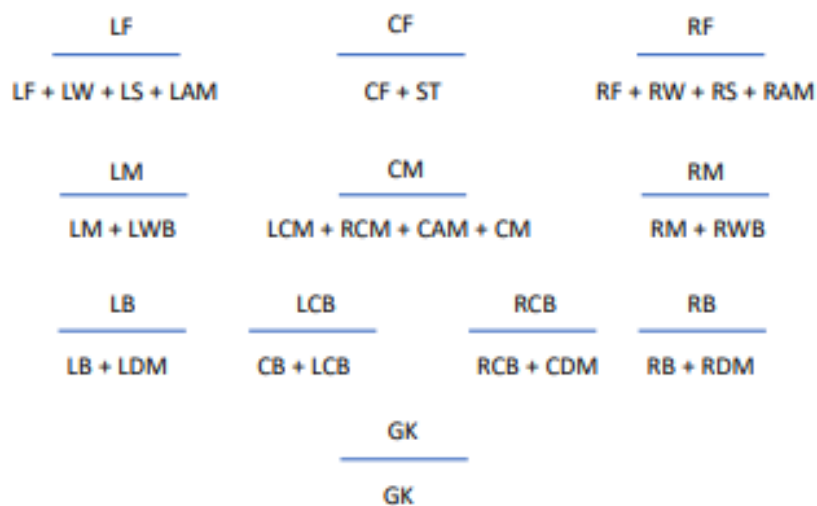


Percepção Sensorial

Final Project FIFA

2022/2023



54436 André Ferreira

53579 Eduardo Dias

52755 Filipe Marques

Introduction

The objective of this project was to create a Python genetic algorithm that retrieves information about the last 8 FIFA databases and selects the best team possible based on their overall and their variance. To do so, we collected data on the players' ratings and statistics from the FIFA games and used a combination of algorithms and manual analysis to determine which players had the highest overall ratings and which team presented the lowest variance.

The resulting team represents the best possible combination of players based on the available data and can be used as a reference for creating a strong team in future FIFA games. In this report, we will describe the methodology that we used to create the script and select the best team, present the results of our analysis, and discuss the implications of our findings. The data used for this project is from the popular game FIFA, particularly from FIFA 15 to FIFA 22. The final objective of this project is to get the best team with the best average overall and low variance when given a certain budget for the transfer market and wages. Through these inputs, it's possible to get the best team for the Caparica Football Club (CFC).

Data filtering

To achieve this objective, first, we filter the data to not only remove unwanted players like reserves or substitutes but also remove players with missing data.

```
year_dict = {
    15 : 0,
    16 : 1,
    17 : 2,
    18 : 3,
    19 : 4,
    20 : 5,
    21 : 6,
    22 : 7
}
budget = 150000000
mensal_budget = 15000000
year = 22
df_alvo = dfs[year_dict[year]]
df_alvo.drop(df_alvo.index[df_alvo['club_position'] == 'SUB'], inplace=True)
df_alvo.drop(df_alvo.index[df_alvo['club_position'] == 'RES'], inplace=True)
df_alvo.dropna(axis=0, how='any', thresh=None, subset=['value_eur'], inplace=True)
df_alvo = df_alvo.reset_index(drop=True)

df_alvo.sort_values(by='value_eur', ascending=False)
df_aux = df_alvo.filter(['value_eur', 'club_position'], axis=1)
df_aux = df_aux.tail(30)
min_val_player = (budget/11) * 0.1
df_alvo = df_alvo[df_alvo['value_eur'] > min_val_player]
#df_alvo = df_alvo[df_alvo['value_eur'] > 0]
df_alvo = df_alvo.reset_index(drop=True)
df_alvo
```

After that, it's defined as a minimum value to be considered being spent on a single player. We do this because it will filter out very low-value players and it will increase the performance of our program. After filtering all the data, we only consider 4 important statistics from the player, their overall, value on the transfer market, their wage value, and the position in which he plays for their club. This is the only data that we need to get the best-valued team.

Algorithm used

For this project, we use a genetic algorithm to get the best team possible. This algorithm can be used to analyze data on players to determine the optimal combination of players to form a team. The algorithm starts by getting a group of randomly generated teams, and then use the principles of natural selection to choose the fittest team based on some measure of team performance. These fittest teams then are used to create a new generation of teams through combinations and mutation of their player data. This process is then repeated over multiple generations until a satisfactory team is found.

Why utilized this algorithm? Well, genetic algorithms are commonly used in optimization problems because they can search through a very large space of potential solutions in a relatively efficient manner. They are particularly useful for problems where the space of possible solutions is too large to search exhaustively, or where there is no known algorithm for finding the optimal solution.

Application of algorithm

Considering that we are using a genetic algorithm as explained before, we first start by making a list of 10 random teams based on the budget that we have initially. For this project, we decided to divide the budget to be spent equally amongst players, so every position has the same quantity available to be spent.

```
In [22]: def make_random_lists(): # faz 20 equipas
list_of_teams = []
list_val = []
val_per_player = budget/11 # 900 000
wage_per_player = mensal_budget/11
for i in range(10):
    team = []
    doPosList()
    max_val = 0
    max_wage = 0
    z = 0
    while (len(team) != 11): # 930 000
        player_num = random.randint(0, len(df_alvo)-1)
        if (pos_dict[df_alvo['club_position']][player_num] == 1):
            continue
        if (len(team) == 10):
            resto1 = (max_val - df_alvo['value_eur'][player_num])
            resto2 = (max_wage - df_alvo['wage_eur'][player_num])
        else:
            resto1 = (val_per_player - df_alvo['value_eur'][player_num])
            resto2 = (wage_per_player - df_alvo['wage_eur'][player_num])
        if (resto1 < 0 or resto2 < 0):
            if (z >= 3000):
                pos_dict[df_alvo['club_position']][player_num] = 1
                team.append(player_num)
                max_val += df_alvo['value_eur'][player_num]
                max_wage += df_alvo['wage_eur'][player_num]
                z = z+1
            continue
        else:
            pos_dict[df_alvo['club_position']][player_num] = 1
            team.append(player_num)
            max_val += df_alvo['value_eur'][player_num]
            max_wage += df_alvo['wage_eur'][player_num]
        #print(team)
    list_of_teams.append(team)
    list_val.append(max_val)
for team in list_of_teams:
    print("Team: " + str(team) + " Spend: " + str(teamCost(team)[0]) + " Save: " + str(teamCost(team)[1]) + " Ov: " + str(
#return List_of_teams
make_random_lists()
```

```
Team: [2021, 2923, 3399, 7, 3901, 1207, 896, 3830, 2959, 2841, 1561] Spend: 48200000.0 Save: 101800000.0 Ov: 3965.92
Team: [2720, 2252, 1137, 3851, 3671, 3448, 3832, 2415, 1836, 2077, 2302] Spend: 23100000.0 Save: 126900000.0 Ov: 3897.04
Team: [2716, 3610, 2885, 2929, 1282, 3608, 637, 3598, 828, 2106, 2604] Spend: 27400000.0 Save: 122600000.0 Ov: 3674.7
Team: [776, 869, 1709, 2594, 2943, 3072, 2078, 3882, 1262, 1895, 1427] Spend: 43800000.0 Save: 106200000.0 Ov: 3947.44
Team: [2516, 1608, 2278, 3673, 1284, 3551, 2361, 1327, 3591, 2044, 1434] Spend: 25500000.0 Save: 124500000.0 Ov: 3617.55
Team: [813, 1068, 1175, 3168, 2857, 3885, 1304, 3865, 1971, 700, 891] Spend: 45000000.0 Save: 105000000.0 Ov: 3949.76
Team: [3243, 3902, 2334, 2100, 3591, 1803, 3850, 1115, 2110, 1443, 2274] Spend: 30400000.0 Save: 119600000.0 Ov: 3966.6
Team: [2632, 2739, 1571, 1005, 3850, 2151, 2784, 574, 2962, 853, 3293] Spend: 36300000.0 Save: 113700000.0 Ov: 3914.89
Team: [2819, 1980, 3175, 890, 3454, 1357, 2715, 1037, 3617, 2944, 3367] Spend: 32100000.0 Save: 117900000.0 Ov: 3683.87
Team: [1470, 3187, 2226, 810, 1090, 839, 3205, 3814, 2134, 2330, 1614] Spend: 37300000.0 Save: 112700000.0 Ov: 3878.61
```

After we generate these 10 teams, we proceed to evaluate them to get the two best teams in terms of their overall/variance, meaning that we get the two teams with the biggest overall and lowest variance.

In the figure beneath it is possible to observe out of the list of teams, the ones that presented the best score, which means high overall and low variance, where the teams positioned in the third and second position of the team's list. The following 2 numbers are respectively their overalls.

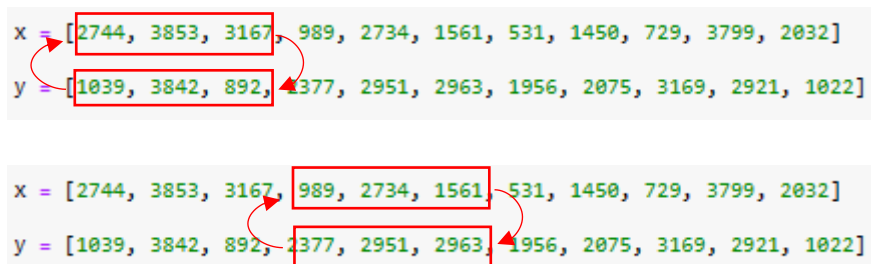
```
In [24]: def avalFun(varList): # avaliar a performance de cada equipa
    best_score = 0
    bestSec_score = 0
    average = 0
    variance = 0
    score = 0
    i = 0
    bestPos = 0
    bestSegPos = 0
    team_tuple = []
    for team in varList:
        if type(team) is int:
            break
        for x in team:
            team_tuple.append(df_alvo['overall'][x])
        average = sum(team_tuple) / len(team_tuple)
        variance = statistics.variance(team_tuple)
        #print('variance' + str(variance))
        score = average * 0.5 + (1 - variance) * 0.5
        if score > best_score:
            bestSec_score = best_score
            best_score = score
            bestSegPos = bestPos
            bestPos = i
        i = i + 1
    if(len(varList) > 2):
        return (bestPos,bestSegPos,best_score,bestSec_score)
    else:
        return (bestPos,best_score)

listt = [[1211, 3357, 1399, 1158, 1510, 839, 2190, 3724, 396, 853, 2117],
[1447, 3408, 394, 2872, 2703, 1470, 2839, 2847, 2020, 1398, 3668],
[3318, 2587, 826, 2816, 3776, 2706, 659, 2000, 2400, 3416, 2447],
[942, 526, 3143, 2569, 3850, 2580, 2450, 1048, 2802, 490, 2202],
[2785, 1301, 1711, 1024, 167, 3111, 1197, 1327, 3452, 1613, 237],
[421, 3465, 3670, 1277, 2929, 2322, 805, 3650, 3453, 3917, 320],
[1201, 1271, 3251, 1343, 2461, 3468, 2739, 1248, 3721, 2858, 1265],
[3117, 3127, 530, 2949, 1583, 2798, 3583, 2477, 2759, 3832, 3640],
[2243, 2438, 1884, 3828, 2864, 2485, 3601, 813, 1916, 2097, 1014],
[2805, 1076, 798, 3360, 2628, 1242, 3235, 3206, 3099, 1296, 830]]
avalFun(listt)

Out[24]: (2, 1, 30.481329852029123, 30.411101741744854)
```

After that, we run a combination and mutation process with these two resulting teams to see if we can get a better team from these.

The figure presented beneath demonstrates a case where the original team was disrupted into better ones. For this, there is a cycle and for each value of that cycle, the team will be cut in different positions. For our specific case, there is only 3 combinations:



The diagram illustrates two combinations of teams, x and y, with specific segments highlighted and arrows indicating a cycle of mutations.

Combination 1:

- x = [2744, 3853, 3167, 989, 2734, 1561, 531, 1450, 729, 3799, 2032]
- y = [1039, 3842, 892, 2377, 2951, 2963, 1956, 2075, 3169, 2921, 1022]

Combination 2:

- x = [2744, 3853, 3167, 989, 2734, 1561, 531, 1450, 729, 3799, 2032]
- y = [1039, 3842, 892, 2377, 2951, 2963, 1956, 2075, 3169, 2921, 1022]

Arrows indicate a cycle of mutations between the highlighted segments in the two combinations.

```

x = [2744, 3853, 3167, 989, 2734, 1561, 531, 1450, 729, 3799, 2032]
y = [1039, 3842, 892, 2377, 2951, 2963, 1956, 2075, 3169, 2921, 1022]

```

For each cycle, the new team formed is re-assessed to determine whether it fits the budget or not, and if improved the quality of the team. If all these factors are verified the team passes to the next phase, otherwise the function is the previous function before the commutations.

```

In [72]: def trocaPlayers(x,y,budget): # trocar jogadores entre duas equipas
    tempEquipa1=[]
    tempEquipa2=[]
    antigoMelhorOv = aux_fun(x)
    antigoSecOv = aux_fun(y)
    ov1 = antigoMelhorOv
    ov2 = antigoSecOv
    equipaMomento = []
    equipaMomento2 = []
    for i in range(0,10,3):
        equipa1=x
        equipa2=y
        equipa1 = posJog(equipa1)
        equipa2 = posJog(equipa2)

        tempEquipa1[0: i+3 ] = equipa2[0: i+3 ]
        tempEquipa1[ i+3 :11] = equipa1[ i+3 :11]
        tempEquipa2[0: i+3 ] = equipa1[0: i+3 ]
        tempEquipa2[ i+3 :11] = equipa2[ i+3 :11]

        if ((budget - teamCost(tempEquipa1)[0]) > 0):
            if ov1 < aux_fun(tempEquipa1):
                ov1 = aux_fun(tempEquipa1)
                equipaMomento = tempEquipa1.copy()
            else: continue
        if ((budget - teamCost(tempEquipa2)[0]) > 0):
            if aux_fun(tempEquipa2) > ov2:
                ov2 = aux_fun(tempEquipa2)
                equipaMomento2 = tempEquipa2.copy()
            else: continue

        if ov1 < antigoMelhorOv :
            if ov2 < antigoSecOv :
                return [equipaMomento, equipaMomento2];
            else:
                return [equipaMomento,y];
        else:
            if ov2 > antigoSecOv :
                return [x, equipaMomento2];
            else:
                return [x,y];
    x = [2744, 3853, 3167, 989, 2734, 1561, 531, 1450, 729, 3799, 2032]
    y = [1039, 3842, 892, 2377, 2951, 2963, 1956, 2075, 3169, 2921, 1022]
    trocaPlayers(x,y,budget)

```

```

Out[72]: [[2744, 3853, 3167, 989, 2734, 1561, 531, 1450, 729, 3799, 2032],
          [1561, 3167, 2032, 2951, 1022, 2377, 2921, 2075, 3842, 3169, 1039]]

```

Particularly, in this case, it is possible to understand that the first team remained still which indicates that a better overall or fit budget was obtained. The second team, in another hand, suffered a mutation of the players **[1561 , 3167 , 2032]** which means that the new players brought to the team a good contribution. From this last step, we save the best team, and we repeat the whole process 15 times. After this, we have a resulting list of 15 teams.

With these 15 teams, we get the two best teams from the list, again based on their high overall and low variance and then we repeat the above-presented process of mutations and combinations. This process is done 2 times with the objective of fine-tuning at maximum each team so that in the next steps less computational burden is required. After this 2-stage process is finished, we get the best team. What we noticed is that this best team doesn't always spend the maximum amount of budget possible, so we go for each position, and based on the amount that we still have left on the budget (save), we try and get better players for each position if possible.

```
for i in overallJog(twobest[betterOne]):
    rslt_df = df_alvo[df_alvo['club_position'] == df_alvo['club_position'][i[1]]]
    rslt_df = rslt_df[rslt_df['overall'] > i[0]]
    rslt_df = rslt_df.reset_index(drop=True)
    rslt_df.sort_values(by='overall', ascending=False)
    novo_valor_per_player = df_alvo['value_eur'][i[1]] + valor_para_melhorias/x
    x = x - 1*0.4
    #print(x)
    for ind in rslt_df.index:
        if (rslt_df['value_eur'][ind] <= novo_valor_per_player):
            final_team.append(rslt_df['sofifa_id'][ind])
            positions_occupied.append(rslt_df['club_position'][ind])
            break
    if df_alvo['club_position'][i[1]] not in positions_occupied:
        final_team.append(df_alvo['sofifa_id'][i[1]])
return final_team
```

This process makes a separate df from each position of the team and sorts it descending. The value to spend with each player is highlighted in the picture. The only aspect relevant is that the value to spend with each player gets bigger with each iteration to avoid a huge amount of saved money.

Finally, to run the program, we run the two processes explained before ten times, getting 10 resulting "best" teams. From these 10 teams, we calculate their respective score based on their variance and overall, with the formula:

$$\text{score} = \text{calc_var}(\text{team})[0] * 0.50 + (1 - \text{calc_var}(\text{team})[1]) * 0.50$$

The team that gets the highest score is considered the best team overall and is the one chosen by our algorithm.

```
%%time
def main():
    final_team = []
    aux = []
    i = 0
    best_score = 0
    if(budget > 1380000): # 22
        while(i < 10):
            team = second_iter(first_iter())
            if (teamCost2(team)[1] > 0):
                score = calc_var(team)[0] * 0.50 + (1 - calc_var(team)[1]) * 0.50
                if score > best_score:
                    best_score = score
                    final_team = team
            else:
                continue
            i = i + 1
        print(best_score)
        print(teamCost2(final_team))
        return display_team(final_team)
    else:
        print("Não dá para fazer uma equipa com os jogadores disponíveis com um budget inferior a 1380000 euros")
    main()
```

Results

We tested the algorithm for the various FIFA databases, and we noticed that the lower the value we have available for our budget, the higher the time needed to run. This happens because there are a lot more players in the database for lower values than higher values in terms of wage and transfer values. We ran numerous tests with the various databases, and we got run times ranging from 1 second (high budget) to 1.5 minutes (low budget). This highly depends not only on the budget chosen by the user but also on the FIFA database used, since some databases contain more players than others.

With a budget of 150M in 2022 database:

```
Team's score: 41.334835657977074
Team's cost: 141400000.0 total save: 141400000.0
CPU times: total: 1.8 s
Wall time: 2.75 s
```

```
Out[212]: [['K. Huntelaar',
            'CF',
            'S. Gerrard',
            'LB',
            'N. Gaitán',
            'LM',
            'A. Candreva',
            'RF',
            'Xabi Alonso',
            'CM',
            'J. Terry',
            'LCB',
            'R. Weidenfeller',
            'GK',
            'Jesús Navas',
            'RM',
            'Dani Alves',
            'RB',
            'P. Mertensacker',
            'RCB',
            'Nani',
            'LF']]
```

With a budget of 150M in 2015 database:

```
Team's score: 41.55435088191645
Team's cost: 143400000.0 total save: 143400000.0
CPU times: total: 1.3 s
Wall time: 2.61 s
```

```
Out[173]: [['Jesús Navas',
            'RM',
            'N. Gaitán',
            'LM',
            'K. Huntelaar',
            'CF',
            'S. Gerrard',
            'LB',
            'Pedro',
            'RF',
            'E. Lavezzi',
            'LF',
            'J. Terry',
            'LCB',
            'Dani Alves',
            'RB',
            'R. Weidenfeller',
            'GK',
            'Xabi Alonso',
            'CM',
            'P. Mertensacker',
            'RCB']]
```

With a budget of 15M in 2022 database:

```
Team's score: 36.95345370176326
Team's cost: 13625000.0 total save: 13625000.0
CPU times: total: 1.83 s
Wall time: 2.7 s
```

```
Out[103]: [['A. Hurtado',
            'LM',
            'Enaldo Praz',
            'LB',
            'Salva Sevilla',
            'RB',
            'M. Ninković',
            'LF',
            'G. Bergessio',
            'RF',
            'Muriqui',
            'CF',
            'Dante',
            'LCB',
            'G. Cahill',
            'RCB',
            'M. Pérez García',
            'RM',
            'C. Pellerano',
            'CM',
            'M. Stekelenburg',
            'GK']]
```

With a budget of 15M in 2015 database:

```
Team's score: 35.585495823807776
Team's cost: 13750000.0 total save: 13750000.0
CPU times: total: 1.48 s
Wall time: 2.15 s
```

```
Out[160]: [['J. Morales',
            'CM',
            'C. Jallet',
            'RB',
            'J. Villar',
            'GK',
            'A. Di Natale',
            'RF',
            'Álvaro Rubio',
            'LB',
            'F. Totti',
            'CF',
            'J. Elmander',
            'LF',
            'T. Henry',
            'LM',
            'T. Buffel',
            'RM',
            'R. Márquez',
            'RCB',
            'W. Brown',
            'LCB']]
```

With a budget of 1,5M in 2022 database:

```
Team's score: 17.73704704352268
Team's cost: 1255000.0 total save: 1255000.0
CPU times: total: 52.9 s
Wall time: 1min 39s
```

```
Out[121]: [['A. Mannus',
            'GK',
            'N. Cabrera',
            'LCB',
            'E. Gutiérrez',
            'LF',
            'J. Machado',
            'RB',
            'Liu Le',
            'LM',
            'C. Skuse',
            'CM',
            'J. Álvarez',
            'LB',
            'N. Topor-Stanley',
            'RCB',
            'Li Xiang',
            'CF',
            'B. Singh',
            'RF',
            'Wang Fei',
            'RM']]
```

With a budget of 150M in 2015 database:

```
Team's score: 27.988908883955997
Team's cost: 1350000.0 total save: 1350000.0
CPU times: total: 3.14 s
Wall time: 5.3 s
```

```
Out[139]: [['Jesús Vázquez',
            'LB',
            'J. Broerse',
            'LCB',
            'Tamudo',
            'CF',
            'A. Głowacki',
            'RCB',
            'R. Foran',
            'RB',
            'L. Bernardi',
            'CM',
            'R. Lowe',
            'RF',
            'C. Ochoa',
            'LF',
            'C. Edwards',
            'RM',
            'T. Andreassen',
            'LM',
            'A. Čović',
            'GK']]
```

Conclusion

By analyzing data on the players' ratings and statistics from the last 7 FIFA games, we were able to identify the best possible team based on overall and variance. Our methodology, which involved using a combination of algorithms and manual analysis, proved effective in selecting a well-balanced team with strong potential for future games.

Through the analysis of the results, it is possible to observe that as the budget goes down, also the score (high overall and small variance) of the team goes down. This happens because with less budget the players bought are not as good as they could be. Also with our methodology, we realized that the 2022 dataset is the densest to explore which will produce much higher values than expected.

In some work, it would be interesting to consider other factors that may impact team strength, such as player positions and chemistry. It would also be useful to evaluate the performance of the selected team in actual games. Overall, this project has provided valuable insights into the process of selecting a strong team for FIFA games.