Final Project:

NBA Analytics

Web App

Algorithm and Programming

—

Francesco Emmanuel Setiawan
2602209620 | L1AC

Table of Contents

Overview

1. Background

For the Algorithm and Programming Computer Science course, students are tasked to design a final project program using the skills in Python that have been taught over the course of the odd 2022 semester. This report is a documentation of the development of my program which includes — its research, design process, final result, as well as my own reflection on the lessons that I've learned over the course of making this project.

Choosing what to do for this project has been quite a daunting challenge for me, as making sure that it would be difficult yet enjoyable and doable to finish before the deadline was a priority. I wanted to make something that would allow me to learn something new and acquire new skills by the time I finished the project. And so, I choose to make an NBA analytics web app. This was a very fun project as I am a huge NBA fan, but also very challenging and educational as I had no experience with data science and machine learning on Python.

2. The function of the program

In the world of basketball where everyone goes crazy over dunks, 3 pointers and ankle breakers, what has always fascinated me were the statistics. Statistics opens up a door to a different world and what this project will aim to do is to allow people to see basketball from a whole different lens like never before.

Thus by taking datasets that can be found on www.kaggle.com, and web scraping www.basketball-reference.com and www.landofbasketball.com, I created an NBA analytics web app where you can see up to date NBA player's statistics, some fun data visualization of NBA statistics, a forecast of every team's win/loss percentage for the next one year, as well as the probability of a player to win MVP and be an All-Star, with the use of machine learning models.

Program Specifications

1. Libraries

When creating this program, I used the following libraries:

A. Streamlit

In the process of making this project, I researched many mediums that would allow me to easily make my program interactive. I looked at frameworks such as django and dash, but ultimately decided to choose streamlit as it allows for the rapid build and sharing of beautiful machine learning and data science web apps, and is a Python-based library specifically designed for machine learning engineers. It also does not require any prior HTML skills like django does, and can easily be done purely with Python.

B. Pandas

Pandas is a library that is practically crucial for projects involving Data Science and machine learning. It is used for analyzing, manipulating, exploring, as well as cleaning data. In particular, it offers data structures and operations for manipulating numerical tables and time series. I will use this library to clean and manipulate my data, as well as explore it before doing machine learning and displaying the data.

C. Scikit-learn

Scikit-learn is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This will be used for the machine learning that I will do in order to predict the likelihood of a player to win MVP and be an All-star, using a Random Forest Regression as well as a Logicstic Regression model.

D. Plotly

This data visualization library is very easy to use and allows for easy customization, interactivity, and flexibility. I will be using this to show the scatter graphs and bar charts in my Streamlit app.

E. Matplotlib

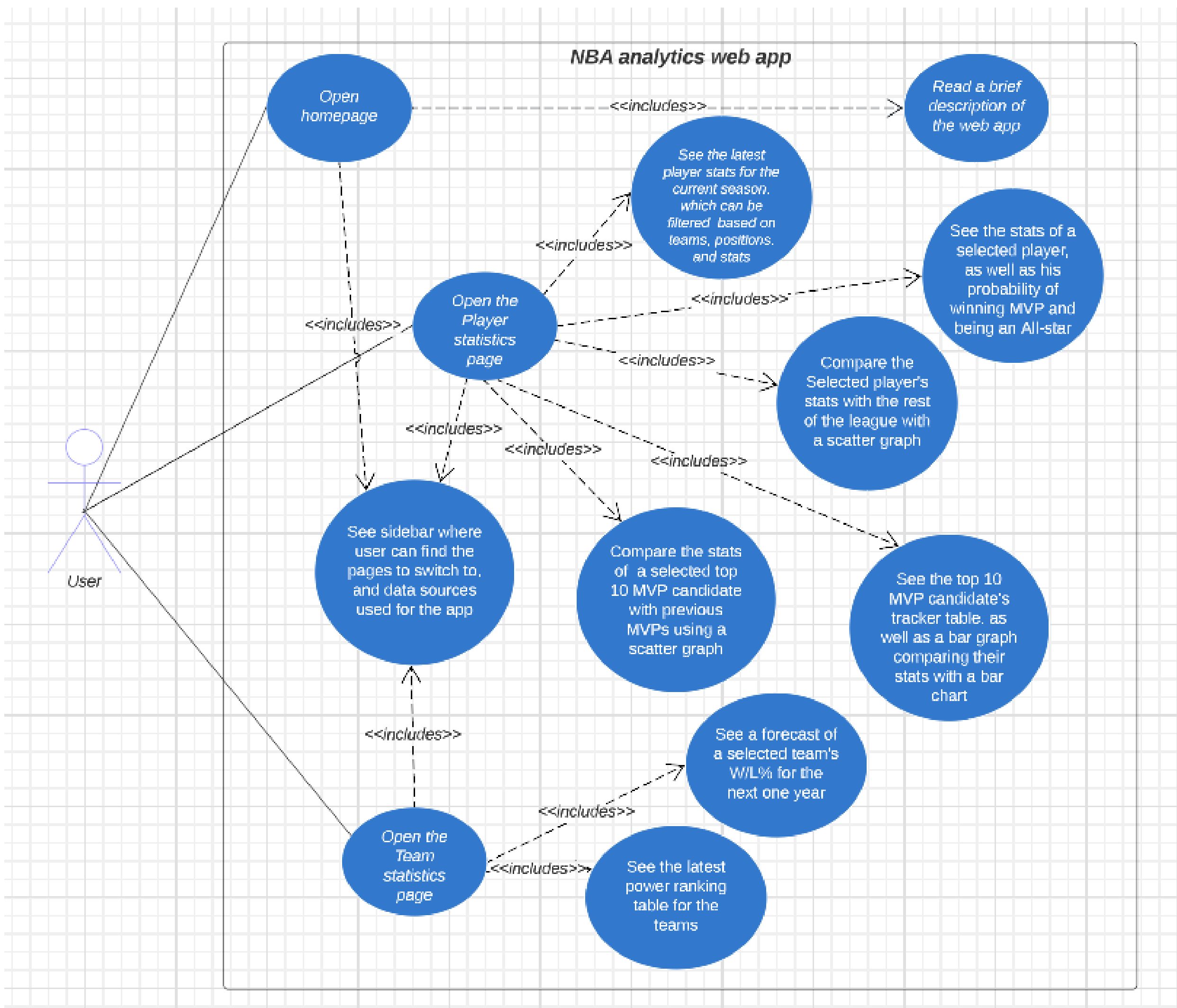
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. I used this library for the data exploration I did before proceeding with machine learning.

F. Prophet

This is a library developed by Facebook that can be used to make forecasts based on time series datasets. This was used to forecast the win/loss percentage of a team for the next one year.

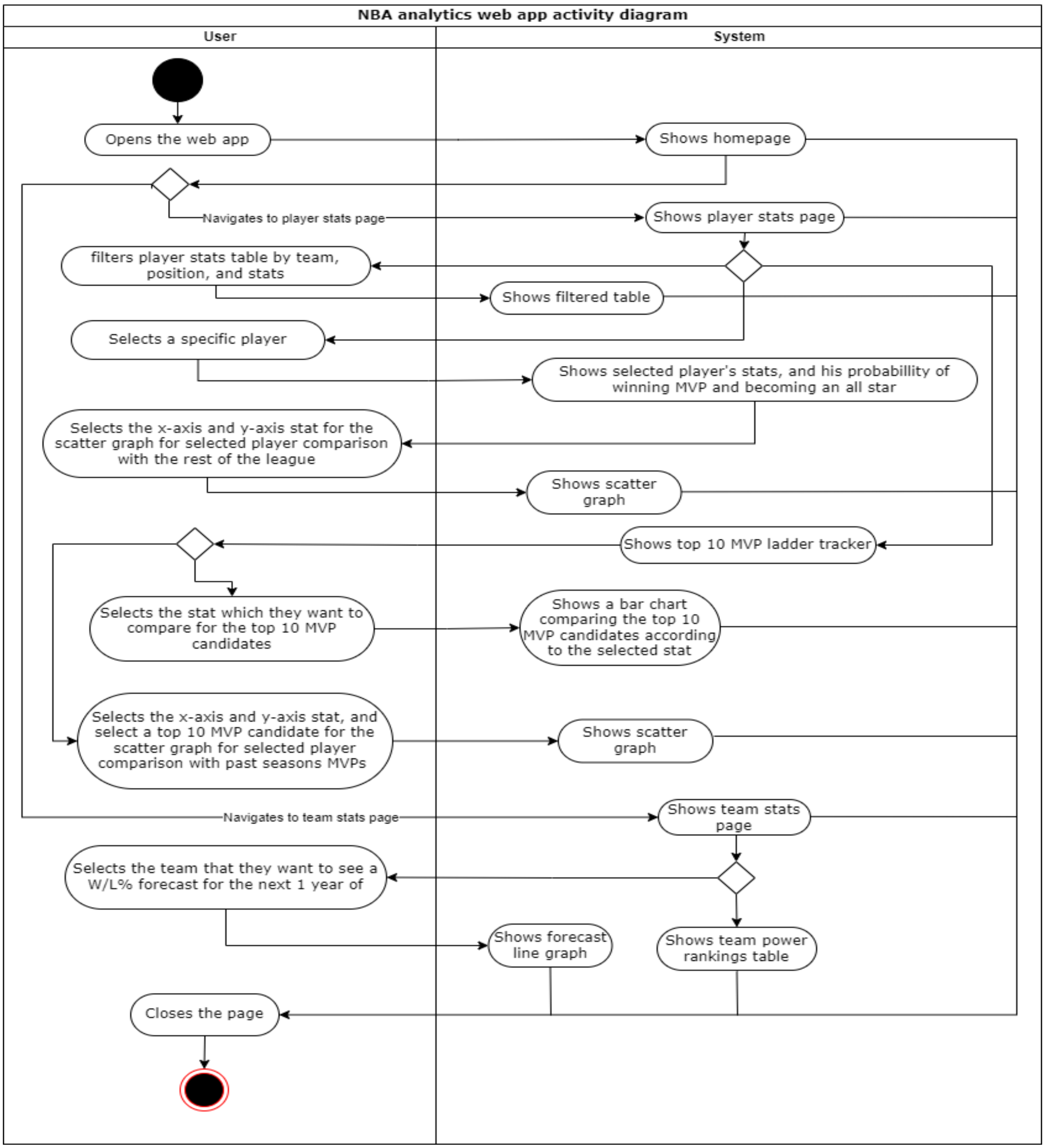
2. Use-case Diagram

The following is a use-case diagram for a better understanding on what users can do, and where they can do it:



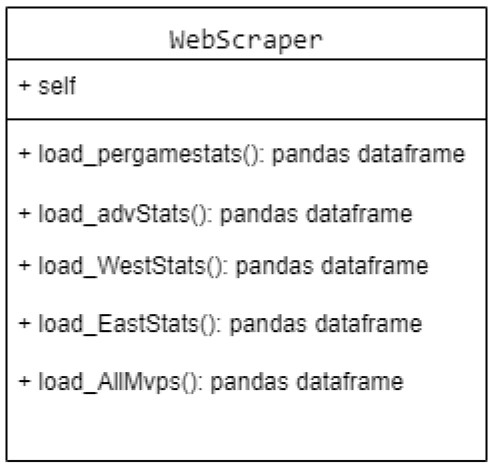
3. Activity Diagram

The following is an activity diagram for my web app, to further understand the course of actions users can take with my program:



4. Class Diagram

Due to the nature of my project/program, I did not need that many classes to achieve the best version of my program. I only have one single class called 'webscraper' which is used for web scraping. The following is a class diagram that I have made:



Essential Algorithms

This section is a documentation for the codes and data manipulation I did when building this web app. There are 3 files for the Streamlit app, which are for the 3 different pages (Homepage, Player statistics, Team statistics). I also have 2 more separate Jupyter notebook files that were used for data cleaning, data exploration, data processing, as well as training the machine learning models.

Data processing and preparation

The first step of this project would be to find the datasets that I will be using, as well as the websites I will scrape data from. These were obtained from the following sources:

- Player stats data for the current 2022/2023 NBA season and all past season MVPs stats, basketball-reference.com
- Team stats data for the current 2022/2023 NBA season, landofbasketball.com

All of these data will go through data cleaning and processing which was done on Jupyter labs, so that I can see how the data frames have been manipulated. After that, I copied all the code from this file into the Streamlit app code file.

1. Data preparation for the latest player statistics data frame on Player_Statistics.py:

[4]:
```
import pandas as pd

perGameStatsUrl = "https://www.basketball-reference.com/leagues/NBA_2023_per_game.html"
perGameStatsDf = pd.read_html(perGameStatsUrl)[0]
```

[129]:
```
perGameStatsDf
```

[129]:

	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
0	1	Precious Achiuwa	C	23	TOR	19	0	18.1	2.9	7.1	...	.791	1.5	3.6	5.1	1.1	0.5	0.6	1.1	1.5	8.1
1	2	Steven Adams	C	29	MEM	38	38	26.9	3.6	6.2	...	.342	4.9	6.4	11.4	2.3	0.8	1.2	1.9	2.3	8.3
2	3	Bam Adebayo	C	25	MIA	39	39	35.0	8.6	16.0	...	.802	2.8	7.3	10.1	3.1	1.1	0.8	2.7	3.0	21.5
3	4	Ochai Agbaji	SG	22	UTA	23	0	11.3	1.3	3.0	...	.500	0.5	0.9	1.4	0.4	0.1	0.1	0.1	1.1	3.5
4	5	Santi Aldama	PF	22	MEM	39	16	22.7	3.4	7.0	...	.722	1.2	3.8	5.0	1.3	0.7	0.8	0.6	2.2	9.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
520	491	Delon Wright	PG	30	WAS	14	0	19.9	1.9	4.9	...	.824	0.7	1.9	2.6	3.1	2.1	0.4	0.9	1.2	5.4
521	492	McKinley Wright IV	PG	24	DAL	13	0	9.4	0.6	1.6	...	.625	0.3	0.7	1.0	1.2	0.5	0.2	0.6	1.0	1.6
522	493	Thaddeus Young	PF	34	TOR	35	9	16.5	2.2	4.0	...	.737	1.6	2.0	3.6	1.6	1.1	0.1	0.7	1.8	4.9

Figure 1: scraping the latest normal per-game player stats

After exploring this data frame, we can see that it has 525 rows and 30 columns. I then removed the "Rk" column as it will not be needed, and filled all the Na values with 0, as after further data exploration, I found out that all of the Na values are on the percentage columns and is due to a player never attempting that specific stat.

```
perGameStatsDf.drop('Rk', inplace = True, axis =1)
perGameStatsDf = perGameStatsDf.fillna(0)
```

Figure 2: Removing 'Rk' and filling all Na cells with 0

I then checked for players who have multiple rows due to them moving teams in the middle of the season.

```
def show_duplicated_players(df):
    playerColumns = pd.DataFrame(df, columns = ["Player"])
    duplicate = playerColumns [playerColumns .duplicated('Player')]
    return duplicate

show_duplicated_players(perGameStatsDf)
```

Figure 3: checking for players who have multiple rows

	Player
41	Player
62	Player
83	Player
104	Player
125	Player
146	Player
167	Player
188	Player
209	Player
230	Player
251	Player
272	Player
276	A.J. Lawson
277	A.J. Lawson

Figure 4: Data frame showing players with multiple rows

I then combined these rows into one, to remove duplicate values.

```
def make_1_row(df):
    if df.shape[0] > 1:
        row = df[df["Tm"] == "TOT"]
        row["Tm"] = df.iloc[-1,:]["Tm"]
        return row

    else:
        return df
```

```
perGameStatsDf = perGameStatsDf.groupby(["Player"]).apply(make_1_row)
perGameStatsDf.index = perGameStatsDf.index.droplevel()
perGameStatsDf = perGameStatsDf.reset_index(drop=True)
```

Figure 5: Combining duplicate rows into one by only taking the TOT row (total stats for duplicated players during the season) and changing the player's team into the most recent one

After that, I scraped the advanced per-game player stats and did the same previous process.

```
playerStatsAdvUrl= "https://www.basketball-reference.com/leagues/NBA_2023_advanced.html"
advStatsDf = pd.read_html(playerStatsAdvUrl)[0]
```

```
advStatsDf
```

	Rk	Player	Pos	Age	Tm	G	MP	PER	TS%	3PAr	...	Unnamed: 19	OWS	DWS	WS	WS/48	Unnamed: 24	OBPM	DBPM
0	1	Precious Achiuwa	C	23	TOR	19	343	14.0	.500	.321	...	NaN	0.0	0.4	0.4	.053	NaN	-2.3	-1.
1	2	Steven Adams	C	29	MEM	38	1022	17.2	.555	.004	...	NaN	1.0	1.9	3.0	.140	NaN	-0.4	0.
2	3	Bam Adebayo	C	25	MIA	39	1366	20.5	.586	.016	...	NaN	1.8	2.1	3.9	.138	NaN	0.6	0.
3	4	Ochai Agbaji	SG	22	UTA	23	259	8.2	.545	.551	...	NaN	0.2	0.1	0.3	.059	NaN	-2.2	-1.
4	5	Santi Aldama	PF	22	MEM	39	887	15.4	.604	.489	...	NaN	1.6	1.4	3.0	.164	NaN	0.4	1.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.

Figure 6: Scraping the advanced per-game player stats

```
advStatsDf.drop(["Unnamed: 19","Unnamed: 24","Rk"], inplace =True, axis=1)
advStatsDf.rename(columns = {'MP':'Total_MP'}, inplace = True)
advStatsDf = advStatsDf.fillna(0)
advStatsDf =advStatsDf.groupby(["Player"]).apply(make_1_row)
advStatsDf.index = advStatsDf.index.droplevel()
advStatsDf = advStatsDf.reset_index(drop=True)
```

```
advStatsDf
```

	Player	Pos	Age	Tm	G	Total_MP	PER	TS%	3PAr	FTr	...	TOV%	USG%	OWS	DWS	WS	WS/48	OBPM	DBPM
0	A.J. Green	SG	23	MIL	20	192	15.0	.678	.833	.056	...	5.1	17.0	0.5	0.2	0.6	.160	1.6	-0.3
1	A.J. Lawson	SG	22	DAL	7	17	0.8	.313	.625	.000	...	0.0	21.3	-0.1	0.0	-0.1	-0.175	-9.5	-5.2
2	AJ Griffin	SF	19	ATL	39	810	13.4	.580	.515	.049	...	8.1	18.8	0.6	0.7	1.3	.077	-0.2	0.1
3	Aaron Gordon	PF	27	DEN	37	1102	20.7	.650	.235	.433	...	11.3	20.9	2.8	1.1	3.9	.171	2.9	-0.5

Figure 7: Removed unnecessary columns and made players with multiple rows have only one row

Now that both the normal and advanced per-game player stats data frame is cleaned, I can combine them into one data frame.

```
FullStats = pd.merge(perGameStatsDf, advStatsDf, on=["Player","Pos","Age","Tm","G"], how='outer').reset_index(drop=True)
```

Figure 8: Combining the normal and advanced stats data frame

2. Data preparation for the latest team statistics data frame that will be used on Player_Statistics.py and Team_Statistics.py:

First, I scraped the tables for the west and east teams' standings table

```
TeamStatsURL = "https://www.landofbasketball.com/yearbyyear/2022_2023_standings.htm"
TeamStatsWest= pd.read_html(TeamStatsURL)[0]
TeamStatsEast= pd.read_html(TeamStatsURL)[1]
```

```
TeamStatsWest
```

	0	1	2	3	4	5	6
0	NaN	Team	W	L	Pct	GB	NaN
1	1.0	Denver Nuggets	29	13	.690	-	NaN
2	2.0	Memphis Grizzlies	29	13	.690	-	NaN
3	3.0	New Orleans Pelicans	26	17	.605	3.5	NaN
4	4.0	Sacramento Kings	23	18	.561	5.5	NaN
5	5.0	Dallas Mavericks	24	20	.545	6.0	NaN
6	6.0	Los Angeles Clippers	22	22	.500	8.0	NaN
7	7.0	Golden State Warriors	21	21	.500	8.0	NaN

Figure 9: Scraping the east and west teams standings table

Upon observation, we can see that the actual header of the data frame is on the first row and not the header, there is an extra unnecessary NaN column on the last column, and an unnecessary ranking column on the first column. To fix this, I did the following steps:

```
TeamStatsWest = TeamStatsWest.dropna(axis = 1)
TeamStatsEast =TeamStatsEast.dropna(axis = 1)

def make_row1_header(df):
    df.columns = df.iloc[0]
    df = df[1:]
    return df

TeamStatsWest = make_row1_header(TeamStatsWest)
TeamStatsEast = make_row1_header(TeamStatsEast)
```

```
TeamStatsWest
```

	Team	W	L	Pct	GB
1	Denver Nuggets	29	13	.690	-
2	Memphis Grizzlies	29	13	.690	-
3	New Orleans Pelicans	26	17	.605	3.5
4	Sacramento Kings	23	18	.561	5.5
5	Dallas Mavericks	24	20	.545	6.0
6	Los Angeles Clippers	22	22	.500	8.0
7	Golden State Warriors	21	21	.500	8.0

Figure 10: Made the first row into the header and removed unnecessary columns

Now that both the east and west team data frames are cleaned, I can combine them into one data frame.

```
TeamStats = pd.concat([TeamStatsWest,TeamStatsEast], axis=0).reset_index(drop = True)
```

```
TeamStats
```

	Team	W	L	Pct	GB
0	Denver Nuggets	29	13	.690	-
1	Memphis Grizzlies	29	13	.690	-
2	New Orleans Pelicans	26	17	.605	3.5
3	Sacramento Kings	23	18	.561	5.5
4	Dallas Mavericks	24	20	.545	6.0
5	Los Angeles Clippers	22	22	.500	8.0
6	Golden State Warriors	21	21	.500	8.0
7	Minnesota Timberwolves	22	22	.500	8.0

Figure 11: Combined the two data frames

For machine learning, I also need the player's team's W/L% which can only be found on the team standings data frame. In order to do this, I have to first add an additional column on the player stats data frame (only has the team name abbreviated) for the player's full team name, by creating a dictionary using a CSV file containing the abbreviated version mapped to the full name version of the team name.

```
abbrevdict = {}

with open("TeamAbbreviationDictionary.csv") as a:
    lines = a.readlines()
    for line in lines[1:]:
        abbreviation, name = line.replace("\n","").split(";")
        abbrevdict[abbreviation] = name
```

```
abbrevdict
```

```
{'ATL': 'Atlanta Hawks',
 'BRK': 'Brooklyn Nets',
 'BOS': 'Boston Celtics',
 'CHO': 'Charlotte Hornets',
 'CHI': 'Chicago Bulls',
 'CLE': 'Cleveland Cavaliers',
 'DAL': 'Dallas Mavericks',
 'DEN': 'Denver Nuggets',
 'DET': 'Detroit Pistons',
 'GSW': 'Golden State Warriors',
 'HOU': 'Houston Rockets',
 'IND': 'Indiana Pacers',
```

Figure 12: Making the team name dictionary

Now that I have the team name dictionary, I can map all the full team names to every player, and get the W/L% for each player. I then dropped the last row of the data frame, as it is just fully Na.

```
FullStats["Team"] = FullStats["Tm"].map(abbrevdict)
FullStats= FullStats.merge(TeamStats, how = "outer", on="Team")
FullStats.drop(["W","L","GB","Team"],axis=1,inplace=True)
FullStats.drop(FullStats.tail(1).index,inplace=True)
```

Figure 13: Obtaining the W/L% for each player

I then did some more data exploration and discovered that the entire data frame has the wrong data type, and had to fix these data types..

```
FullStats[["Age","G","GS","Total_MP"]] = FullStats[["Age","G","GS","Total_MP"]].astype(int)
FullStats.iloc[:, 6:29] = FullStats.iloc[:, 6:29].astype(float)
FullStats.iloc[:, 30:51]= FullStats.iloc[:, 30:51].astype(float)
```

Figure 14: Fixing the data types

3. Data preparation for the past season MVPs stats data frame that will be used for the scatter graph on Player_Statistics.py:

To obtain the past seasons MVPs stats, I first had to scrape this data from the internet

```
mvpStatsUrl = "https://www.basketball-reference.com/awards/mvp.html"
mvpStats = pd.read_html(mvpStatsUrl)[0]
```

```
mvpStats
```

	Unnamed: 0_level_0	Unnamed: 1_level_0	Unnamed: 2_level_0	Unnamed: 3_level_0	Unnamed: 4_level_0	Unnamed: 5_level_0	Unnamed: 6_level_0	Per Game								
	Season	Lg	Player	Voting	Age	Tm	G	MP	PTS	TRB	AST	STL	BLK	FG%	3
0	2021-22	NBA	Nikola Jokić	(V)	26	DEN	74	33.5	27.1	13.8	7.9	1.5	0.9	0.583	0.3
1	2020-21	NBA	Nikola Jokić	(V)	25	DEN	72	34.6	26.4	10.8	8.3	1.3	0.7	0.566	0.3
2	2019-20	NBA	Giannis Antetokounmpo	(V)	25	MIL	63	30.4	29.5	13.6	5.6	1.0	1.0	0.553	0.3
3	2018-19	NBA	Giannis Antetokounmpo	(V)	24	MIL	72	32.8	27.7	12.5	5.9	1.3	1.5	0.578	0.2
4	2017-18	NBA	James Harden	(V)	28	HOU	72	35.4	30.4	5.4	8.8	1.8	0.7	0.449	0.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
62	1959-60	NBA	Wilt Chamberlain	(V)	23	PHW	72	46.4	37.6	27.0	2.3	NaN	NaN	0.461	N

Figure 15: Scraping the past season MVPs stats

This data frame has an unnecessary multi-level header and some columns that will not be needed. So the next step I did was to delete all of this.

```
mvpStats.columns = [col[1] for col in mvpStats.columns]
```

```
mvpStats.drop(["Lg","Voting"],axis=1,inplace=True)
mvpStats = mvpStats.fillna(0)
```

```
mvpStats
```

	Season	Player	Age	Tm	G	MP	PTS	TRB	AST	STL	BLK	FG%	3P%	FT%	WS	WS/48
0	2021-22	Nikola Jokić	26	DEN	74	33.5	27.1	13.8	7.9	1.5	0.9	0.583	0.337	0.810	15.2	0.296
1	2020-21	Nikola Jokić	25	DEN	72	34.6	26.4	10.8	8.3	1.3	0.7	0.566	0.388	0.868	15.6	0.301
2	2019-20	Giannis Antetokounmpo	25	MIL	63	30.4	29.5	13.6	5.6	1.0	1.0	0.553	0.304	0.633	11.1	0.279
3	2018-19	Giannis Antetokounmpo	24	MIL	72	32.8	27.7	12.5	5.9	1.3	1.5	0.578	0.256	0.729	14.4	0.292
4	2017-18	James Harden	28	HOU	72	35.4	30.4	5.4	8.8	1.8	0.7	0.449	0.367	0.858	15.4	0.289
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
62	1959-60	Wilt Chamberlain	23	PHW	72	46.4	37.6	27.0	2.3	NaN	NaN	0.461	NaN	0.582	17.0	0.245
63	1958-59	Bob Pettit	26	STL	72	39.9	29.2	16.4	3.1	NaN	NaN	0.438	NaN	0.759	14.8	0.246

Figure 16: Dropped unnecessary columns, filled Na with 0, and removed the multi-level header

Training the machine learning models

1. MVP prediction model

For this machine learning model, I used a data set that contains past seasons' players' stats alongside the amount of MVP votes they got, which can be found on kaggle.com

Before proceeding with training my model, the first step that I have to take is to first clean and balance my data.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
statsDf = pd.read_csv("1982-2022_NBA_MVP_Stats.csv") #sets a variable for my 1982-2022 Nba Player Mvp voting stats data se
```

```
statsDf
```

	season	player	pos	age	team_id	g	gs	mp_per_g	fg_per_g	fga_per_g	...	ws	ws_per_48	obpm	dbpm	bpm	vorp
0	1982	Kareem Abdul-Jabbar	C	34	LAL	76	76	35.2	9.9	17.1	...	10.7	0.192	3.8	1.2	5.0	4.7
1	1982	Alvan Adams	C	27	PHO	79	75	30.3	6.4	13.0	...	7.2	0.144	1.4	2.2	3.6	3.4
2	1982	Mark Aguirre	SF	22	DAL	51	20	28.8	7.5	16.1	...	1.9	0.061	2.3	-1.6	0.7	1.0
3	1982	Danny Ainge	SG	22	BOS	53	1	10.6	1.5	4.2	...	0.5	0.042	-3.7	1.0	-2.7	-0.1

Figure 17: Loading the past seasons' stats data set

After loading the data set into my Juptyer notebook, I filled all the Na with 0, renamed the column names so that it is the same as all my other data frames (found on the previous sections), and removed the columns that I will not be using.

```
#Renaming the columns of the dataframe to match the name of the columns for the current season
#stats dataframe that I will webscrape
statsDf.columns = ["season","Player","Pos","Age","Tm","G","GS","MP","FG","FGA",
                   "FG%","3P","3PA","3P%","2P","2PA","2P%",
                   "eFG%","FT","FTA","FT%","ORB","DRB","TRB","AST","STL","BLK","TOV","PF","PTS",
                   "Total_MP","PER","TS%","3PAr","FTr","ORB%","DRB%","TRB%","AST%",
                   "STL%","BLK%","TOV%","USG%","OWS","DWS","WS","WS/48", "OBPM","DBPM","BPM","VORP","award_share",
                   "mov","mov_adj", "W/L%"]

#Dropped 2 columns that I will not be using
statsDf.drop(['mov', 'mov_adj'], axis=1, inplace= True)
```

Figure 18

When doing machine learning, it is important to have a balanced data set. Data that is unbalanced means that one subset of observations for the target class is significantly more numerous than another subset of observations. The majority class refers to the bigger subset, whereas the minority class refers to the smaller subset. In my scenario, there are significantly fewer players who received MVP votes than there are players who did not. My model could train itself to predict players who will not receive votes, even though it would be largely right, this would have no bearing on picking the next MVP. Therefore, it is preferable to balance the data to enable my algorithm to accurately understand the connection between a player's statistics and receiving MVP votes. In this situation, we wish to correctly estimate the players who will receive MVP votes, but this will be difficult because they are a minority class. Thus to solve this, I need to undersample the majority class by removing players that have an extremely low chance of receiving votes. To find these players with low chances, I did some more data exploration using Matplotlib and found the following:

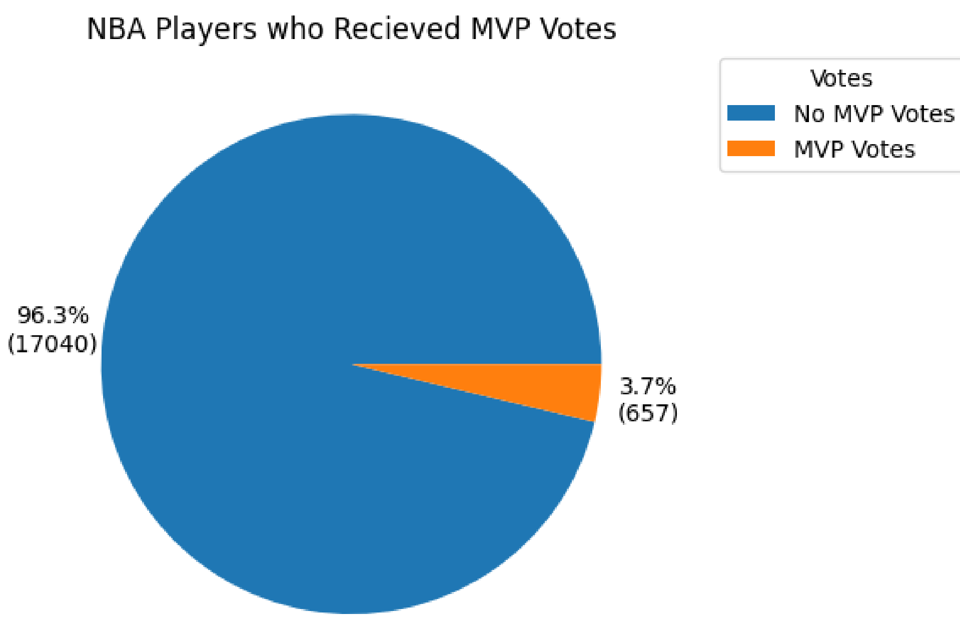


Figure 19: Shows that there are way more players who did not get any MVP votes

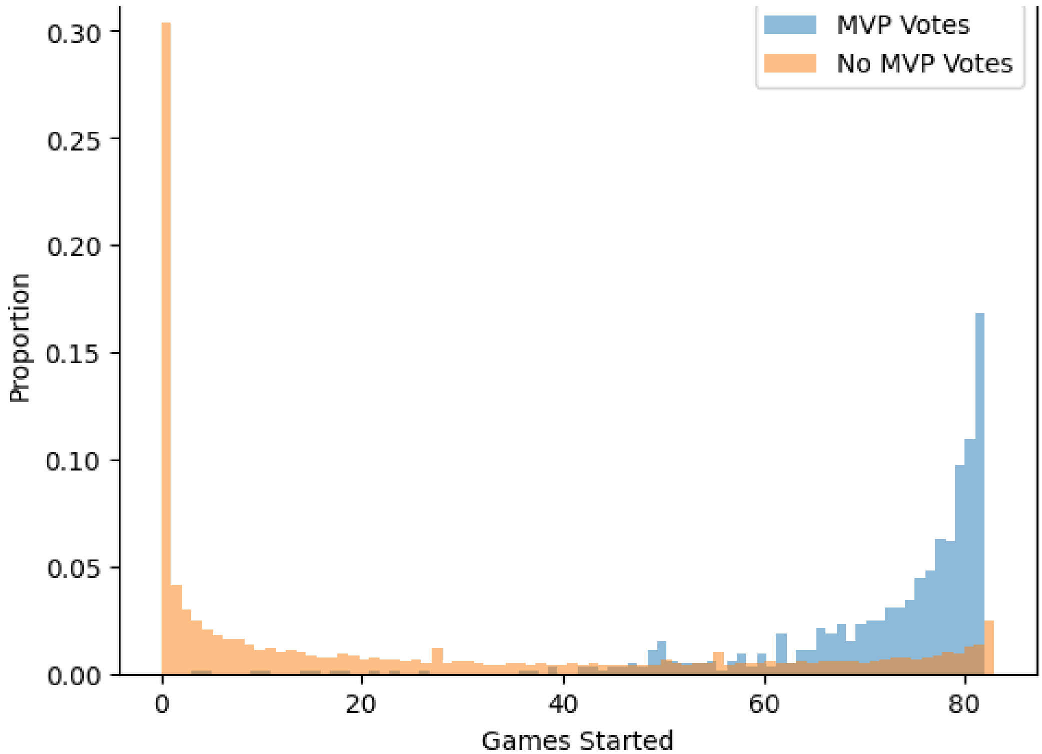


Figure 20: Shows that the majority of players that gets a vote must start at least 30 games

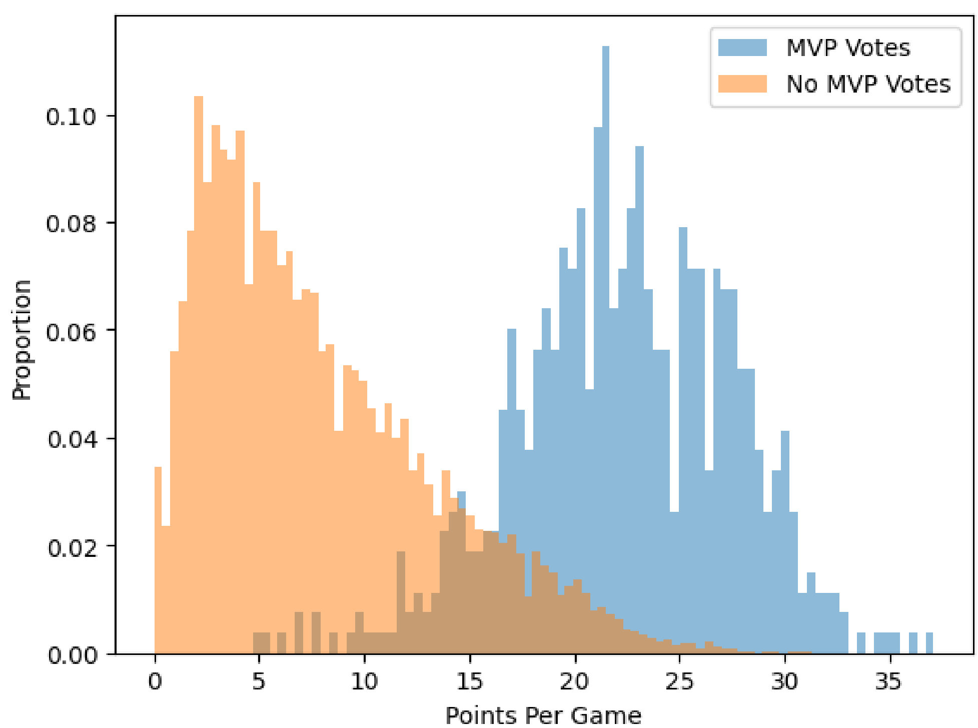


Figure 21: Shows that the majority of players that gets a vote must score at least 7 points per game

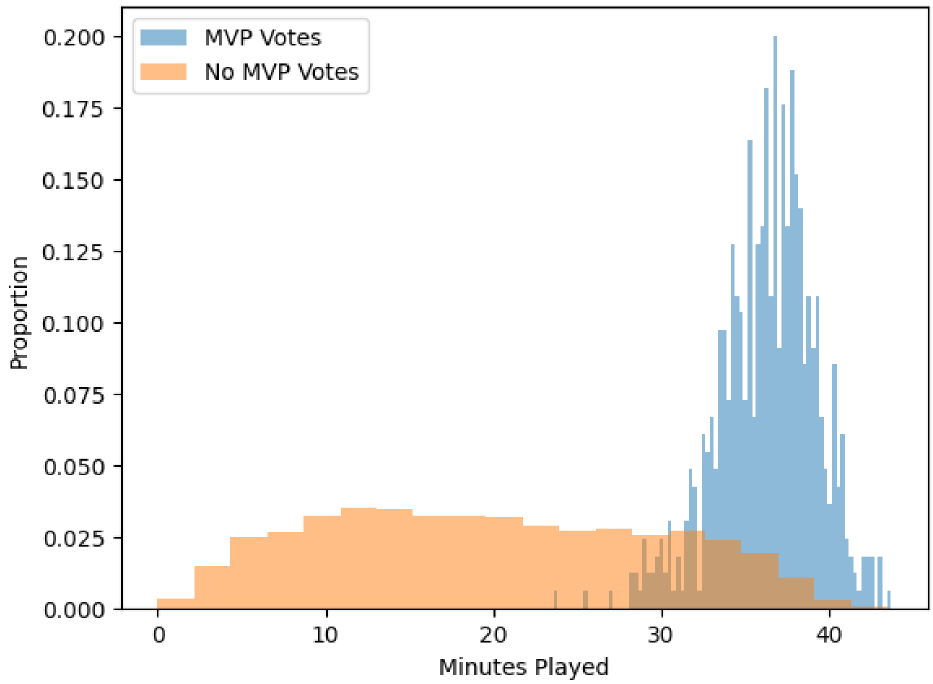


Figure 22: Shows that the majority of players that gets a vote must play at least 27.6 minutes per game

So after discovering all of this, I reduced the data set to only players who played more than 27.6 minutes per game, scored 7 points per game, and start more than 30 games.

```
statsDf = statsDf[statsDf["GS"]>30]
statsDf = statsDf[(statsDf["PTS"] >6.9)].reset_index(drop = True)
statsDf = statsDf[(statsDf["MP"] >27.6)].reset_index(drop = True)
```

Figure 23: reduced the data set to only players who played more than 27.6 minutes per game, scored 7 points per game, and start more than 30 games.

My data set looks a bit better now that I have undersampled the majority class;

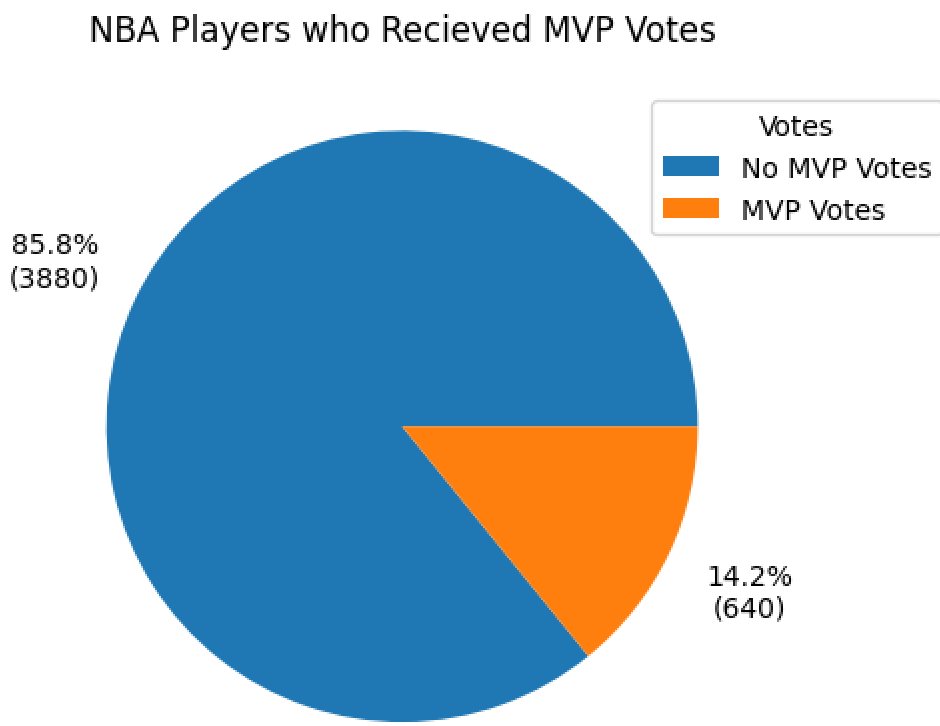


Figure 24: shows the share of players who gets votes after undersampling the majority class

Now that I have cleaned and somewhat balanced my data, I will proceed with choosing and training a machine-learning model. Before this, however, I need to divide my data set into two - a set for training (player stats before 2022) and a set for testing (2022 player stats).
I then chose to use a Ridge Regression model as the first model to test out as it is usually used when the data suffers from multicollinearity (independent variables are highly correlated). In my case, I am using multiple stats to predict the amount of MVP shares a player will get, so it will be fitting to use this model. This model can be imported from the 'Sci-kit learn' library.

```
trainStatsDf = statsDf[statsDf["season"]<2022]
testStatsDf = statsDf[statsDf["season"] == 2022]
```

```
from sklearn.linear_model import Ridge

Rmodel = Ridge(alpha=0.1)
```

```
predictors = ["Age","G","GS","MP","FG","FGA",
                      "FG%","3P","3PA","3P%","2P","2PA","2P%",
                      "eFG%","FT","FTA","FT%","ORB","DRB","TRB","AST","STL","BLK","TOV","PF","PTS",
                      "MP","PER","TS%","3PAr","FTr","ORB%","DRB%","TRB%","AST%",
                      "STL%","BLK%","TOV%","USG%","OWS","DWS","WS","WS/48", "OBPM","DBPM","BPM","VORP","W/L%"]
```

```
Rmodel.fit(trainStatsDf[predictors], trainStatsDf["award_share"])
```

```
Ridge
Ridge(alpha=0.1)
```

```
predictions = Rmodel.predict(testStatsDf[predictors])
```

Figure 25: Divided the dataset, imported the machine learning model, trained the model to predict the MVP award share a player will get using the stats that are used as predictors, and tested this model on the test data set.

I then created a data frame to compare the predicted results with the actual result of the test data set.

```
[33]: predictions = Rmodel.predict(testStatsDf[predictors])

[34]: predictions = pd.DataFrame(predictions, columns = ["predictions"], index=testStatsDf.index)

[35]: compare = pd.concat([testStatsDf[["Player","award_share"]],predictions], axis =1)

[37]: compare= compare.sort_values("award_share", ascending = False)
      compare["Rk"]= list(range(1,compare.shape[0]+1))

[38]: compare = compare.sort_values("predictions", ascending = False)
      compare["Predicted_Rk"]= list(range(1,compare.shape[0]+1))

[39]: compare
```

[39]:

	Player	award_share	predictions	Rk	Predicted_Rk
4469	Nikola Jokić	0.875	0.368845	1	1
4397	Giannis Antetokounmpo	0.595	0.303620	3	2
4439	Joel Embiid	0.706	0.229479	2	3
4434	Luka Dončić	0.146	0.192527	5	4
4519	Trae Young	0.000	0.181081	125	5

Figure 26: Dataframe comparing the actual MVP rankings with the one my model predicted

To see how accurate this model is, I did backtesting and tested my model for every season since 1996 by doing the following method:

```
def backtest(stats,model,year,predictors):
    aps = []
    all_predictions = []
    for year in years[:5]:
        trainStatsDf = statsDf[statsDf["season"]< year]
        testStatsDf = statsDf[statsDf["season"] == year]
        model.fit(trainStatsDf[predictors], trainStatsDf["award_share"])
        predictions = model.predict(testStatsDf[predictors])
        predictions = pd.DataFrame(predictions, columns = ["predictions"], index=testStatsDf.index)
        compare = pd.concat([testStatsDf[["Player","award_share"]],predictions], axis =1)
        all_predictions.append(compare)
        aps.append(find_ap(compare))
    return sum(aps)/len(aps), aps , pd.concat(all_predictions)
```

```
mean_ap,aps,all_predictions = backtest(statsDf,Rmodel,years[5:],predictors)
```

```
mean_ap
```

```
0.7896349206349205
```

Figure 27: From this, I can say that this model has an accuracy of 79%

The next model that I tried out was a Random Forest Regression model, which many have said to work very well on most data sets. This was the result obtained using the same previous process:

```
from sklearn.ensemble import RandomForestRegressor

rfmodel = RandomForestRegressor(n_estimators=500, random_state =1, min_samples_split=5)

mean_ap,aps,all_predictions = backtest(statsDf,rfmodel,years[5:],predictors)
```

```
mean_ap
```

```
0.8937532467532467
```

Figure 27: Training a Random Forest Regression model

I can then conclude that the Random Forest Regression model performs better, with an accuracy of 89%. Now the next step would be to save this model as a pickle file so that I can later deploy it on my Streamlit app code file.

```
import pickle
pickle.dump(rfmodel, open('rfmodel.pkl', 'wb'))
```

Figure 28: Saving the model as a pickle file

2. All-star prediction model

For this machine learning model, I used a data set that contains past seasons' players' stats alongside a boolean column that determines if a player is an All-star that season, which can be found on kaggle.com. The model that I chose to use for this is a Logistic Regression Model as it's used when the prediction is categorical, like yes or no, true or false, 0 or 1. The data set that I am using, shows a boolean of either 1 or 0, to determine if the player is an all-star that season. With this in mind, the Logistic Regression Model is the best choice.

The process that I went with this model is similar to the one I did for the MVP model

```
allstar =pd.read_csv('2016-2019_total stats.csv')
```

```
allstar =allstar.fillna(0)
```

```
allstar.columns
```

```
Index(['Season', 'Player', 'Pos', 'Age', 'Tm', 'G', 'GS', 'MP', 'FG', 'FGA',
       'FG%', '3P', '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA',
       'FT%', 'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS',
       'PER', 'TS%', 'USG%', 'WS', 'BPM', 'All-Star'],
      dtype='object')
```

```
allstarPredictors = allstar.columns[7 : 35].to_list()
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(allstar[allstarPredictors], allstar['All-Star'],
                                                    train_size=0.8, random_state=1)
```

Figure 29: Imported the data frame, filled all Na with 0, determined the predictors, and imported the model

```
from sklearn.linear_model import LogisticRegression

logregModel = LogisticRegression(solver='lbfgs', max_iter=3000)
logregprediction = logregModel.fit(X_train, y_train)
print('training accuracy: {}'.format(logregprediction.score(X_train, y_train).round(3)))
print('test accuracy: {}'.format(logregprediction.score(X_test, y_test).round(3)))
```

```
training accuracy: 0.98
test accuracy: 0.987
```

```
pickle.dump(logregModel, open('AllStarslogregModel.pkl', 'wb'))
```

Figure 30: Trained the model and saved it as a Pickle file to deploy later on my Streamlit app code file.

Streamlit

1.Homepage.py

This is the Python code file for first page that the user will see when opening the web app.

- Importing streamlit and pandas library, setting the page configuration, as well as putting my data sources on the sidebar.

```
Homepage.py > ...
import streamlit as st
import pandas as pd

#setting the configuration of this page
st.set_page_config(
    page_title="Homepage",
    initial_sidebar_state="expanded",
    layout='centered'
)


# list of my data sources, put on the sidebar
DataSourcesinfo = st.sidebar.expander("Data sources")
DataSourcesinfo.write("""[Player Data](https://www.basketball-reference.com)\n
[Team Data](https://www.landofbasketball.com/yearbyyear/2022_2023_standings.htm)\n
[Past MVP Dataset for machine learning](https://www.kaggle.com/code/robertsunderhaft/predicting-the-nba-mvp/data
[Past All-Star Dataset for machine learning](https://www.kaggle.com/datasets/toniabiru/nba-stats-20162019-seasor
[Time series dataset of every team's W/L%](https://www.kaggle.com/datasets/nathanlauga/nba-games)\n
[Github Source code](https://github.com/FrancescoEmmanuel/NBA-Analytics-Web-App)""")
```

- Shows the title and description of my web app

```
st.title("NBA Analytics App")
st.image("images/nba-basketball-logo.jpg", use_column_width=True)
st.header("About")
st.markdown("""
In the world of basketball where everyone goes crazy over dunks, 3 pointers and ankle breakers,
what have always fascinated me were the statistics. Statistics opens up a door to a different world and
what this app will aim to do is to allow you to see basketball from a whole different lens like never before.
\n
This is an NBA analytics web app where you can see up to date NBA player's statistics,
some fun data visualization of NBA statistics, a forecast of every team's win/loss precentage for the next one
of a player to win MVP and be an All-Star, with the use of machine learning models.
\n

""")
```

- Created a class for web scraping. Details and code of the web scraping process have been explained in the pervious sections of this report. I used st.cache to make my program run faster, as this saves the output of the functions - meaning that the program doesn't always have to run the functions.

```python
class WebScraper:
    def __init__(self):
        pass

    # Fetching all players' per game stats this season so far
    @st.cache(allow_output_mutation=True) #st.cache caches the function which basically saves the output of the
    # "allow_output_mutation = True" allows the saved output to be changed when the input is changed as the scra

    def load_pergamestats(self):
        perGameStatsUrl = "https://www.basketball-reference.com/leagues/NBA_2023_per_game.html"
        Df = pd.read_html(perGameStatsUrl)[0]
        Df.drop('Rk', inplace = True, axis =1) #Dropping the columns that won't be needed during machine learnin
        Df = Df.fillna(0)
        Df = Df.groupby(["Player"]).apply(make_1_row) # Setting all players who moved to different teams into or
        Df.index = Df.index.droplevel()
        Df = Df.reset_index(drop=True)
        return Df

    # Fetching all players' advanced game stats this season so far
```

2.Player_Statistics.py

- Importing all the necessary libraries, as well as the functions and class from Homepage.py. Configured the page, and put my data sources on the sidebar.

```python
import streamlit as st
import pandas as pd
import pickle
import plotly.express as px
from Homepage import *


#setting the configuration of this page
st.set_page_config(
    page_title="Player Statistics",
    initial_sidebar_state="expanded",
    layout='centered'
)

# list of my data sources, put on the sidebar
DataSourcesinfo = st.sidebar.expander("Data sources")
DataSourcesinfo.write("""[Player Data](https://www.basketball-reference.com)\n
[Team Data](https://www.landofbasketball.com/yearbyyear/2022_2023_standings.htm)\n
[Past MVP Dataset for machine learning](https://www.kaggle.com/code/robertsunderhaft/predicting-the-nba-mvp/data
[Past All-Star Dataset for machine learning](https://www.kaggle.com/datasets/tonjahiru/nba-stats-20162019-seasor
```

- Set the variables for the scraped data

```python
#This whole section below is pre data procession to load all the necessary data and dataframes

#set variables for the scraped data
perGameStatsDf = WebScraper()
advStatsDf= WebScraper()
TeamStatsWest= WebScraper()
TeamStatsEast= WebScraper()
mvpstats= WebScraper()
perGameStatsDf=perGameStatsDf.load_pergamestats()
advStatsDf =advStatsDf.load_advStats()
TeamStatsWest=TeamStatsWest.load_WestStats()
TeamStatsEast=TeamStatsEast.load_EastStats()
mvpstats= mvpstats.load_AllMvps()
```

- I then did the data processing and cleaning on the player and team stats – which has been explained in the previous section, and deployed my machine learning model for the MVP and all-star prediction.

```python
rfmodel = pickle.load(open('Machine Learning models/rfmodel.pkl', 'rb')) #deploying the random forest regressor

#Determining the columns that will be used to predict
predictors = ["Age","G","GS","MP","FG","FGA",
                        "FG%","3P","3PA","3P%","2P","2PA","2P%",
                        "eFG%","FT","FTA","FT%","ORB","DRB","TRB","AST","STL","BLK","TOV","PF","PTS",
                        "MP","PER","TS%","3PAr","FTr","ORB%","DRB%","TRB%","AST%",
                        "STL%","BLK%","TOV%","USG%","OWS","DWS","WS","WS/48", "OBPM","DBPM","BPM","VORP","W/L%"]

#setting the model
MvpPrediction= rfmodel.predict(FullStats[predictors])

#making the predictions made by the model into a dataframe
Mvppredictions = pd.DataFrame(MvpPrediction, columns = ["predictions"], index=FullStats.index)

#combining the predictions dataframe with the Fullstats dataframe
FullDf = pd.concat([FullStats,Mvppredictions], axis =1)
FullDf = FullDf.sort_values(by = "predictions", ascending = False)
```

```python
#deploying the logic regressor machine learning model that will be used to predict the chance of being an all-st
AllStarLogregModel = pickle.load(open('Machine Learning models/AllStarslogregModel.pkl', 'rb'))

# Determining the columns that will be used to predict the chance of becoming an allstar
AllStarPredictors = ['BLK', 'DRB', 'WS', 'STL', 'AST', 'USG%', '3P', '3P%', 'MP', '2PA', 'TOV', 'TRB', 'FGA', 'F
AllstarPrediction = AllStarLogregModel.predict(FullStats[AllStarPredictors])

#making the predictions made by the model into a dataframe
AllstarPredictions = pd.DataFrame(AllstarPrediction, columns = ["Allstar_Prediction"], index=FullStats.index)

#Combining the Allstar prediction dataframe with th Full Stats dataframe
AllstarDf = pd.concat([FullStats,AllstarPredictions], axis =1)
```

- I then displayed the scraped and fully cleaned live NBA player stats data frame - which can be sorted based on a stat, and filtered based on the teams and positions. The stats can be selected by a select box, and the position and teams can be filtered by streamlit's multi-select feature.

```
#This section shows the live player stats for the current season
st.header("2022-2023 NBA Player Stats")
sorted_unique_team = sorted(FullDf.Tm.unique()) # set a variable for all the teams sorted
selected_team = st.multiselect('Team', sorted_unique_team, sorted_unique_team) #set a variable for the team that
unique_pos = ['C','PF','SF','PG','SG'] # made a list of all NBA positions
selected_pos = st.multiselect('Position', unique_pos, unique_pos)#set a variable for the position that the user
df_selected_team = FullDf[(FullDf.Tm.isin(selected_team)) & (FullDf.Pos.isin(selected_pos))] #made a dataframe b

 #made a multiselect box for the stats that a user can use to sort the table on descending order and show the le
selected_leader_filter = st.selectbox("Leaders",list(FullDf.iloc[:, 4:51]))
TheDf = df_selected_team.sort_values(by = selected_leader_filter, ascending= False).reset_index(drop =True)

st.dataframe(TheDf ) #show the data frame
```

- Next thing you can find is the in-depth analysis section which shows the selected player's stats, his probability of winning MVP (using the deployed MVP model), if he has a chance of being an all-star(using the deployed all star model), and a scatter graph to compare this selected player with the rest of the league, using stats that the user selects. Again, streamlit's selectbox feature is used for the users to select the player and stats.

```
#This section shows an in depth analysis of a user selected player
st.header("In depth analysis")

selected_player = st.selectbox("Player",list(FullDf["Player"])) #selectbox for the user to select a player

SelectedStats = FullDf[FullDf["Player"] == selected_player] #made a dataframe for the selected player
SelectedAllstarPredict = AllstarDf[AllstarDf["Player"] == selected_player] #made the allstar prediction datafram
st.dataframe(SelectedStats) #shows only the stats of the selected player
st.write(f"{selected_player}'s probability of winning MVP: ",(SelectedStats.iloc[0]["predictions"])*100,"%") #sh

#show if this selected player has a chance to be an allstar
if SelectedAllstarPredict["Allstar_Prediction"].all() == 1:
    st.write(f"{selected_player} has a chance to be an All-star this season")
else:
    st.write(f"{selected_player} has no chance to be an All-star this season")

#This section shows a scatter graph to compare the selected player with the rest of the league based on stats th
st.subheader(f"Scatter graph of {selected_player} and the rest of the league")
Scatterx = st.selectbox("Choose the x-axis stat",list(FullDf.iloc[:, 4:51])) #selectbox for the user to choose t
Scattery = st.selectbox("Choose the y-axis stat",list(FullDf.iloc[:, 4:51])) #selectbox fort the user to choose

figure = px.scatter(FullDf, x=Scatterx, y=Scattery,hover_name = FullDf["Player"])

#highlights the selected player in yellow
figure.add_traces(
    px.scatter(FullDf[FullDf["Player"]==selected_player],
    x=Scatterx, y=Scattery,hover_name =SelectedStats["Player"]).update_traces(marker_color="yellow").data
)

st.plotly_chart(figure) #shows the graph
```

-

- The next section is the top 10 MVP ladder tracker which shows the top 10 MVP candidates (based on my MVP model), a bar chart comparing the top 10 candidates using a stat that the user selects, as well as a scatter graph to compare the selected candidate with the past seasons MVPs (data taken from the data set I found on Kaggle).

```
#This section shows a table for the top 10 MVP ladder tracker
st.subheader("Top 10 MVP Ladder Tracker") #shows the dataframe for the top 10 MVP candidates

Top10Df = FullDf.head(10)# made a dataframe for the top 10 MVP candidates
st.write(Top10Df.reset_index(drop = True))

stat1= st.selectbox("Choose a stat to compare with",list(FullDf.iloc[:, 4:51])) #selectbox to choose the stat
st.subheader(f"Comparing the top 10 MVP candidate's {stat1} ")
fig2 = px.bar(Top10Df,x=Top10Df["Player"],y= stat1) #made a barchart comparing the top 10 mvp candidates based
st.plotly_chart(fig2) #display the barchart

#this section is a scatter graph that compares a user selected top 10 mvp candidates with the past season MVPs
selectedTop10 = st.selectbox("Select a player",list(Top10Df["Player"])) #selectbox for the user to choose a to
selectedTop10Df = Top10Df[Top10Df["Player"] == selectedTop10]

#Sets the columns to only the columns of stats that are available for the past season MVPs dataset that i foun
selectedTop10Df = selectedTop10Df[["Player","G","MP","PTS","TRB","AST","STL","BLK","FG%","3P%","FT%","WS","WS/


fig3Df = mvpstats.append(selectedTop10Df) # made a dataframe that combined the current top 10 MVP candidates'
st.subheader(f"Scatter graph of {selectedTop10} with past seasons MVPs")
statX2= st.selectbox("Choose the x-axis stat",list(fig3Df[["G","MP","PTS","TRB","AST","STL","BLK","FG%","3P%",
statY2= st.selectbox("Choose the y-axis stat",list(fig3Df[["G","MP","PTS","TRB","AST","STL","BLK","FG%","3P%",

fig3 = px.scatter(fig3Df , x=statX2, y=statY2,hover_name = fig3Df["Player"])

#highlights the selected player in yellow
fig3.add_traces(
    px.scatter(fig3Df[fig3Df["Player"]==selectedTop10],
    x=statX2, y=statY2,hover_name =selectedTop10Df["Player"]).update_traces(marker_color="yellow").data)

st.plotly_chart(fig3) #displays the scatter graph
```

3.Team_Statistics.py

- Importing all the necessary libraries, as well as the functions and class from Homepage.py. Configured the page, put my data sources on the sidebar, and set variables for the scraped team data.

```python
import streamlit as st
import pandas as pd
from prophet import Prophet
from prophet.plot import plot_plotly
from Homepage import *


#setting the configuration of this page
st.set_page_config(
    page_title="Team Statistics",
    initial_sidebar_state="expanded",
    layout='centered'
)



# list of my data sources, put on the sidebar
DataSourcesinfo = st.sidebar.expander("Data sources")
DataSourcesinfo.write("""[Player Data](https://www.basketball-reference.com)\n
[Team Data](https://www.landofbasketball.com/yearbyyear/2022_2023_standings.htm)\n
[Past MVP Dataset for machine learning](https://www.kaggle.com/code/robertsunderhaft/predicting-the-nba-mvp/da
[Past All-Star Dataset for machine learning](https://www.kaggle.com/datasets/toniabiru/nba-stats-20162019-seas
[Github Source code](https://github.com/FrancescoEmmanuel/NBA-Analytics-Web-App)""")


st.header("Team Statistics")

#set variables for the scraped data
TeamStatsWest= WebScraper()
TeamStatsEast= WebScraper()
TeamStatsWest=TeamStatsWest.load_WestStats()
TeamStatsEast=TeamStatsEast.load_EastStats()
```


- Displays live NBA team power rankings, by combining the east and west team stats data frame in descending order according to their W/L%

```python
#This section is for the team power ranking table
TeamStats = pd.concat([TeamStatsWest,TeamStatsEast], axis=0).reset_index(drop = True) #combined the west and ea
TeamStats["Pct"]=TeamStats["Pct"].astype(float) #set this column datatype as float
TeamStats = TeamStats.rename(columns = {"Pct" : "W/L%"})
TeamStats.sort_values(by= ["W/L%"], ascending=False,inplace = True) #sort the values in decending order based o
TeamStats =TeamStats.reset_index(drop =True)
st.subheader(" 2022-2023 Live power rankings")
st.write(TeamStats) #displays the power rankings dataframe
```

- The next section and last section of this web app shows a forecast of a selected team's W/L% for the next year. First I loaded a time series dataset that I found on Kaggle - which contains every team's W/L% everyday since 2006. And appended all the team names from the data set for the user to select using the selectbox feature.

```
#This section is for the W/L% forecast section
timeSeriesTeams = pd.read_csv("datasets/ranking.csv") # loaded the dataset for all teams' win lost percentage ev
#Appending the team names in timeSeries teams to a list for selectbox
Teams=[]
for a in timeSeriesTeams['TEAM']:
    # check if value is not already in the list
    if a  not in Teams:
        # if it is not, append it to the list
        Teams.append(a)

st.subheader("Forecast")
st.markdown("This is a forecast for the W/L Percentage of a team of your choice")
selected_team = st.selectbox("Select the team",list(Teams)) #Select box the for the user to choose a team
```

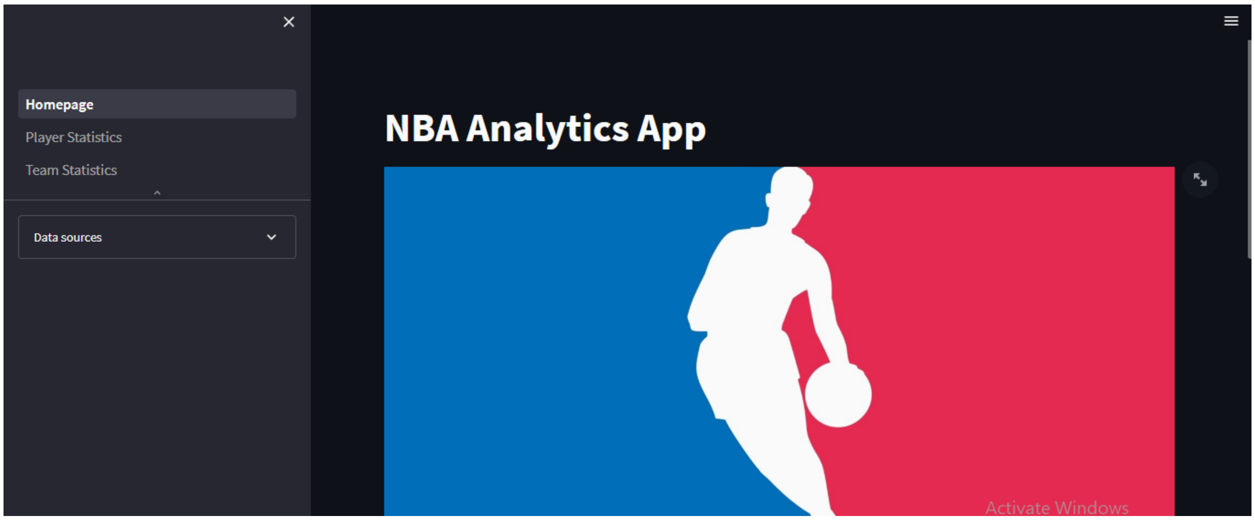
- I then fit this data set into the Prophet model, set the W/L% as the y-axis and the time as the x-axis. The forecast made by the Prophet model is then displayed on to Streamlit

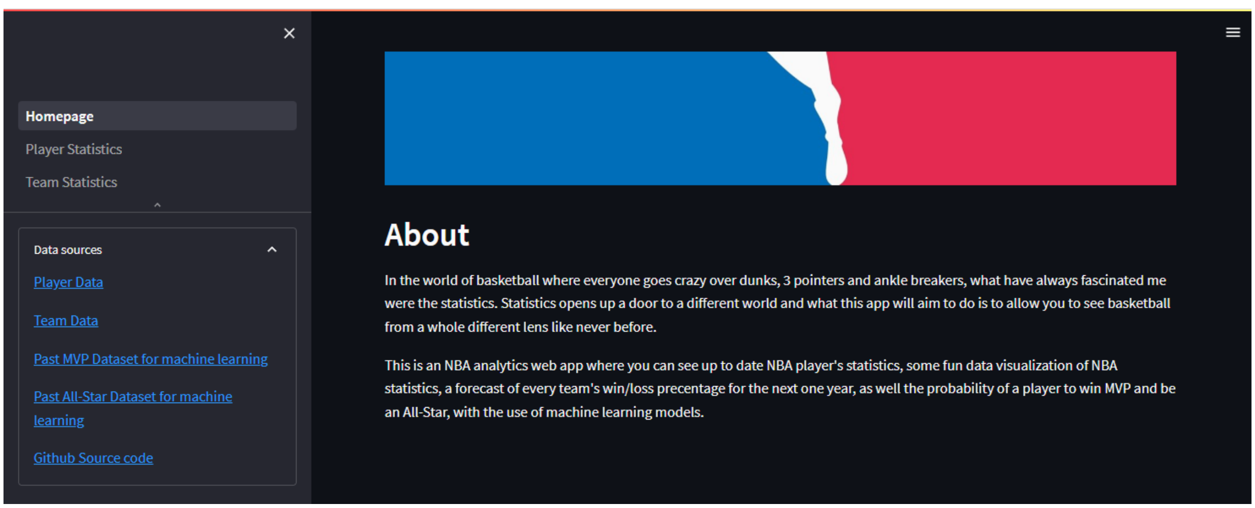
```
#selecting the column needed to show the forecast
timeSeriesTeams= timeSeriesTeams[["TEAM","W_PCT","STANDINGSDATE"]]
timeSeriesTeams = timeSeriesTeams[timeSeriesTeams["TEAM"] == selected_team].drop("TEAM",axis=1)
timeSeriesTeams=timeSeriesTeams.rename(columns={"STANDINGSDATE":"ds", "W_PCT":"y"})
timeSeriesTeams["y"] = timeSeriesTeams["y"]*100

pmodel = Prophet(interval_width=0.95) #set the forecast model
pmodel.fit(timeSeriesTeams) #trained the model
future_dates = pmodel.make_future_dataframe(periods=365)  #sets the period of forecast
forecast = pmodel.predict(future_dates)#forecast
fig = plot_plotly(pmodel, forecast) #plotted the forecast into a line graph
st.plotly_chart(fig) #shows the forecast graph
```

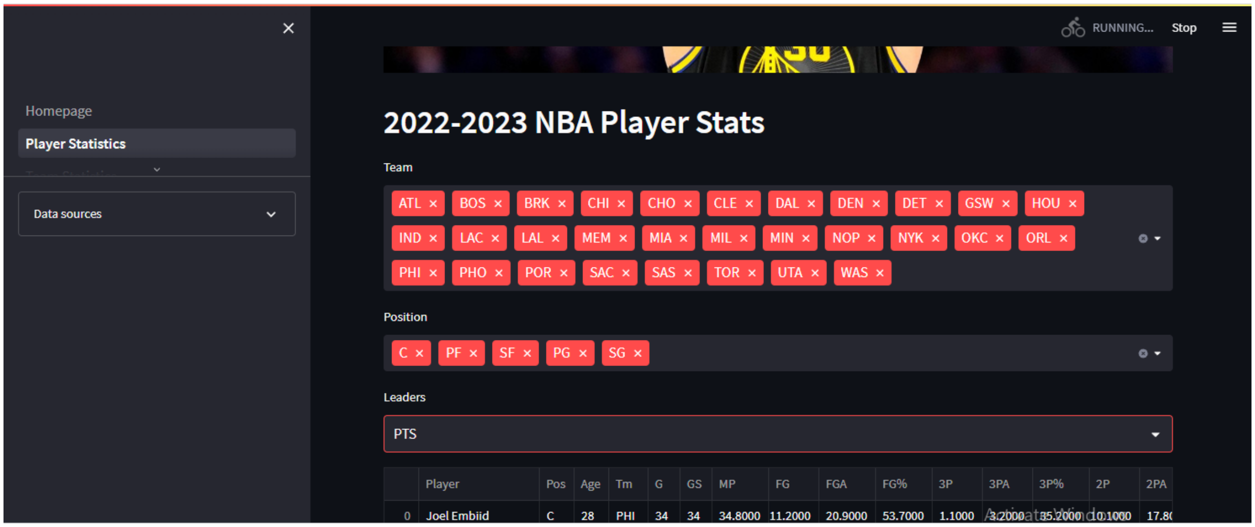
Evidence of a working program

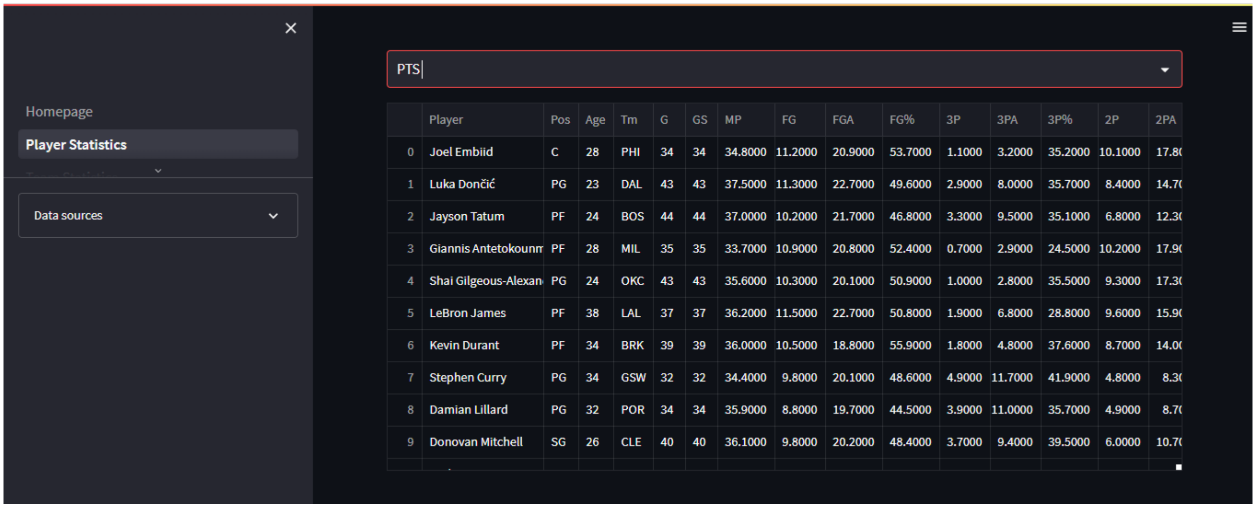
Homepage

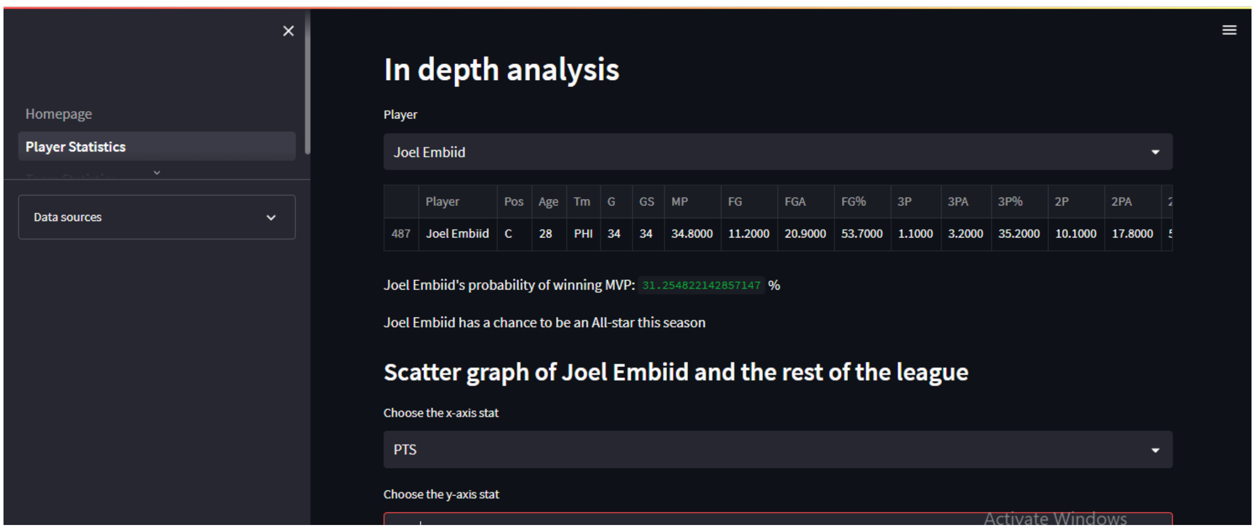




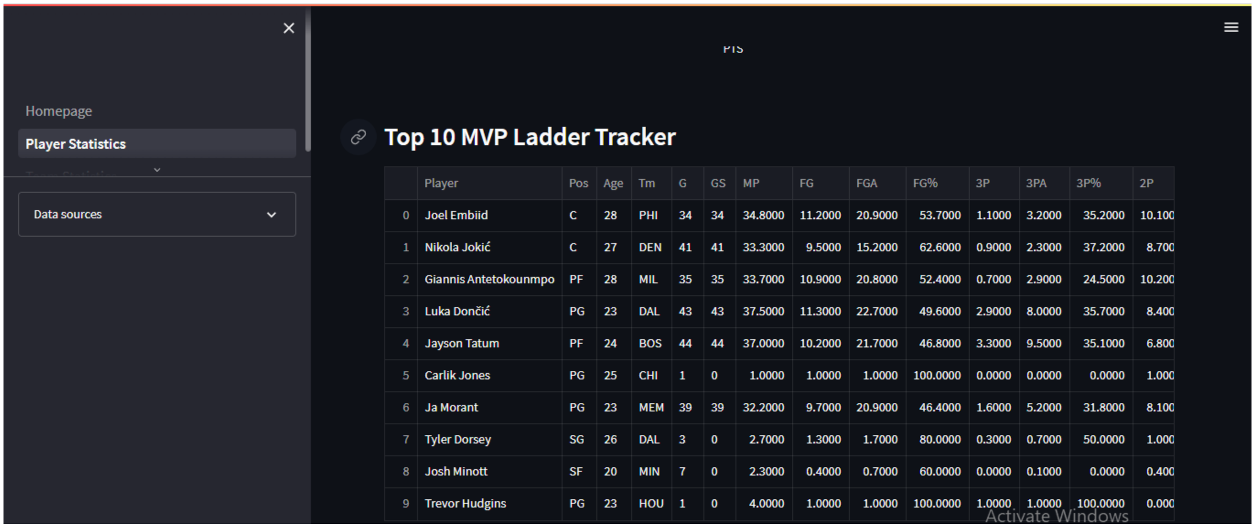
Player Statistics page



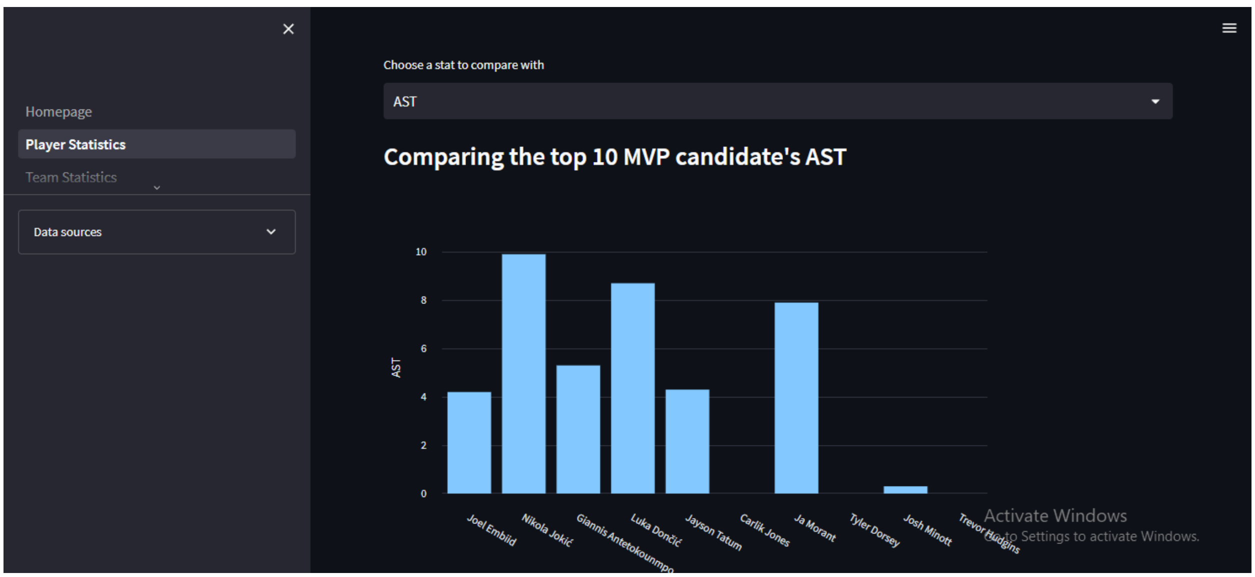

Homepage
Player Statistics
Data sources
PTS
Player Pos Age Tm G GS MP FG FGA FG% 3P 3PA 3P% 2P 2PA
0 Joel Embiid C 28 PHI 34 34 34.8000 11.2000 20.9000 53.7000 1.1000 3.2000 35.2000 10.1000
1 Luka Dončić PG 23 DAL 43 43 37.5000 11.3000 22.7000 49.6000 2.9000 8.0000 35.7000 8.4000
2 Jayson Tatum PF 24 BOS 44 44 37.0000 10.2000 21.7000 46.8000 3.3000 9.5000 35.1000 6.8000
3 Giannis Antetokounm PF 28 MIL 35 35 33.7000 10.9000 20.8000 52.4000 0.7000 2.9000 24.5000 10.2000
4 Shai Gilgeous-Alexan PG 24 OKC 43 43 35.6000 10.3000 20.1000 50.9000 1.0000 2.8000 35.5000 9.3000
5 LeBron James PF 38 LAL 37 37 36.2000 11.5000 22.7000 50.8000 1.9000 6.8000 28.8000 9.6000
6 Kevin Durant PF 34 BRK 39 39 36.0000 10.5000 18.8000 55.9000 1.8000 4.8000 37.6000 8.7000
7 Stephen Curry PG 34 GSW 32 32 34.4000 9.8000 20.1000 48.6000 4.9000 11.7000 41.9000 4.8000
8 Damian Lillard PG 32 POR 34 34 35.9000 8.8000 19.7000 44.5000 3.9000 11.0000 35.7000 4.9000
9 Donovan Mitchell SG 26 CLE 40 40 36.1000 9.8000 20.2000 48.4000 3.7000 9.4000 39.5000 6.0000

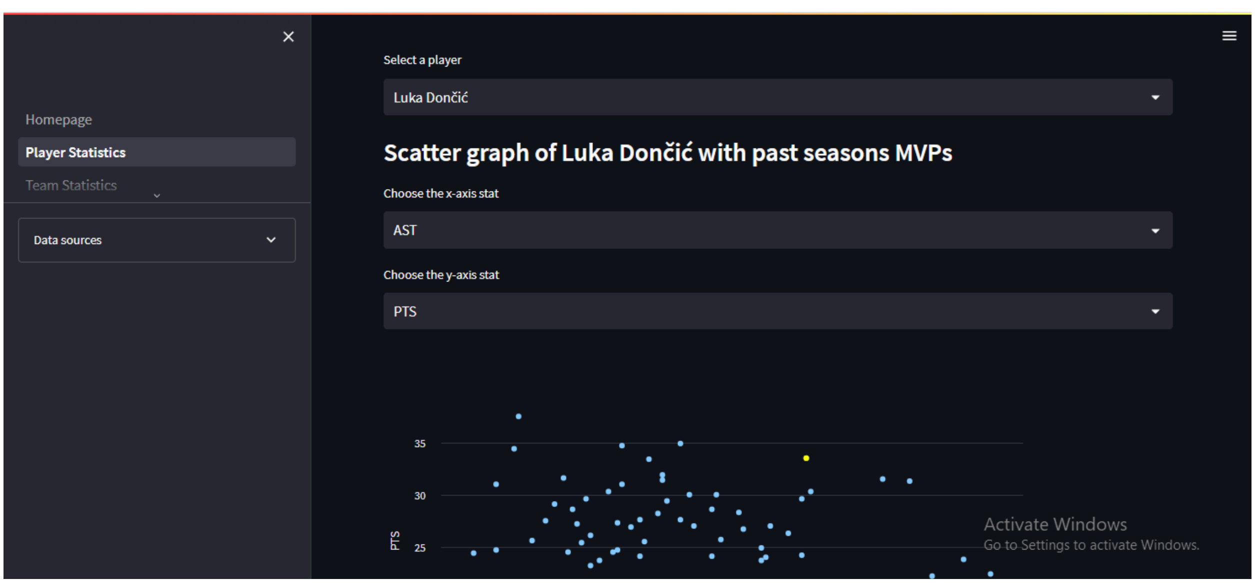

Homepage
Player Statistics
Data sources
In depth analysis
Player
Joel Embiid
Player Pos Age Tm G GS MP FG FGA FG% 3P 3PA 3P% 2P 2PA
487 Joel Embiid C 28 PHI 34 34 34.8000 11.2000 20.9000 53.7000 1.1000 3.2000 35.2000 10.1000 17.8000
Joel Embiid's probability of winning MVP: 31.254822142857147 %
Joel Embiid has a chance to be an All-star this season
Scatter graph of Joel Embiid and the rest of the league
Choose the x-axis stat
PTS
Choose the y-axis stat
Activate Windows


Homepage
Player Statistics
Data sources
Choose the y-axis stat
TRB
TRB
PTS
Activate Windows

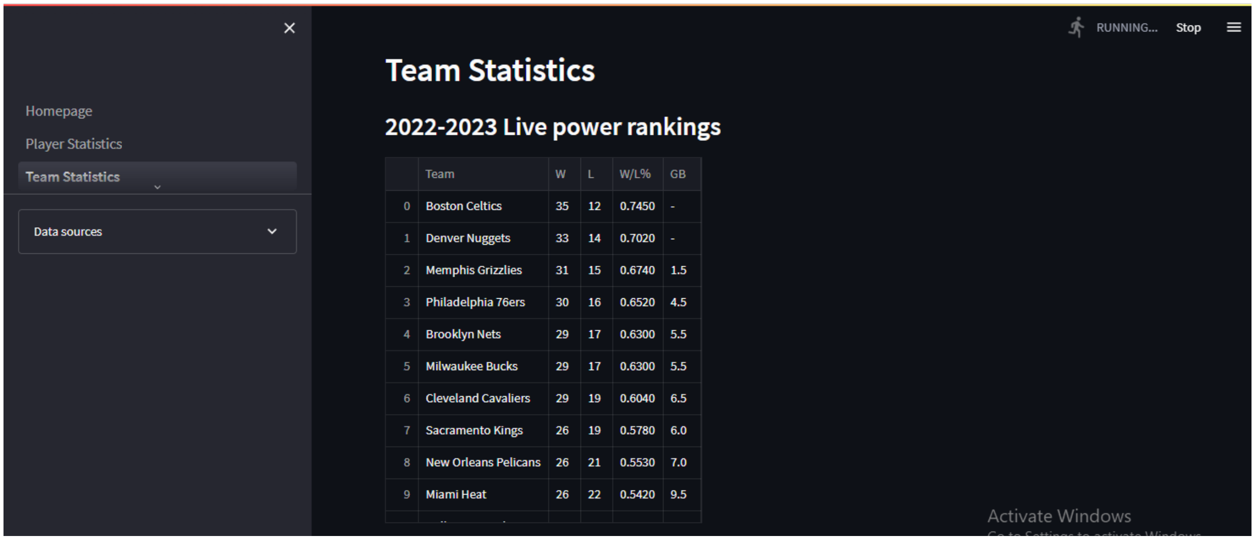


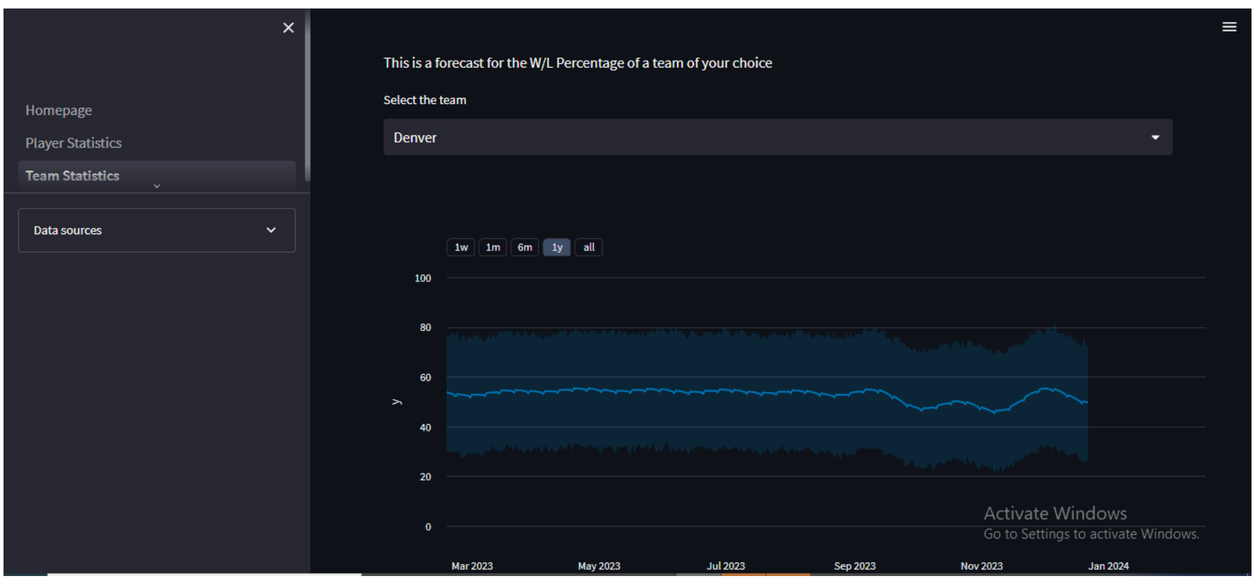
	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P
0	Joel Embiid	C	28	PHI	34	34	34.8000	11.2000	20.9000	53.7000	1.1000	3.2000	35.2000	10.100
1	Nikola Jokić	C	27	DEN	41	41	33.3000	9.5000	15.2000	62.6000	0.9000	2.3000	37.2000	8.700
2	Giannis Antetokounmpo	PF	28	MIL	35	35	33.7000	10.9000	20.8000	52.4000	0.7000	2.9000	24.5000	10.200
3	Luka Dončić	PG	23	DAL	43	43	37.5000	11.3000	22.7000	49.6000	2.9000	8.0000	35.7000	8.400
4	Jayson Tatum	PF	24	BOS	44	44	37.0000	10.2000	21.7000	46.8000	3.3000	9.5000	35.1000	6.800
5	Carlik Jones	PG	25	CHI	1	0	1.0000	1.0000	1.0000	100.0000	0.0000	0.0000	0.0000	1.000
6	Ja Morant	PG	23	MEM	39	39	32.2000	9.7000	20.9000	46.4000	1.6000	5.2000	31.8000	8.100
7	Tyler Dorsey	SG	26	DAL	3	0	2.7000	1.3000	1.7000	80.0000	0.3000	0.7000	50.0000	1.000
8	Josh Minott	SF	20	MIN	7	0	2.3000	0.4000	0.7000	60.0000	0.0000	0.1000	0.0000	0.400
9	Trevor Hudgins	PG	23	HOU	1	0	4.0000	1.0000	1.0000	100.0000	1.0000	1.0000	100.0000	0.000





Team Statistics page





Reflection

Even though I have created a program that I'm satisfied with, I feel that there is still room for lots of improvements and additional features. The MVP prediction is very accurate when it comes to the top 5 candidates, but for the rest however, I feel that there are players who are predicted to be higher or lower than they actually deserve to be. I can see some players who I've never even heard of get a higher probability of being MVP than NBA stars like Kevin Durant. I'm also looking to improve the All-Star prediction, as my current program does not predict the 24 possible all-stars for the season, but only shows if a player has a chance to be an All-Star. Due to this, my program has about 100+ players who are deemed to have a chance to be an All-star. To further improve the usability of my program, I would also like to add more features to the program. I was thinking of showing which NBA legend a selected player is most similar to, forecast the stats of a selected player for the next 3 seasons, show the probability of a player to be inducted to the hall of fame, some more fun data visualization comparing the current season overall to past seasons, and many more - the possibilities are endless. I also feel that Streamlit takes too much time when running my program, and would like to switch to faster frameworks like Django or Dash.

Making this project has taught me so many things, as I took on a project that requires a lot of skills that I have not been taught yet - specifically in the world of Data Science and Machine Learning. I had to learn new foreign libraries such as Pandas, Streamlit, Plotly, Prophet, and Scikit-learn, as well as the function of machine learning models like Ridge regression, Random forest regression, and Logistic Regression. Due to this, I had to dedicate many hours of my time to looking up youtube tutorials, browsing websites such as stackoverflow.com and w3schools.com, and exploring many library documentations. I also encountered a lot of bugs throughout the process of making this project, which improved my problem-solving and research skills, and has also taught me how to be more patient. Implementing a class to my program has also been one of the toughest challenges that I encountered as my program did not need classes, I solved this problem by making a class for the web scraping process. Furthermore, I was also on a holiday abroad when making this project, which significantly lessened the amount of time I can invest. Even though this has been a challenging project for me, I'm really glad I chose to do this. It has been such a rewarding experience that taught me the value of patience, as well as many skills that I would have learned much later on. I hope to not let this go to waste, and further hone my data science skills in the future.

Sources

Data sets - https://www.kaggle.com/

Web scraping

live player data - https://www.basketball-reference.com/

live team data - https://www.landofbasketball.com/

Library documentations

Streamlit - https://docs.streamlit.io/

Plotly - https://plotly.com/python/

Pandas - https://pandas.pydata.org/docs/

Prophet - https://facebook.github.io/prophet/docs/quick_start.html#python-api

Scikit-learn - https://scikit-learn.org/0.21/documentation.html

Matplotlib - https://matplotlib.org/stable/index.html