Machine Learning and AI Project

Introduction

Since the birth of the "Moneyball" era of baseball originally crafted by the Oakland Athletics in 2002, data analytics have become a prevalent part of professional sports development. A wide range of professional sports programs now divert a solid portion of their funding to analytic departments designed to increase player performance, devise new game plans, and make informed decisions. Not only has this data-driven approach revolutionized the way organizations structure their teams but it has also bled into the world of sports betting, where complex algorithms based around historical data, player performance, and various environmental conditions are used to create odds and team favorites.

The scope of this project is to mimic the methods used by larger professional organizations and tailor them to a volleyball data set, specifically the Volleyball Nations League (VNL) results from the 2021-2022 seasons. By leveraging machine learning techniques such as Random Forest, Decision Tree, and Logistic Regression this project aims to predict the likelihood of a particular country to win a volleyball match in the VNL based on historical data.

IMPORTING LIBRARIES AND DATA SETS

The datasets utilized in this project are sourced from Kaggle. The primamry data set is available at https://www.kaggle.com/datasets/sumbowdy/volleyball-nations-league-2021-2022-data and highlights the amtch results for the 2021-2022 volleyball seasons of the VNL.

To aid the above data set is the following: https://www.kaggle.com/code/zakirpasha/eda-vnl-viz-sql which provides a copmrehensive list of player metrics and team rosters for each team that played in the 2021-2023 VNL seasons.

The focus of this project is to employ logistic regression, decision tree, and random forest algorithms through use of Python libraries such as Panda's, NumPy, Seaborn, and Matplotlib.pyplot. These algorithms are chosen for their efficiency when analyzing larger numeric datasets with multiple variables, as well are their abilities to model complex decision making processes.

To initialize this project, the first step as with any coding-based project, is to import and pre-process the datasets. This will be followed by an exploratory data analysis to assess trends amongst the various countries. The machine learning models will then be trained and evaluated on the selected predictor variables and analyzed based their accuracy scores to identify influential variables of team performance.

In [127…
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [128…
```
#import CSV's from our saved files
roster = pd.read_csv('C:/Users/wtrag/OneDrive/Documents/WR Portfolio/Zakir VBall Data/df_mens_rosters_21_23.csv')
individuals = pd.read_csv('C:/Users/wtrag/OneDrive/Documents/WR Portfolio/Zakir VBall Data/df_mens_indv_21_23.csv')
VNL = pd.read_csv('C:/Users/wtrag/OneDrive/Documents/WR Portfolio/Zakir VBall Data/VNL_DATA.csv')
```

In [129…
```
roster.describe(include='all') # a quick look at the summary of all our columns for rosters
#this set shows us what player plays for each country and their individual stats
```

Out[129]:

	No.	Player Name	Position	Player_ID	Year	Country_Name	Nationality	
count	893.000000	941	941	893.000000	941.000000	941	893	893.00
unique	NaN	567	7	NaN	NaN	18	18	
top	NaN	Sanchez Matias	OH	NaN	NaN	United States	United States	
freq	NaN	3	278	NaN	NaN	61	58	
mean	14.412094	NaN	NaN	150543.433371	2021.950053	NaN	NaN	27.03
std	11.652111	NaN	NaN	19832.950780	0.747088	NaN	NaN	4.21
min	1.000000	NaN	NaN	112869.000000	2021.000000	NaN	NaN	18.00
25%	7.000000	NaN	NaN	136851.000000	2021.000000	NaN	NaN	24.00
50%	13.000000	NaN	NaN	152405.000000	2022.000000	NaN	NaN	26.00
75%	20.000000	NaN	NaN	164022.000000	2023.000000	NaN	NaN	30.00
max	99.000000	NaN	NaN	192453.000000	2023.000000	NaN	NaN	43.00

In [130…
```
individuals.describe(include='all')# a quick look at the summary of all our columns for indiviudals
#this shows us what teams played each other and the contributions from each player during that match
```

Out[130]:

	Player_ID	Year	TeamA	TeamB	Serve_Points	Serve_Errors	Serve_Attempts	Tota
count	8890.000000	8890.000000	8782	8782	8890	8890	8890	
unique	NaN	NaN	18	18	11	11	28	
top	NaN	NaN	Japan	Italy	0	0	0	
freq	NaN	NaN	863	738	6863	4564	3524	
mean	148602.457705	2021.896625	NaN	NaN	NaN	NaN	NaN	
std	19041.906013	0.809995	NaN	NaN	NaN	NaN	NaN	
min	112869.000000	2021.000000	NaN	NaN	NaN	NaN	NaN	
25%	136442.000000	2021.000000	NaN	NaN	NaN	NaN	NaN	
50%	150703.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	
75%	162870.000000	2023.000000	NaN	NaN	NaN	NaN	NaN	
max	192453.000000	2023.000000	NaN	NaN	NaN	NaN	NaN	

11 rows × 29 columns

In [131…

```
VNL.describe(include='all') # a quick look at the summary of all our
columns for VNL
#this will give us our historical match results from the years 2021-2022
```

Out[131]:

	MATCHID	TEAM1	TEAM2	SCORES	SETSCORES	ATTACKS1	ATTACKS2	BLOCKS1	B
count	224.000000	224	224	224	224	224.000000	224.000000	224.000000	224
unique	NaN	17	17	6	222	NaN	NaN	NaN	
top	NaN	Poland	Slovenia	3\n:\n0	22-25\n23-25\n19-25	NaN	NaN	NaN	
freq	NaN	18	18	54	2	NaN	NaN	NaN	
mean	12635.611607	NaN	NaN	NaN	NaN	46.544643	45.816964	7.196429	7
std	967.824196	NaN	NaN	NaN	NaN	11.272721	11.155142	3.541780	3
min	11700.000000	NaN	NaN	NaN	NaN	0.000000	0.000000	0.000000	0
25%	11755.750000	NaN	NaN	NaN	NaN	38.750000	37.000000	5.000000	5
50%	11811.500000	NaN	NaN	NaN	NaN	47.000000	46.000000	7.000000	7
75%	13695.250000	NaN	NaN	NaN	NaN	55.000000	54.000000	9.000000	9
max	13751.000000	NaN	NaN	NaN	NaN	70.000000	75.000000	19.000000	18

11 rows × 21 columns

These datasets will be used to determine average points scored by individuals for each country then utilized to predict based on head-to-head country performance which teams, statistically,

have a better chance at winning in future matches. Not much can be derived from this data as is, so the next focus of this assignment is to clean up our data sets and perform a few exploratory analyses before getting into the meat of the code.

Preliminary Analysis and Data Cleansing

The first order of business is to merge all the data sets into one. This will create an easier way of navigation moving forward and will create less work for coding our algorithms in the future. Unfortunately, this will not be as easy as it seems. There will be certain columns used that will need to be converted into numeric and a group by function will need to be present merged by the country names.

In [132…

```
#convert our Average by match column into a numeric value
roster['Average by Match'] = pd.to_numeric(roster['Average by Match'],
errors='coerce')

#create a data frame here (named test as I was not sure this would work)
and group our variables by country name and year 2021-2023
test_df = roster.groupby(['Country_Name','Year'])['Average by
Match'].agg('sum') #creates a sum of all averages per player to create a
total for the country
test_df = pd.DataFrame(test_df) #convert our column to a data frame

#combine both roster and indiviudal on the player id tokens through a
inner join
df_combined = pd.merge(roster, individuals, on='Player_ID', how='inner')

#creating two different data frames (A/B) to differentiate team A's
average poitns by match and Team B's
#creates data frame with average points by match for team A (created from
merging on the columns Year_x and Year/Team A and Country)
df_finalA = pd.merge(df_combined, test_df, left_on= ['Year_x','TeamA'],
right_on=['Year','Country_Name'])

#creates data frame with average points by match for team B (created from
merging on the columns Year_x and Year/Team B and Country)
df_finalB = pd.merge(df_combined, test_df, left_on= ['Year_x','TeamB'],
right_on=['Year','Country_Name'])
```

```
#renaming these columns to make them easier to find and view in our data frame
df_finalA = df_finalA.rename(columns={'Average by Match_y': 'Average by Match Team A'})
df_finalB = df_finalB.rename(columns={'Average by Match_y': 'Average by Match Team B'})

#lets merge both these to have both columns on the same data frame
df_final_both = pd.merge(df_finalA,df_finalB)
df_final_both #double check that the data frame has the expected columns
```

Out[132]:

	No.	Player Name	Position	Player_ID	Year_x	Country_Name	Nationality	Age	Height
0	1.0	Sanchez Matias	S	142412.0	2021	Argentina	Argentina	25.0	175.0
1	2.0	Pereyra Federico	O	117445.0	2021	Argentina	Argentina	33.0	200.0
2	3.0	Martinez Franchi Jan	U	142410.0	2021	Argentina	Argentina	24.0	190.0
3	6.0	Poglajen Cristian	OH	119853.0	2021	Argentina	Argentina	32.0	195.0
4	7.0	Conte Facundo	OH	118886.0	2021	Argentina	Argentina	32.0	197.0
...	...	...	...	...	...	...	...	...	...
18639	12.0	Brehme Anton	MB	162081.0	2023	Germany	Germany	24.0	206.0
18640	31.0	Böhme Yan	O	184322.0	2023	Germany	Germany	26.0	203.0
18641	20.0	Homayonfarmanesh Shahrooz	OH	123413.0	2023	Iran	Iran	34.0	192.0
18642	22.0	Afshin Far Armin	OH	189602.0	2023	Iran	Iran	25.0	207.0
18643	27.0	Valizadeh Mohammad	MB	174373.0	2023	Iran	Iran	22.0	204.0

18644 rows × 50 columns

In [133…

```
#next a combination of the newly created data frame above and our vnl data must happen
df_mega = pd.merge(df_final_both, VNL, left_on = ['TeamA', 'TeamB'], right_on=['TEAM1', 'TEAM2'])

#look at the columns to make sure everything was added
df_mega.columns
```

Out[133]:
```
Index(['No.', 'Player Name', 'Position', 'Player_ID', 'Year_x', 'Country_Name',
       'Nationality', 'Age', 'Height', 'Total Points', 'Average by Match_x',
       'Attack Points', 'Efficiency', 'Attack Avg Points', 'Block Points',
       'Block Success', 'Block Avg Points', 'Serve Points', 'Serve Success',
       'Serve Avg Points', 'Year_y', 'TeamA', 'TeamB', 'Serve_Points',
       'Serve_Errors', 'Serve_Attempts', 'Total_Serves', 'Match_Date',
       'Set_Successes', 'Set_Errors', 'Set_Attempts', 'Total_Sets',
       'Attack_Success', 'Attack Errors', 'Attack_Attempts', 'Total_Attacks',
       'Block_Successful', 'Block_Errors', 'Block_Rebounds', 'Total_Blocks',
       'Reception_Successful', 'Reception_Errors', 'Reception_Attemps',
       'Total_Receptions', 'Dig_Success', 'Dig_Errors', 'Dig_Attempts',
       'Total_Digs', 'Average by Match Team A', 'Average by Match Team B',
       'MATCHID', 'TEAM1', 'TEAM2', 'SCORES', 'SETSCORES', 'ATTACKS1',
       'ATTACKS2', 'BLOCKS1', 'BLOCKS2', 'SERVES1', 'SERVES2',
       'OPPONENT_ERRORS1', 'OPPONENT_ERRORS2', 'TOTAL1', 'TOTAL2', 'DIGS1',
       'DIGS2', 'RECEPTION1', 'RECEPTION2', 'SETS1', 'SETS2'],
      dtype='object')
```

The data frame above now has all the information combined and an average by match total for both countries who competed in a match from 2021-2022. While this works for general analysis, the data cannot be processed in our machine learning algorithms as it has too many unnecessary columns and rows that would present errors in further exploration. To make this data usable, df_mega will need to cut any rows containing the year 2023 as there is no winner defined in the VNL data frame. Along with this, over half of the columns in the data frame will be dropped as they are not relevant to the predictions of match outcome.

In [134…
```
#drop the columns that will not be used in the analysis
df_cleaned_up = df_mega.drop(['No.', 'Position', 'Player_ID','SETSCORES',
'ATTACKS1','ATTACKS2', 'BLOCKS1', 'BLOCKS2', 'SERVES1', 'SERVES2',
                              'OPPONENT_ERRORS1', 'OPPONENT_ERRORS2',
'TOTAL1', 'TOTAL2', 'DIGS1',
                              'DIGS2', 'RECEPTION1', 'RECEPTION2',
'SETS1', 'SETS2','Set_Errors',
                              'Serve_Errors', 'Serve_Attempts',
'Total_Serves','Nationality','Block_Successful', 'Block_Errors',
'Block_Rebounds', 'Total_Blocks',
                              'Reception_Errors',
'Reception_Attemps','Total_Receptions','Dig_Errors', 'Dig_Attempts',
                              'Total_Digs','Block Success','Serve
Success','Set_Attempts', 'Total_Sets','MATCHID',
'Match_Date','Attack_Success', 'Attack Errors', 'Attack_Attempts',
       'Total_Attacks', 'Attack Points', 'Block Points', 'Serve Points',
'Serve_Points'], axis=1)
```

```
df_cleaned_up.head() #check that the columns did indeed get dropped
```

Out[134]:

	Player Name	Year_x	Country_Name	Age	Height	Total Points	Average by Match_x	Efficiency	Attack Avg Points	Block Avg Points	...	Tea
0	Sanchez Matias	2021	Argentina	25.0	175.0	4	0.57	1	0.29	0	...	
1	Pereyra Federico	2021	Argentina	33.0	200.0	147	9.80	0.49	8.33	0.8	...	
2	Martinez Franchi Jan	2021	Argentina	24.0	190.0	53	4.42	0.42	4.25	0.17	...	
3	Poglajen Cristian	2021	Argentina	32.0	195.0	49	4.45	0.41	4.09	0.27	...	
4	Conte Facundo	2021	Argentina	32.0	197.0	112	7.47	0.39	6.13	1.2	...	

5 rows × 22 columns

In [135…

```
#check the columns still available
df_cleaned_up.columns
```

Out[135]:

```
Index(['Player Name', 'Year_x', 'Country_Name', 'Age', 'Height',
       'Total Points', 'Average by Match_x', 'Efficiency', 'Attack Avg Points',
       'Block Avg Points', 'Serve Avg Points', 'Year_y', 'TeamA', 'TeamB',
       'Set_Successes', 'Reception_Successful', 'Dig_Success',
       'Average by Match Team A', 'Average by Match Team B', 'TEAM1', 'TEAM2',
       'SCORES'],
      dtype='object')
```

In [136…

```
#drop all rows with the Year_x = to 2023 as we do not have the data to confirm the winners of each match for those
df_cleaned_up = df_cleaned_up[(df_cleaned_up['Year_x']==2021)|(df_cleaned_up['Year_x']==2022)]
df_cleaned_up #look at the data frame and check everything is indeed in the right place
```

Out[136]:

	Player Name	Year_x	Country_Name	Age	Height	Total Points	Average by Match_x	Efficiency	Attack Avg Points	Block Avg Points	..
0	Sanchez Matias	2021	Argentina	25.0	175.0	4	0.57	1	0.29	0	.
1	Pereyra Federico	2021	Argentina	33.0	200.0	147	9.80	0.49	8.33	0.8	.
2	Martinez Franchi Jan	2021	Argentina	24.0	190.0	53	4.42	0.42	4.25	0.17	.
3	Poglajen Cristian	2021	Argentina	32.0	195.0	49	4.45	0.41	4.09	0.27	.
4	Conte Facundo	2021	Argentina	32.0	197.0	112	7.47	0.39	6.13	1.2	.
...	...	...	...	...	...	...	...	...	...	...	.
23676	Petric Nemanja	2022	Serbia	35.0	203.0	8	1.00	0.37	0.88	0.12	.
23677	Katic Milan	2022	Serbia	29.0	202.0	26	2.17	0.44	1.92	0.17	.
23678	Masulovic Nemanja	2022	Serbia	27.0	205.0	65	5.42	0.6	4.33	0.92	.
23679	Lisinac Srecko	2022	Serbia	30.0	205.0	49	6.12	0.71	5	1	.
23680	Koprivica Lazar	2022	Serbia	31.0	200.0	0	0.00	-	-	-	.

17411 rows × 22 columns

In [137…

```
#though we have a set winner from our VNL package we have to convert this
to a numeric value to test on
#creating a function here that replaces our text in the scores column with
a blank space as they area on seperate lines right now
def remove_newline(text):
    return text.replace('\n', '')

#this will apply the fourmula above to each line in the cores column
df_cleaned_up['SCORES'] = df_cleaned_up['SCORES'].apply(remove_newline)

#split the scores column into two different columns 'Scores1' and 'Scores
2' which allows us to see each teams set wins per match
df_cleaned_up[['SCORES1', 'SCORES2']] =
```

```
df_cleaned_up['SCORES'].str.split(':', expand=True)

# Convert the new columns to numeric type as we can utilize it easier this
way
df_cleaned_up['SCORES_TeamA'] = pd.to_numeric(df_cleaned_up['SCORES1'])
df_cleaned_up['SCORES_TeamB'] = pd.to_numeric(df_cleaned_up['SCORES2'])

#insert a new column called 'Winner' that looks at the new scores column
and determines a winner based one which column has the higher value
df_cleaned_up['Winner'] = df_cleaned_up.apply(lambda row: 1 if
row['SCORES1'] > row['SCORES2'] else 2, axis=1) #gives 1 if column 1
country wins 2 otherwise
df_cleaned_up.head(50) #double check the code works as intended
```

Out[137]:

	Player Name	Year_x	Country_Name	Age	Height	Total Points	Average by Match_x	Efficiency	Attack Avg Points	B Pc
0	Sanchez Matias	2021	Argentina	25.0	175.0	4	0.57	1	0.29	
1	Pereyra Federico	2021	Argentina	33.0	200.0	147	9.80	0.49	8.33	
2	Martinez Franchi Jan	2021	Argentina	24.0	190.0	53	4.42	0.42	4.25	
3	Poglajen Cristian	2021	Argentina	32.0	195.0	49	4.45	0.41	4.09	
4	Conte Facundo	2021	Argentina	32.0	197.0	112	7.47	0.39	6.13	
5	Loser Agustin	2021	Argentina	24.0	198.0	111	9.25	0.68	7.17	
6	Danani Santiago	2021	Argentina	26.0	176.0	0	0.00	-	-	
7	Lima Bruno	2021	Argentina	26.0	198.0	81	11.57	0.54	10.86	
8	Palacios Ezequiel	2021	Argentina	29.0	198.0	21	1.91	0.4	1.45	
9	De Cecco Luciano	2021	Argentina	33.0	191.0	18	1.20	0.35	0.53	
10	Palonsky Luciano	2021	Argentina	22.0	198.0	15	1.25	0.37	1.17	
11	Mendez Nicolas	2021	Argentina	29.0	191.0	45	3.00	0.41	2.53	
12	Ramos Martin	2021	Argentina	30.0	197.0	72	4.80	0.6	3.33	
13	Massimino Franco	2021	Argentina	33.0	177.0	0	0.00	-	-	
14	Ebadipour Ghara H. Milad	2021	Iran	28.0	196.0	99	7.62	0.4	6.31	
15	Abedini Reza	2021	Iran	30.0	203.0	18	1.64	0.67	0.73	
16	Marouflakrani Mir Saeid	2021	Iran	36.0	189.0	10	0.77	0.4	0.62	
17	MoUnited Statesvi Eraghi Seyed Mohammad	2021	Iran	34.0	203.0	103	7.36	0.55	4.71	
18	Fayazi D. Purya	2021	Iran	29.0	195.0	2	0.18	0.13	0.18	
19	Hazratpourtalatappeh Mohammadreza	2021	Iran	22.0	187.0	0	0.00	-	-	
20	Gholami Masoud	2021	Iran	31.0	204.0	63	4.20	0.54	2.73	
21	Ghafour Amir	2021	Iran	30.0	202.0	36	7.20	0.57	6.6	
22	Kazemi Saber	2021	Iran	23.0	205.0	177	12.64	0.45	10.86	
23	Sharifi Morteza	2021	Iran	22.0	193.0	26	2.36	0.42	1.91	
24	Salehi Meisam	2021	Iran	23.0	198.0	160	10.67	0.55	9.2	
25	Salehi Arman	2021	Iran	29.0	180.0	0	0.00	-	-	
26	Saadat Bardia	2021	Iran	19.0	205.0	75	5.36	0.5	4.86	
27	Karimisouchelmaei Javad	2021	Iran	24.0	204.0	8	1.33	0.2	0.17	

	Player Name	Year_x	Country_Name	Age	Height	Total Points	Average by Match_x	Efficiency	Attack Avg Points	B P
28	Sanchez Matias	2022	Argentina	26.0	175.0	8	0.67	0.45	0.42	
29	Martinez Franchi Jan	2022	Argentina	25.0	190.0	-	NaN	-	-	
30	Conte Facundo	2022	Argentina	33.0	197.0	-	NaN	-	-	
31	Loser Agustin	2022	Argentina	25.0	198.0	129	10.75	0.58	7.92	
32	Danani Santiago	2022	Argentina	27.0	176.0	0	0.00	-	-	
33	Lima Bruno	2022	Argentina	27.0	198.0	158	13.17	0.51	11.33	
34	Palacios Ezequiel	2022	Argentina	30.0	198.0	148	12.33	0.49	10.83	
35	De Cecco Luciano	2022	Argentina	34.0	191.0	10	2.50	0.25	0.5	
36	Palonsky Luciano	2022	Argentina	23.0	198.0	78	6.50	0.5	6.33	
37	Mendez Nicolas	2022	Argentina	30.0	191.0	-	NaN	-	-	
38	Ramos Martin	2022	Argentina	31.0	197.0	27	2.25	0.51	1.67	
39	Massimino Franco	2022	Argentina	34.0	177.0	0	0.00	-	-	
40	Ebadipour Ghara H. Milad	2022	Iran	29.0	196.0	146	11.23	0.48	8.77	
41	Abedini Reza	2022	Iran	31.0	203.0	5	1.67	0.5	0.67	
42	Hazratpourtalatappeh Mohammadreza	2022	Iran	23.0	187.0	0	0.00	-	-	
43	Kazemi Saber	2022	Iran	24.0	205.0	11	2.75	0.28	1.75	
44	Sharifi Morteza	2022	Iran	23.0	193.0	50	3.85	0.55	3.08	
45	Salehi Meisam	2022	Iran	24.0	198.0	-	NaN	-	-	
46	Saadat Bardia	2022	Iran	20.0	205.0	5	0.56	0.5	0.44	
47	Karimisouchelmaei Javad	2022	Iran	25.0	204.0	-	NaN	-	-	
61	Graham Beau	2021	Australia	27.0	203.0	101	6.73	0.38	6.4	
62	Graham Beau	2021	Australia	27.0	203.0	101	6.73	0.38	6.4	

50 rows × 27 columns

The data is finally processed enough to begin working on it. How about a simple test to look at the top scorers of the VNL through the last couple years?

In [138…

```
#first we need to change some our point data to numeric values
df_cleaned_up['Total Points'] = pd.to_numeric(df_cleaned_up['Total
Points'], errors='coerce')
```

```
print('The amount of players in the data set
is',len(df_cleaned_up),'Player Name')
print('The total amount of points scored is',df_cleaned_up['Total
Points'].sum())

#to find a percentage of points scored by individual players we can use a
group by function
individual_points = df_cleaned_up.groupby('Player Name')['Total
Points'].sum()

#next create a varibale for the total points column
total_points = df_cleaned_up['Total Points'].sum()

#now divide the two

percentage = (individual_points/total_points) * 100

#since we have almost 1000 player lets look at the top ten overall point
scoreres

top_10 = percentage.sort_values(ascending=False).head(10)
top_10= round(top_10, 2)
print(top_10)

#number 3 on this list Torey Defalco is actually someone I played against
when I was younger. HOW COOL!
```

```
The amount of players in the data set is 17411 Player Name
The total amount of points scored is 917892.0
Player Name
Abdel-Aziz Nimir          2.14
Ishikawa Yuki             1.87
Defalco Torey             1.80
Patry Jean                1.55
Čebulj Klemen             1.51
Souza Ricardo Lucarelli   1.46
Možič Rok                 1.46
Pajenk Alen               1.42
Michieletto Alessandro    1.41
Urnaut Tine               1.33
Name: Total Points, dtype: float64
```

The above shows the top ten players based on their total points scored in the 2021-2022 season as a percentage of overall points scored in the VNL. Almost half of the above players are based in Slovenia, which should give us some indicator of how well Slovenia should perform against most teams. However, this alone cannot provide a basis for the predictive model and more information will need to be gathered.

In [139…

```
#to find a percentage of points scored by individual players we can use a
group by function
country_points = df_cleaned_up.groupby('Country_Name')['Total
Points'].sum()

#next create a data frame teh points scored per country
country_points_df = country_points.reset_index()

#sort new data frame in descending order
sorted_country_points = country_points_df.sort_values(by='Total Points',
ascending=False)

print(sorted_country_points)
```

```
    Country_Name  Total Points
15       Slovenia       87684.0
6          France       86580.0
16  United States       75159.0
2          Brazil       74255.0
12         Poland       74054.0
10          Japan       69015.0
11    Netherlands       59178.0
9           Italy       57915.0
7         Germany       54077.0
0       Argentina       52541.0
4          Canada       47178.0
14         Serbia       42276.0
8            Iran       42098.0
3        Bulgaria       41968.0
1       Australia       32704.0
13         Russia       14129.0
5           China        7081.0
```

These results tell a tale of the overall top scoring countries giving a better indication for future matches of who will win. More points should, in theory, result in a better team with a better record.

Another valuable data realm to explore is the attributes of height, age, and individual points as a feature of the country each player represents. These variables should prove to be influential in predictions on match outcomes. To explore this measure a graph correlating to each country

and the respective attributes to be measured will be made. To do so, a group by clause for country names will be present at the beginning of the code with the respective variables being plotted on x axis.

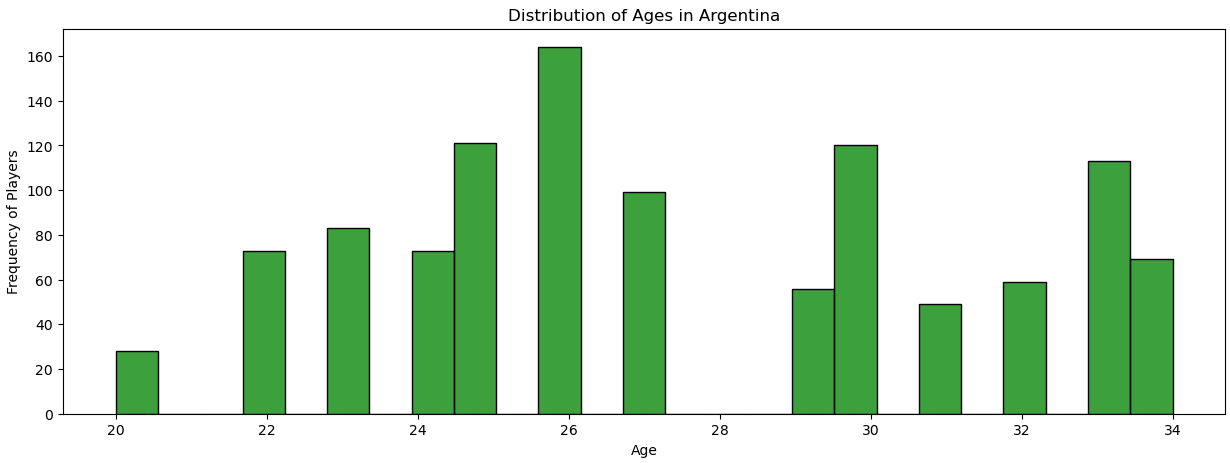
In [140…

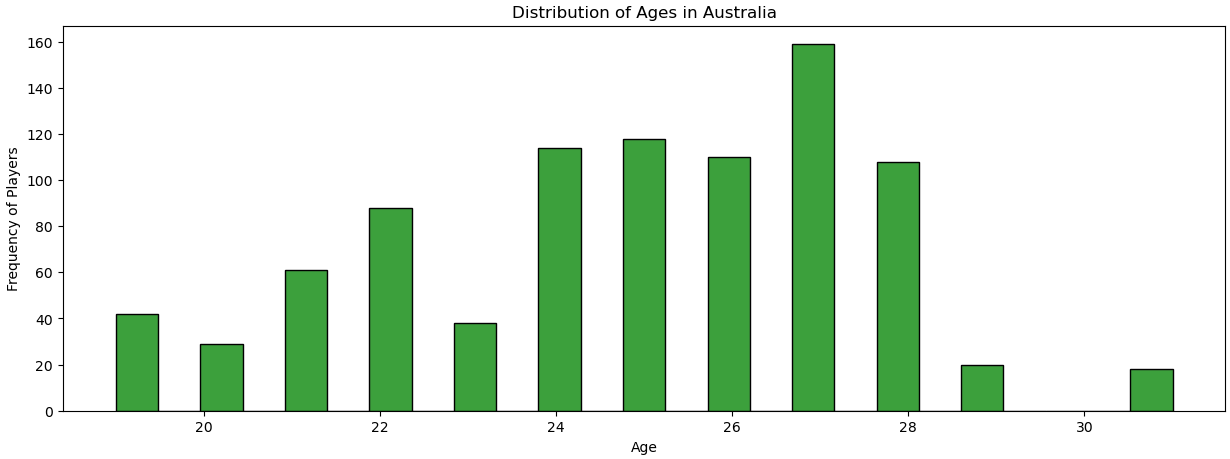
```
#Lets find out how many unique countries we have to make sure each country
is being shown
country_totals = df_cleaned_up['Country_Name'].nunique()

print("Number of unique values in the column:", country_totals)
#with 17 unique countires there should be 17 graphs per variable
```

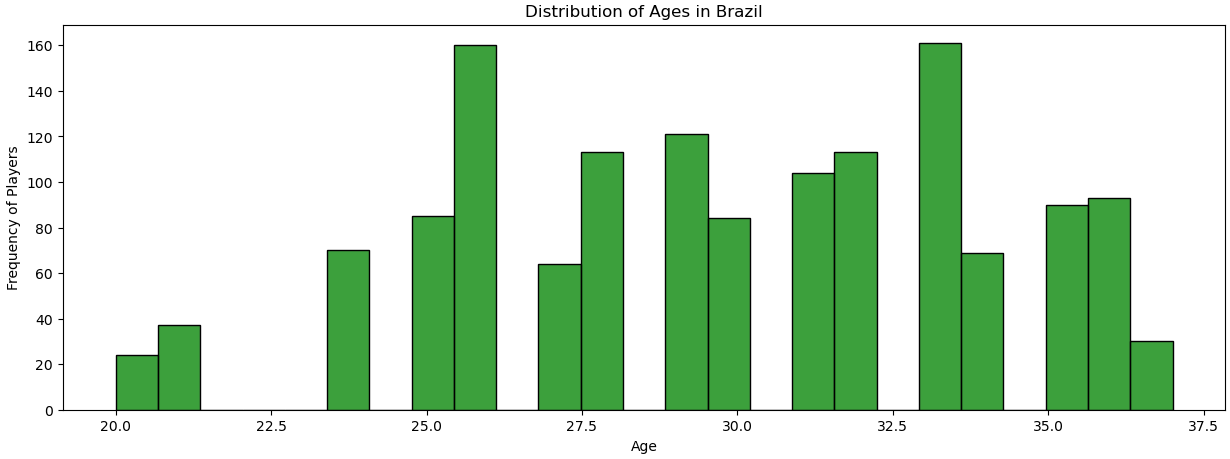
```
Number of unique values in the column: 17
```

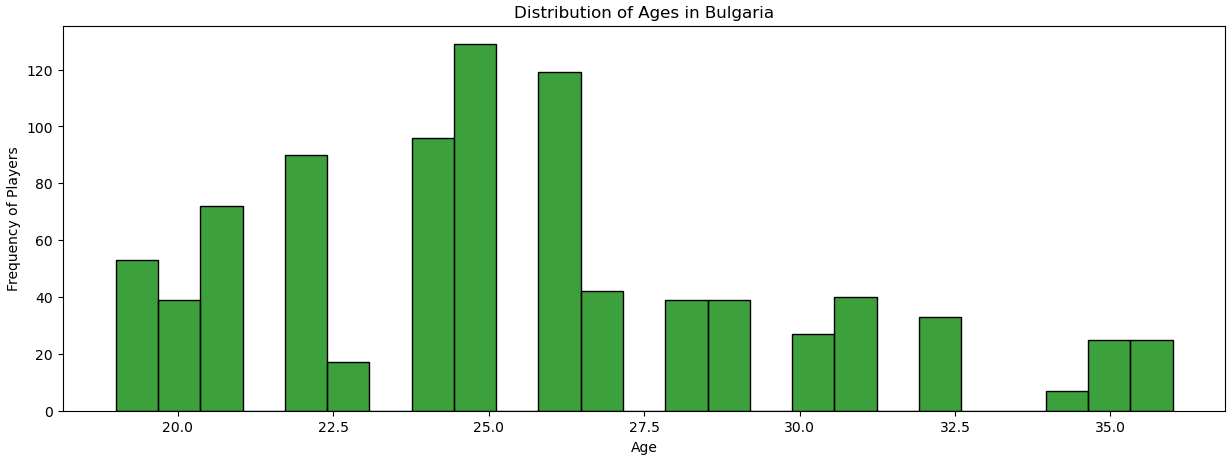
In [141…

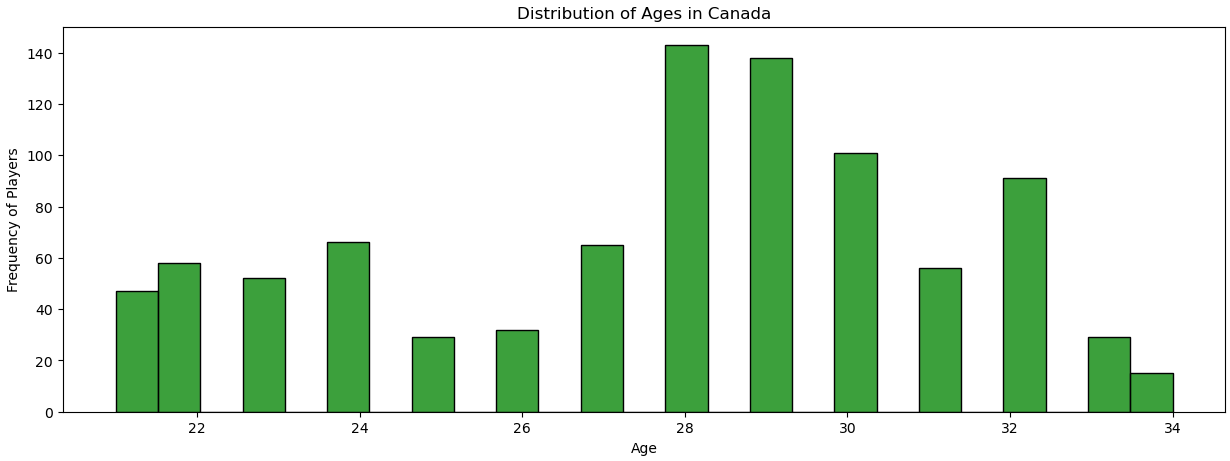
```
#first group our data by countries as we want to each unique country
available
grouped_countries = df_cleaned_up.groupby('Country_Name')

# Lets make our historgrams now
for country, data in grouped_countries:
    plt.figure(figsize=(15, 5))
    sns.histplot(data=data, x='Age', bins=25, element="bars",
color='green')
    plt.title(f'Distribution of Ages in {country}')
    plt.xlabel('Age')
    plt.ylabel('Frequency of Players')
    plt.savefig('age of players by country')
    plt.show()
```

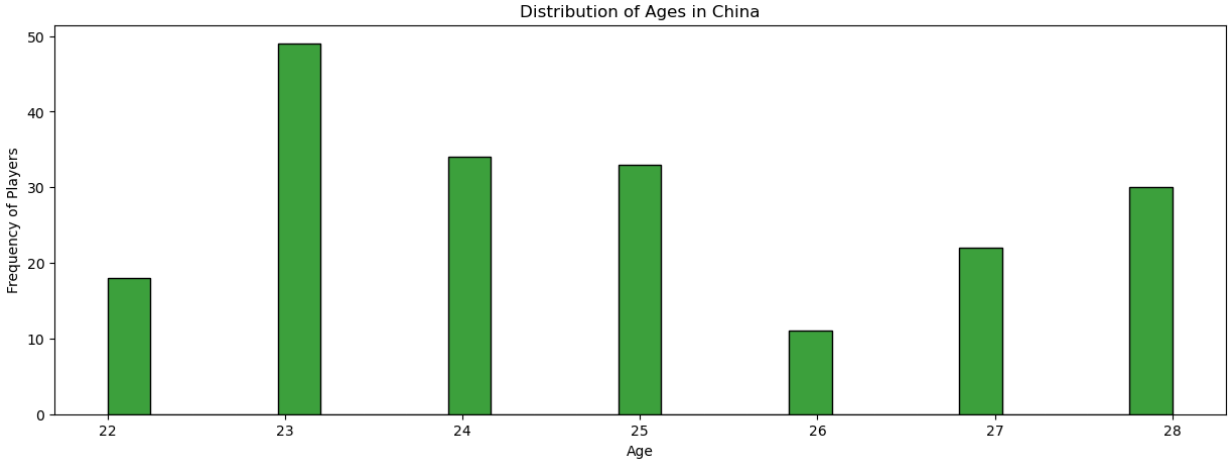


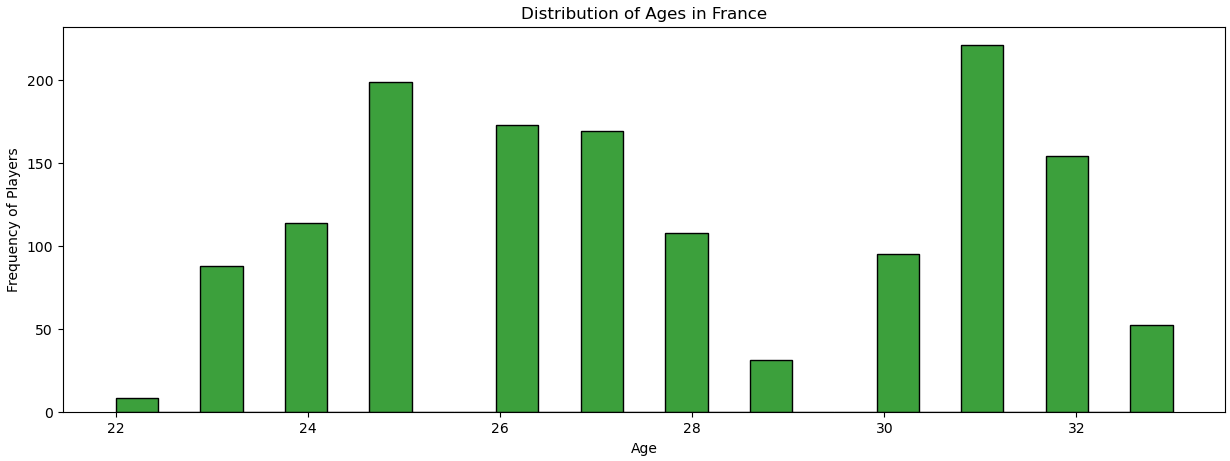

Distribution of Ages in Australia
Frequency of Players
Age

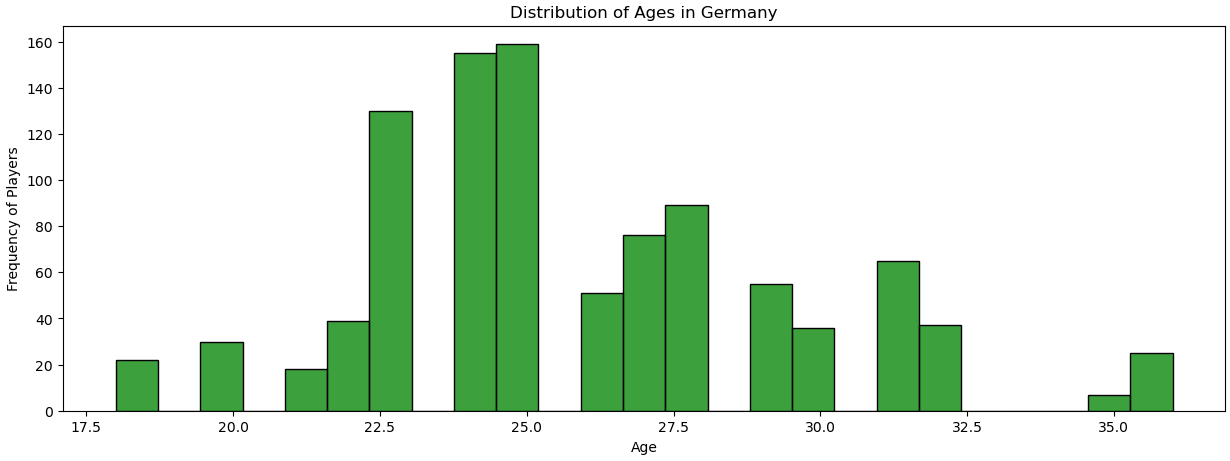

Distribution of Ages in Brazil
Frequency of Players
Age

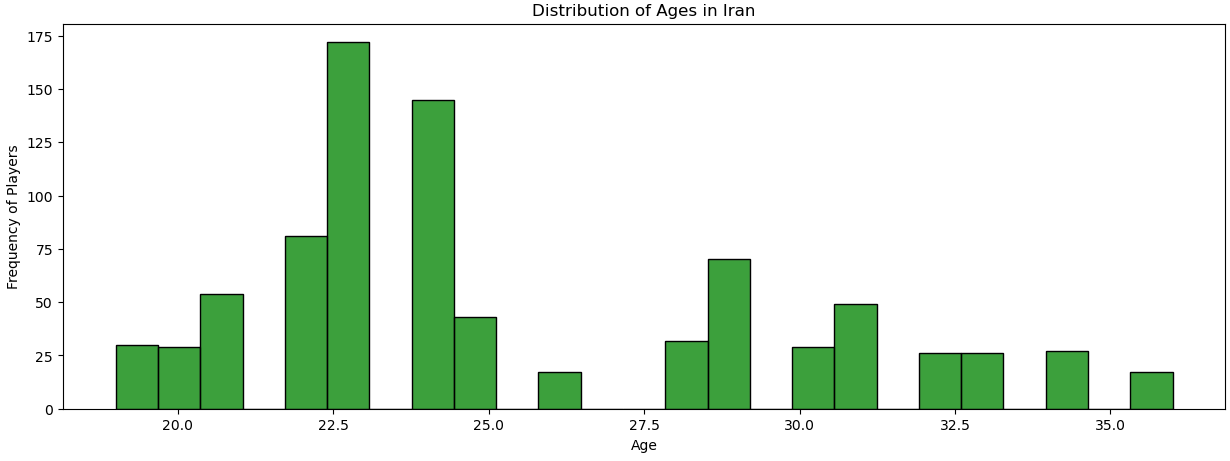

Distribution of Ages in Bulgaria
Frequency of Players
Age

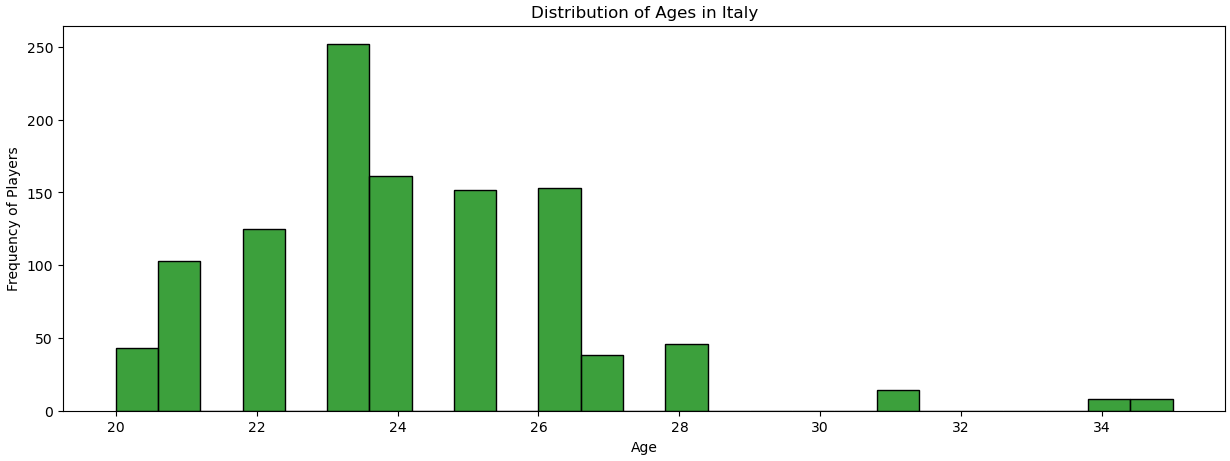

Distribution of Ages in Canada
Frequency of Players
Age

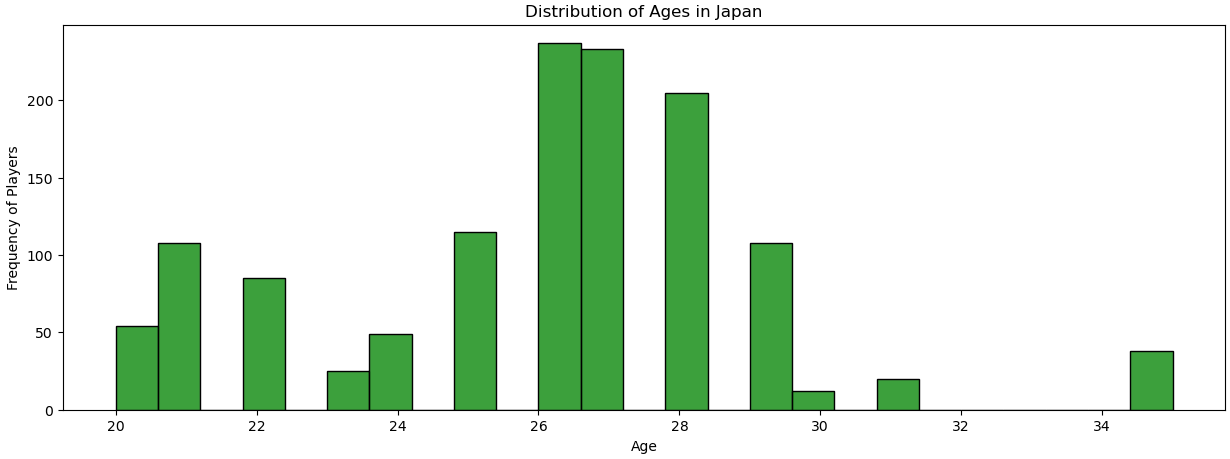

Distribution of Ages in China
Frequency of Players
Age
0
10
20
30
40
50
22
23
24
25
26
27
28

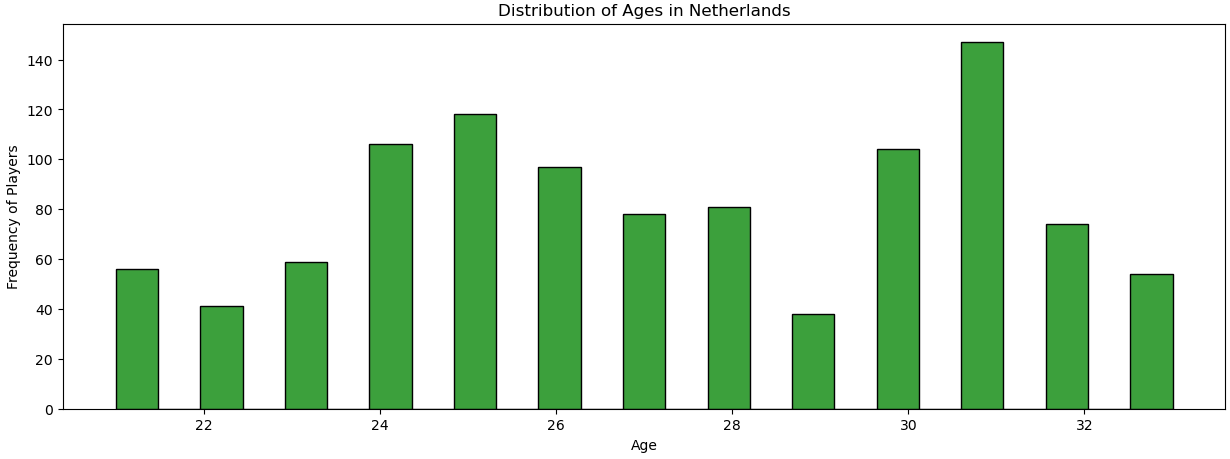

Distribution of Ages in France
Frequency of Players
Age
0
50
100
150
200
22
24
26
28
30
32

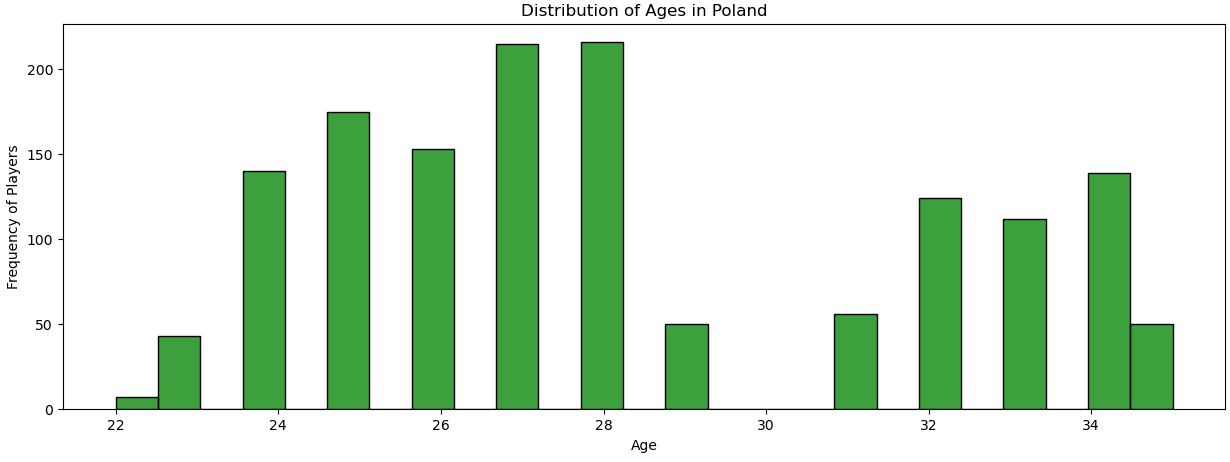

Distribution of Ages in Germany
Frequency of Players
Age
0
20
40
60
80
100
120
140
160
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0

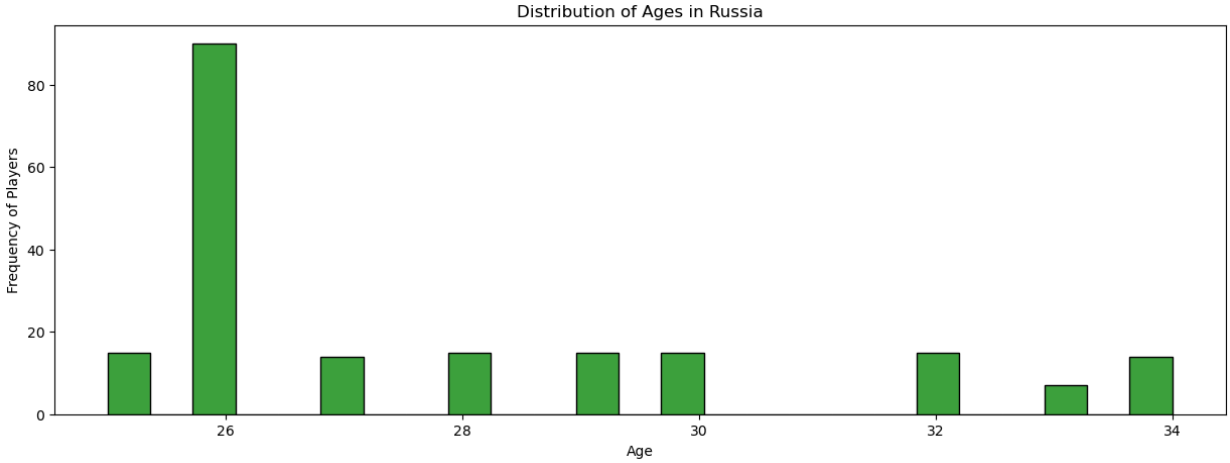

Distribution of Ages in Iran
Frequency of Players
Age
0
25
50
75
100
125
150
175
20.0
22.5
25.0
27.5
30.0
32.5
35.0

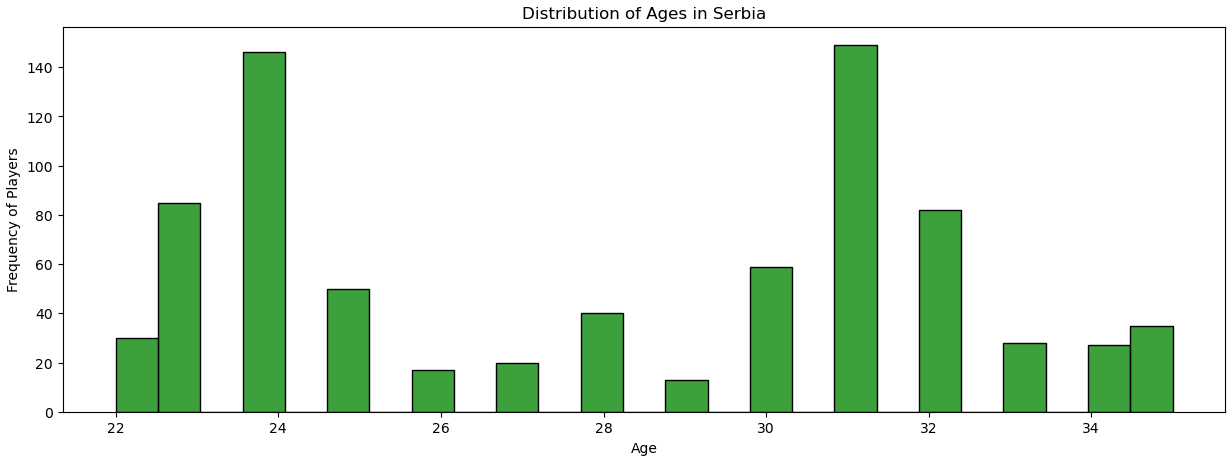

Distribution of Ages in Italy
Frequency of Players
Age
0
50
100
150
200
250
20
22
24
26
28
30
32
34

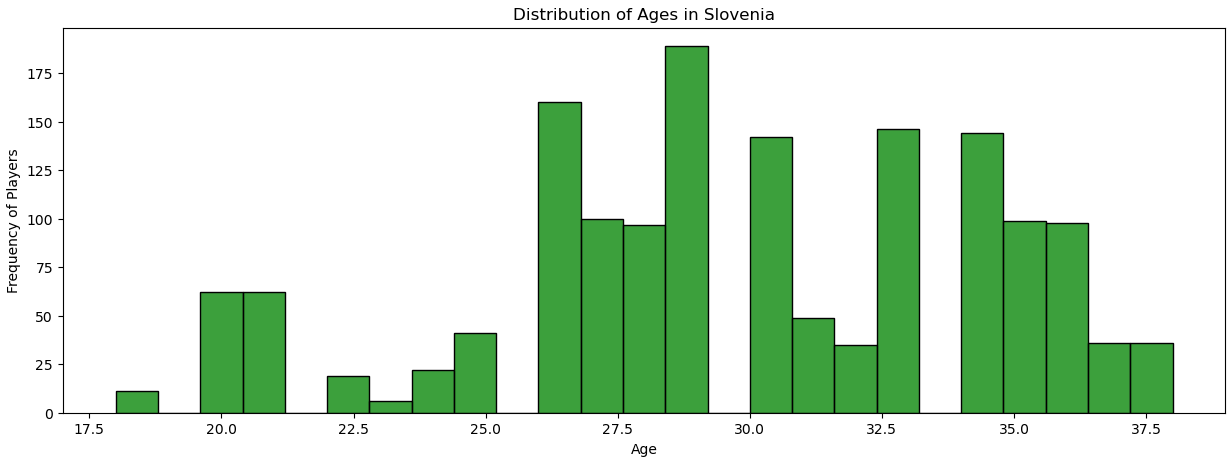

Distribution of Ages in Japan
Frequency of Players
Age
0
50
100
150
200
20
22
24
26
28
30
32
34

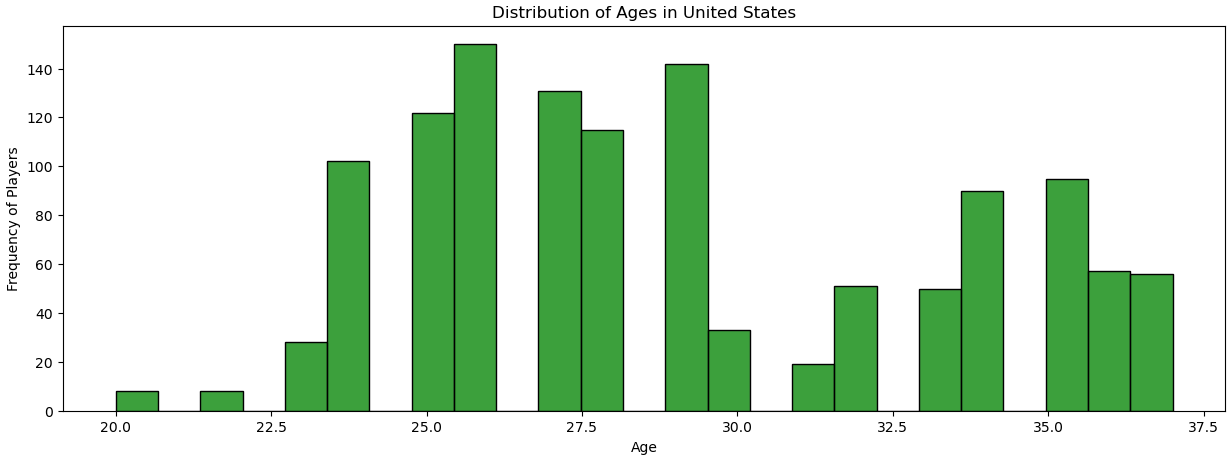

Distribution of Ages in Netherlands
Frequency of Players
Age
0
20
40
60
80
100
120
140
22
24
26
28
30
32


Distribution of Ages in Poland
Frequency of Players
Age
0
50
100
150
200
22
24
26
28
30
32
34


Distribution of Ages in Russia
Frequency of Players
Age


Distribution of Ages in Serbia
Frequency of Players
Age

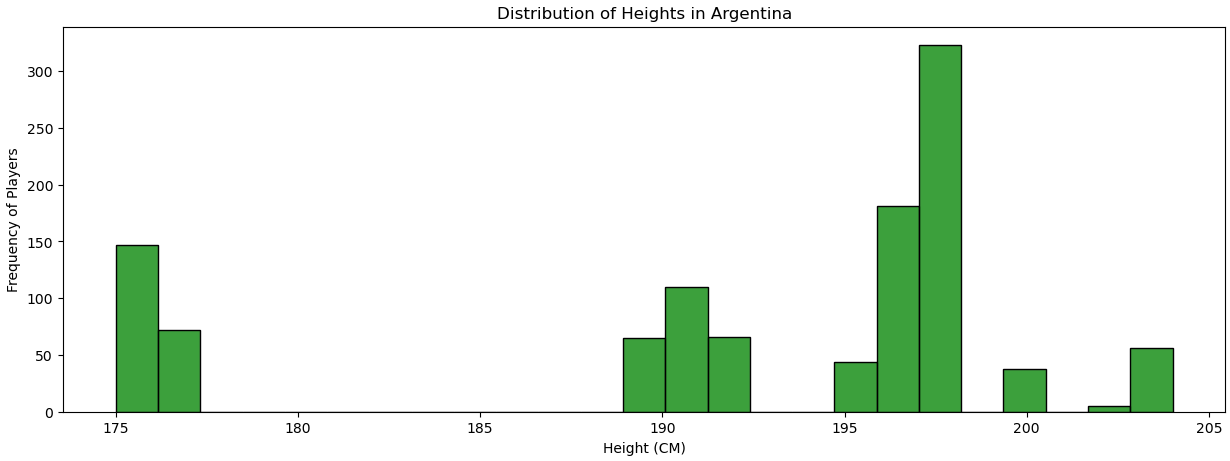

Distribution of Ages in Slovenia
Frequency of Players
Age

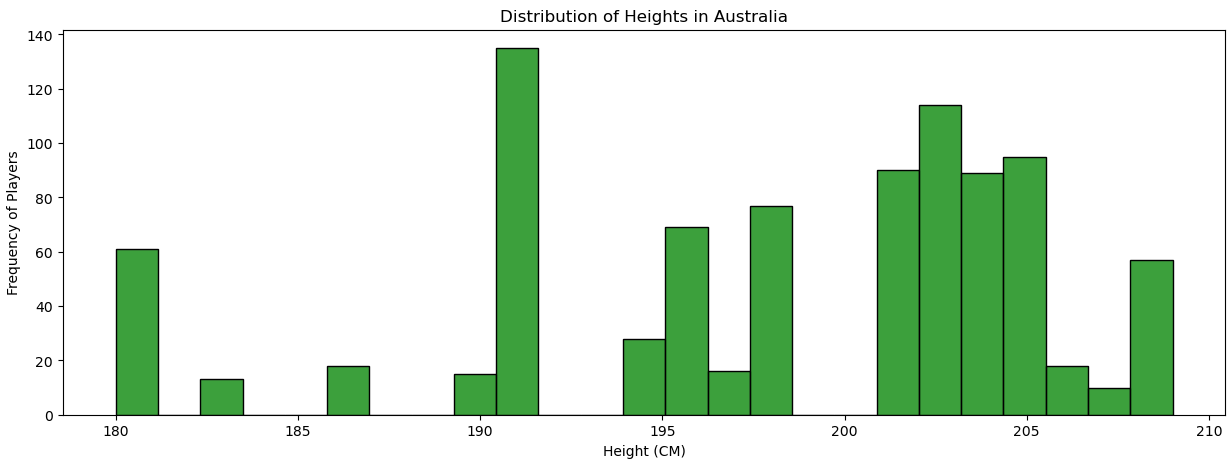

Distribution of Ages in United States
Frequency of Players
Age

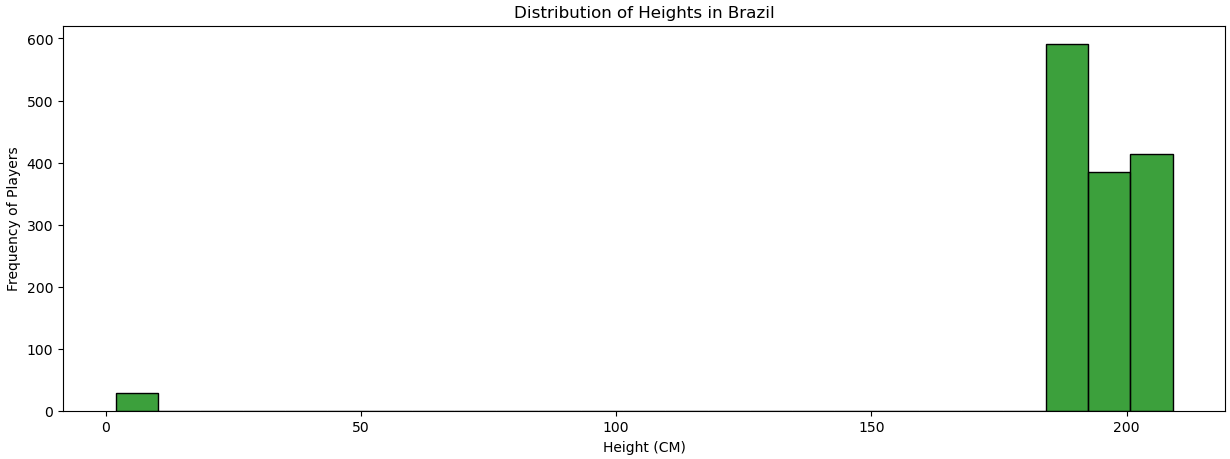
The above graphs tell a story about the typical age range of each player. Most countries tend to build a roster of players in their mid 20's as this tends to be the 'Prime' of a young adult's athletic life. There do tend to be some outliers with players being well into their thirties and even a few still in their teens!

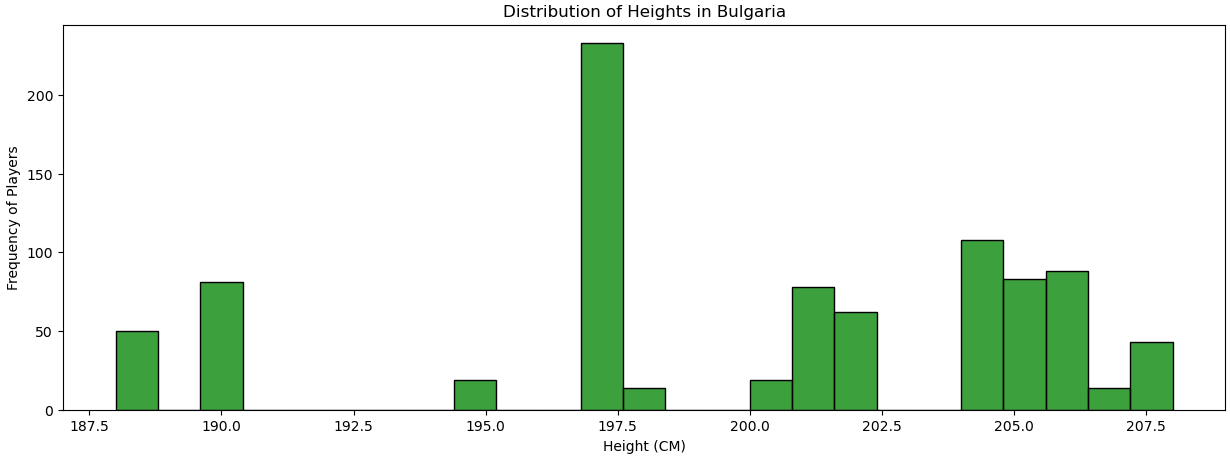
In [142…

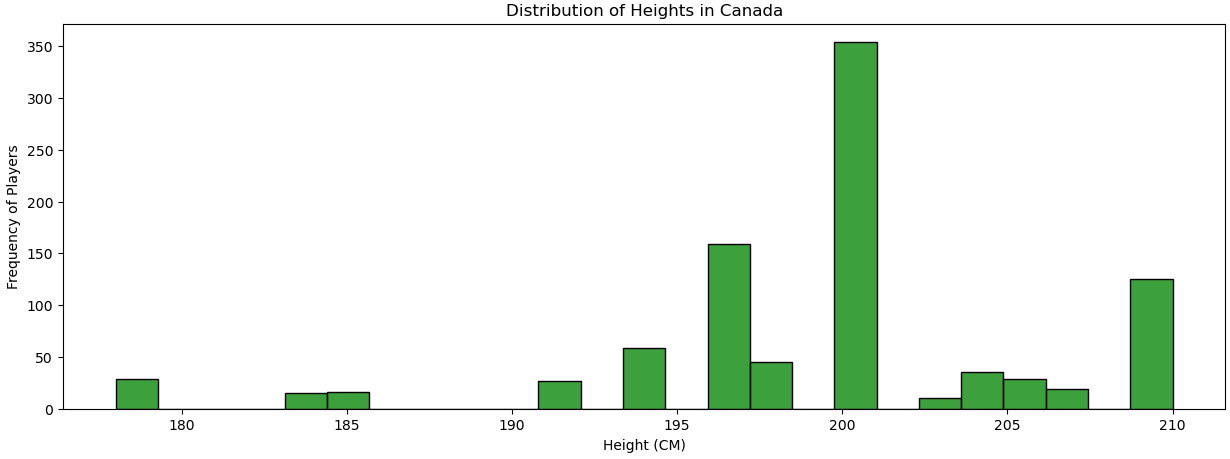
```
grouped_countries = df_cleaned_up.groupby('Country_Name')

# Lets make our historgrams now
for country, data in grouped_countries:
    plt.figure(figsize=(15, 5))
    sns.histplot(data=data, x='Height', bins=25, element="bars",
color='green')
    plt.title(f'Distribution of Heights in {country}')
    plt.xlabel('Height (CM)')
    plt.ylabel('Frequency of Players')
    plt.savefig('height of players by country')
    plt.show()
```

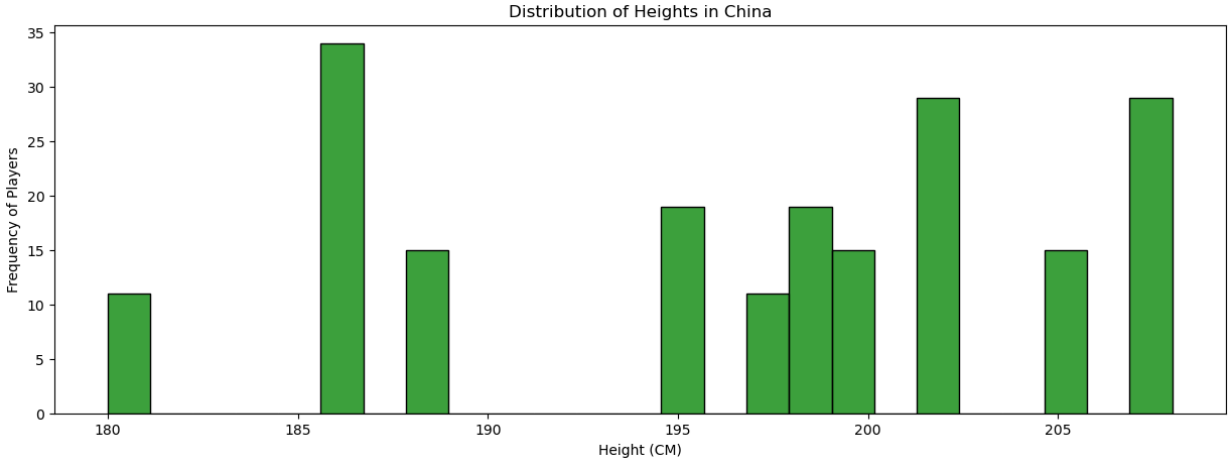


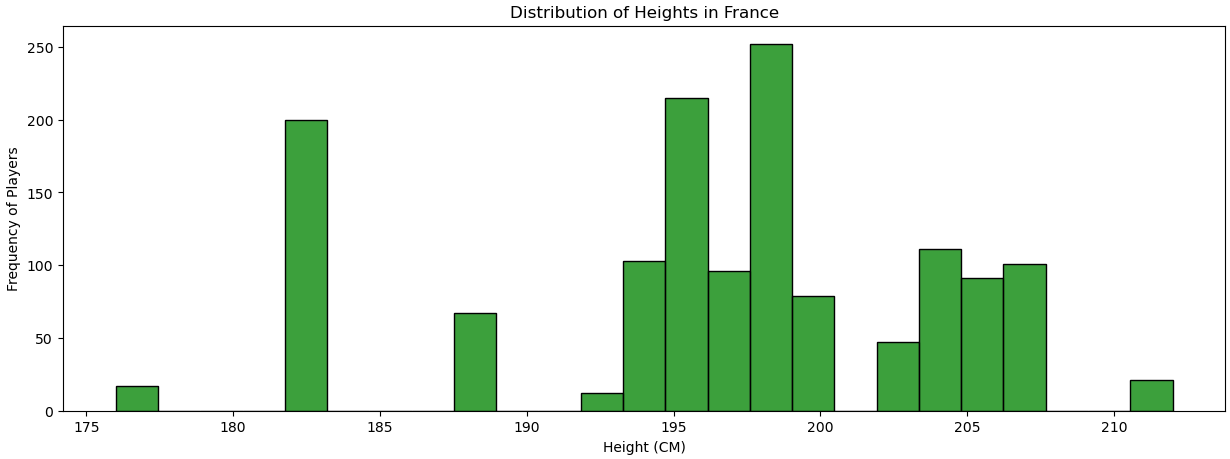


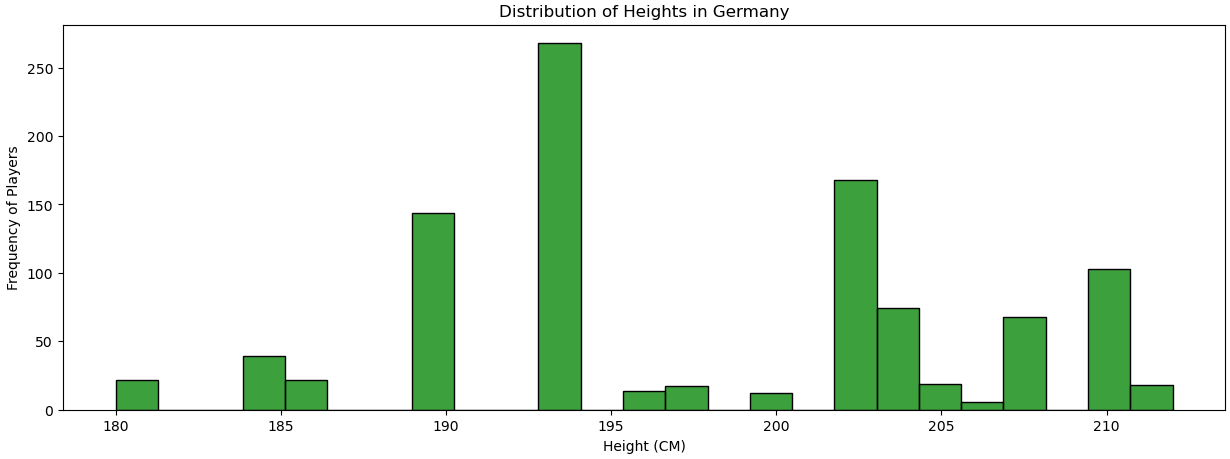

Distribution of Heights in Brazil
Frequency of Players
Height (CM)

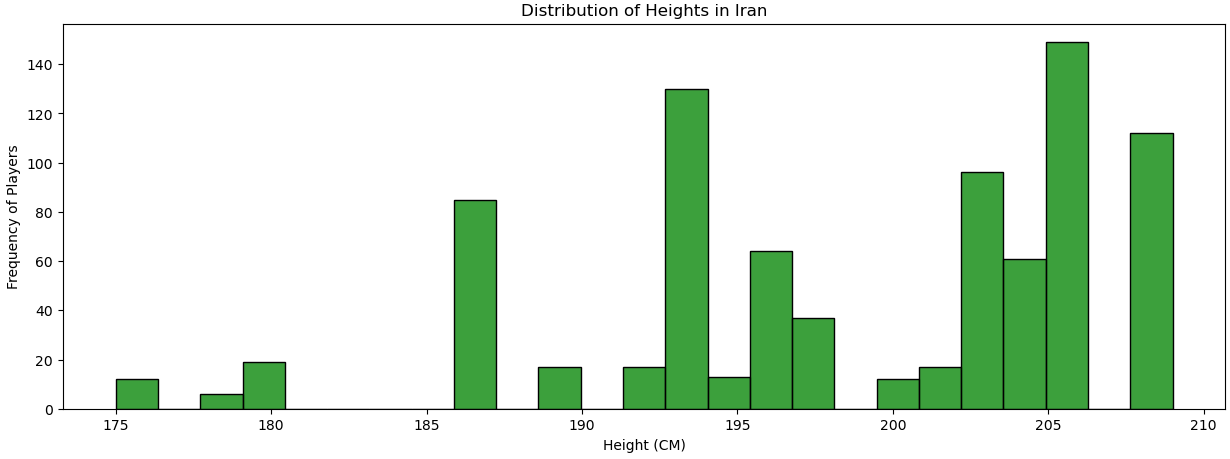

Distribution of Heights in Bulgaria
Frequency of Players
Height (CM)

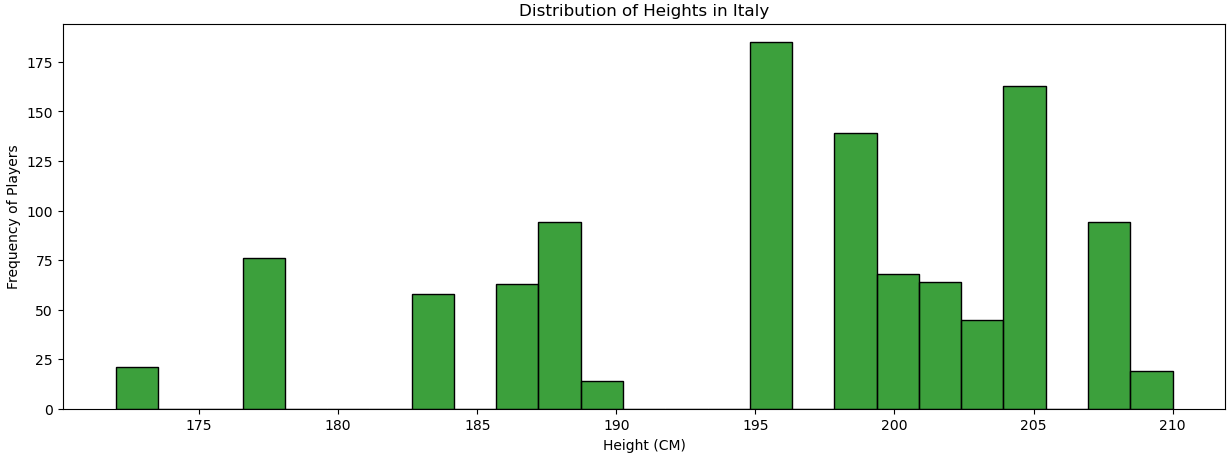

Distribution of Heights in Canada
Frequency of Players
Height (CM)

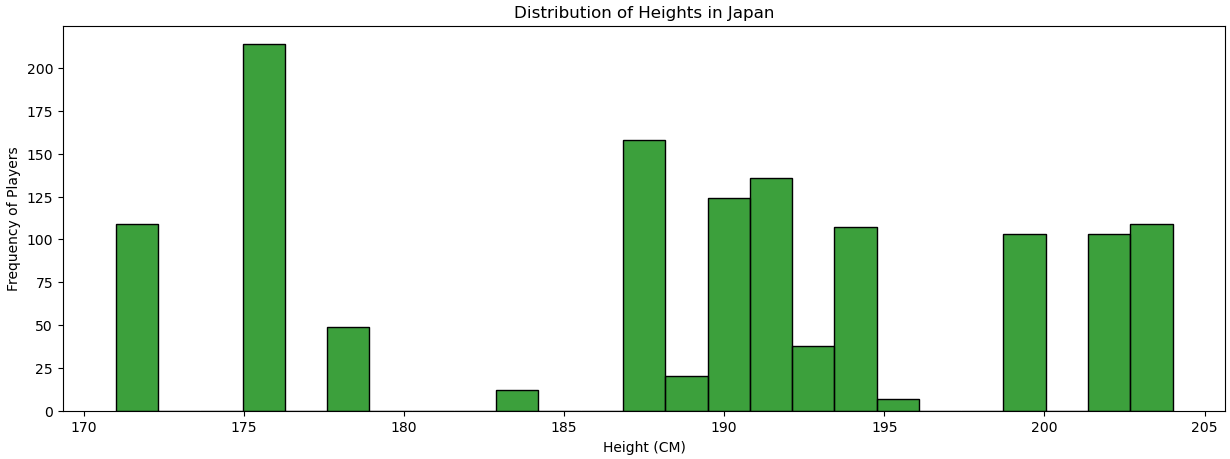

Distribution of Heights in China
Frequency of Players
Height (CM)

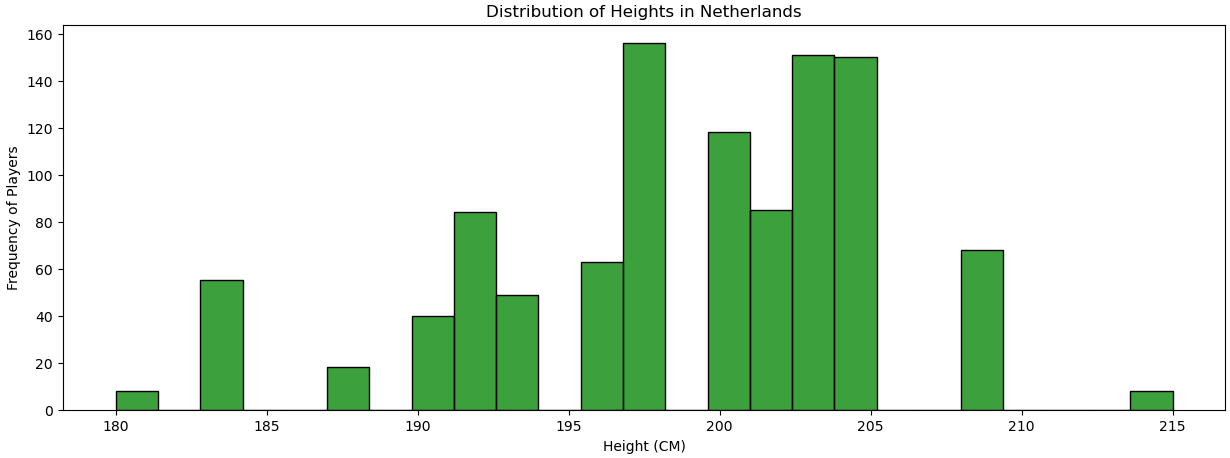

Distribution of Heights in France
Frequency of Players
Height (CM)

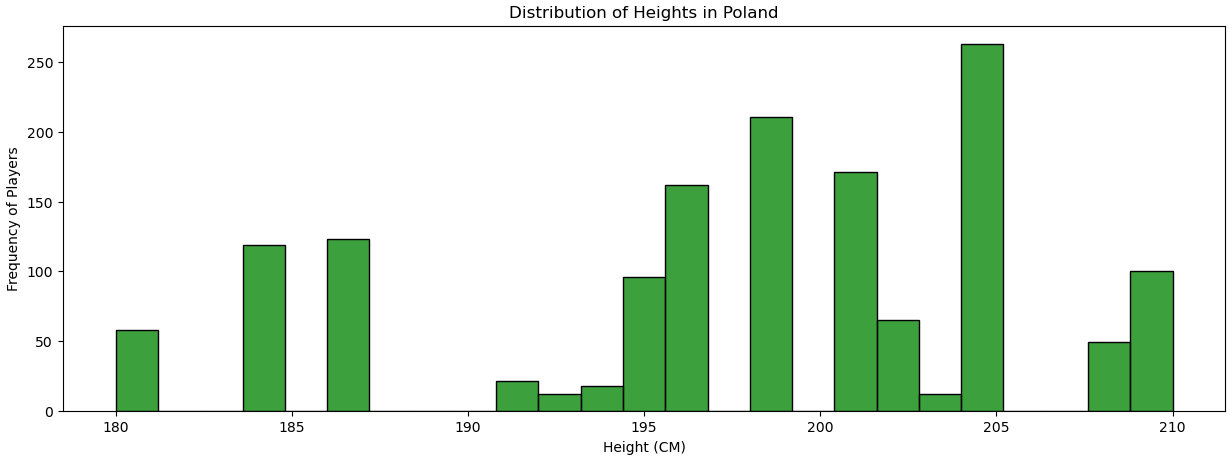

Distribution of Heights in Germany
Frequency of Players
Height (CM)

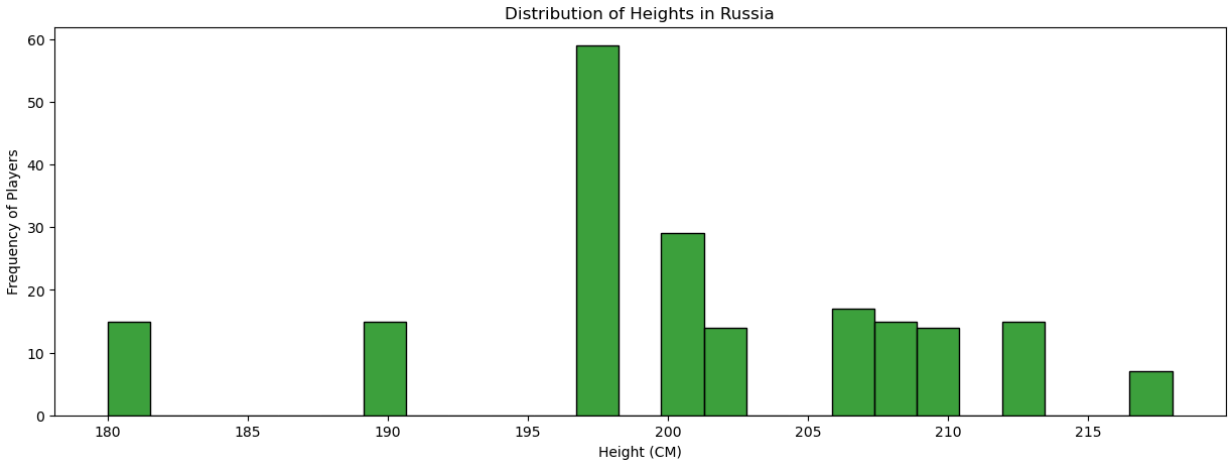

Distribution of Heights in Iran
Frequency of Players
Height (CM)

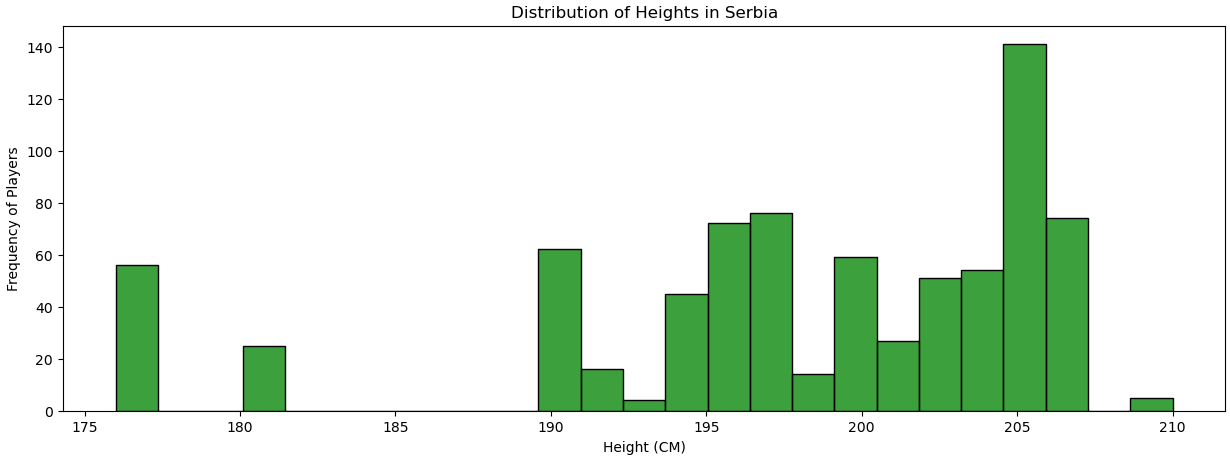

Distribution of Heights in Italy
Frequency of Players
Height (CM)

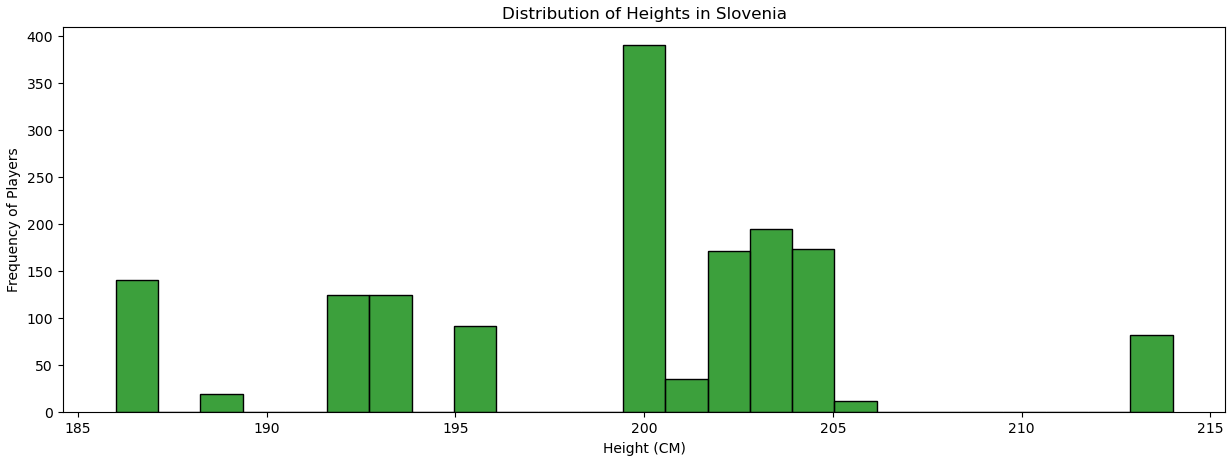

Distribution of Heights in Japan
Frequency of Players
Height (CM)

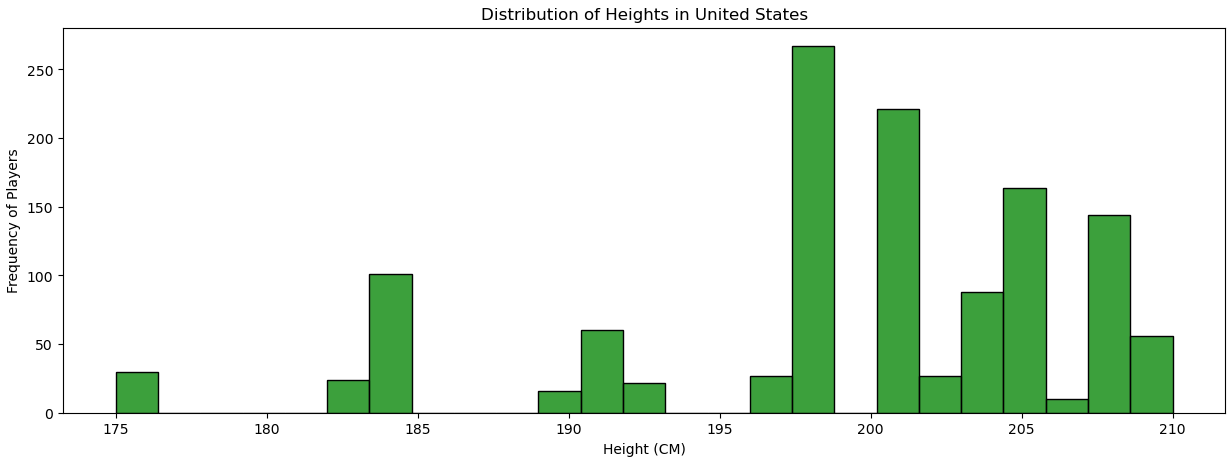

Distribution of Heights in Netherlands
Frequency of Players
Height (CM)


Distribution of Heights in Poland
Frequency of Players
Height (CM)


Distribution of Heights in Russia
Frequency of Players
Height (CM)





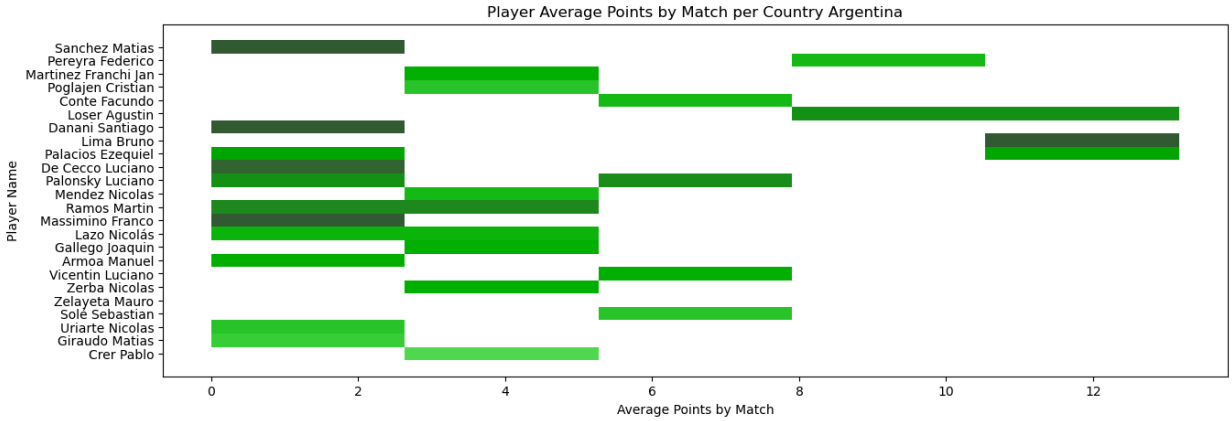


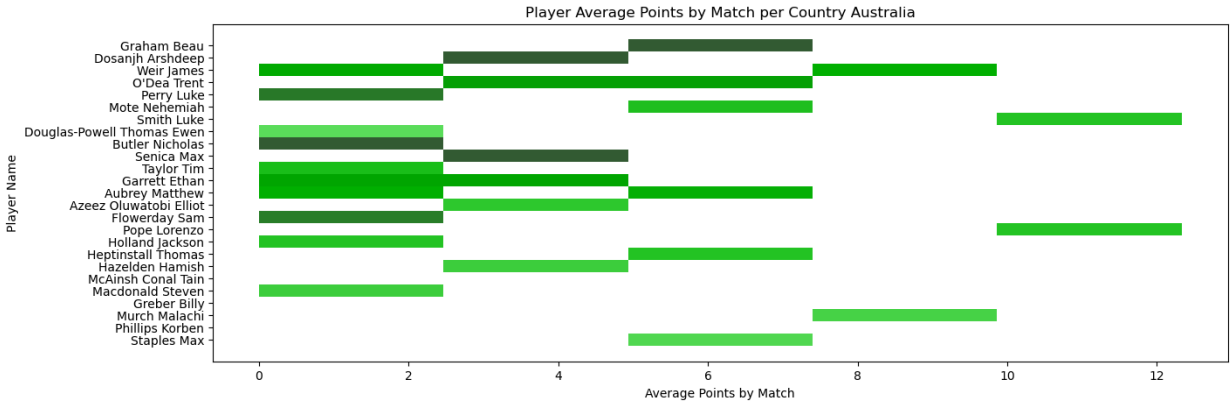
As with the age range graphs above, these graphs tell a similar story of rostered players heights per each country. Volleyball is often thought to be a sport designed for taller individuals which can be clearly seen in the distributions above. Each country seems to bring on players from about 185cm and above. The outliers that fall below this range are probably liberos for the team or shorter players that have unbelievable athletic abilities.

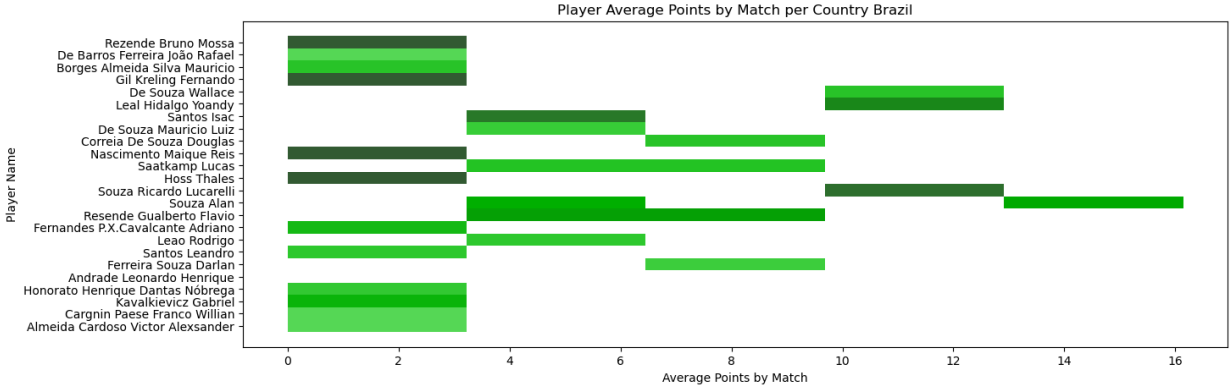
In [143…

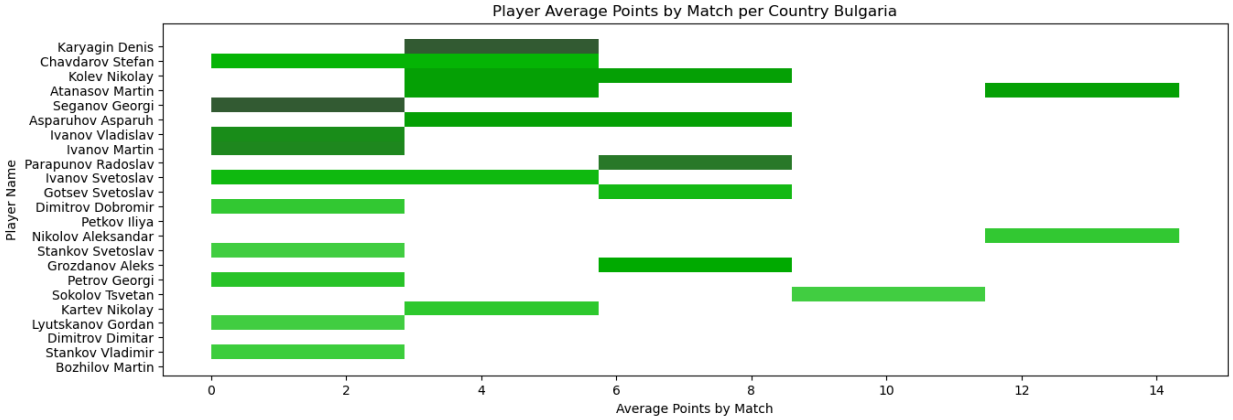
```
#first group our data by countries as we want to each unique country
available
grouped_countries = df_cleaned_up.groupby('Country_Name')
```

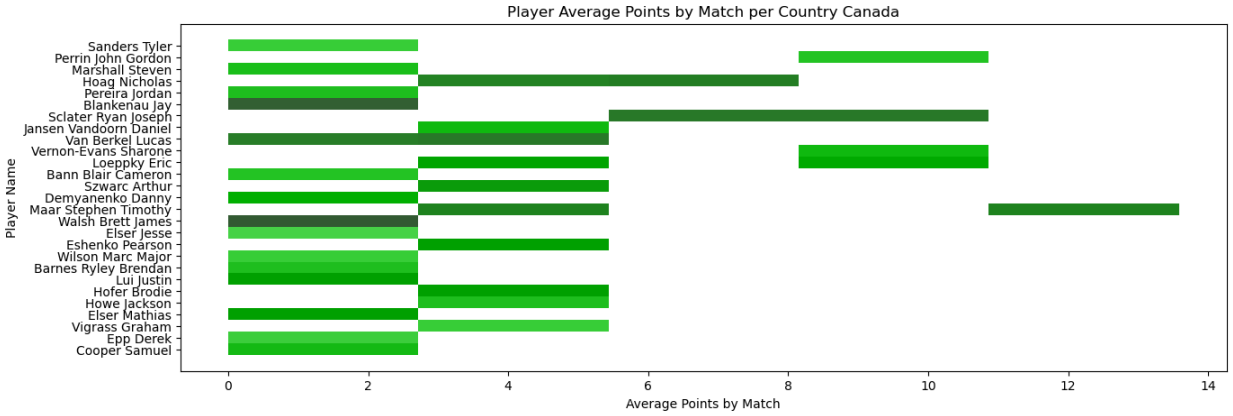
```
# Lets make our historgrams now
for country, data in grouped_countries:
    plt.figure(figsize=(15, 5))
    sns.histplot(data=data, x='Average by Match_x', y='Player Name',
bins=5, element="bars", color='green')
    plt.title(f'Player Average Points by Match per Country {country}')
    plt.xlabel('Average Points by Match')
    plt.ylabel('Player Name')
    plt.savefig('average points of players by country')
    plt.show()
```

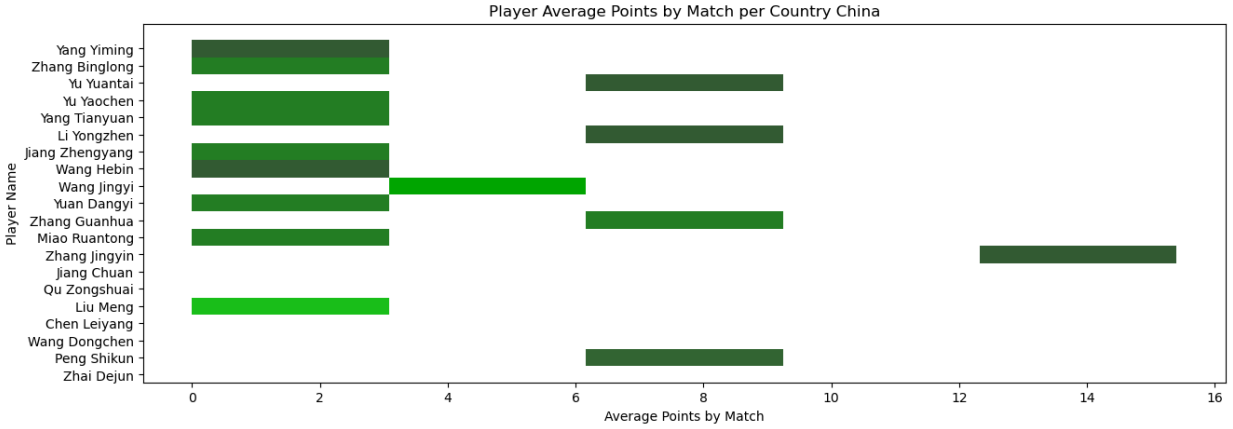


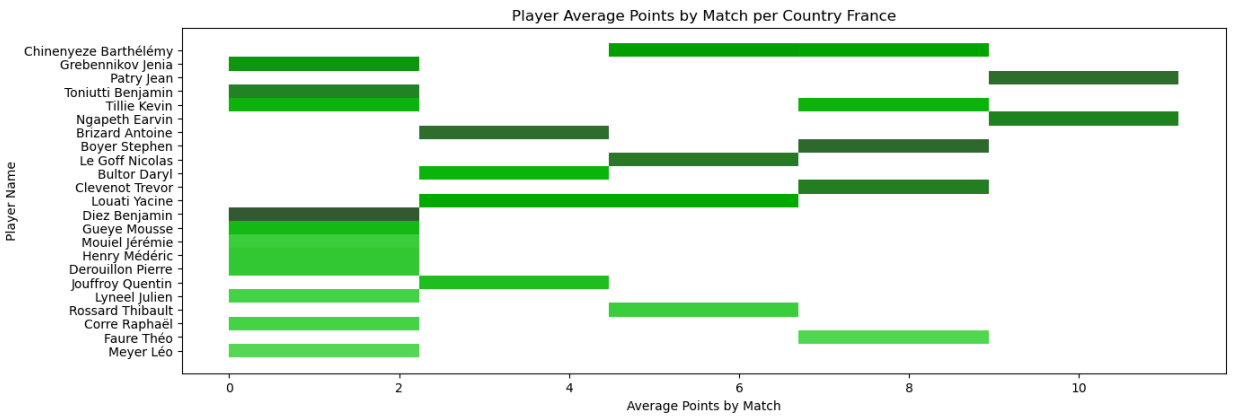


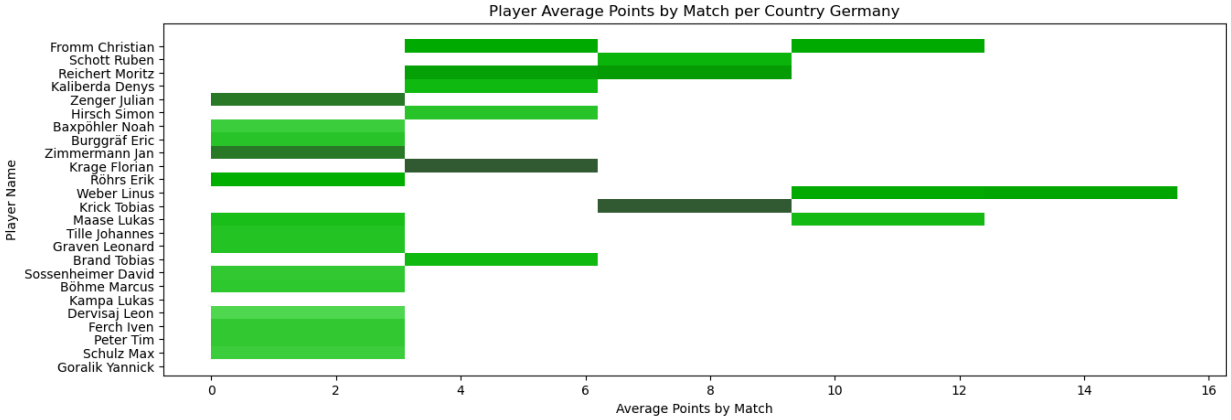


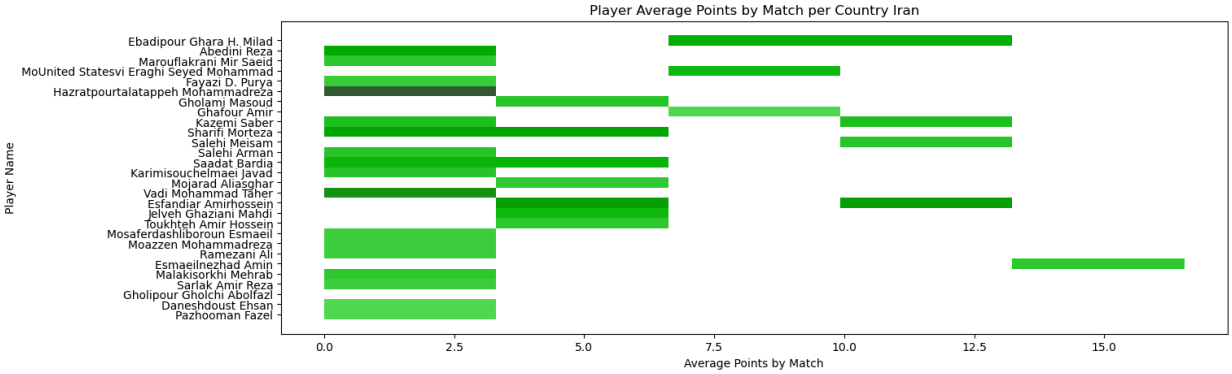

Player Average Points by Match per Country Bulgaria
Karyagin Denis
Chavdarov Stefan
Kolev Nikolay
Atanasov Martin
Seganov Georgi
Asparuhov Asparuh
Ivanov Vladislav
Ivanov Martin
Parapunov Radoslav
Ivanov Svetoslav
Gotsev Svetoslav
Dimitrov Dobromir
Petkov Iliya
Nikolov Aleksandar
Stankov Svetoslav
Grozdanov Aleks
Petrov Georgi
Sokolov Tsvetan
Kartev Nikolay
Lyutskanov Gordan
Dimitrov Dimitar
Stankov Vladimir
Bozhilov Martin
Player Name
0
2
4
6
8
10
12
14
Average Points by Match

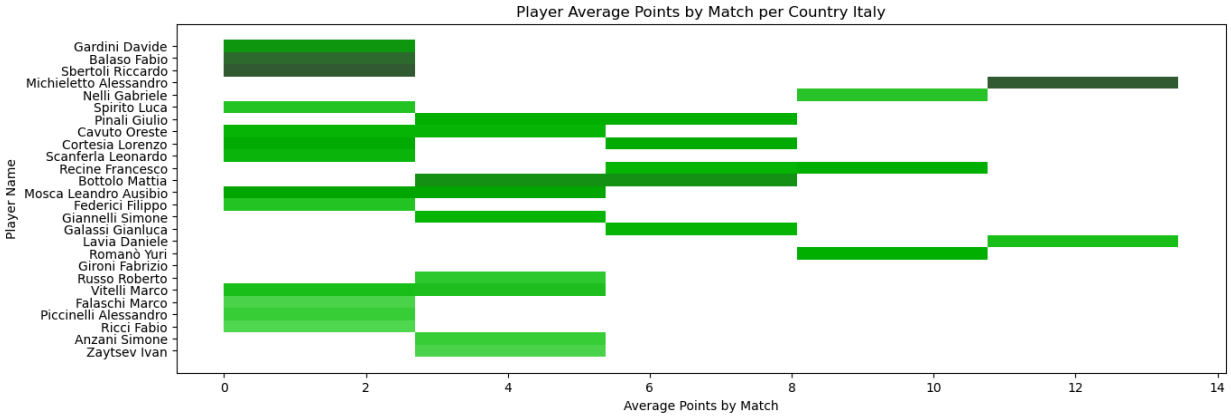

Player Average Points by Match per Country Canada
Sanders Tyler
Perrin John Gordon
Marshall Steven
Hoag Nicholas
Pereira Jordan
Blankenau Jay
Sclater Ryan Joseph
Jansen Vandoorn Daniel
Van Berkel Lucas
Vernon-Evans Sharone
Loeppky Eric
Bann Blair Cameron
Szwarc Arthur
Demyanenko Danny
Maar Stephen Timothy
Walsh Brett James
Elser Jesse
Eshenko Pearson
Wilson Marc Major
Barnes Ryley Brendan
Lui Justin
Hofer Brodie
Howe Jackson
Elser Mathias
Vigrass Graham
Epp Derek
Cooper Samuel
Player Name
0
2
4
6
8
10
12
14
Average Points by Match

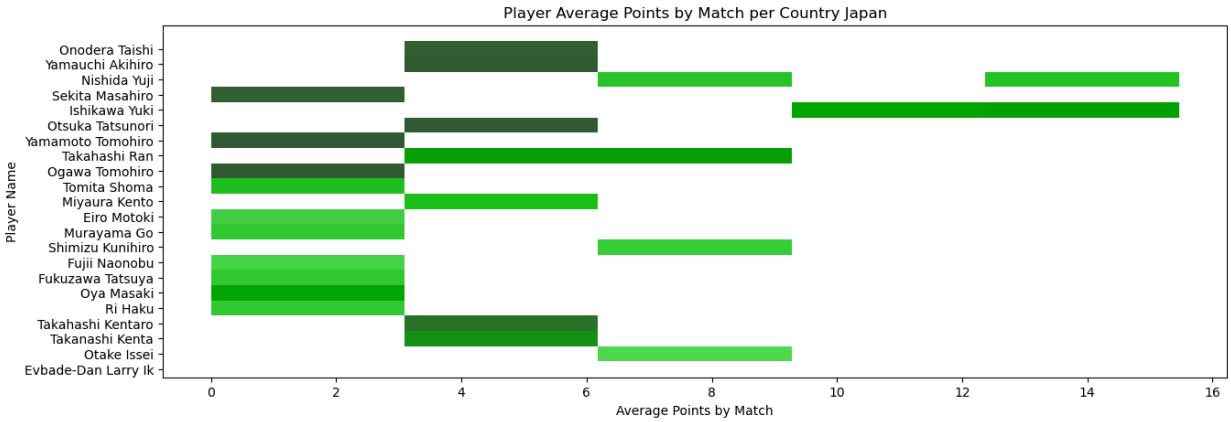

Player Average Points by Match per Country China
Yang Yiming
Zhang Binglong
Yu Yuantai
Yu Yaochen
Yang Tianyuan
Li Yongzhen
Jiang Zhengyang
Wang Hebin
Wang Jingyi
Yuan Dangyi
Zhang Guanhua
Miao Ruantong
Zhang Jingyin
Jiang Chuan
Qu Zongshuai
Liu Meng
Chen Leiyang
Wang Dongchen
Peng Shikun
Zhai Dejun
Player Name
0
2
4
6
8
10
12
14
16
Average Points by Match

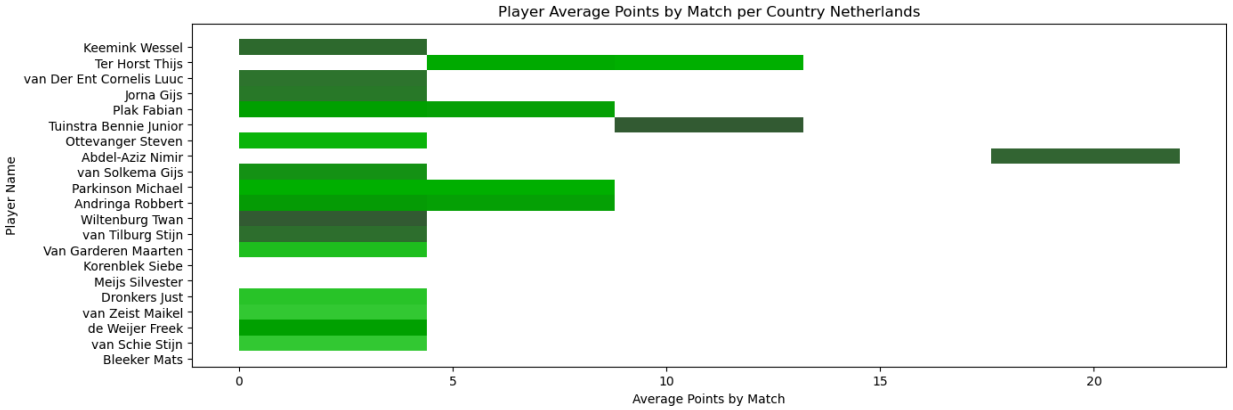

Player Average Points by Match per Country France
Chinenyeze Barthélémy
Grebennikov Jenia
Patry Jean
Toniutti Benjamin
Tillie Kevin
Ngapeth Earvin
Brizard Antoine
Boyer Stephen
Le Goff Nicolas
Bultor Daryl
Clevenot Trevor
Louati Yacine
Diez Benjamin
Gueye Mousse
Mouiel Jérémie
Henry Médéric
Derouillon Pierre
Jouffroy Quentin
Lyneel Julien
Rossard Thibault
Corre Raphaël
Faure Théo
Meyer Léo
Player Name
0
2
4
6
8
10
Average Points by Match

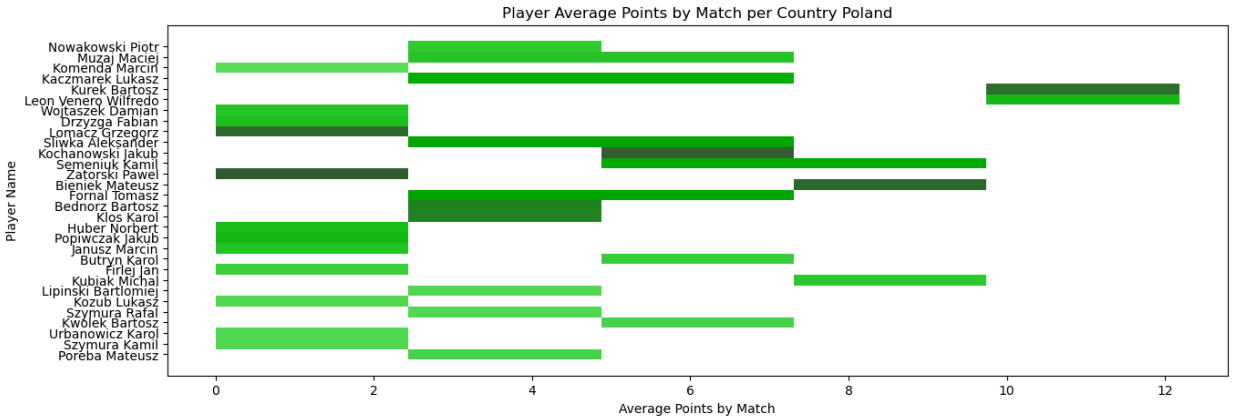

Player Average Points by Match per Country Germany
Fromm Christian
Schott Ruben
Reichert Moritz
Kaliberda Denys
Zenger Julian
Hirsch Simon
Baxpöhler Noah
Burggräf Eric
Zimmermann Jan
Krage Florian
Röhrs Erik
Weber Linus
Krick Tobias
Maase Lukas
Tille Johannes
Graven Leonard
Brand Tobias
Sossenheimer David
Böhme Marcus
Kampa Lukas
Dervisaj Leon
Ferch Iven
Peter Tim
Schulz Max
Goralik Yannick
Player Name
0
2
4
6
8
10
12
14
16
Average Points by Match

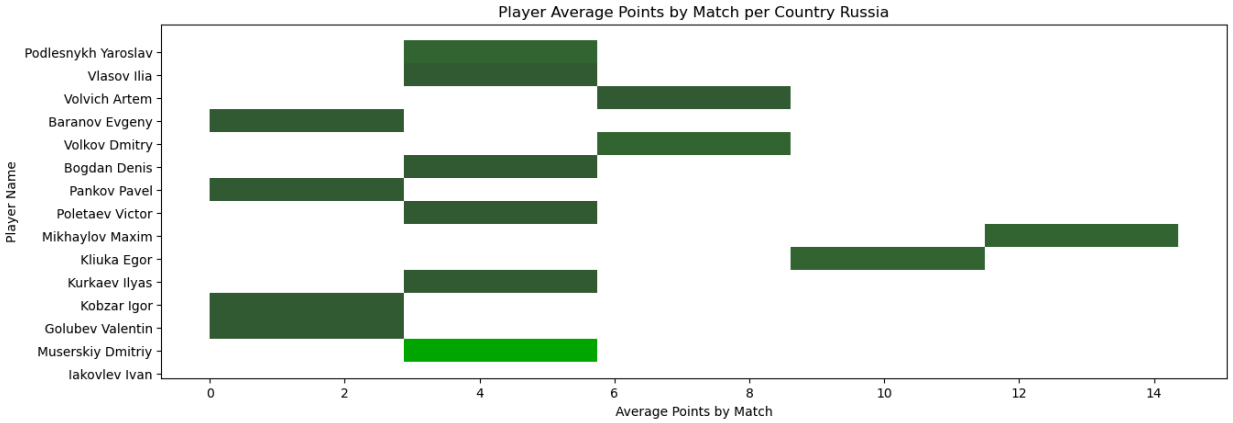

Player Average Points by Match per Country Iran
Ebadipour Ghara H. Milad
Abedini Reza
Marouflakrani Mir Saeid
MoUnited Statesvi Eraghi Seyed Mohammad
Fayazi D. Purya
Hazratpourtalatappeh Mohammadreza
Gholami Masoud
Ghafour Amir
Kazemi Saber
Sharifi Morteza
Salehi Meisam
Salehi Arman
Saadat Bardia
Karimisouchelmaei Javad
Mojarad Aliasghar
Vadi Mohammad Taher
Esfandiar Amirhossein
Jelveh Ghaziani Mahdi
Toukhteh Amir Hossein
Mosaferdashliboroun Esmaeil
Moazzen Mohammadreza
Ramezani Ali
Esmaeilnezhad Amin
Malakisorkhi Mehrab
Sarlak Amir Reza
Gholipour Gholchi Abolfazl
Daneshdoust Ehsan
Pazhooman Fazel
Player Name
0.0
2.5
5.0
7.5
10.0
12.5
15.0
Average Points by Match

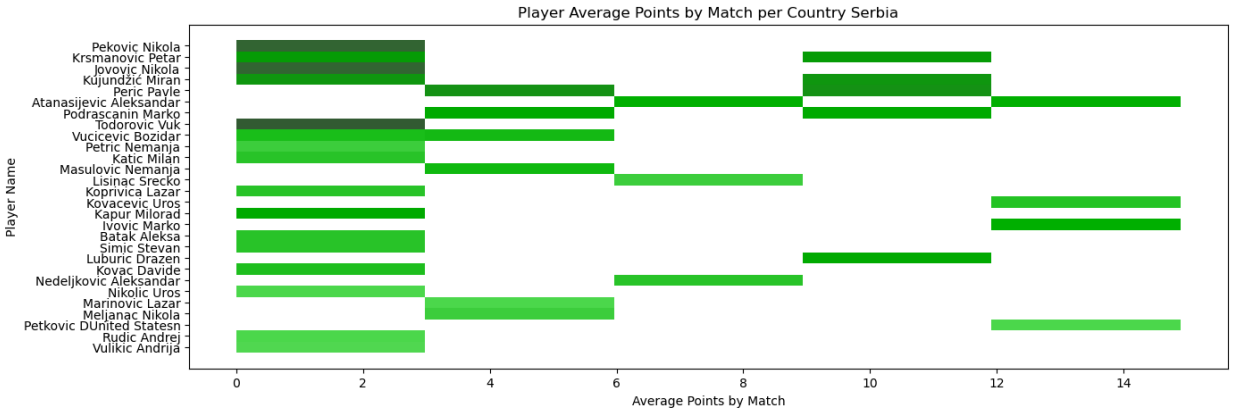

Player Average Points by Match per Country Italy
Gardini Davide
Balaso Fabio
Sbertoli Riccardo
Michieletto Alessandro
Nelli Gabriele
Spirito Luca
Pinali Giulio
Cavuto Oreste
Cortesia Lorenzo
Scanferla Leonardo
Recine Francesco
Bottolo Mattia
Mosca Leandro Ausibio
Federici Filippo
Giannelli Simone
Galassi Gianluca
Lavia Daniele
Romanò Yuri
Gironi Fabrizio
Russo Roberto
Vitelli Marco
Falaschi Marco
Piccinelli Alessandro
Ricci Fabio
Anzani Simone
Zaytsev Ivan
Player Name
0
2
4
6
8
10
12
14
Average Points by Match

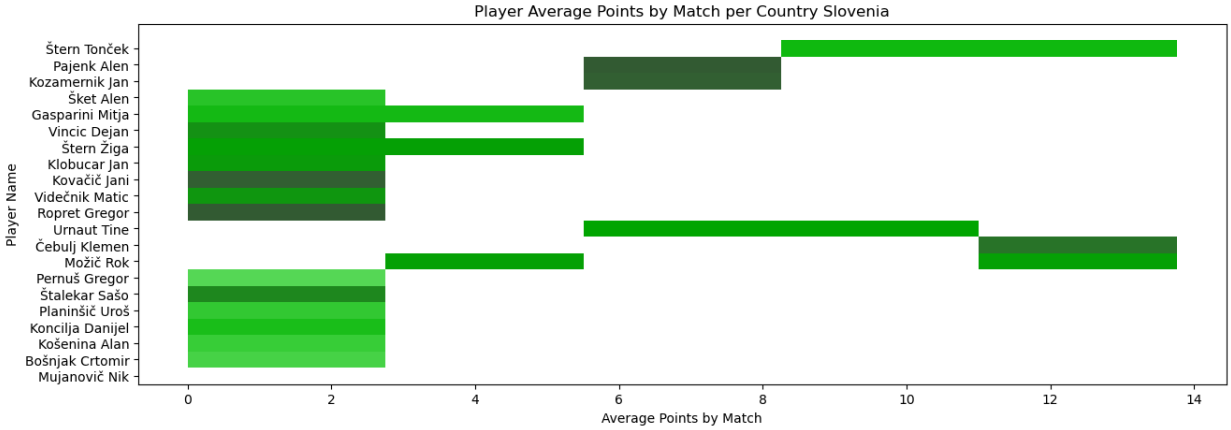

Player Average Points by Match per Country Japan
Onodera Taishi
Yamauchi Akihiro
Nishida Yuji
Sekita Masahiro
Ishikawa Yuki
Otsuka Tatsunori
Yamamoto Tomohiro
Takahashi Ran
Ogawa Tomohiro
Tomita Shoma
Miyaura Kento
Eiro Motoki
Murayama Go
Shimizu Kunihiro
Fujii Naonobu
Fukuzawa Tatsuya
Oya Masaki
Ri Haku
Takahashi Kentaro
Takanashi Kenta
Otake Issei
Evbade-Dan Larry Ik
Player Name
0
2
4
6
8
10
12
14
16
Average Points by Match

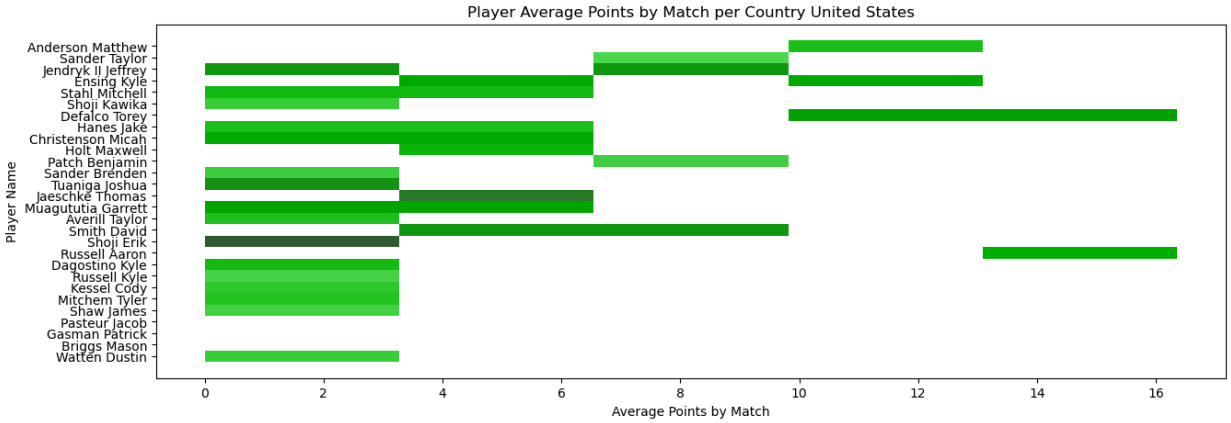

Player Average Points by Match per Country Netherlands
Keemink Wessel
Ter Horst Thijs
van Der Ent Cornelis Luuc
Jorna Gijs
Plak Fabian
Tuinstra Bennie Junior
Ottevanger Steven
Abdel-Aziz Nimir
van Solkema Gijs
Parkinson Michael
Andringa Robbert
Wiltenburg Twan
van Tilburg Stijn
Van Garderen Maarten
Korenblek Siebe
Meijs Silvester
Dronkers Just
van Zeist Maikel
de Weijer Freek
van Schie Stijn
Bleeker Mats
Player Name
0
5
10
15
20
Average Points by Match


Player Average Points by Match per Country Poland
Nowakowski Piotr
Muzaj Maciej
Komenda Marcin
Kaczmarek Lukasz
Kurek Bartosz
Leon Venero Wilfredo
Wojtaszek Damian
Drzyzga Fabian
Lomacz Grzegorz
Sliwka Aleksander
Kochanowski Jakub
Semeniuk Kamil
Zatorski Pawel
Bieniek Mateusz
Fornal Tomasz
Bednorz Bartosz
Klos Karol
Huber Norbert
Popiwczak Jakub
Janusz Marcin
Butryn Karol
Firlej Jan
Kubiak Michal
Lipinski Bartlomiej
Kozub Lukasz
Szymura Rafal
Kwolek Bartosz
Urbanowicz Karol
Szymura Kamil
Poreba Mateusz
Player Name
0
2
4
6
8
10
12
Average Points by Match


Player Average Points by Match per Country Russia
Podlesnykh Yaroslav
Vlasov Ilia
Volvich Artem
Baranov Evgeny
Volkov Dmitry
Bogdan Denis
Pankov Pavel
Poletaev Victor
Mikhaylov Maxim
Kliuka Egor
Kurkaev Ilyas
Kobzar Igor
Golubev Valentin
Muserskiy Dmitriy
Iakovlev Ivan
Player Name
0
2
4
6
8
10
12
14
Average Points by Match


Player Average Points by Match per Country Serbia
Pekovic Nikola
Krsmanovic Petar
Jovovic Nikola
Kujundzic Miran
Peric Payle
Atanasijevic Aleksandar
Podrascanin Marko
Todorovic Vuk
Vucicevic Bozidar
Petric Nemanja
Katic Milan
Masulovic Nemanja
Lisinac Srecko
Koprivica Lazar
Kovacevic Uros
Kapur Milorad
Ivovic Marko
Batak Aleksa
Simic Stevan
Luburic Drazen
Kovac Davide
Nedeljkovic Aleksandar
Nikolic Uros
Marinovic Lazar
Meljanac Nikola
Petkovic DUnited Statesn
Rudic Andrej
Vulikic Andrija
Player Name
0
2
4
6
8
10
12
14
Average Points by Match



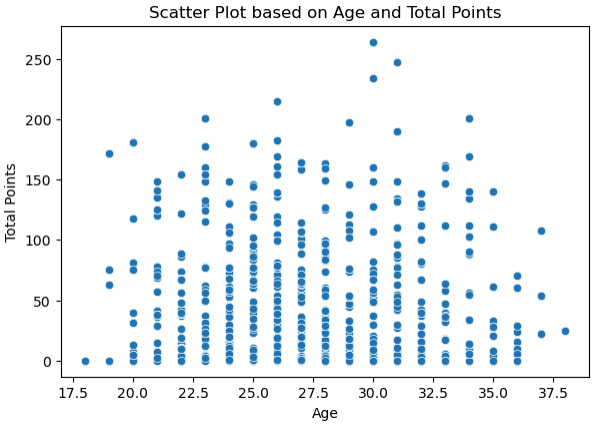


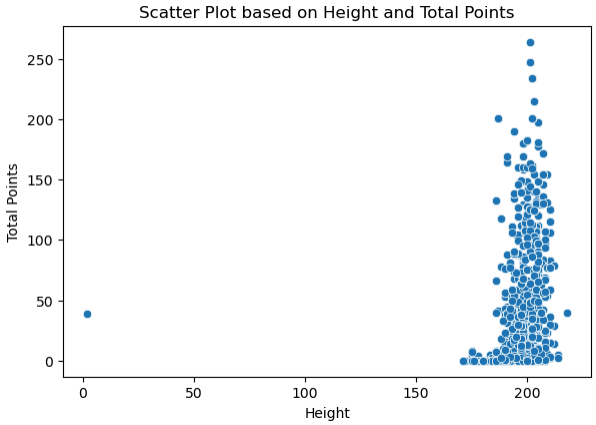
These graphs present the data as a histogram plotting out the average points scored by each player for their respective country. It gives a sense of ranking for each player per country with those who on average score more being the country's top-level player.

Continuing the trend of exploratory analysis, the next items that could show importance are the number of total points scored in correlation with a player's age or height.

In [144…

```
#beging by pairing off groups of variabels we want to test
pairs = [('Age', 'Total Points'), ('Height', 'Total Points')]

#create a scatter plot map to show the variables in comparison
plt.figure(figsize=(15, 10))
for i, pairs in enumerate(pairs, 1):
    plt.subplot(2, 2, i)
    sns.scatterplot(data= df_cleaned_up, x=pairs[0], y= pairs[1])
    plt.title(f'Scatter Plot based on {pairs[0]} and {pairs[1]}')
    plt.xlabel(pairs[0])
    plt.ylabel(pairs[1])
    plt.savefig('scatter plot age and height by points')
```





As expected, most of the points scored are from taller individuals in their late twenties and early thirties. IN theory this should directly correlate to match results for teams with taller players in their late twenties being more successful per match.

Now to begin working machine learning algorithms into the data frames to predict future match results!

Methods and Machine Learning Algorithms

The main goal of this assignment is to create a model with the ability to predict future outcomes of volleyball matches. To create such a model, the use of various machine learning algorithms is essential. In specific the models used in this code are: Random Forest, Decision Tree, and Logistic Regression. With Random Forest and Decision tree algorithms the goal is to classify the importance of each predictor variable being stored in the model. With Decision Tree processing each variable is analyzed as the tree continues to grow down and thus provides a simplified and easy to read anser on key importance of each variable. Unfortunately, decision trees are commonly found to overfit data sets with many variables thus calling on the need for a Random Forest Algorithm as well. With Random Forests the data set is run based on multiple decision trees (amount specific is to be determined in the code) creating results based around the mode of every decision tree created. This allows the algorithm to shift around the overfitting results thus making it a better fit for the data sets in this report.

On top of these two, logistic regression is key to determining our results. While it is just a simple binary classification tool, the ability to have a linear association between the input and target variables provides an easy to interpret result of 'Winner' or 'Loser'.

For this to work some categorical variables need to be shifted to numerical. https://stackoverflow.com/questions/39494001/using-labelencoder-for-a-series-in-scikitlearn This is the posting I found to help me understand how to use label encoder to fit our model. On top of this code from lab 3 Random Forests https://moodle.essex.ac.uk/mod/resource/view.php?id=923148 was used to create the below code.

In [145…

```
#import any missing libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier

#drop the columns we wont need to worry about
X = df_cleaned_up.drop(['Winner', 'Player Name', 'SCORES','SCORES1',
'SCORES2', 'SCORES_TeamA','SCORES_TeamB',], axis=1) #predictors
y = df_cleaned_up['Winner'] #create a predcition variable which will be
the winner of our match
encoder = LabelEncoder() #encodes categorical varibales as numericals
X_en = X.copy() #copies our predictor variables in the X data set above
#encode each variable below into a numeric
X_en['Country_Name'] = encoder.fit_transform(X['Country_Name'])
X_en['TEAM1'] = encoder.transform(X['TEAM1'])
X_en['TEAM2'] = encoder.transform(X['TEAM2'])
X_en['TeamA'] = encoder.transform(X['TeamA'])
X_en['TeamB'] = encoder.transform(X['TeamB'])
#replace all ''-'' values of X_en as we cannot run the code with NaN's
X_en = X_en.replace('-',0)
#drop all NaN's still present
X_en = X_en.dropna()
#checks that only lines present in X_en are being measured
y = y[X_en.index]
#store our X_en variables as feature names for later code
feature_names = X_en.columns.tolist()



#create a training and test set for the models w/ test sixe of 20%
X_train, X_test, y_train, y_test = train_test_split(X_en, y,
test_size=0.2, random_state=13)
```

```
from sklearn.metrics import accuracy_score
clf = RandomForestClassifier() #create our random forest classification
clf.fit(X_train,y_train) #fit it on our training set
y_pred_F = clf.predict(X_test) #create a prediciton variable from our X
test set
DT = accuracy_score(y_test,y_pred_F) #dtermine the accuracy of the set and
print it below
print(f'The accuracy from Random Forest is : {DT}')

#this portion of the code I used from the lob as I found it difficult to
vary it from online sources
#create a random foreset labeled 'forest' with established parameters
forest =
RandomForestClassifier(n_estimators=100,bootstrap=True,max_features='sqrt',
forest.fit(X_train,y_train) #fit the variable to the training sets
y_train_pred = forest.predict(X_train) #create a prediction variable from
the training set
print("Accuracy on training set:",round(metrics.accuracy_score(y_train,
y_train_pred),2)) #print results
y_test_pred = forest.predict(X_test) #do the same as above but with the
test set
print("Accuracy on test set:",round(metrics.accuracy_score(y_test,
y_test_pred),2)) #pirint results
print("OOB score:",round(forest.oob_score_,2)) #print out of bag score

#logsitic REgression

#this code was influenced by code found here
https://stackoverflow.com/questions/24255723/sklearn-logistic-regression-
important-features
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

model = LogisticRegression(solver = 'liblinear') #create the logisitic
regression model
model.fit(X_train,y_train) #fit it to our training sets
y_pred = model.predict(X_test) #create a predicitve variable based on the
test set
```

```
LR = accuracy_score(y_test,y_pred) #create a label an accuracy score for
our logistic regression

print(f'Accuracy of the regression: {LR}') #print the score
```

```
The accuracy from Random Forest is : 0.7529305680793508
Accuracy on training set: 0.88
Accuracy on test set: 0.75
OOB score: 0.75
Accuracy of the regression: 0.6639615269011121
```

Based on the outcomes from the tests performed above, it is apparent that the Random Forest performs better in comparison to the Logistic Regression model, as indicated by the 75% accuracy scored achieved.

Though the Random Forest Algorithm does indeed prove to be a better test, there are notable discrepancies between the training and test sets the model runs on. While the model runs with 88% accuracy on the training set, it dwindles almost 13% on the test set only resulting in an accuracy score of 75%. This issue could be due to potential overfitting, where the model begins to tailor itself to the training data and cannot generalize to the unseen data.

In [146…

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
accuracy = accuracy_score(y_test, y_pred) #create a variable 'accuracy'
which compares the test score to the prediciton scores
print("Accuracy:", accuracy)

# #print a calassification report for the above
print("Classification Report:")
print(classification_report(y_test, y_pred))

#print a confusion matirx for the above
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.6639615269011121
Classification Report:
              precision    recall  f1-score   support

           1       0.67      0.70      0.69      1748
           2       0.65      0.63      0.64      1579

    accuracy                           0.66      3327
   macro avg       0.66      0.66      0.66      3327
weighted avg       0.66      0.66      0.66      3327

Confusion Matrix:
[[1219  529]
 [ 589  990]]
```

The classification report above analyzes the data based on classes 1 and 2, broken down by precision, recall, and f-1 scores. Precision in this report is measuring the ratios of correctly predicted outcomes in comparison to the total outcomes predicted to be positive per class. Recall on the other hand predicts measures the ratio of correctly predicted instances compared to actual instances per class. The f-1 score showcases the mean values between precision and recall, giving a more balanced outcome for the model's performance. The overall report shows a bit of variance per class.

The model achieved a precision of 67% for class 1 and 65% for class 2, indicating that it correctly identified 67% of class 1 instances and 65% of class 2 instances among all instances predicted as positive. The recall values of 70% for class 1 and 63% for class 2 indicate that the model correctly identified 70% of class 1 instances and 63% of class 2 instances among all actual instances of each class. The f-1 scores present a more reasonable balance between the two metrics. Giving 69% for class 1 and 64% for class 2.

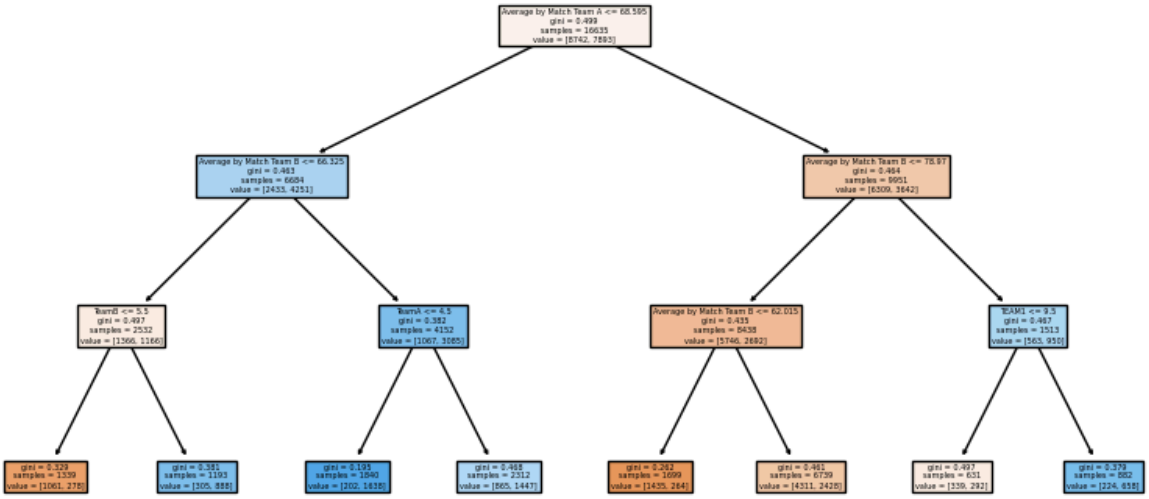
The confusion Matrix provides a visual into the model's performance by showing the amount of tried and false positives determined per class. For this specific model 1219 instances were classified correctly for class 1 and 990 for class 2. Unfortunately, it did still misclassify 529 instance for class 1 and 589 for class 2. Overall the model performs well, but there is definite need for improvement.

In [147…

```
from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics

#create a decision tree and choose tree depth with random state to match
the above
model_tree = tree.DecisionTreeClassifier(max_depth = 3, random_state=13)
model_tree.fit(X_train,y_train) #fit our model on trainin sets for the
predictor and target varaibles
y_pred=model_tree.predict(X_test) #create another predcition based on
```

```
decision tree

print('Accuracy of the test set in the decision tree is',
np.round(metrics.accuracy_score(y_test, y_pred),2))
# Fit the decision tree model for our X/y variables
model_tree.fit(X_en, y)

#feature importance based on model above
feature_importance = model_tree.feature_importances_

feature_importance_df = pd.DataFrame({'Features': X_train.columns,
'Importance': feature_importance}) #create a new data fram
sorted_feature_importance_df =
feature_importance_df.sort_values(by='Importance', ascending=False) #sort
the data frame by feture imporatance
print(sorted_feature_importance_df)
# Plot the decision tree
fig = plt.figure(figsize=(10, 5))
_ = plot_tree(model_tree, feature_names=feature_names, filled=True)
plt.savefig('feature decisiotn tree')
plt.show()
```

```
Accuracy of the test set in the decision tree is 0.72
                  Features  Importance
17  Average by Match Team B    0.346765
16  Average by Match Team A    0.331522
12                    TeamB    0.206707
11                    TeamA    0.081435
18                    TEAM1    0.033571
0                    Year_x    0.000000
1              Country_Name    0.000000
15              Dig_Success    0.000000
14     Reception_Successful    0.000000
13            Set_Successes    0.000000
10                   Year_y    0.000000
9          Serve Avg Points    0.000000
8          Block Avg Points    0.000000
7         Attack Avg Points    0.000000
6                Efficiency    0.000000
5        Average by Match_x    0.000000
4              Total Points    0.000000
3                    Height    0.000000
2                       Age    0.000000
19                    TEAM2    0.000000
```



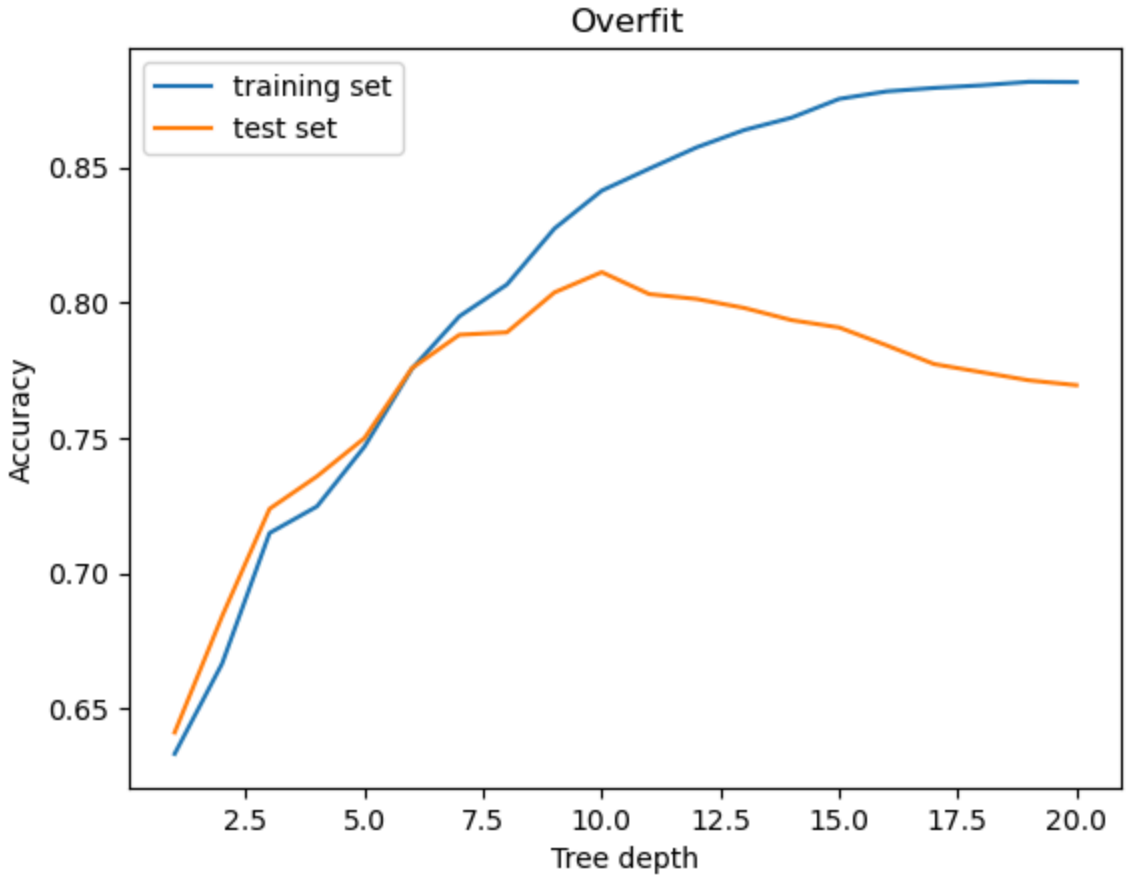
The decision tree model, while underperforming compared to the random forest, did indeed provide a high accuracy score of 72% as well. Closer inspection of the model will show the rankings of feature importance, having the two most significant features being 'Average by Team A' and 'Average by Team B' with scores of 34.68% and 33.15% respectively. Following behind these two is 'Team B', which makes up 20.67% of the predictive capabilities. There then becomes a noticeable decrease in the next features with 'Team A' and 'Team 1' only making up 8.14% and 3.36% respectively.

Oddly enough, the rest of the variables classified as predictors were deemed to be insignificant. Each variable accounted for 0% importance of the score. This is more than likely caused by the average by match categories already being influenced by the individual variables that proved to be non significant.

In [148…

```
#lets create a graph to show accuracy as a result of our max depth as
defined above (pulled from the lab for decisions trees)
max_depth_vals = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
accuracytrain_list=[]
accuracytest_list=[]
for i in range(0,len(max_depth_vals)):
    decision_tree =
tree.DecisionTreeClassifier(max_depth=max_depth_vals[i], random_state=i)
    decision_tree.fit(X_train,y_train)
    # Training
    ytrain_pred = decision_tree.predict(X_train)
    accuracy_train = metrics.accuracy_score(y_train, ytrain_pred)
    accuracytrain_list.append(accuracy_train)
    # Test
```

```
    ytest_pred = decision_tree.predict(X_test)
    accuracy_test = metrics.accuracy_score(y_test, ytest_pred)
    accuracytest_list.append(accuracy_test)

# Plot our graph with accuracy as our frequency and y axis and max depth
as our x
fig = plt.figure()
ax = plt.axes()
line1, = ax.plot(max_depth_vals,accuracytrain_list,label='training set')
line2, = ax.plot(max_depth_vals,accuracytest_list,label='test set')
plt.legend(handles=[line1, line2])
plt.title("Overfit")
plt.xlabel("Tree depth")
plt.ylabel("Accuracy")
plt.savefig('over fit model')
plt.show()
```



As the maximum depth of the decision tree increases an improvement in accuracy can be seen with both the training and test sets. This is visible until reaching a tree depth of 10, where a noticeable decline in test set accuracy occurs. Overfitting is the reason for this, as the models begin to memorize patterns of the training data instead of continuing to understand patterns.

Thus, when applied to unseen data, the model begins to perform worse resulting in lower accuracy scores.

In [149…

```
from sklearn.model_selection import cross_val_score, KFold
import math

#define the model/decision tree
model = tree.DecisionTreeClassifier(max_depth=5, random_state=13)

#create k fold object
maxDepth = 5
k = 5  # Number of folds to test on
kf = KFold(n_splits=k, shuffle=True, random_state=13)

# get our kf scores
KF_scores = cross_val_score(model, X_en, y, cv=kf)
#show our results
print("Cross-validated scores:", KF_scores)
print("Mean accuracy:", KF_scores.mean())

# lets find the average of the accuracy
average = sum(KF_scores)/len(KF_scores)
standard_deviation = np.std(KF_scores, ddof=1)
print('Average performance for a tree depth
of',maxDepth,'is:',np.round(average*100,1),'+/-
',np.round(standard_deviation*100,1),'%')
```

```
Cross-validated scores: [0.74992486 0.73760144 0.75503457 0.74391344 0.73429516]
Mean accuracy: 0.7441538923955515
Average performance for a tree depth of 5 is: 74.4 +/- 0.9 %
```

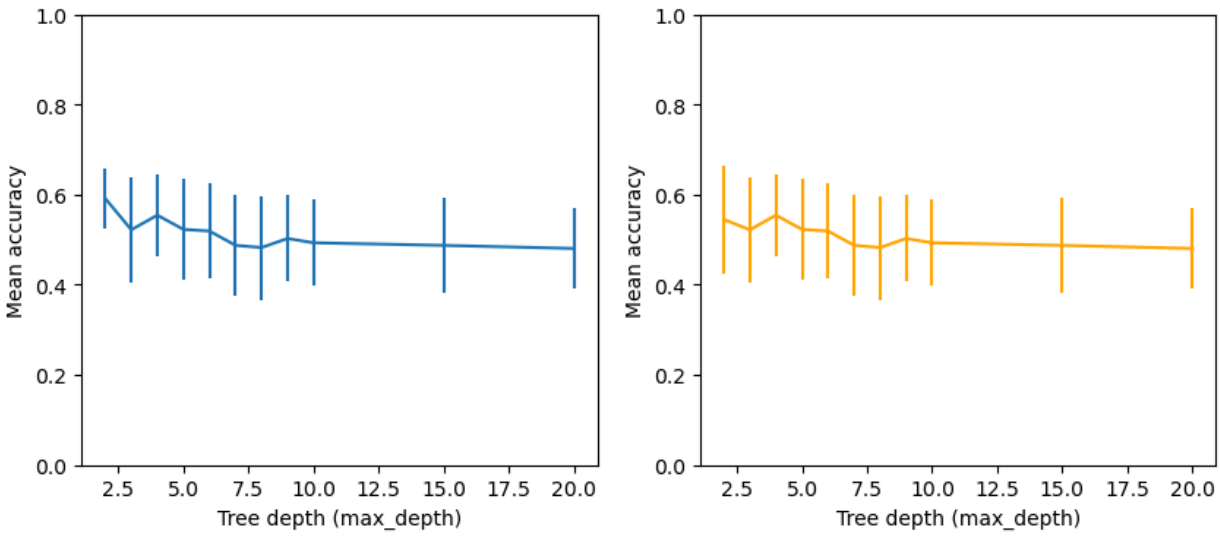
The cross validation results above can be interpretted as seen below. The first fold of the model resulted in a score of 74.92% The second fold of the model resulted in a score of 73.76% The third fold of the model resulted in a score of 75.50% The fourth fold of the model resulted in a score of 74.39% The fifth fold of the model resulted in a score of 73.43% The mean accuracy of the model was 74.42% The numbers above indicate that, on average, the model will work 74.42% of the time with a standard deviation for error of 0.9% in either direction when tested on a 5 fold measure. This proves the model is quite accurate and consistent with its results!

In [150…

```
#this code was also taken from lab 3 as I could not come up with another
way to create the cahrts below
```

```
from sklearn.model_selection import cross_val_score
import math

max_depth_vals = [2,3,4,5,6,7,8,9,10,15,20]

k=5 #store number of k fodls to 5
mean_accuracy_cv_k5 = [] #stores the mean accuracy of a blank list
sd_cv_k5 = [] #stores standard devaition of a blnak list

#for loop to fill innthe blank lsits from above
for i in range(0,len(max_depth_vals)): #pulls each value of max depth and
runs the decision tree classifier below
    decision_tree =
tree.DecisionTreeClassifier(max_depth=max_depth_vals[i], random_state=i)
    cv_scores = cross_val_score(decision_tree, X_en, y, cv=k) #calculates
accuracy score
    avg = sum(cv_scores)/len(cv_scores) #calcultaes the average
    sd = math.sqrt(sum((cv_scores-avg)**2)/(len(cv_scores)-1)) #calculates
the standard deviation
    mean_accuracy_cv_k5.append(avg) #stores the values into the blank list
above
    sd_cv_k5.append(sd)#stores the values into the blank list above


    #perform the same funstinos above, but this time with a k fold score
of 10
    k=10
mean_accuracy_cv_k10 = []
sd_cv_k10 = []

for i in range(0,len(max_depth_vals)):#pulls each value of max depth and
runs the decision tree classifier below
    decision_tree =
tree.DecisionTreeClassifier(max_depth=max_depth_vals[i], random_state=i)
    cv_scores = cross_val_score(decision_tree, X_en, y, cv=k)#calculates
accuracy score
    avg = sum(cv_scores)/len(cv_scores)#calcultaes the average
    sd = math.sqrt(sum((cv_scores-avg)**2)/(len(cv_scores)-1))#calculates
the standard deviation
```

```
    mean_accuracy_cv_k10.append(avg)#stores the values into the blank list
above
    sd_cv_k10.append(sd)#stores the values into the blank list above

# Make the graphs
import matplotlib.pyplot as plt
#create a figure with two subplots from above
fig, ax = plt.subplots(1,2, figsize=(10, 4))

#create and error bar and plot the points for max depths and accuracy of
kfold score of 5
ax[0].errorbar(max_depth_vals,mean_accuracy_cv_k5,yerr=sd_cv_k5)
ax[0].set_xlabel("Tree depth (max_depth)")#label x aixs
ax[0].set_ylabel("Mean accuracy") #label y axis
ax[0].set_ylim([0,1.0]) #set limits for the y axis from 0-10

#create and error bar and plot the points for max depths and accuracy of
kfold score of 10
ax[1].errorbar(max_depth_vals,mean_accuracy_cv_k10,yerr=sd_cv_k10,
color="orange")
ax[1].set_xlabel("Tree depth (max_depth)") #label x aixs
ax[1].set_ylabel("Mean accuracy")#label y axis
ax[1].set_ylim([0,1.0]) #set limits for the y axis from 0-10
plt.savefig('tree depth by accuracy')
plt.show()
```



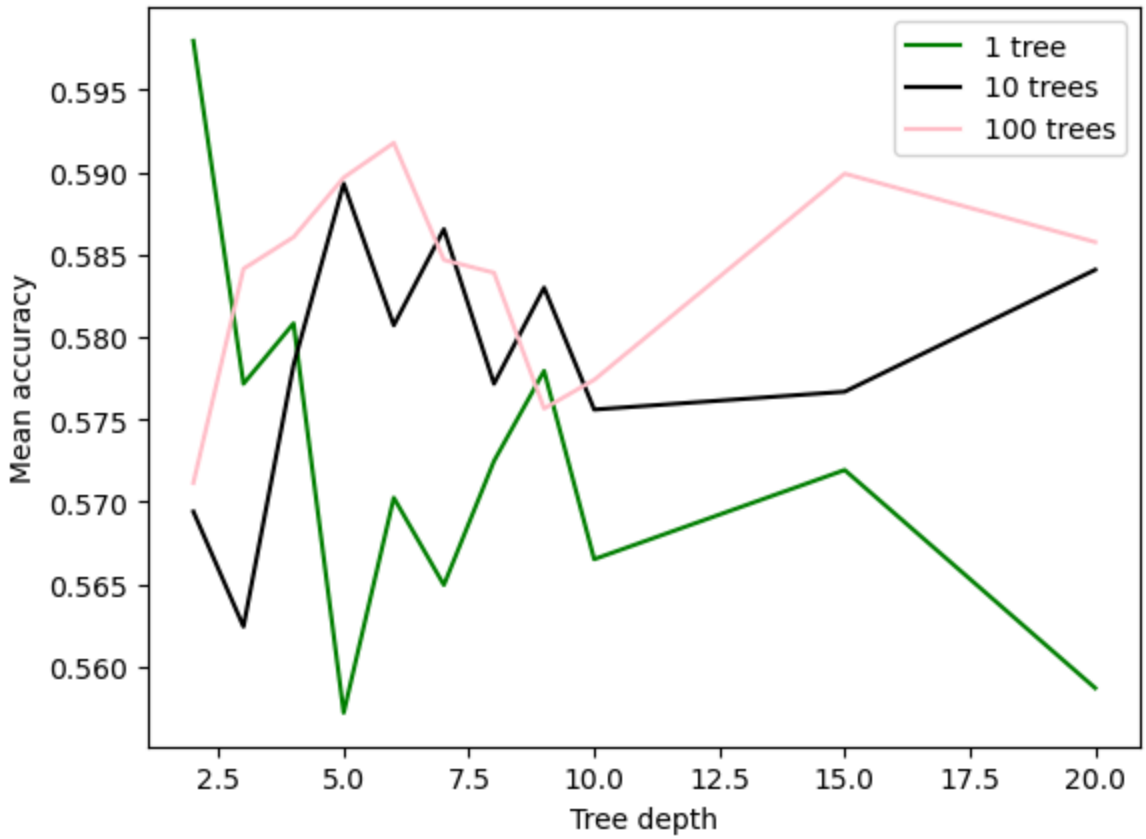
The peak accuracy is achieved when the model reaches a tree depth of around 2-5, suggesting that less complexity in the model will lead to a better performance. When evaluating on a five -

fold cross-validation score, a greater consistency is noticeable when compared to the ten-fold cross-validation.

In [151…

```
#this code was designed to mirror the code from lab 3, but wihtout
copying. the overall method is virtually the same, but the plotting was
chanegd
# Define parameters
max_depth_vals = [2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20]
n_estimators_vals = [1, 10, 100]
k = 5

# create blank lists for our functions below on both mean of teh accuracy
and standard deviation
mean_accuracy_store = []
sd_accuracy_store = []

# write a function to test cross validation scores and input them into our
blank strings for each estimator 1, 10, 100
for n_estimators in n_estimators_vals:
    mean_accuracy_cv = []
    sd_cv = []

    # create a functino wo test cross calidation scores on each tree depth
level above and store them in the blank lists
    for max_depth in max_depth_vals:
        forest = RandomForestClassifier(n_estimators=n_estimators,
bootstrap=True, max_features="sqrt", criterion='gini',
max_depth=max_depth, random_state=3, oob_score=False)
        cv_scores = cross_val_score(forest, X_en, y, cv=k)
        mean_accuracy_cv.append(np.mean(cv_scores))
        sd_cv.append(np.std(cv_scores))

    # input results into the blank lists
    mean_accuracy_store.append(mean_accuracy_cv)
    sd_accuracy_store.append(sd_cv)

# Plot
fig, ax = plt.subplots()
colors = ['green', 'black', 'pink']  # Colors for lines (chosen to help my
```

```
colorblind eyes see the differences)
labels = ['1 tree', '10 trees', '100 trees']  # Labels for lines

for i, (mean_accuracy, color, label) in enumerate(zip(mean_accuracy_store,
colors, labels)):#pairs the elements together
    ax.plot(max_depth_vals, mean_accuracy, color=color, label=label)

plt.legend()
plt.xlabel("Tree depth")
plt.ylabel("Mean accuracy")
plt.savefig('random forest tree depth by accuracy')
plt.show()
```



The decision tree model above reveals a trend in accuracy because of tree depth. For the single tree model, a sizeable decrease in accuracy occurs from depths 0-5, followed by an ever so slight improvement from 5-10. Beyond a depth of 10 the model begins another decline in accuracy.

On the contrary, the 10 and 100 tree models show an increase in accuracy from 0-6 depth levels. They do show a small decrease when approaching a depth level of 10, but beyond this level they continue their trend upwards in respect to accuracy.

These results could be caused by the tendency of models to overfit, highlighting the importance of tree depths on model accuracy.

Results

Model Performce

Random Forest:

The Random Forest model provided the best results for the data set with a score of 88% on the training set. The accuracy on the test set did fall a bit lower coming in around 75%, but overall this results in quite succesfful tests. The Out of Bag Score was 75% which is also considered a success. Overall the model performed very well with the data set provided. OOB score: 0.75

Logistic Regression

This model tested a bit lower than the other two models run coming in at 66.39% which is still quite accurate on average.

Decision Tree

The decision tree produced results on the Training set with an accuracy score of 72%. The importance of the features is broken down as follows. The top five Releveant Features:

```
    1. Average by Match Team B: 34.67%
    Average by Match Team A: 33.15%
    Team: 20.67%
    TeamA: 8.14%
    TEAM1: 3.36%
 The other predictor variables provided no importance to training
the model.
```

Cross Validation

Tree depths ranging from 2-20 were used to evaluate on k scores of 5 and 10. For both scores the accuracy of the model was found to be around 74% with a standard devaiation of 0.9% in

either direction.

Classification Report and Confusion Matrix

The overall accuracy of the modesls was 66.4% provided by the precision, recall, f-1, and support instances for each class (1/2).

The confusion matrix gave a result of true positives (1219), false positives (529), true negatives (589), and false ngatives (990).

Conclusion

In conclusion, the models created in this project performed well and provide promising results for the predictive nature of volleyball match outcomes based on historical data. These models can be utilized by both teams and individuals looking for insights into the possible performance of a team against a specified opponent.

Adjustments to the model could be looked at for future renditions of this model. For example, having a larger data and utilizing the full extent of the features available for more than two seasons. Fine tuning of the model around more specialized features could also allow teams to tailor their game play toward the strengths the model produces.

With a deeper understanding of the Python coding language and the data sets provided a much more exploratory and accurate model could be developed. Optimizing key factors like predictor variables and model evaluation techniques could result in higher accuracy scores.

Overall, the model exceeded expectations despite the limited knowledge and capabilities of the data provided. Continued refinement of the model is necessary before rolling out to a public setting, but as it stands the models performance is capable of basic predictions for the analysis of professional volleyball.

References

1.Volleyball Nations League 2021-2022 Data [Internet]. www.kaggle.com. [cited 2024 Apr 11]. Available from: https://www.kaggle.com/datasets/sumbowdy/volleyball-nations-league-2021-2022-data

2.EDA_VNL_Viz_SQL [Internet]. kaggle.com. [cited 2024 Apr 11]. Available from: https://www.kaggle.com/code/zakirpasha/eda-vnl-viz-sql

3.Using LabelEncoder for a series in scikitlearn [Internet]. Stack Overflow. [cited 2024 Apr 11]. Available from: https://stackoverflow.com/questions/39494001/using-labelencoder-for-a-series-

in-scikitlearn

4.Gutiérrez-Roig MGR, Voigt L. Decision_Trees_Titanic_Solutions [Internet]. Moodle. Available from: https://moodle.essex.ac.uk/course/view.php?id=15076§ion=10

In [ ]: