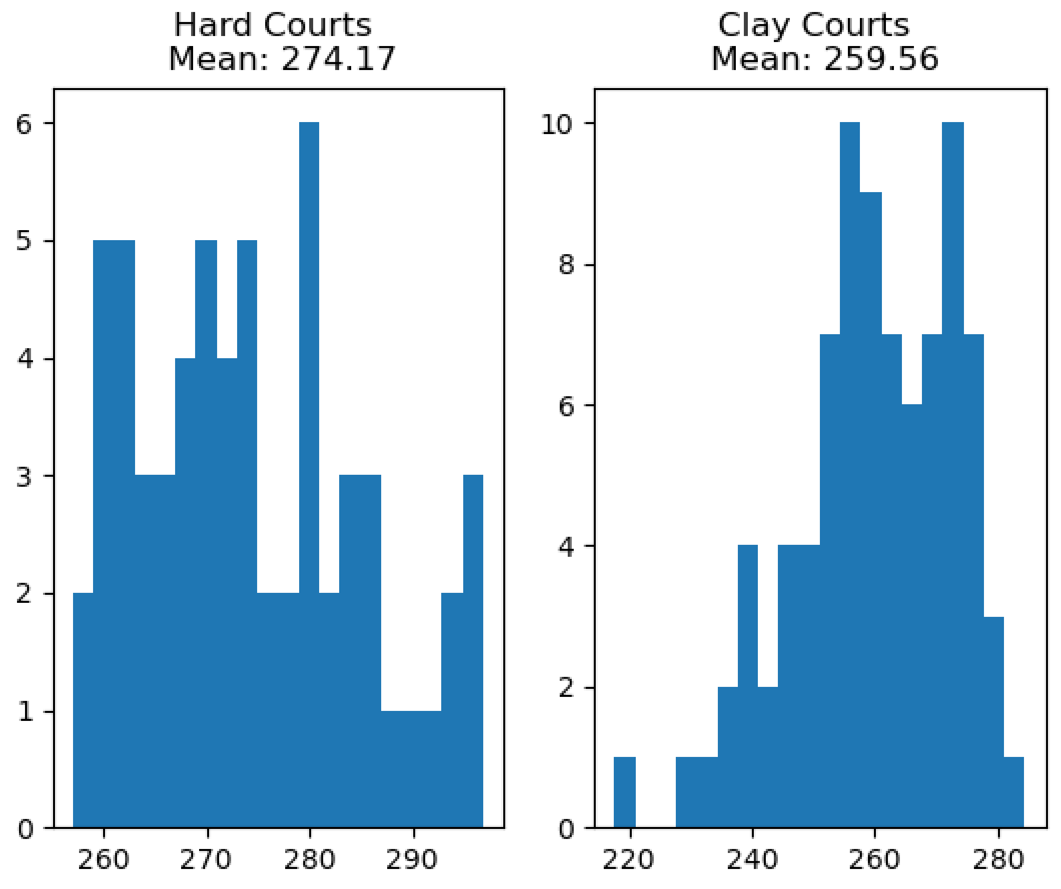
In [ ]:
```
clay_df.head()
```

Out[ ]:

	Name	Serve Rating
0	Nicolas Jarry	284.4
1	Stefanos Tsitsipas	279.4
2	Alexei Popyrin	279.3
3	Karen Khachanov	278.7
4	Carlos Alcaraz	277.6

Just by looking at the distributions and the means for each, we can tell they are different.

In [ ]:
```
hard_mean = round(np.mean(hard_df['Serve Rating']),2)
clay_mean = round(np.mean(clay_df['Serve Rating']),2)
f, (ax1, ax2) = plt.subplots(1, 2, width_ratios = [1,1])
ax1.hist(hard_df['Serve Rating'], bins = 20)
ax1.set_title('Hard Courts \n Mean: ' + str(hard_mean))
ax2.hist(clay_df['Serve Rating'], bins = 20)
ax2.set_title('Clay Courts \n Mean: ' + str(clay_mean));
```



In order to do a two sample t-test, we first need to make sure each sample is more or less normally distributed. We do this by testing for normality

In [ ]:
```
hard_courts = hard_df['Serve Rating'].to_numpy().astype('int')
test = scipy.stats.normaltest(hard_courts)
print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.105
```

In [ ]:
```
clay_courts = clay_df['Serve Rating'].to_numpy().astype('int')
test = scipy.stats.normaltest(clay_courts)
print('pvalue = ' + str(round(test.pvalue,3)))
```

```
pvalue = 0.064
```

Since both p-values are greater than 0.05, we fail to reject the null - each sample is nearly normally distributed. However, since we have unequal sample sizes, we will use a Welch's t-test to test for independece.

In [ ]:
```
t_test = scipy.stats.ttest_ind(hard_courts, clay_courts, equal_var = False)
print('pvalue = ' + str(t_test.pvalue))
```

```
pvalue = 1.1600311174431524e-11
```

Based on this p-value significantly smaller than 0.05, we can reject the null hypothesis. Players do indeed serve differently on clay and hard courts.

Spain vs. the World

Spain is routinely ranked by many as the number 1 country in tennis. This is in large due to the fact that Spain places much more importance on tennis than other sports, compared to other countries. This leads to more time and money being spent on tennis training and facilities in Spain compared to other countries

Here we will try to find whether the data agrees with this. We will take the nationalities of all the players, and generate probabilities for Spain winning a match vs. another country winning a match. We will create a normal distribution using the probablility of an average non-Spain country winning a tennis match, and compare this to the amount of wins that Spain has accumulated this year.

The nationalities previously weren't include in the overall DataFrame used. We have webscraped the nationalities for the serve and return leaders separately, so they are included in their respective DataFrames and must be combined with the overall_df DataFrame, containing all of the tennis matches played in 2023

In [ ]:
```
overall_df = pd.read_csv('overall_df.csv')
serve_df = pd.read_csv('serve_leader_df.csv')
return_df = pd.read_csv('serve_return.csv')

winner_country = []
loser_country = []

for i in range(len(overall_df)):
  if serve_df[serve_df.name == overall_df.loc[i].winner_name].country.iloc[0] is None:
    winner_country.append(return_df[return_df.Name == overall_df.loc[i].winner_name].country.iloc[0])
  else: winner_country.append(serve_df[serve_df.name == overall_df.loc[i].winner_name].country.iloc[0])

  if serve_df[serve_df.name == overall_df.loc[i].loser_name].country.iloc[0] is None:
    loser_country.append(return_df[return_df.Name == overall_df.loc[i].loser_name].country.iloc[0])
  else: loser_country.append(serve_df[serve_df.name == overall_df.loc[i].loser_name].country.iloc[0])

overall_df['winner_country'] = winner_country
overall_df['loser_country'] = loser_country

overall_df = overall_df.drop(overall_df.columns[0], axis=1)
overall_df
```

Out[ ]:

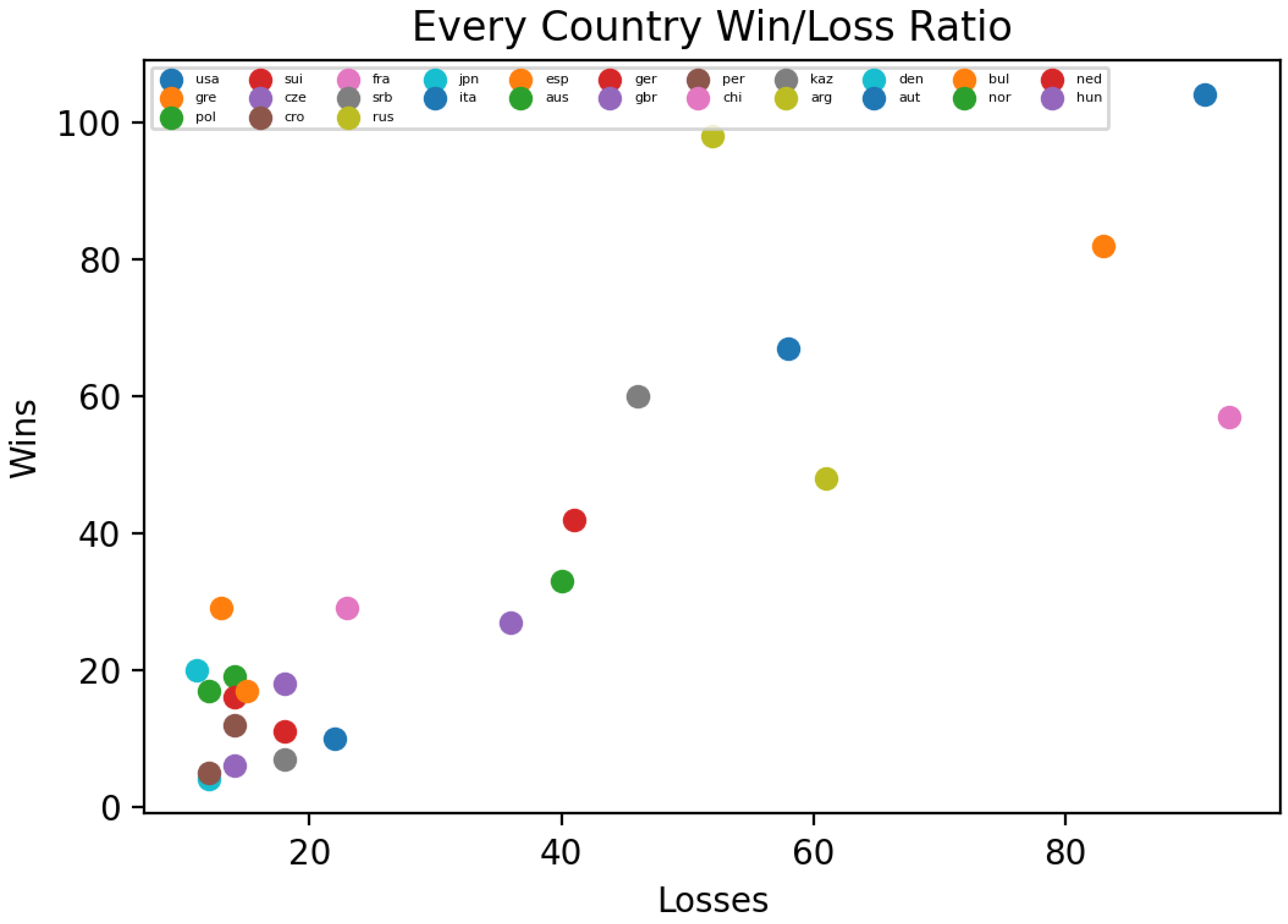
	winner_name	loser_name	winner_rank	loser_rank	winner_serve_rating	loser_serve_rating	winner_return_rating	loser_return_rating	rank_diff	serve_diff	return_diff	winner_country	loser_country
0	Frances Tiafoe	Lorenzo Musetti	2310.0	1470.0	282.3	265.3	137.8	152.4	840.0	17.0	-14.6	usa	ita
1	Taylor Fritz	Hubert Hurkacz	3100.0	3245.0	288.6	295.5	145.4	124.6	-145.0	-6.9	20.8	usa	pol
2	Stefanos Tsitsipas	Borna Coric	4235.0	1135.0	292.0	264.0	133.5	125.6	3100.0	28.0	7.9	gre	cro
3	Cameron Norrie	Taylor Fritz	1940.0	3100.0	270.6	288.6	147.0	145.4	-1160.0	-18.0	1.6	gbr	usa
4	Frances Tiafoe	Daniel Evans	2310.0	1131.0	282.3	260.8	137.8	136.7	1179.0	21.5	1.1	usa	gbr
...	...	...	...	...	...	...	...	...	...	...	...	...	...
855	Taylor Fritz	Aslan Karatsev	3100.0	1193.0	288.6	268.0	145.4	131.9	1907.0	20.6	13.5	usa	rus
856	Lorenzo Sonego	Aslan Karatsev	990.0	1193.0	277.2	268.0	132.2	131.9	-203.0	9.2	0.3	ita	rus
857	Andrey Rublev	Aslan Karatsev	4805.0	1193.0	279.4	268.0	146.7	131.9	3612.0	11.4	14.8	rus	rus
858	Frances Tiafoe	Aslan Karatsev	2310.0	1193.0	282.3	268.0	137.8	131.9	1117.0	14.3	5.9	usa	rus
859	Ben Shelton	Aslan Karatsev	2215.0	1193.0	282.8	268.0	127.3	131.9	1022.0	14.8	-4.6	usa	rus

860 rows × 13 columns

We then find the individual number of wins and losses for every country found in the above DataFrame, and create a visualization of the win/loss ratio for the countries

In [ ]:

```python
countries = []
countries_win = {}
countries_loss = {}

for i in range(len(overall_df)):
  if overall_df.winner_country[i] not in countries and overall_df.loser_country[i] not in countries:
    countries.append(overall_df.winner_country[i])

  if overall_df.winner_country[i] not in countries_win:
    countries_win[overall_df.winner_country[i]] = 1
  else:
    countries_win[overall_df.winner_country[i]] += 1

  if overall_df.loser_country[i] not in countries_loss:
    countries_loss[overall_df.loser_country[i]] = 1
  else:
    countries_loss[overall_df.loser_country[i]] += 1

country_points = []
for country in countries:
  country_points.append([countries_loss[country], countries_win[country]])

plt.figure(figsize=(6,4), dpi=200)
for i in range(len(countries)):
  plt.scatter(np.array(country_points)[i,0], np.array(country_points)[i,1])
plt.xlabel('Losses')
plt.ylabel('Wins')
plt.title('Every Country Win/Loss Ratio')
plt.legend(countries, fontsize='4', ncol=11)
plt.show()
```



Now we calculate the win and loss ratios for all the countries. For each country, we calculate their win and loss ratio by only looking at the outcomes of the games that included a player with their nationality. We made sure to account for the fact that for some games both of the players are Spanish players, so we double counted those as they are two attempts for a Spanish player trying to win a match, not just one. So for example, for the Spanish win and loss ratios, we only include games that include Spanish players. We then combined all the non-Spanish win and loss ratios to create an average win and loss ratio for non-Spanish players.

In [ ]:
```
countries_win_ratio = {}
countries_loss_ratio = {}

for countries in countries_win.keys():
  total = len(overall_df[(overall_df.winner_country == countries) | (overall_df.loser_country == countries)])
  total += len(overall_df[(overall_df.winner_country == countries) & (overall_df.loser_country == countries)])
  countries_win_ratio[countries] = countries_win[countries] / total
```

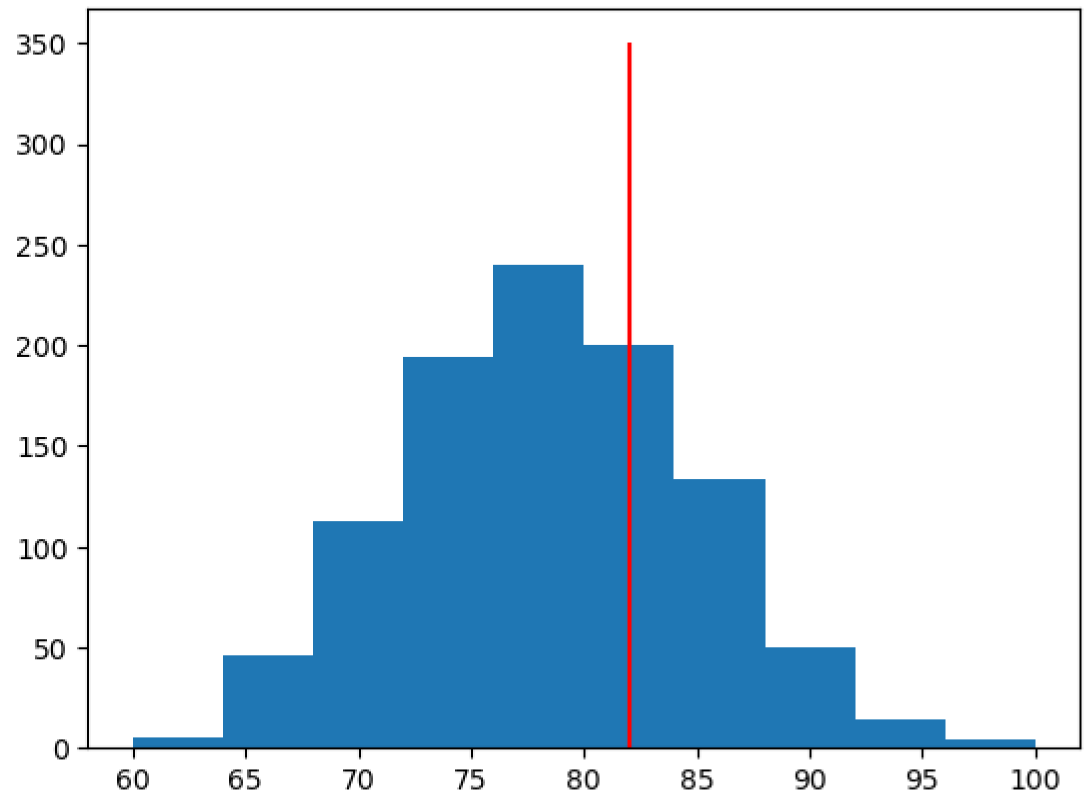
```
for countries in countries_loss.keys():
  total = len(overall_df[(overall_df.winner_country == countries) | (overall_df.loser_country == countries)])
  total += len(overall_df[(overall_df.winner_country == countries) & (overall_df.loser_country == countries)])
  countries_loss_ratio[countries] = countries_loss[countries] / total

avg_win_ratio = 0
for countries in countries_win_ratio.keys():
  if countries != 'esp': avg_win_ratio += countries_win_ratio[countries]
  else: esp_win_ratio = countries_win_ratio[countries]
avg_win_ratio = avg_win_ratio / (len(countries_win_ratio) - 1)

avg_loss_ratio = 0
for countries in countries_loss_ratio.keys():
  if countries != 'esp': avg_loss_ratio += countries_loss_ratio[countries]
  else: esp_loss_ratio = countries_loss_ratio[countries]
avg_loss_ratio = avg_loss_ratio / (len(countries_win_ratio) - 1)

print("avg_win_ratio: "+str(avg_win_ratio))
print("esp_win_ratio: "+str(esp_win_ratio))
print("avg_loss_ratio: "+str(avg_loss_ratio))
print("esp_loss_ratio: "+str(esp_loss_ratio))
```

```
avg_win_ratio: 0.4716058696429138
esp_win_ratio: 0.49696969696969695
avg_loss_ratio: 0.5283941303570863
esp_loss_ratio: 0.503030303030303
```

Now we create a binomial distribution of the Non-Spain country win ratio and plot the number of games a Spanish player has won to see whether the null hypothesis that Spanish players are not better than other players can be rejected. We calculate the p-values to do so.

In [ ]:
```
esp_num_games = len(overall_df[(overall_df.winner_country == 'esp') |
                               (overall_df.loser_country == 'esp')])
esp_num_games += len(overall_df[(overall_df.winner_country == 'esp') &
                               (overall_df.loser_country == 'esp')])

win_dist = np.random.binomial(esp_num_games, avg_win_ratio, size=1000)
plt.hist(win_dist, bins=10)
plt.vlines(x=countries_win['esp'], ymin=0, ymax=350, color='red')
plt.plot()

mean = esp_num_games*avg_win_ratio
std = np.sqrt(esp_num_games*avg_win_ratio*(1-avg_win_ratio))
z_score = (countries_win['esp'] - mean)/std
print("z-score: "+str(z_score))

p_value = scipy.stats.norm.cdf(abs(z_score))
print('p-value: ' + str(p_value))
```

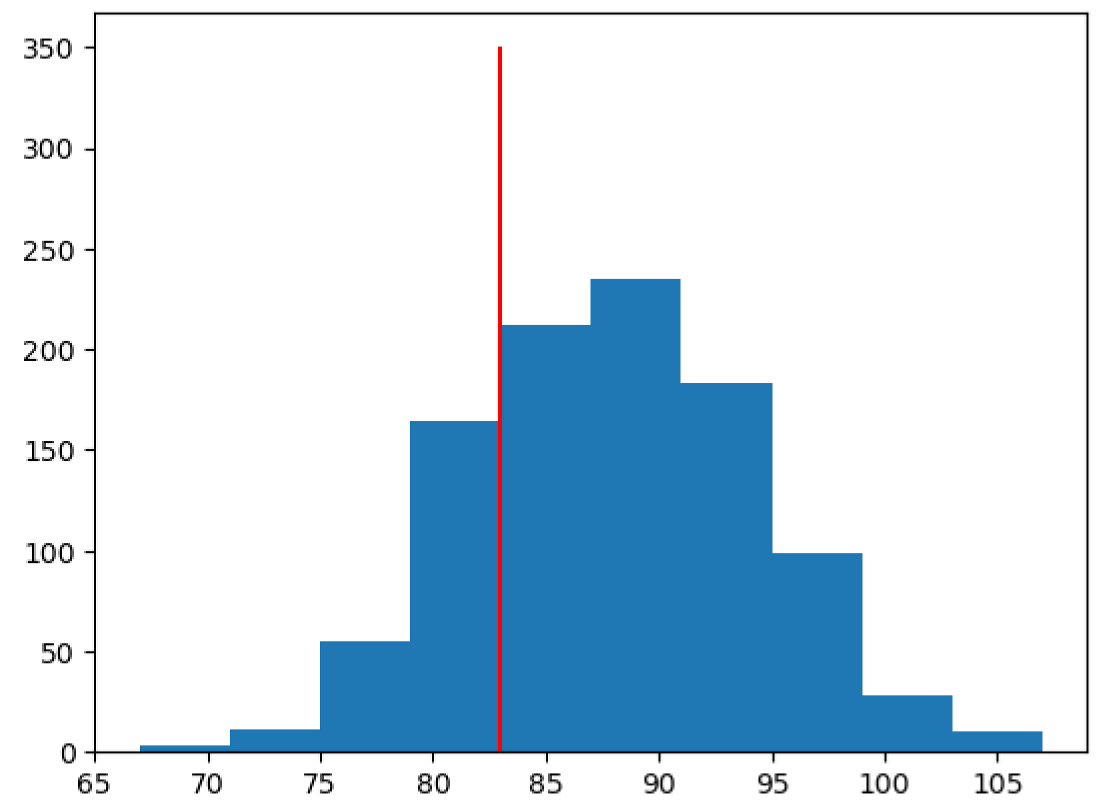
```
z-score: 0.6526617586200043
p-value: 0.74301281939148
```



We do the same for the losses

```
In [ ]: loss_dist = np.random.binomial(esp_num_games, avg_loss_ratio, size=1000)
        plt.hist(loss_dist, bins=10)
        plt.vlines(x=countries_loss['esp'], ymin=0, ymax=350, color='red')
        plt.plot()

        mean = esp_num_games*avg_loss_ratio
        std = np.sqrt(esp_num_games*avg_loss_ratio*(1-avg_loss_ratio))
        z_score = (countries_loss['esp'] - mean)/std
        print("z_score: "+str(z_score))

        p_value = scipy.stats.norm.cdf(abs(z_score))
        print('p-value: ' + str(p_value))
```

```
z_score: -0.6526617586200066
p-value: 0.7430128193914808
```



As one can see, the z_score shows that the number of wins or losses for a Spanish players is under than 1 standard deviation from the median, these correspond to p-values above 0.05, so we fail to reject the null hypothesis, Spanish players do not perform statistically better than non-Spanish players, at least in 2023.

Conclusions and Questions for Reviewers

Predicting Tennis Match Outcomes with Logistic Regression

We have shown that, by considering rankings and server/return ratings compiled by the ATP organization of men's tennis players, we can train a logistic regression to predict the outcome of a tennis match based on the differences between the rankings and ratings of the two players in the match. We have shown that this model performs better than randomly selecting the outcome of tennis matches at a level far in excess of 5%. These models have typical roc-auc scores between 0.6 and 0.7, which is closer to random guessing than being a perfect classifier.

We had good reason to expect that our model would perform better than guessing on a large sample, since tennis players that are generally thought of as "better" according to experts in the field should, more often than not, beat tennis players that are generally thought of as "worse". However, the fact that our model is closer to randomly guessing than being a perfect classifier is also to be expected, for reasons that we will go into in the following paragraphs.

Because of the high degree of multicollinearity between serve rating, return rating, and overall rating, we find that the predictive power of our model does not improve when taking into account the serve rating and the return rating in addition to the overall ranking. We further find that the serve rating is a more powerful classifer based on auc-roc score than the return rating.

What does our model say about tennis?

Tennis is a "one-on-one" sport (unlike many other sports, like American football, where as many as 22 players see the field simultaneously), and so it is easy to trick oneself into thinking that all players can be ranked on a spectrum from "good" to "bad", and that "better" players will always defeat "worse players." Any tennis aficionado could have already told you that better players do not always beat worse players, and that some players "match up" well against players who are typically ranked above them. In fact, if it were the case that better players always beat their less-skilled opponents, there would be almost no point in watching tennis, since you could just look up a player's rankings and decide the outcome of the match before it is even played. Fortunately, our model bears out these facts of tennis experience.

Ranking Players Doesn't Tell You Everything about the Matchup

Our analyses show that while there is substantial amount of multicollinearity between skill in different categories of tennis, the correlations between the ranking metrics that we considered does not exceed 0.5. In other words, we ought not to expect that a high-ranked tennis player is automatically good at both serving and returning compared to other top professional tennis players. Because there are so many factors that go into a tennis match, including circumstances about the match (to be explored in the section about playing surfaces below), and also circumstances about the strengths and weaknesses of different players, not to mention interactions between the strengths and weaknesses of different players, we really can't say with much confidence whether one player will beat another, even if we can say that on average one player is better than another. In short, rankings are descriptive of players over a long sample size of games, but they are not totally prescriptive about the outcome of matches.

How Good are Predictions Based on Tennis Ranking and Skill Ratings Alone

Our model earns a ROC-AUC score of about 0.7 on a big data set. We have shown, through cross-validation and significance testing, that this score is not a "fluke". Therefore, we know that tennis matches are not truly random, and rankings do contain information about the skill level of players. However, our model fails because of three major blind spots. (1) The skill level of players varies over shorter time scales than our metrics are sensitive to. If a player has a "bad day" or the stomach flu, we wouldn't know anything about it from our match. Even if metrics like ranking can track the long-term trajectory of a player, maybe as they mature and age, they won't be able to keep up with the day to day fluctuations in a player's skill. (2) Interactions between different effects are not accounted for in our model. If one player has a really strong serve, and the other has a really weak return, this will matter more than either of these effects in isolation. If we had more time, we likely would have been well-advised to explore interactions between different inputs to our model. (3) As we will discuss below, many things go into a tennis match, including where/when it is played, what surface it is played on, and the context of the match, that we do not model for by treating every tennis match as an independent event.

Additional Analyses

To further flesh out our understanding of what factors affect the outcome of tennis matches and at what level they influence the outcome, we performed two additional analyses that we preregistered in phase III. First, we attempted to determine whether the strength of the tennis program in the home country of a tennis players may affect their likelihood to win a match by considering the difference in win probability between Spain and countries other than Spain. Spain was selected because it has a world-famous tennis program. We further assessed how the surface of court where the tennis match affects the quality of serves by tennis players. We speculated that for certain players who specialize in a strong serve, the playing surface may significantly increase or decrease their probability of coming out of a match victorious.

We have also shown that Spanish players do not perform significantly better than non-Spanish players, through looking at the history of matches in 2023. The difference in the probability of Spanish players winning or losing vs Non-Spanish players winning or losing is not statistically significant.

Based on the result of the Welch’s t-test regarding hard vs clay courts, we found that there is a significant difference in mean serve rating (p value = 1.16e-11). One reason for this difference may be that tennis players practice more on hard courts compared to clay ones. Another difference is that clay courts cause the game to play “slower”, meaning much of the ball’s energy is lost when it hits the ground. This makes a serve easier to receive as opposed to a faster surface, like a hard court. Because of this, in a future model it would be worthwhile to include the playing surface as an additional input variable that would potentially increase the precision of the model.

Overall Conclusions

Tennis is a very subtle game. We have shown that many factors enter into the outcome of tennis matches at the statistically-significant level, and some surprising factors do not. While our model is unlikely to make us a fortune in Vegas from predicting the outcomes of matches, we feel that this analysis has shed some light on the game of tennis by drawing our attention to factors that do affect the outcome of matches and "mythbusting" some factors that appear not to affect the outcome of tennis matches.

We feel that we addressed most of the points of our research question.

Works Cited

Using the locale function: https://docs.python.org/3/library/locale.html

ATP Match results: https://github.com/JeffSackmann/tennis_atp/blob/master/atp_matches_2023.csv

ATP tennis stats: https://www.wheeloratings.com/tennis_atp_stats_last52.html

ATP men's singles rankings: https://www.atptour.com/rankings/singles

The Washington Post Tennis Betting: https://www.washingtonpost.com/world/interactive/2023/tennis-match-fixing-itf-grigor-sargsyan/#:~:text=Gambling%20on%20tennis%20has%20exploded,to%20roughly%20%2450%20billion%20annually.

Web Scraping Images: https://plainenglish.io/blog/web-scraping-images-with-python-and-selenium-792e452abd70

Appendix - Data Cleaning

Data Cleaning

Overall Rankings

We will begin by querying the ATP tour website. We will use beautiful soup to parse the html file, closely following the procedure from Discussion 3.

In [ ]:
```
atp_rankings_url = "https://www.atptour.com/rankings/singles"
atp_rankings_result = requests.get(atp_rankings_url)
if atp_rankings_result.status_code != 200:
  print("something went wrong:", atp_rankings_result.status_code, atp_rankings_result.reason)
with open("atp_ranking.html", "w") as writer:
  writer.write(atp_rankings_result.text)
with open("atp_ranking.html", "r") as reader:
  html_source = reader.read()
page = BeautifulSoup(html_source, "html.parser")
```

Next, we will locate the table where the player rankings are stored on the ATP website. I will iterate through each row in the table. The number of items in each row is unlabeled and highly irregular, so we will need to take some ugly shortcuts to determine which values correspond to the player's name, and which corresponds to their ranking as compiled by ATP. Essentially, for each item in each row, we try to convert it to a float - if we get an error, we know that the item in the row is a string. The only string value in each row is the player name, so we know that item corresponds to the player name. If we are successful in converting our item to a float, then the only values in the table which will exceed 610 will be the player rankings. So we check if the value is larger than 610, and if it is, we add it to the player rankings list. At the end, we verify that the length of these lists are the same length, confirming that we didn't miss any values.

This method is extremely messy because it is not invariant under changes in the format of the table - if extra values are added to each row that are strings, this step will fail. If there is a new row added to the table with floats that are in excess of 610, this step will fail. However, data cleaning is messy and we only need to get this data into a csv one time for it to work reliably, so we will proceed with this method for now. Perhaps in later phases we can go back and clean up this step.

In [ ]:
```
atp_rankings = page.find("table", {"id":"player-rank-detail-ajax"})
item_count = 0
items = []
rankings = []
players = []
for row in atp_rankings.tbody.findAll('tr'):
    for item in row.findAll('td'):
        items.append(item)
        item_count += 1
        try:
            items[item_count -1] = locale.atof(items[item_count-1].text)

            if(items[item_count-1] > 610):
                rankings.append(items[item_count -1])
        except:
            if len(items[item_count-1].text.strip()) > 0:
                players.append(items[item_count-1].text.strip())

atp_items = page.find_all("td", {"data-type": "text_align:left"})
rankings_df = pd.DataFrame({"player":players, "ranking":rankings})
```

```
rankings_df.to_csv("atp_rankings.csv")
rankings_df
```

```
100
```

Out[ ]:

	player	ranking
0	Novak Djokovic	9945.0
1	Carlos Alcaraz	8455.0
2	Daniil Medvedev	7200.0
3	Jannik Sinner	5490.0
4	Andrey Rublev	4805.0
...	...	...
95	Quentin Halys	640.0
96	Jason Kubler	631.0
97	Flavio Cobolli	619.0
98	Alex Michelsen	619.0
99	Maximilian Marterer	616.0

100 rows × 2 columns

Serve Leaders

This code is using the Firefox webdriver to get the HTML data from the atp website. We are using a webdriver because the data were are trying to scrape is dynamically loaded, and as such doesn't appear when BeautifulSoup is used.

In [ ]:
```
atp_serve_url = 'https://www.atptour.com/en/stats/leaderboard?boardType=serve&'
+'timeFrame=52Week&surface=all&versusRank=all&formerNo1=false'
driver = webdriver.Firefox(executable_path=str(str(os.getcwd())+'/geckodriver'))
driver.get(atp_serve_url)
driver.implicitly_wait(2)
```

We are finding every HTML instance of the tag name 'tr', which includes the rows of data for every player in the top 80 serve leaders. These are stored in r, an array of webdriver objects

In [ ]:
```
table = driver.find_element_by_id('leaderboardTable')
r = table.find_elements_by_tag_name('tr')
c = table.find_elements_by_tag_name('img')

print("r length: "+str(len(r)))
print("r index 0: "+str(r[0]))
```

```
r length: 78
r index 0: <selenium.webdriver.firefox.webelement.FirefoxWebElement (session="ec13a12d-122d-4359-bed0-22405e2cafc4", element="b0c3bf23-1ca6-4e34-b600-cfd84521958a")>
```

For every row in the atp serve leaders website, we are taking the corresponding webdriver object and taking out the name of the player and the serve rating and serve-related-data of the player, storing them in names and serve_df

In addition, we separately webscrape the nationalities of all the players using a separate .get_attribute() method. The nationality isn't explicity stated, rather it is stored inside the name for a .svg file link. .get_attribute('src') is able to scrape the .svg file name and the nationality can be extracted from that. The nationality is stated in a three-letter code, for example 'esp' corresponds to Spain

In [ ]:
```
names = []
serve_df = []
for row in r:
  name = row.text.split('\n')[1]
```

```
    row = np.array(row.text.split('\n')[2].split(' '))
    for i in range(len(row)):
      if '%' in row[i]: row[i] = row[i].replace('%', '')

    names.append(name)
    serve_df.append(row.astype(float))

print("names length: "+str(len(names)))
print("names first 5: "+str(names[0:5]))
print("serve_df length: "+str(len(serve_df)))
print("serve_df first 5: "+str(serve_df[0:5]))

countries = []
index = 0
for row in c:
  if index%2 == 1: countries.append(row.get_attribute('src')[-7:-4])
  index += 1
```

```
names length: 78
names first 5: ['Hubert Hurkacz', 'Stefanos Tsitsipas', 'Nicolas Jarry', 'Novak Djokovic', 'Christopher Eubanks']
serve_df length: 78
serve_df first 5: [array([295.5,  64. ,  79.6,  50.7,  88. ,  15.2,   2. ]), array([291.6,  63.9,  78.3,  55.1,  88.7,   7.8,   2.2]), array([290.6,  65.1,  76.8,  53.9,  87.8,
9.1,   2.1]), array([290.6,  63.9,  76.4,  57.9,  88.5,   6.9,   3. ]), array([289.7,  68.8,  73.9,  52.4,  85.6,  12.1,   3.1])]
```

Here we are processing the names and serve_df data and putting it all into one dataframe

In [ ]:
```
serve_df = pd.DataFrame(serve_df)
serve_df = serve_df.rename(columns={0:'serve_rating', 1:'1st_serve_%',
                                    2:'1st_serve_points_won_%',
                                    3:'2nd_serve_points_won_%',
                                    4:'service_games_won_%',
                                    5:'avg_aces_per_match',
                                    6:'avg_double_faults_per_match'})

serve_df = pd.concat([pd.Series(names), pd.Series(countries), serve_df], axis=1)
serve_df = serve_df.rename(columns={0:'name', 1:'country'})

serve_df.to_csv('serve_leader_df.csv')
serve_df
```

Out[ ]:

	name	country	serve_rating	1st_serve_%	1st_serve_points_won_%	2nd_serve_points_won_%	service_games_won_%	avg_aces_per_match	avg_double_faults_per_match
0	Hubert Hurkacz	pol	295.5	64.0	79.6	50.7	88.0	15.2	2.0
1	Novak Djokovic	srb	292.2	64.3	76.8	57.9	89.0	7.0	2.8
2	Stefanos Tsitsipas	gre	292.0	64.1	78.3	55.2	88.7	7.9	2.2
3	Nicolas Jarry	chi	290.6	65.1	76.8	53.9	87.8	9.1	2.1
4	Christopher Eubanks	usa	289.7	68.8	73.9	52.4	85.6	12.1	3.1
...	...	...	...	...	...	...	...	...	...
73	Richard Gasquet	fra	252.4	57.8	69.5	49.2	74.0	4.8	2.9
74	Sebastian Ofner	aut	248.6	57.0	70.9	46.8	71.4	6.0	3.5
75	Jaume Munar	esp	246.9	68.0	65.3	45.2	69.0	2.4	3.0
76	Diego Schwartzman	arg	244.0	67.6	62.4	47.5	67.9	1.2	2.6
77	Bernabe Zapata Miralles	esp	242.8	66.4	63.5	45.7	67.4	1.8	2.0

78 rows × 9 columns

In [ ]:
```
serve_df = pd.read_csv("serve_leader_df.csv")
```

Return Leaders

This first section is initializing the webdriver object using Selenium. Because the atp website is dynamically loaded, we can't use BeautifulSoup to access the html.

In [ ]:
```
link = "https://www.atptour.com/en/stats/leaderboard?boardType=return&timeFrame"
+"=52Week&surface=all&versusRank=all&formerNo1=false"
browser = webdriver.Firefox(executable_path=str(str(os.getcwd())+'/geckodriver'))
browser.get(link)
```

Next, we found the table containing all the information of return leaders in 2023. First we accessed the table element as a whole, then created a list where each element is a separate row.

In [ ]:
```
table = browser.find_element_by_id('leaderboardTable')
r = table.find_elements_by_tag_name('tr')
c = table.find_elements_by_tag_name('img')
```

This is a quick function to make the process of getting each player's information easier. This function parses the text of each row to return a dictionary containing all the pertinent information about a player, like their name and return rating.

In [ ]:
```
def GetInfo(id, row = r):
    t = row[id].text
    first_slash = t.index('\n')
    standing = t[:first_slash]
    name = t[first_slash+1:t.rfind('\n')]
    stats = t[t.rfind('\n')+1:].split(' ')

    return {'Name': name, 'Standing': standing, 'Return Rating': float(stats[0]),
            '% 1st Serve Return Points W'\
            : float(stats[1][:-1]),\
            '% 2nd Serve Return Points W':float(stats[2][:-1]),
            '% Return Games Won': float(stats[3][:-1]),\
            '% Break Points Converted': float(stats[4][:-1])}
```

Finally, this cell is where the dataframe containing all the information is created. We used the GetInfo() function to create the first row of the data frame, and then a for loop to fill it in with all the other player's information.

The nationalities of the return leaders is scraped in the same way as the serve leaders.

In [ ]:
```
y = GetInfo(0)
return_df = pd.DataFrame(y, index = [0])
for x in range(len(r))[1:]:
    record = pd.Series(GetInfo(x))
    return_df = pd.concat([return_df, record.to_frame().T], ignore_index = True)

countries = []
index = 0
for row in c:
  if index%2 == 1: countries.append(row.get_attribute('src')[-7:-4])
  index += 1

return_df = pd.concat([pd.Series(countries), return_df], axis=1)
return_df = return_df.rename(columns={0:'country'})

return_df.to_csv('serve_return.csv')
return_df
```

Out[ ]:

	country	Name	Standing	Return Rating	% 1st Serve Return Points W	% 2nd Serve Return Points W	% Return Games Won	% Break Points Converted
0	rus	Daniil Medvedev	1	163.7	32.9	54.0	30.4	46.4
1	esp	Carlos Alcaraz	2	162.6	35.8	53.9	32.5	40.4
2	ita	Jannik Sinner	3	160.0	33.1	54.7	29.8	42.4
3	srb	Novak Djokovic	4	158.4	33.0	54.1	28.6	42.7
4	aus	Alex de Minaur	5	157.7	33.6	51.9	29.0	43.2
...	...	...	...	...	...	...	...	...
73	can	Felix Auger-Aliassime	74	122.0	27.2	46.4	16.2	32.2
74	aus	Thanasi Kokkinakis	75	121.9	25.9	44.8	15.9	35.3
75	usa	Brandon Nakashima	76	119.6	27.4	46.3	15.2	30.7
76	aus	Alexei Popyrin	77	119.5	26.0	46.0	15.2	32.3
77	usa	Christopher Eubanks	78	114.6	22.3	45.5	12.2	34.6

78 rows × 8 columns

In [ ]:
```
return_df = pd.read_csv("serve_return.csv")
```

Joining the different tables

Now, we are loading in a dataframe (described in the data description section) compiled by a tennis statistician, with the results of every ATP men's tennis match that occured in the calendar year of 2023, plus a ton of other data.

In [ ]:
```
head_to_head_df = pd.read_csv('atp_matches_2023.csv')
head_to_head_df.head()
```

Out[ ]:

	tourney_id	tourney_name	surface	draw_size	tourney_level	tourney_date	match_num	winner_id	winner_seed	winner_entry	...	l_1stIn	l_1stWon	l_2ndWon	l_SvGms	l_bpSaved	l_bpFaced	winner_rank	win
0	2023-9900	United Cup	Hard	18	A	20230102	300	126203	3.0	NaN	...	62.0	47.0	15.0	12.0	9.0	9.0	9.0	
1	2023-9900	United Cup	Hard	18	A	20230102	299	126207	NaN	NaN	...	12.0	8.0	3.0	4.0	1.0	3.0	19.0	
2	2023-9900	United Cup	Hard	18	A	20230102	296	126203	3.0	NaN	...	62.0	51.0	7.0	12.0	2.0	2.0	9.0	
3	2023-9900	United Cup	Hard	18	A	20230102	295	126207	NaN	NaN	...	41.0	26.0	12.0	9.0	6.0	9.0	19.0	
4	2023-9900	United Cup	Hard	18	A	20230102	292	126774	1.0	NaN	...	58.0	48.0	18.0	16.0	1.0	2.0	4.0	

5 rows × 49 columns

From the above dataframe, make a dataframe that contains the winner name of each match, the loser name of each match, and the winner's ATP overall ranking (scraped above) by INNER JOINING with the overall ATP rank df where the player name from the ranking dataframe matches the winner name. The inner join is necessary because we don't want to consider matches between unranked players.

In [ ]:
```
winners_df = duckdb.sql("SELECT winner_name, loser_name, \
    rankings_df.ranking AS winner_rank FROM head_to_head_df  \
    INNER JOIN rankings_df ON head_to_head_df.winner_name = rankings_df.player").df()
winners_df.head()
```

Out[ ]:

	winner_name	loser_name	winner_rank
0	Taylor Fritz	Matteo Berrettini	800.0
1	Frances Tiafoe	Lorenzo Musetti	2570.0
2	Taylor Fritz	Hubert Hurkacz	800.0
3	Frances Tiafoe	Kacper Zuk	2570.0
4	Stefanos Tsitsipas	Matteo Berrettini	5205.0

Next, we need to add the loser ranking to the above dataframe, inner joining on where the name of the player in the ranking df matches the loser name from the match results dataframe.

In [ ]:

```
winners_and_losers_df = duckdb.sql("SELECT winner_name, loser_name, winners_df.winner_rank, rankings_df.ranking\
                                    AS loser_rank FROM winners_df  INNER JOIN rankings_df ON\
                                       winners_df.loser_name = rankings_df.player").df()
winners_and_losers_df.head()
```

Out[ ]:

	winner_name	loser_name	winner_rank	loser_rank
0	Taylor Fritz	Matteo Berrettini	800.0	691.0
1	Frances Tiafoe	Lorenzo Musetti	2570.0	1525.0
2	Taylor Fritz	Hubert Hurkacz	800.0	3500.0
3	Stefanos Tsitsipas	Matteo Berrettini	5205.0	691.0
4	Stefanos Tsitsipas	Borna Coric	5205.0	1193.0

Next, I will load in the csv that we generated in an earlier step of the serve leaders (player names and serve statistics). I will then immediately discard all columns that are not player name and serve rating.

In [ ]:

```
serve_rating = pd.read_csv("serve_leader_df.csv")
serve_rating = duckdb.sql("SELECT name, serve_rating FROM serve_rating").df()
serve_rating.head()
```

Out[ ]:

	name	serve_rating
0	Hubert Hurkacz	295.5
1	Novak Djokovic	292.2
2	Stefanos Tsitsipas	292.0
3	Nicolas Jarry	290.6
4	Christopher Eubanks	289.7

Next, we will add the winner's serve rating to the dataframe by inner joining the serve rating dataframe with the existing dataframe where the player name in the serve rating equals the player name in the existing dataframe.

In [ ]:

```
overall_df = duckdb.sql("SELECT winner_name, loser_name, winner_rank, loser_rank, serve_rating AS\
                         winner_serve_rating FROM winners_and_losers_df INNER JOIN serve_rating ON \
                        winners_and_losers_df.winner_name = serve_rating.name").df()
overall_df.head()
```

Out[ ]:

	winner_name	loser_name	winner_rank	loser_rank	winner_serve_rating
0	Taylor Fritz	Matteo Berrettini	800.0	691.0	288.6
1	Frances Tiafoe	Lorenzo Musetti	2570.0	1525.0	282.3
2	Taylor Fritz	Hubert Hurkacz	800.0	3500.0	288.6
3	Stefanos Tsitsipas	Matteo Berrettini	5205.0	691.0	292.0
4	Stefanos Tsitsipas	Borna Coric	5205.0	1193.0	292.0

We will do the same process for the serve rating of the loser of the match. We will continue inner joining to ensure that we do not consider matches between players for whom we don't have data.

In [ ]:
```
overall_df = duckdb.sql("SELECT winner_name, loser_name, winner_rank, loser_rank, winner_serve_rating, \
serve_rating AS loser_serve_rating FROM overall_df INNER JOIN serve_rating ON overall_df.loser_name = \
serve_rating.name").df()
overall_df.head()
```

Out[ ]:

	winner_name	loser_name	winner_rank	loser_rank	winner_serve_rating	loser_serve_rating
0	Frances Tiafoe	Lorenzo Musetti	2570.0	1525.0	282.3	265.3
1	Taylor Fritz	Hubert Hurkacz	800.0	3500.0	288.6	295.5
2	Stefanos Tsitsipas	Borna Coric	5205.0	1193.0	292.0	264.0
3	Cameron Norrie	Taylor Fritz	2310.0	800.0	270.6	288.6
4	Frances Tiafoe	Daniel Evans	2570.0	1178.0	282.3	260.8

Next, we will execute essentially the same process using the return ratings csv generated above. First, we will load in the csv and discard all of the columns that do not contain relevant data to our analysis.

In [ ]:
```
return_rating = pd.read_csv("serve_return.csv")
return_rating.rename(columns = {'Return Rating':'return_rating'}, inplace = True)

return_rating = duckdb.sql("SELECT Name, return_rating FROM return_rating").df()
```

We will inner join the return ratings df where the match winner name equals the name in the return rating, exactly as we have done several times above.

In [ ]:
```
overall_df = duckdb.sql("SELECT winner_name, loser_name, winner_rank, loser_rank, winner_serve_rating, \
loser_serve_rating, return_rating AS winner_return_rating FROM overall_df INNER JOIN return_rating ON \
overall_df.winner_name = return_rating.Name").df()
overall_df.head()
```

Out[ ]:

	winner_name	loser_name	winner_rank	loser_rank	winner_serve_rating	loser_serve_rating	winner_return_rating
0	Frances Tiafoe	Lorenzo Musetti	2570.0	1525.0	282.3	265.3	137.8
1	Taylor Fritz	Hubert Hurkacz	800.0	3500.0	288.6	295.5	145.4
2	Stefanos Tsitsipas	Borna Coric	5205.0	1193.0	292.0	264.0	133.5
3	Cameron Norrie	Taylor Fritz	2310.0	800.0	270.6	288.6	147.0
4	Frances Tiafoe	Daniel Evans	2570.0	1178.0	282.3	260.8	137.8

Repeating the same step as above, except now for the loser's return rating:

In [ ]:
```
overall_df = duckdb.sql("SELECT winner_name, loser_name, winner_rank, loser_rank, winner_serve_rating, \
loser_serve_rating, winner_return_rating, return_rating AS loser_return_rating FROM overall_df INNER JOIN \
return_rating ON overall_df.loser_name = return_rating.Name").df()
overall_df.head()
```

Out[ ]:

	winner_name	loser_name	winner_rank	loser_rank	winner_serve_rating	loser_serve_rating	winner_return_rating	loser_return_rating
0	Frances Tiafoe	Lorenzo Musetti	2570.0	1525.0	282.3	265.3	137.8	152.4
1	Taylor Fritz	Hubert Hurkacz	800.0	3500.0	288.6	295.5	145.4	124.6
2	Stefanos Tsitsipas	Borna Coric	5205.0	1193.0	292.0	264.0	133.5	125.6
3	Cameron Norrie	Taylor Fritz	2310.0	800.0	270.6	288.6	147.0	145.4
4	Frances Tiafoe	Daniel Evans	2570.0	1178.0	282.3	260.8	137.8	136.7

Finally, we will check the shape of our dataframe to assess how many matches of data we have to analyze.

We have overall ranking data, return ratings, and serve ratings for both players in 862 ATP tennis matches in the calendar year of 2023. We contend that this is more than enough data to train a model on, at least as a preliminary step.