PYTHON

M1

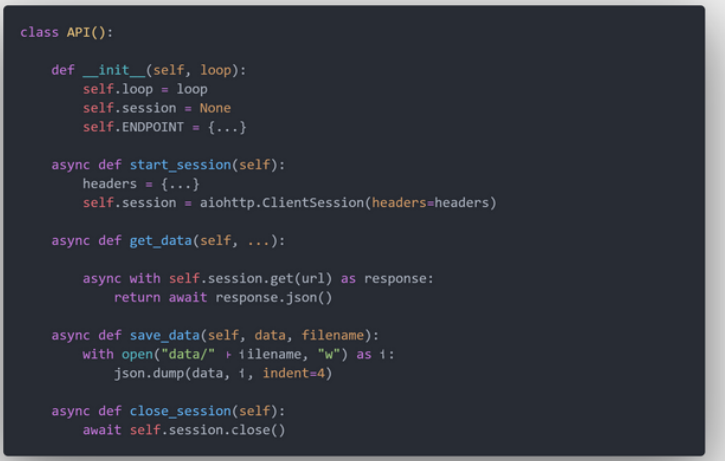
2022 / 2023

BY
CARLOS JIMÉNEZ GARCÍA
PIERRE PINATEAU
JULIA RUIZ CARRERO
FLORENT YANG

STRUCTURE

Explanation of how the program's and classes's layout

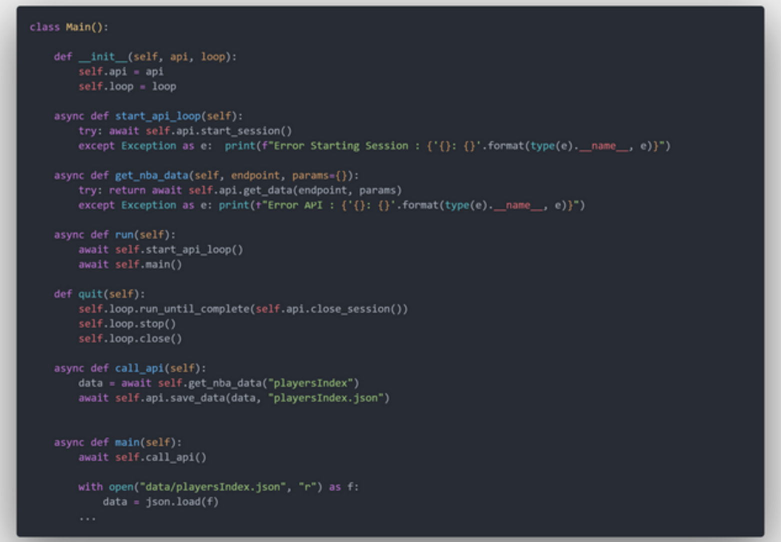
In this file, we have two Class, the first one for the API and the second one for the Main code (plotting, parsing etc.). In the first part, we will explain how the code works globaly and then, the details.



```python
class API():

    def __init__(self, loop):
        self.loop = loop
        self.session = None
        self.ENDPOINT = {...}

    async def start_session(self):
        headers = {...}
        self.session = aiohttp.ClientSession(headers=headers)

    async def get_data(self, ...):

        async with self.session.get(url) as response:
            return await response.json()

    async def save_data(self, data, filename):
        with open("data/" + filename, "w") as f:
            json.dump(data, f, indent=4)

    async def close_session(self):
        await self.session.close()
```

We have 5 functions in the API Class :

- __init__, Python equivalent of the C++ constructor
- start_session, to create aiohttp session with header
- get_data, make the request on an url
- save_data, to..save data in a json format
- close_session, to close the connector

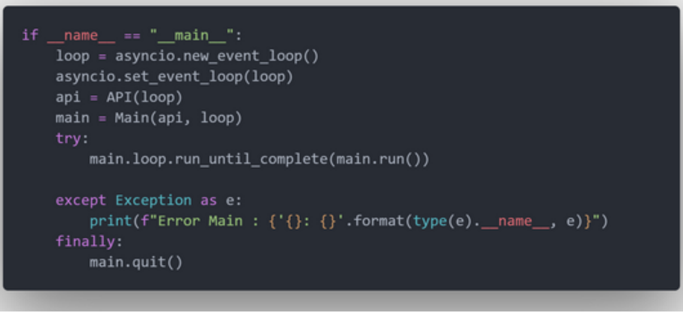


```python
class Main():

    def __init__(self, api, loop):
        self.api = api
        self.loop = loop

    async def start_api_loop(self):
        try: await self.api.start_session()
        except Exception as e:  print(f"Error Starting Session : {'{}: {}'.format(type(e).__name__, e)}")

    async def get_nba_data(self, endpoint, params={}):
        try: return await self.api.get_data(endpoint, params)
        except Exception as e: print(f"Error API : {'{}: {}'.format(type(e).__name__, e)}")

    async def run(self):
        await self.start_api_loop()
        await self.main()

    def quit(self):
        self.loop.run_until_complete(self.api.close_session())
        self.loop.stop()
        self.loop.close()

    async def call_api(self):
        data = await self.get_nba_data("playersIndex")
        await self.api.save_data(data, "playersIndex.json")


    async def main(self):
        await self.call_api()

        with open("data/playersIndex.json", "r") as f:
            data = json.load(f)
        ...
```

In the Main Class we have 6 fonctions (and the __init__)

- start_api_loop
- get_nba_data
- run
- quit
- call_api
- main,

Finally, it remains this small piece of code that allows to launch the whole program.
We find the async loops sent as a parameter to the API() Class and then this same class becomes a parameter of the Main() Class.

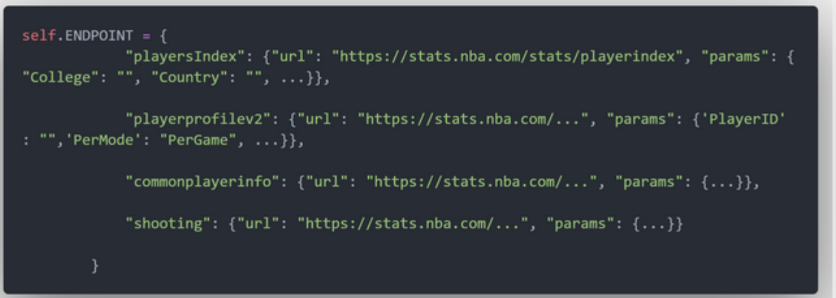


```python
if __name__ == "__main__":
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    api = API(loop)
    main = Main(api, loop)
    try:
        main.loop.run_until_complete(main.run())

    except Exception as e:
        print(f"Error Main : {'{}: {}'.format(type(e).__name__, e)}")
    finally:
        main.quit()
```

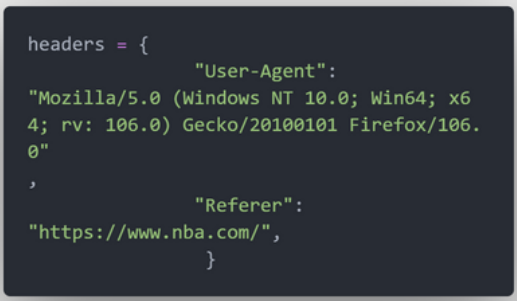
SCRAPPING

Libraries : asyncio, aiohttp, json

For more details at the API level, we find the endpoint URLs used in the NBA site retrieved via network analysis, stored in class attribute in the form of a dictionary.
When the session starts, a session attribute is also created with the minimal headers so that the requests are accepted by the server.

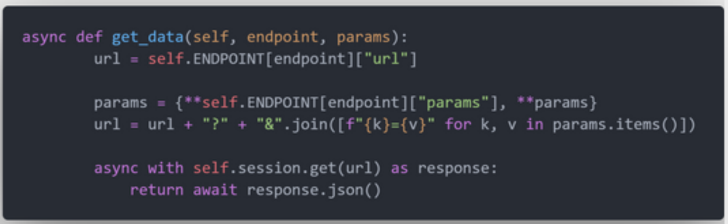


```
self.ENDPOINT = {
            "playersIndex": {"url": "https://stats.nba.com/stats/playerindex", "params": {
"College": "", "Country": "", ...}},

            "playerprofilev2": {"url": "https://stats.nba.com/...", "params": {'PlayerID'
: "",'PerMode': "PerGame", ...}},

            "commonplayerinfo": {"url": "https://stats.nba.com/...", "params": {...}},

            "shooting": {"url": "https://stats.nba.com/...", "params": {...}}

        }
```

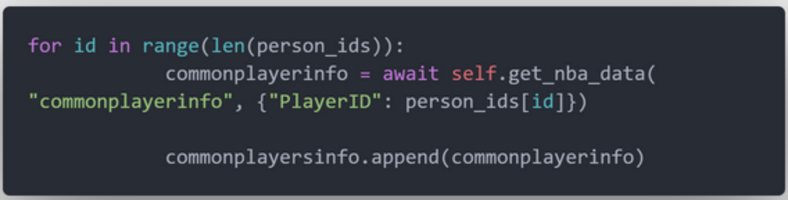


```
headers = {
                "User-Agent":
"Mozilla/5.0 (Windows NT 10.0; Win64; x6
4; rv: 106.0) Gecko/20100101 Firefox/106.
0"
,
                "Referer":
"https://www.nba.com/",
                }
```

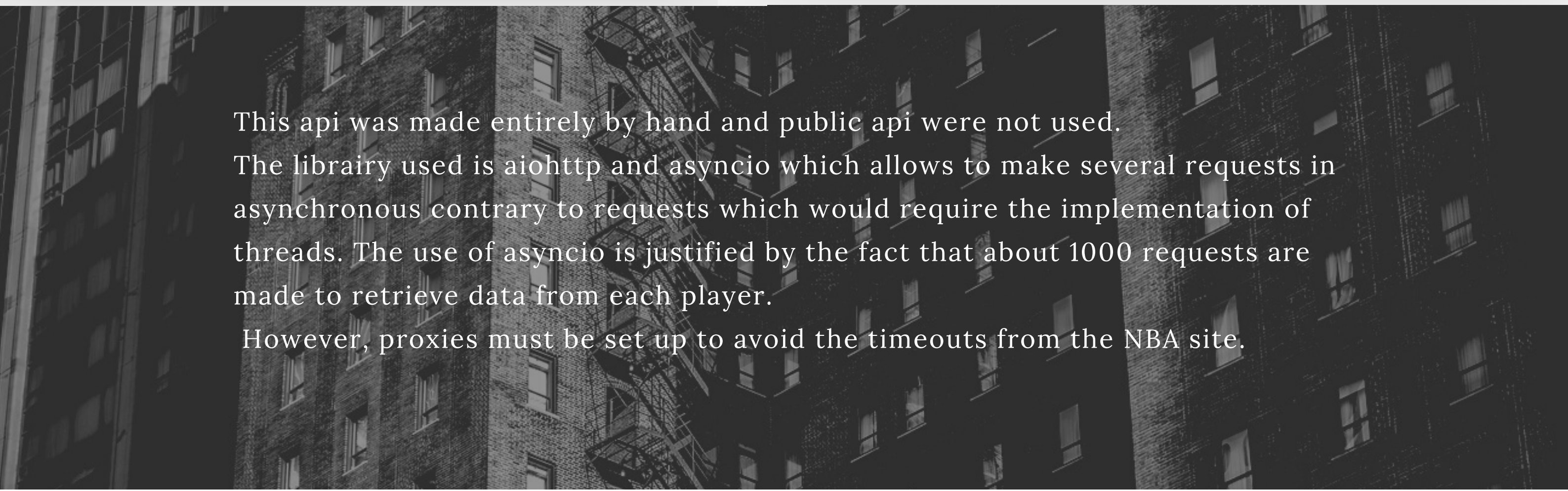
The url is recomposed in the get_data() function with the default parameters indicated in the endpoint. If ever parameters are specified, they will be overiden and will replace the default ones.
An example of a recursive call on the ID of a player can be found on the image below



```
async def get_data(self, endpoint, params):
        url = self.ENDPOINT[endpoint]["url"]

        params = {**self.ENDPOINT[endpoint]["params"], **params}
        url = url + "?" + "&".join([f"{k}={v}" for k, v in params.items()])

        async with self.session.get(url) as response:
            return await response.json()
```



```
for id in range(len(person_ids)):
            commonplayerinfo = await self.get_nba_data(
"commonplayerinfo", {"PlayerID": person_ids[id]})

            commonplayersinfo.append(commonplayerinfo)
```



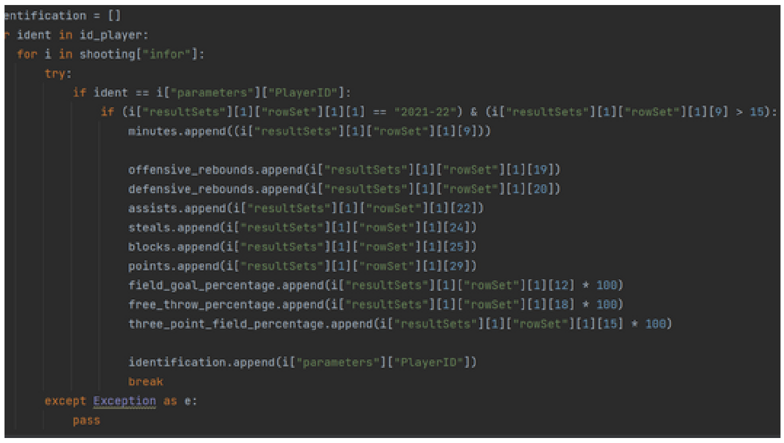
This api was made entirely by hand and public api were not used.
The librairy used is aiohttp and asyncio which allows to make several requests in asynchronous contrary to requests which would require the implementation of threads. The use of asyncio is justified by the fact that about 1000 requests are made to retrieve data from each player.
However, proxies must be set up to avoid the timeouts from the NBA site.

DATA SCIENCE

Based on the information obtained from the web scraping, a series of figures have been elaborated to compare different data. At the same time, the user is allowed to create his own graphs with the desired data so that he can also draw his own conclusions.

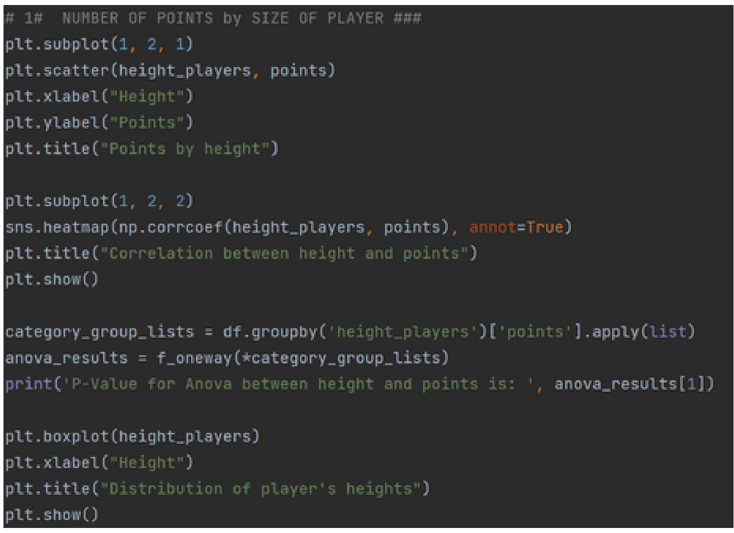
It should be noted that out of all the players analyzed, only those with more than 15 minutes played were selected. If this filter had not been performed, the results obtained would not have been representative, as there would have been players with insignificant statistics that would have been taken into account.
In the code there will be three main parts; the first one used to get all of the values out of the json files; the second one used to plot the variables that were thought to be interesting to see; and the last one that is used to ask the user what variables he will like to plot and with what type of plot he would want to do it.

Example of the first part:



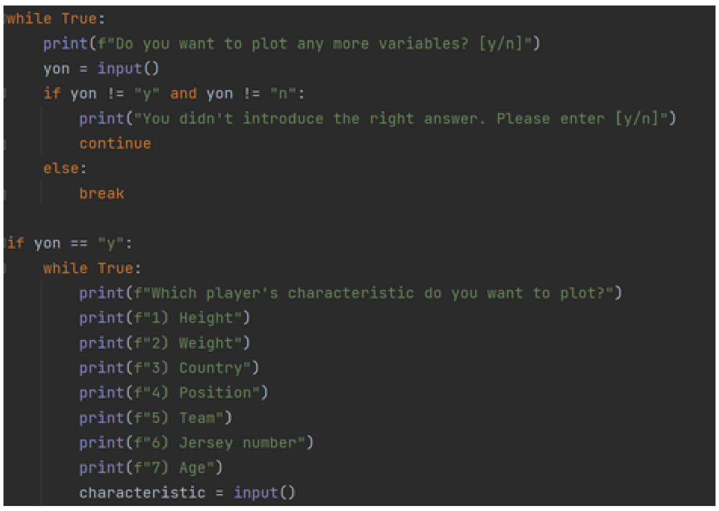
```
entification = []
r ident in id_player:
  for i in shooting["infor"]:
      try:
          if ident == i["parameters"]["PlayerID"]:
              if (i["resultSets"][1]["rowSet"][1][1] == "2021-22") & (i["resultSets"][1]["rowSet"][1][9] > 15):
                  minutes.append((i["resultSets"][1]["rowSet"][1][9]))

                  offensive_rebounds.append(i["resultSets"][1]["rowSet"][1][19])
                  defensive_rebounds.append(i["resultSets"][1]["rowSet"][1][20])
                  assists.append(i["resultSets"][1]["rowSet"][1][22])
                  steals.append(i["resultSets"][1]["rowSet"][1][24])
                  blocks.append(i["resultSets"][1]["rowSet"][1][25])
                  points.append(i["resultSets"][1]["rowSet"][1][29])
                  field_goal_percentage.append(i["resultSets"][1]["rowSet"][1][12] * 100)
                  free_throw_percentage.append(i["resultSets"][1]["rowSet"][1][18] * 100)
                  three_point_field_percentage.append(i["resultSets"][1]["rowSet"][1][15] * 100)

                  identification.append(i["parameters"]["PlayerID"])
                  break
      except Exception as e:
          pass
```

Example of the second part:



```
# 1#  NUMBER OF POINTS by SIZE OF PLAYER ###
plt.subplot(1, 2, 1)
plt.scatter(height_players, points)
plt.xlabel("Height")
plt.ylabel("Points")
plt.title("Points by height")

plt.subplot(1, 2, 2)
sns.heatmap(np.corrcoef(height_players, points), annot=True)
plt.title("Correlation between height and points")
plt.show()

category_group_lists = df.groupby('height_players')['points'].apply(list)
anova_results = f_oneway(*category_group_lists)
print('P-Value for Anova between height and points is: ', anova_results[1])

plt.boxplot(height_players)
plt.xlabel("Height")
plt.title("Distribution of player's heights")
plt.show()
```

Example of the third part:



```
while True:
    print(f"Do you want to plot any more variables? [y/n]")
    yon = input()
    if yon != "y" and yon != "n":
        print("You didn't introduce the right answer. Please enter [y/n]")
        continue
    else:
        break

if yon == "y":
    while True:
        print(f"Which player's characteristic do you want to plot?")
        print(f"1) Height")
        print(f"2) Weight")
        print(f"3) Country")
        print(f"4) Position")
        print(f"5) Team")
        print(f"6) Jersey number")
        print(f"7) Age")
        characteristic = input()
```

In the third part the user can choose between one of the variables that are in the "characteristics" group (weight, height, nationality...) and one between the "statistics" group (points, blocks, assists...).
Depending on what the user chooses he will be able to use different types of plots:

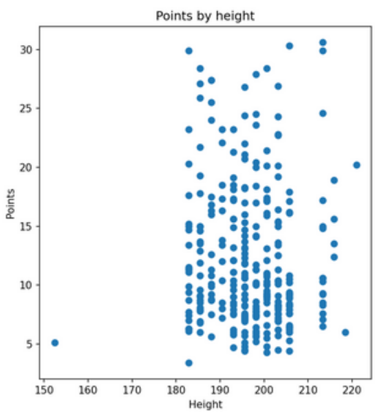
- Scatter plot
- Joint plot
- Histogram
- Box plot
- Correlation

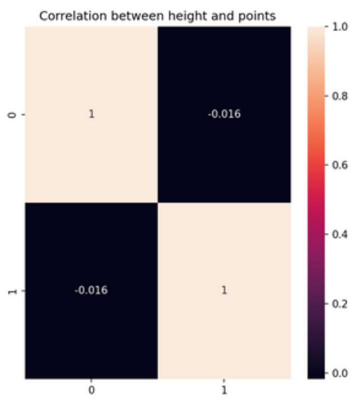
GRAPHICS

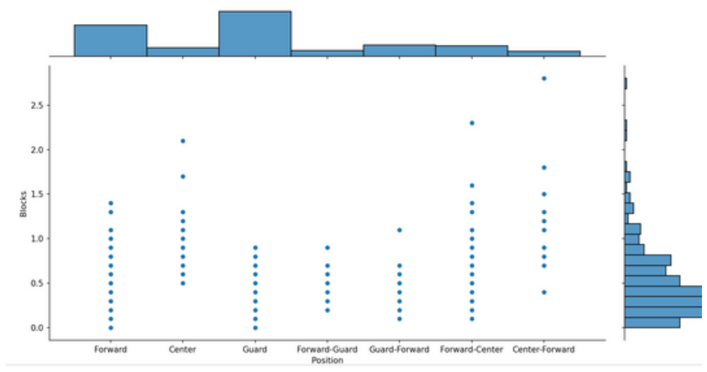
Plotting the variables and analyzing the results

From all of the variables we took from the json file we can differentiate between two types: categorical and numerical.
For the numerical data scatter plots were used to see how the two variables relate to each other but also hetmaps to see if there is a real correlation between the two. This last plot has made it possible to make some conclusions about what it can be seen in the plots. **All of the conclusions can be found in the code.** Also, some histograms were used and boxplots to further explain why the conclusions were taken.

The following plot demonstrates how the information was decided to be shown:





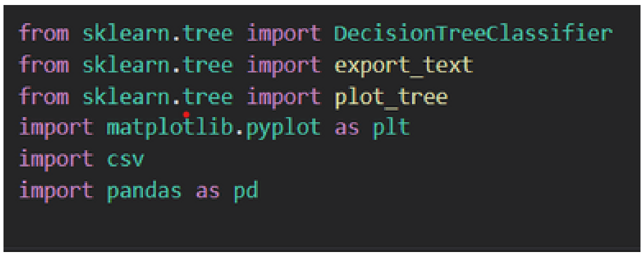


The picture on the left hand side shows a scatter plot between the points made by each player and their height, and next to it can be observed that there is no actual correlation between these two variables since the correlation number is really low.
The picture on the right hand side shows a joint plot between the position each player playes and the number of blocks each player is able to make.
When working with categorical values, it was decided to calculate the p-value between the two variables to see if there were actually realted to each other or not. If the **p-value** is lower than 0.05 then the null hypothesis can be rejected which means that there is a relation between the two variables. This information is **shown trough the output screen**.
When the program runs it will show first all of the plots that were thought to be interesting and, after that, it will ask the user through the screen if he wants to plot anything else. If so, the user must decide what variables and what kind of plot by enetring his decision through the screen.

DECISION TREE

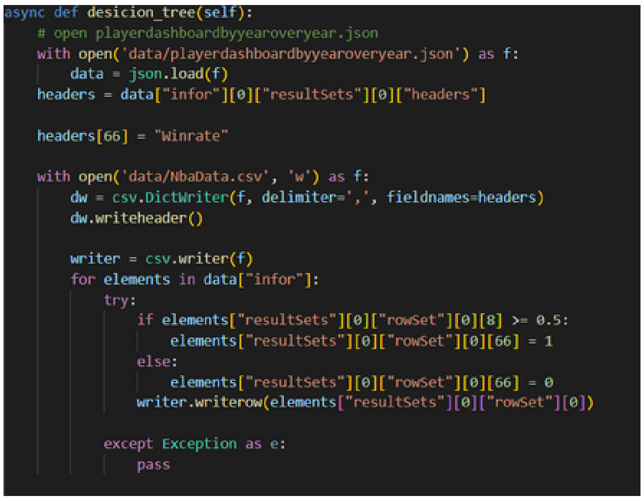
Libraries: csv, json

To build the decision tree, Sklearn.tree will be used

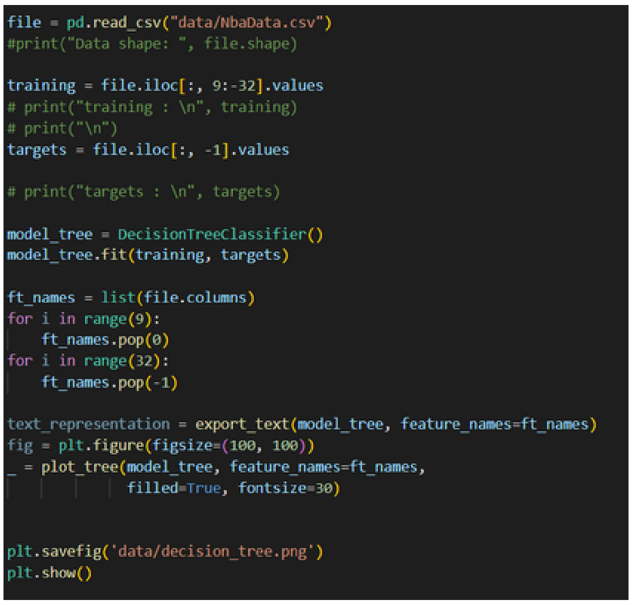


```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import csv
import pandas as pd
```

Then, a json file with all the data that is needed will be used and then converted into csv.
The 67 element will change, if the winrate is 50% or above, the 67 element will be set on 1, else it's 0.
This will set our target



```python
async def desicion_tree(self):
    # open playerdashboardbyyearoveryear.json
    with open('data/playerdashboardbyyearoveryear.json') as f:
        data = json.load(f)
    headers = data["infor"][0]["resultSets"][0]["headers"]

    headers[66] = "Winrate"

    with open('data/NbaData.csv', 'w') as f:
        dw = csv.DictWriter(f, delimiter=',', fieldnames=headers)
        dw.writeheader()

        writer = csv.writer(f)
        for elements in data["infor"]:
            try:
                if elements["resultSets"][0]["rowSet"][0][8] >= 0.5:
                    elements["resultSets"][0]["rowSet"][0][66] = 1
                else:
                    elements["resultSets"][0]["rowSet"][0][66] = 0
                writer.writerow(elements["resultSets"][0]["rowSet"][0])

            except Exception as e:
                pass
```



```python
file = pd.read_csv("data/NbaData.csv")
#print("Data shape: ", file.shape)

training = file.iloc[:, 9:-32].values
# print("training : \n", training)
# print("\n")
targets = file.iloc[:, -1].values

# print("targets : \n", targets)

model_tree = DecisionTreeClassifier()
model_tree.fit(training, targets)

ft_names = list(file.columns)
for i in range(9):
    ft_names.pop(0)
for i in range(32):
    ft_names.pop(-1)

text_representation = export_text(model_tree, feature_names=ft_names)
fig = plt.figure(figsize=(100, 100))
_ = plot_tree(model_tree, feature_names=ft_names,
                   filled=True, fontsize=30)


plt.savefig('data/decision_tree.png')
plt.show()
```

Training : Selection of necessary data to build the tree
Targets : If the winrate is 50% or above, the target is true else, it is false
model_tree : prepare the tree then train it with the training data and targets data

In the end, we plot the decision tree ===>