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
```

Research Question

Can we use two ATP men's tennis players overall rankings, server 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)

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. 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.

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
```

Out[ ]:

	player	ranking
0	Novak Djokovic	11045.0
1	Carlos Alcaraz	8805.0
2	Daniil Medvedev	7355.0
3	Jannik Sinner	5000.0
4	Andrey Rublev	4765.0
...	...	...
95	Hugo Gaston	643.0
96	Taro Daniel	634.0
97	Facundo Diaz Acosta	633.0
98	Juan Manuel Cerundolo	618.0
99	Liam Broady	617.0

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

```
r length: 80
r index 0: <selenium.webdriver.firefox.webelement.FirefoxWebElement (session="a91b1c03-e938-45cd-a456-998033e02186", element="1ef195fd-8eee-4495-9cad-3fad3c5d9b7d")>
```

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 [ ]:
```
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]))
```

```
names length: 80
names first 5: ['Hubert Hurkacz', 'Novak Djokovic', 'Stefanos Tsitsipas', 'Christopher Eubanks', 'Nicolas Jarry']
serve_df length: 80
serve_df first 5: [array([297. ,  63.9,  79.8,  51.3,  88.1,  16. ,   2.1]), array([292.7,  64.5,  77.3,  58.2,  88.7,   6.6,   2.6]), array([292. ,  64.8,  78.4,  54.9,  88.4,
7.6,   2.1]), array([289.9,  68.8,  73.8,  52.9,  85.4,  12.1,   3.1]), array([289.1,  65.1,  76.6,  53.5,  87. ,   9. ,   2.1])]
```

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

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

Out[ ]:

	name	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	297.0	63.9	79.8	51.3	88.1	16.0	2.1
1	Novak Djokovic	292.7	64.5	77.3	58.2	88.7	6.6	2.6
2	Stefanos Tsitsipas	292.0	64.8	78.4	54.9	88.4	7.6	2.1
3	Christopher Eubanks	289.9	68.8	73.8	52.9	85.4	12.1	3.1
4	Nicolas Jarry	289.1	65.1	76.6	53.5	87.0	9.0	2.1
...	...	...	...	...	...	...	...	...
75	Botic van de Zandschulp	251.3	62.3	68.5	46.3	73.1	5.3	4.2
76	Jaume Munar	248.7	68.3	65.6	45.7	69.6	2.5	3.0
77	Sebastian Ofner	248.6	56.9	71.0	47.1	71.8	5.6	3.8
78	Diego Schwartzman	243.7	67.7	62.3	47.6	67.7	1.1	2.7
79	Bernabe Zapata Miralles	242.2	66.2	63.6	45.3	67.3	1.8	2.0

80 rows × 8 columns

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')
```

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.

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)

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

Out[ ]:

	Name	Standing	Return Rating	% 1st Serve Return Points W	% 2nd Serve Return Points W	% Return Games Won	% Break Points Converted
0	Daniil Medvedev	1	166.1	33.6	54.2	31.3	47.0
1	Carlos Alcaraz	2	162.0	35.5	53.8	32.2	40.5
2	Jannik Sinner	3	160.8	33.1	54.9	29.9	42.9
3	Novak Djokovic	4	159.1	33.4	54.8	29.2	41.7
4	Alex de Minaur	5	157.7	33.5	52.1	29.3	42.8
...	...	...	...	...	...	...	...
75	Quentin Halys	76	121.0	25.1	44.5	14.8	36.6
76	Brandon Nakashima	77	118.1	27.1	45.7	14.8	30.5
77	Alexei Popyrin	78	117.1	25.7	45.6	14.4	31.4
78	Christopher Eubanks	79	116.6	22.5	45.6	13.0	35.5
79	Maxime Cressy	80	102.0	22.0	38.5	9.0	32.5

80 rows × 7 columns

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	winn
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	3410.0
1	Frances Tiafoe	Lorenzo Musetti	2650.0
2	Taylor Fritz	Hubert Hurkacz	3410.0
3	Frances Tiafoe	Kacper Zuk	2650.0
4	Stefanos Tsitsipas	Matteo Berrettini	4360.0

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	3410.0	850.0
1	Frances Tiafoe	Lorenzo Musetti	2650.0	1865.0
2	Taylor Fritz	Hubert Hurkacz	3410.0	2900.0
3	Stefanos Tsitsipas	Matteo Berrettini	4360.0	850.0
4	Stefanos Tsitsipas	Borna Coric	4360.0	1328.0

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	297.0
1	Novak Djokovic	292.7
2	Stefanos Tsitsipas	292.0
3	Christopher Eubanks	289.9
4	Nicolas Jarry	289.1

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	3410.0	850.0	286.8
1	Frances Tiafoe	Lorenzo Musetti	2650.0	1865.0	282.1
2	Taylor Fritz	Hubert Hurkacz	3410.0	2900.0	286.8
3	Stefanos Tsitsipas	Matteo Berrettini	4360.0	850.0	292.0
4	Stefanos Tsitsipas	Borna Coric	4360.0	1328.0	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	2650.0	1865.0	282.1	268.2
1	Taylor Fritz	Hubert Hurkacz	3410.0	2900.0	286.8	297.0
2	Stefanos Tsitsipas	Borna Coric	4360.0	1328.0	292.0	267.5
3	Cameron Norrie	Taylor Fritz	2100.0	3410.0	270.9	286.8
4	Frances Tiafoe	Daniel Evans	2650.0	1325.0	282.1	263.8

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	2650.0	1865.0	282.1	268.2	138.3
1	Taylor Fritz	Hubert Hurkacz	3410.0	2900.0	286.8	297.0	145.8
2	Stefanos Tsitsipas	Borna Coric	4360.0	1328.0	292.0	267.5	135.2
3	Cameron Norrie	Taylor Fritz	2100.0	3410.0	270.9	286.8	148.0
4	Frances Tiafoe	Daniel Evans	2650.0	1325.0	282.1	263.8	138.3

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	2650.0	1865.0	282.1	268.2	138.3	153.5
1	Taylor Fritz	Hubert Hurkacz	3410.0	2900.0	286.8	297.0	145.8	121.1
2	Stefanos Tsitsipas	Borna Coric	4360.0	1328.0	292.0	267.5	135.2	124.6
3	Cameron Norrie	Taylor Fritz	2100.0	3410.0	270.9	286.8	148.0	145.8
4	Frances Tiafoe	Daniel Evans	2650.0	1325.0	282.1	263.8	138.3	137.8

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

In [ ]:
```
print(overall_df.shape)
```

```
(862, 8)
```

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.

Data Description

In [ ]:
```
overall_df.to_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.

Exploratory 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.

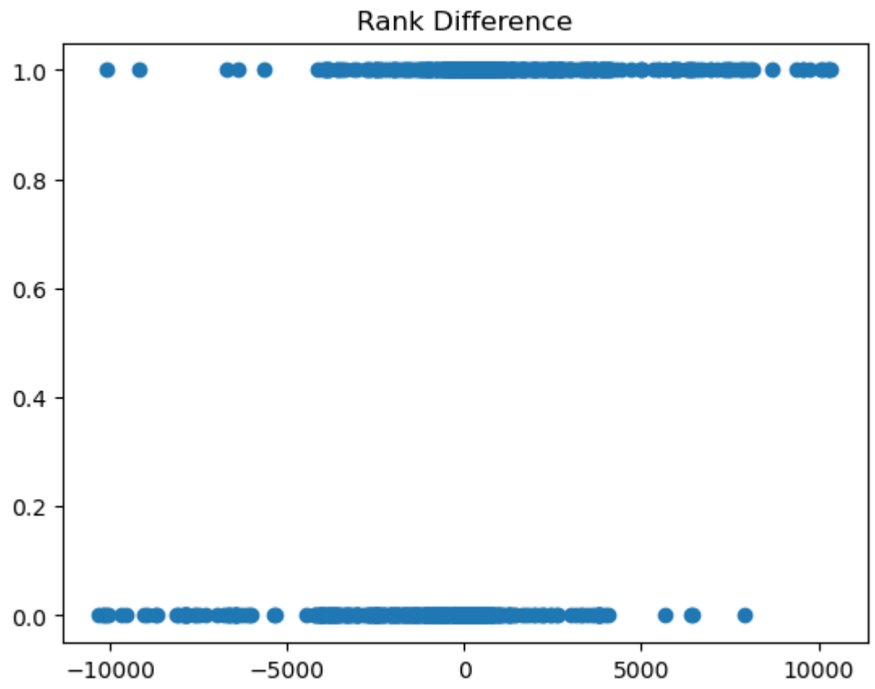
In [ ]:
```
overall_df = pd.read_csv('overall_df.csv')
overall_df = overall_df.drop(['Unnamed: 0'], axis=1)
overall_df['rank_diff'] = overall_df.winner_rank - overall_df.loser_rank
overall_df['serve_diff'] = overall_df.winner_serve_rating - overall_df.loser_serve_rating
overall_df['return_diff'] = overall_df.winner_return_rating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

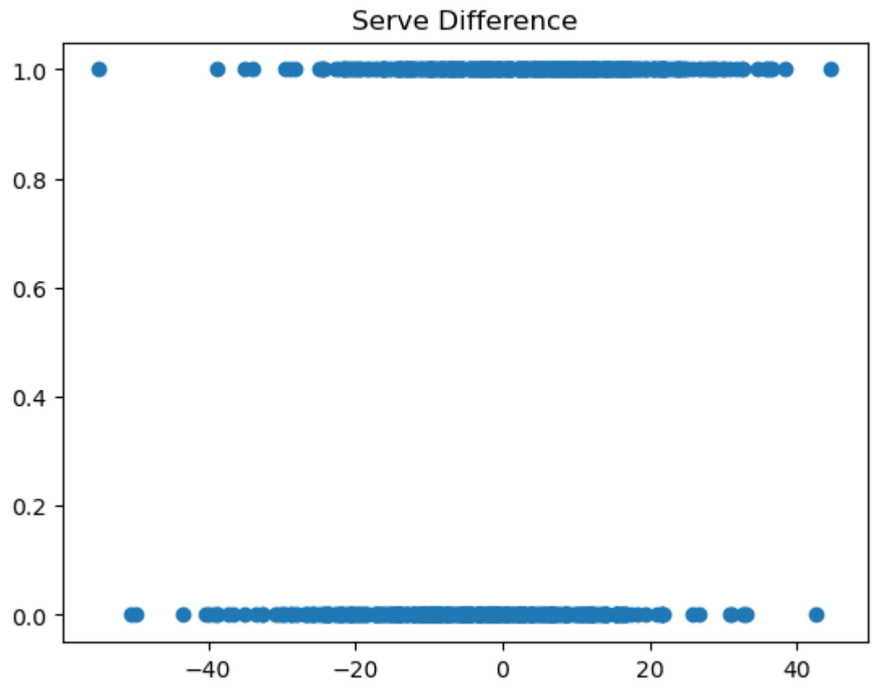
Next, we will make an np array of our inputs, X, formatted in "list of list" form for the purpose of doing a regression on the array. We will also make an np array for our outputs, which are all one (meaning that the ratings are calculated as Player A rating - Player B rating, where Player A is always the winner in our table). We then plot the outcome of the 800+ matches in our dataset vs the differential ratings and rankingsof players in those matches..

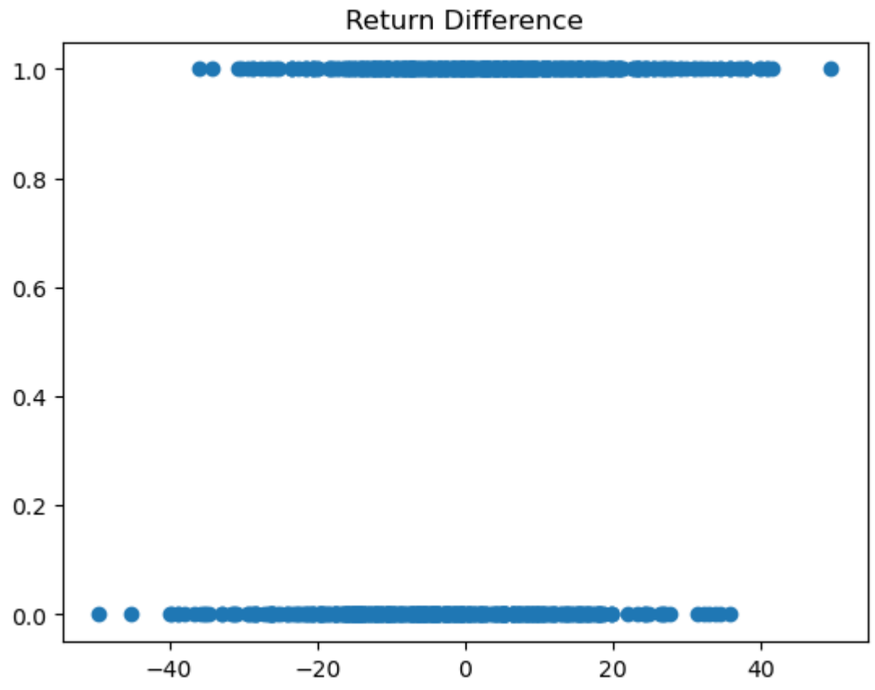
In [ ]:
```
X = np.array(overall_df[['rank_diff', 'serve_diff', 'return_diff']])
y = np.ones((len(X),1))


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

```
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 0x17a8a52d0>







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.

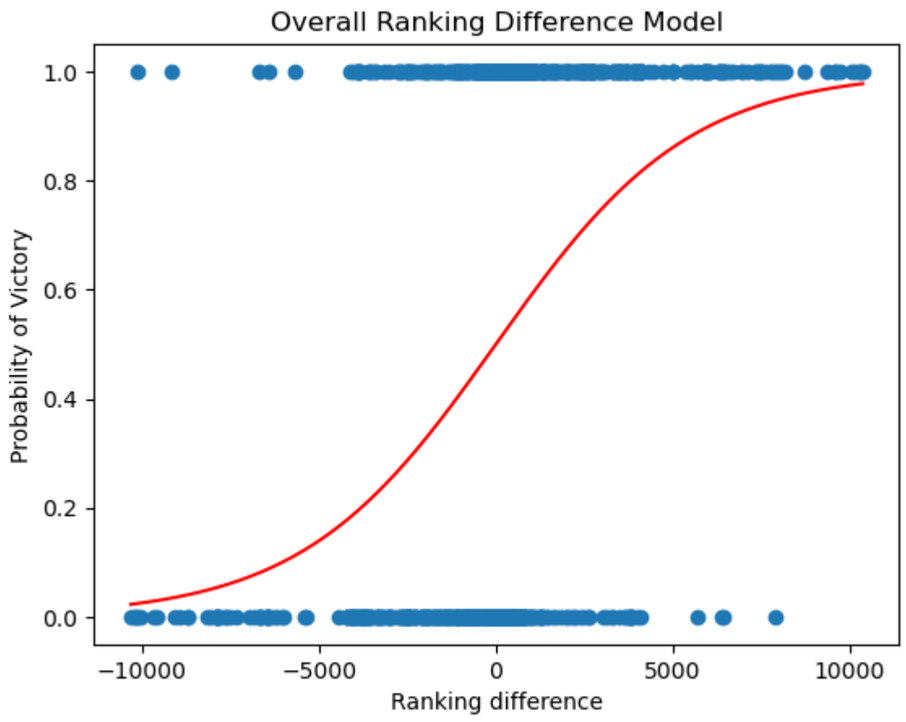
Predicting Winners from Overall Rankings Alone

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

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

```
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

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



In the below cell, 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 precision of our model's prediction.

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 = []
recalls = []
f1s = []
rocaucs = []
for threshold in thresholds:
    yhat = []
    for x in rank_X.flatten():
        if rank_model.predict_proba([[x]])[0, 1] >= threshold:
            yhat.append(1)
        if rank_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))
```

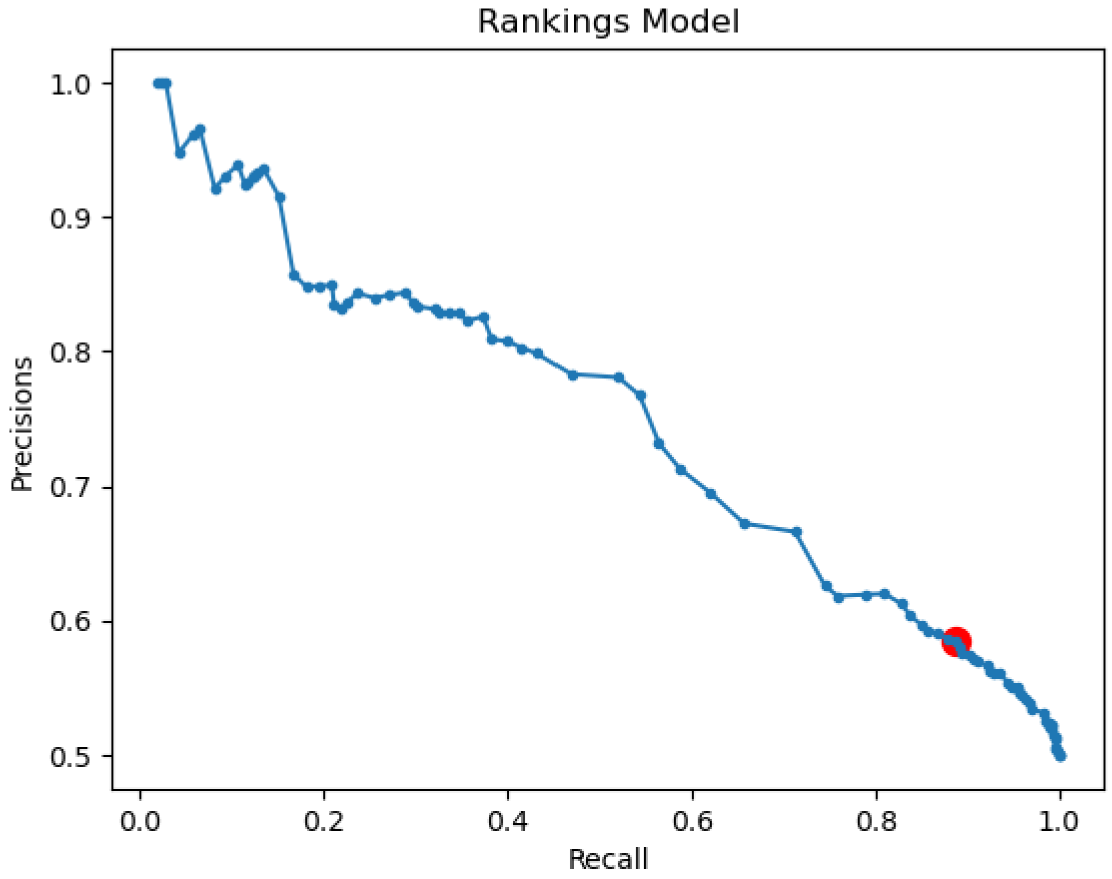
```
    rocaucs.append(roc_auc_score(y, yhat))
precisions = np.array(precisions)
recalls = np.array(recalls)
```

```
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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 above model and 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
[0.38383838]
```

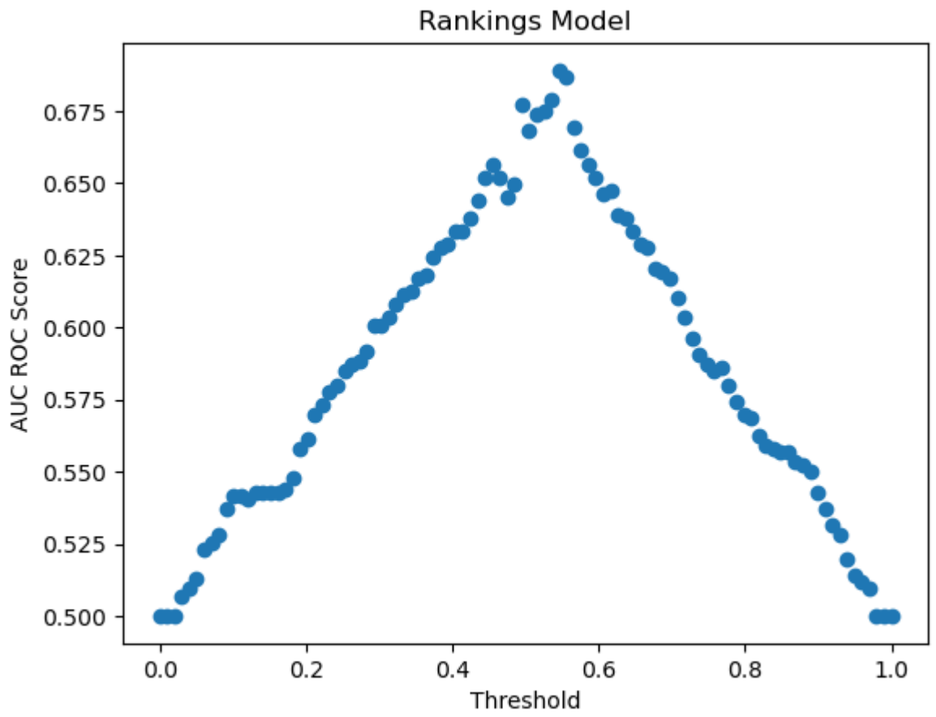


From the above plot, it is clear that the optimization process is essentially optimizing recall at the expense of making precision very close to 0.5, which effectively means that this statistic is prioritizing minimizing false negatives at the expense of tolerating more false positives. At this point 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 at 0.38, a little less than half of the players we predict to win the match would actually lose the match.

In the below plot, we are plotting the rocauc score to determine the optimal threshold and the 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 auc roc score")
print(rocaucs[max_index[0][0]])
```

```
threshold
[0.54545455]
max auc roc score
0.6890951276102089
```



From above, we found that the optimal threshold occurs at .545, with a maximum auc roc score of 0.689. 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 is actually predictive, but is closer to randomness 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.

In the below cells, I will repeat the same steps but making the predictions from server ratings and return ratings.

Predicting Winners from Return Ratings

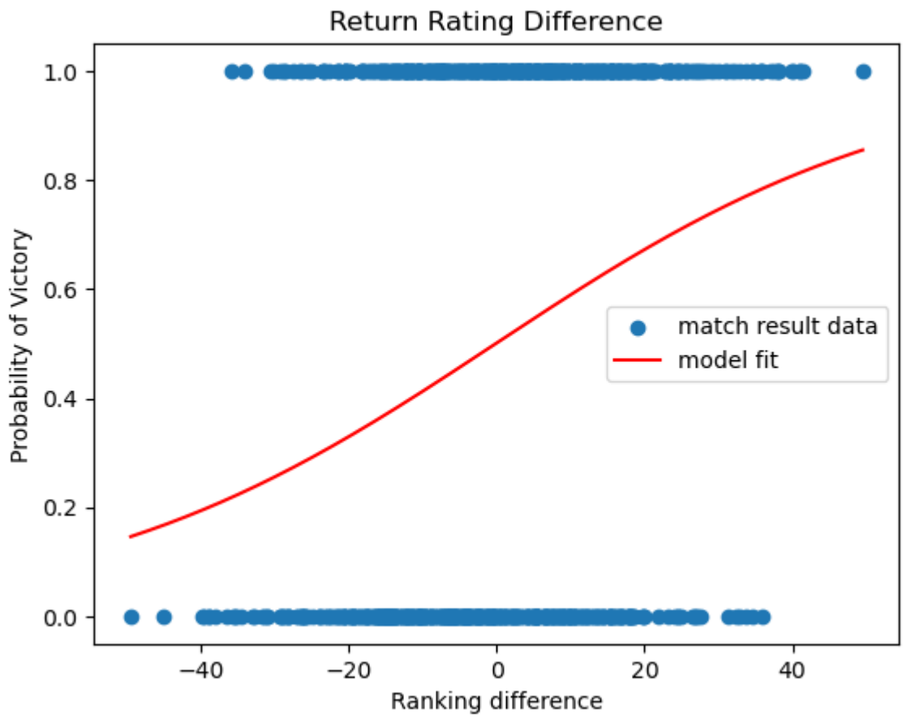
Below, I have plotted a linear regression to the 800+ matches in our dataset based on the return rating differential, exactly as we did in the corresponding above cell from the overall rankings.

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")
```

```
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

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.

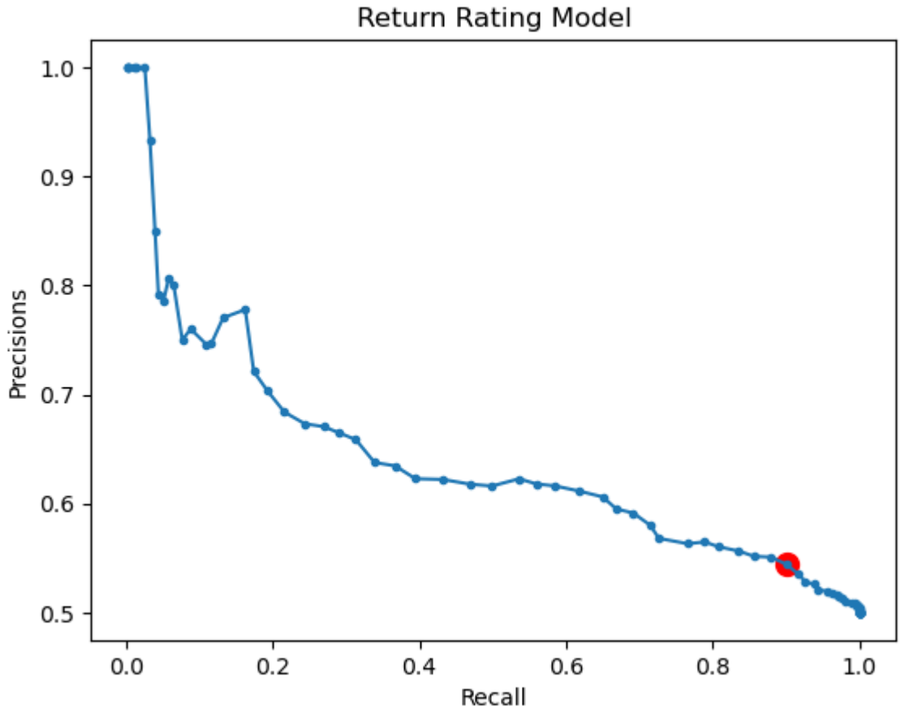
In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)
precisions = []
recalls = []
f1s = []
rocaucs = []
for threshold in thresholds:
    yhat = []
    for x in return_X.flatten():
        if return_model.predict_proba([[x]])[0, 1] >= threshold:
            yhat.append(1)
        if return_model.predict_proba([[x]])[0, 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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
[0.37373737]
```

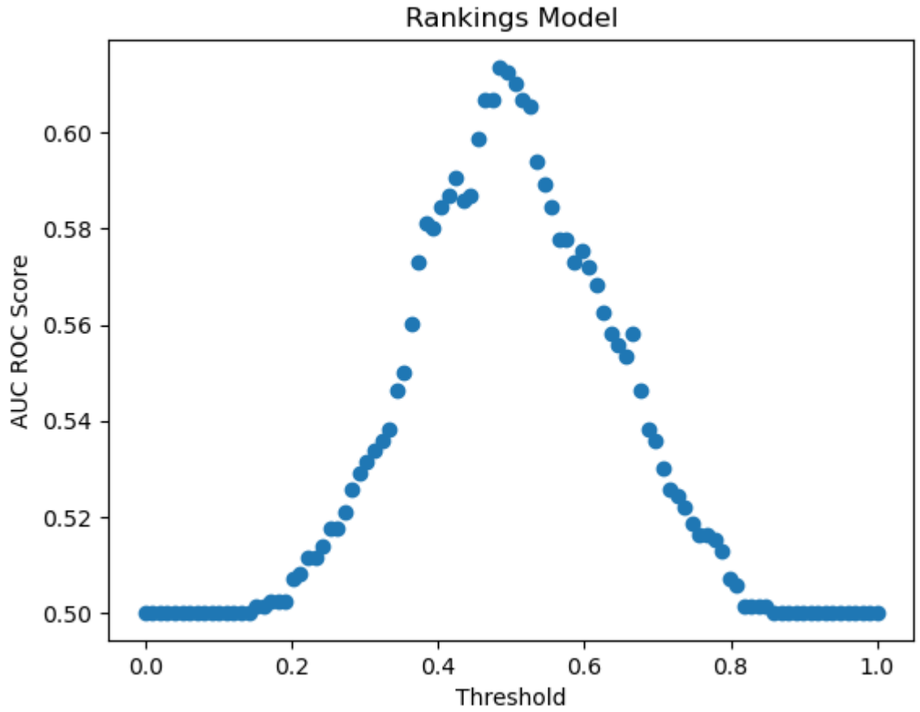


Based on the above plot, we can conclude that a very similar trend is occuring as above - prioritizing recall, and therefore minimizing false negatives at the expense of tolerating more false positives. If we were to set the threshold based on minimization of the F1 score, a little less than half of 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
[0.48484848]
max roc auc score
0.6136890951276102
```



Since the max rocauc score is lower for the model based on return rating rather than 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.

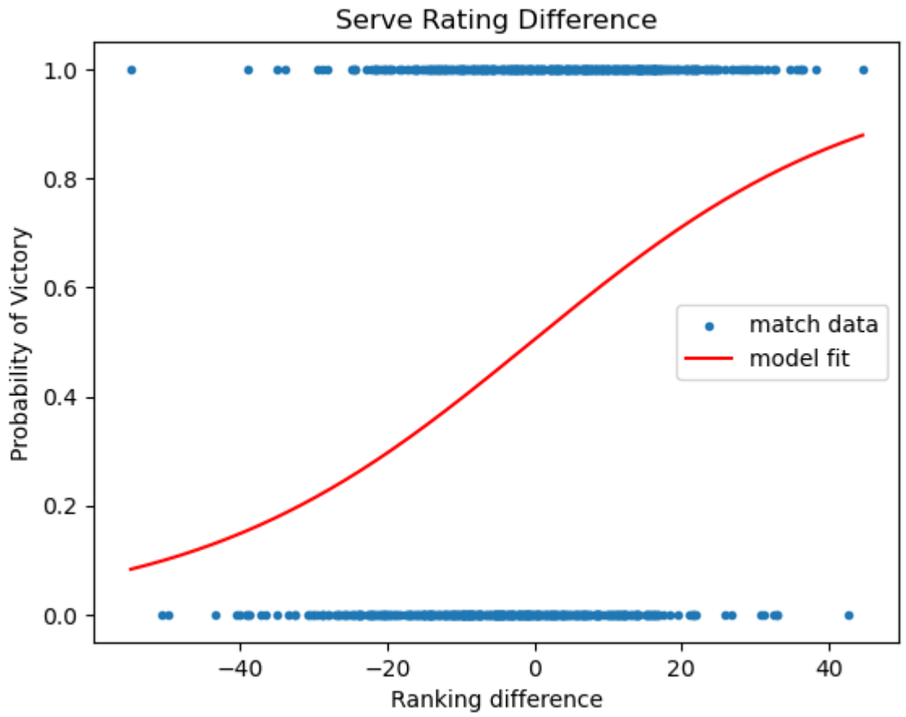
Predicting Winners from Serve Ratings

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")
```

```
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

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)
```

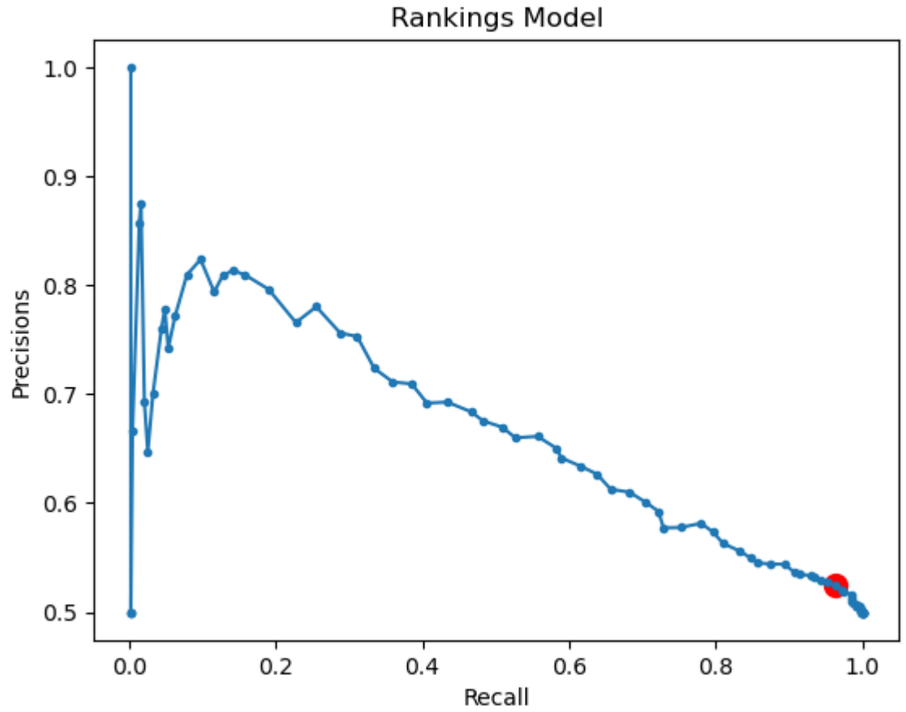
```
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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
[0.28282828]
```

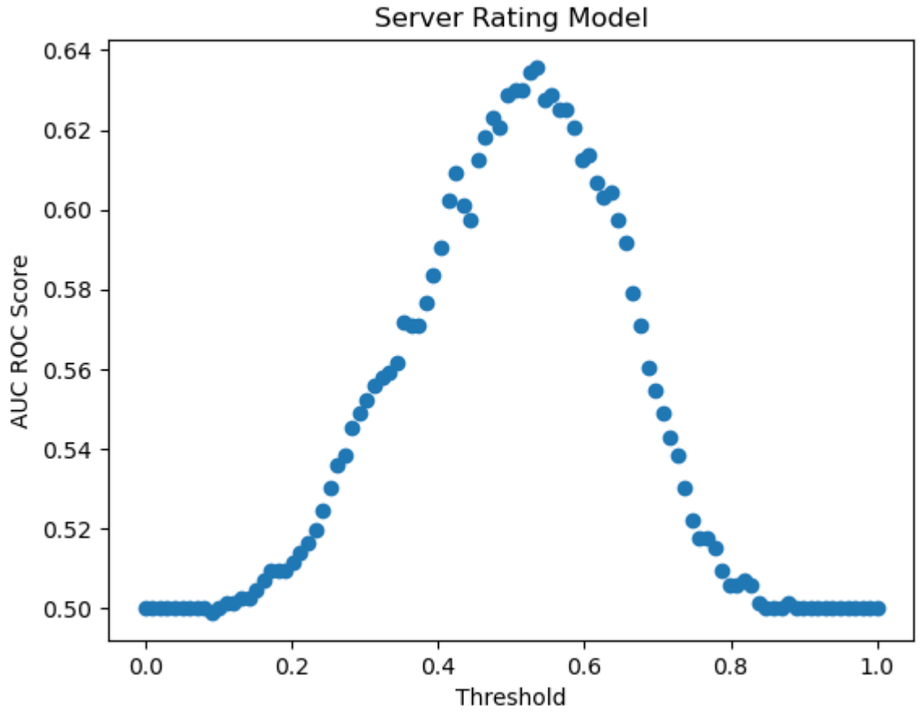


We notice the same trend of a low optimal threshold and the same "hardly better than random" precision with the benefit of a very high recall. This is probably not where we want to optimally set the threshold as false negatives are about as important as false positives in our model.

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
[0.53535354]
max roc auc score
0.6357308584686775
```



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^(0.000364)

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^(0.0441)

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^(0.0357)

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^(0.000187). 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^(0.0335). 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^(0.0259)

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.

In [ ]:
```
model = LogisticRegression().fit(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: 0.00036420371719544817
intercept: -3.9374771857459235e-09

 for serve model
coefficient: 0.044104364120348065
intercept: 0.016442393648689158

 for rank model
coefficient: 0.03574748623621544
intercept: 0.004672702284234553

 For multivariate model
rank, serve, and return coefficients: 0.00018692807701527255, 0.03347634794502018, 0.02585255228487367
intercept: 1.0251277451582133e-05
```

```
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

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 models except this time feeding the model 3 inputs of 1 input.

In [ ]:
```
thresholds = np.linspace(0, 1.0, 100)
precisions = []
recalls = []
f1s = []
rocaucs = []
for threshold in thresholds:
    yhat = []
    for x in X:
        if model.predict_proba([x])[0, 1] >= threshold:
            yhat.append(1)
        if 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)
```

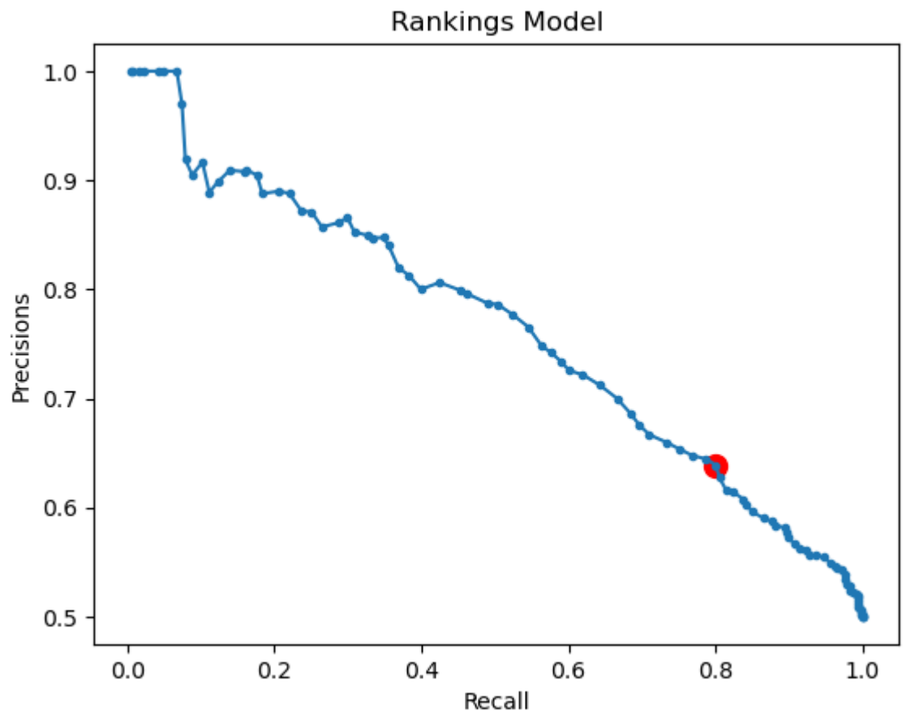
```
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set 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/haonan/anaconda3/envs/info2950/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

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("Rankings Model")
print('threshold')
print(thresholds[max_index])
print("Max recall")
print(recalls[max_index[0][0]])
print("Max precision")
print(precisions[max_index[0][0]])
```

```
threshold
[0.42424242]
Max recall
0.8004640371229699
Max precision
0.6388888888888888
```



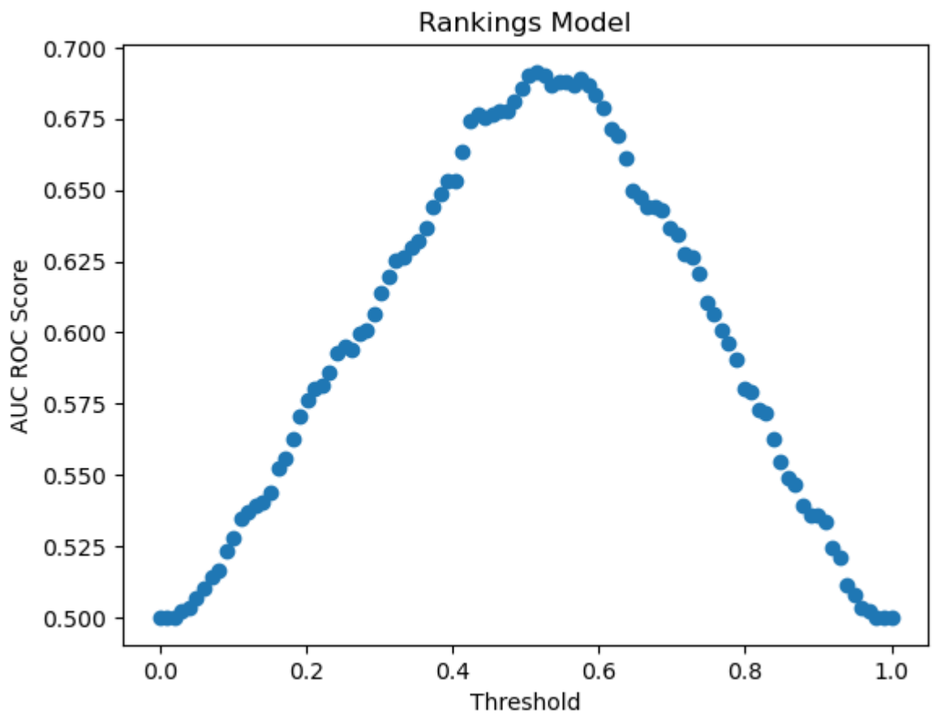
Taking into account the three ratings, we are able to achieve a recall of 0.80 and a precision of 0.64 simultaneously, meaning that of the players we predict to win their match, around 64% of them actually did win their match, and of the players who actually did win their match, we were able to identify 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 threshold plot using the same methodologyas in the 1 dimensional cases.

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
[0.51515152]
max roc auc score
0.691415313225058
```



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.

Conclusions and Questions for Reviewers

Conclusions: Because of the high degree of collinearity between serve rating, return rating, and overall rating, we find that the predictive power of our model is only barely improved 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.

It is clear that we have addressed most of the points of our research question. Therefore, in future phases of this project, we should probably seek to improve our model by making it sensitive to more factors - specifically, we could make models that are sensitive to the play surface of the match, and see if we can improve the auc-roc score of our model by doing so.

We could also try to determine if there subsets of matches where our model performs better or worse than other subsets. For example, it's possible that our model makes better predictions for matches between highly ranked players than between players with low ranks.

Questions for Reviewers

- Have we conclusively answered our research question as stated at the beginning of this document? It seems that we have shown the relative predictive values between serve rating, return rating, and overall ranking based on the auc-roc score vs threshold curve. Is there anything else that we need to do to establish the relative predictive qualities of these three metrics?
- Is it interesting to expand on our research question just by adding more parameters to our model? Like is a model that has a higher max auc-roc curve (say, 80-85% instead of ~70%) an interesting outcome of our research? Or should we focus more on determining subsets of matches where our model performs better or worse?

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