Big Data aplicada als escacs

I. Introducció

Els escacs son un dels jocs de taula més antics de la humanitat. En alguns documents tant antics com de principis del segle 7 DC es poden trobar referències a aquest joc. Cada dia milions de persones hi juguen, la majoria d'ells en línia.
Una cosa més antiga que els escacs és fer trampes. És per això que hi ha hagut vegades en que fins i tot han esquitxat un joc tan maco com el prèviament mencionat. Per a fer trampes només cal jugar els moviments del nostre oponent contra un ordinador (com Stockfish, el millor sistema del món d'escacs) i respondre a la nostra partida tal i com ho fa l'ordinador.

Com a persona que ràpidament va veure's fascinada per ells i que ha jugat milers de partides quan va veure que un dels possibles DataSets que es presentaven era d'escacs no va tenir problema en escollir aquest tema ja que li apassionava. En aquestes partides també m'he trobat amb persones que no jugaven de manera legítima i per això he enfocat el meu projecte en aquesta direcció.

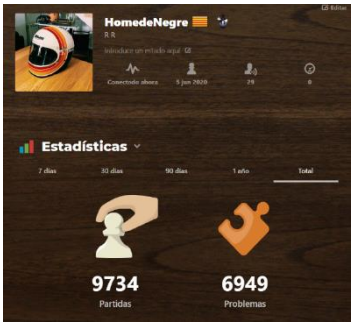


Fig. 1. Un dels meus perfils on hi ha jugades 9734 partides

II. Estructura del projecte

El projecte està dividit en tres parts: la primera es basa en l'estudi de l'Elo com a sistema de puntuació per a classificar els jugadors d'escacs, posteriorment estudiarem els moviments que tant blanques com negres juguen de manera més sovint i finalment s'analitzarà els usuaris del DataSet que han tingut comportaments sospitosos. Després de les explicacions es mostrarà el codi i s'explicarà que fa cada part.

III. Elo i Elo clustering

El sistema de puntuació Elo és un mètode matemàtic, basat en càlcul estadístic, per calcular l'habilitat relativa dels jugadors. Va ser dissenyat per Árpád Élő, que va ser president de la Federació Americana d'Escacs, va dissenyar un mètode de puntuació estadístic que va ser començat a ser utilitzat al voltant dels anys 60 pels americans i a partir dels anys 70 va ser considerat el sistema universal de puntuació pels escacs.

És un sistema en el qual cada jugador rep una qualificació entre 0 i 3000 (alguns ordinadors superen aquest límit de manera notòria però no ens centrarem en aquest aspecte) que determina com de fort és el jugador. Com més proper a 3000 més fort és un jugador. Quan dos jugadors s'enfronten es posen els dos números (un de cada jugador) en una fórmula on es determina quants punts guanyarà cadascú si guanya i quants en perdrà en cas contrari. El jugador amb més Elo guanyarà menys per derrotar algú amb menys Elo però en perdrà més en cas de ser derrotat.

Un cop analitzats els jugadors del DataSet podem observar el següent:

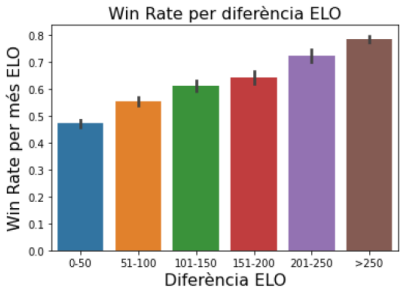


Fig. 2. Winrate per diferència d'Elo

Un cop analitzades les dades podem observar com efectivament a mesura que la diferència de nivell entre jugadors es va fent gran més possibilitats de victòria per al que té un nivell superior. Això concorda de manera exacta amb el mencionat prèviament sobre el sistema de puntuació.

Posteriorment es va decidir realitzar Clustering per poder visualitzar com estaven repartits els jugadors del nostre DataSet.

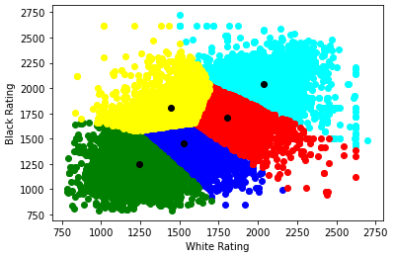


Fig. 3. Elo de Negres vs Elo de Blanques

I es pot veure que tot i que hem emprat 5 centroïdes cap d'elles està centrada per sobre de 2250. Això es deu a que a mesura que augmentem el nivell es redueixen el número de jugadors que poden assolir-lo. També és important veure aquesta distribució perquè els jugadors que arriben en aquests nivells solen ser persones titulades i que juguen amb comptes verificats. Aquestes persones, tot i que no sempre, solen conèixer-se en la vida real i no s'arriscarien a fer trampes ja que podria tenir repercussions en els seus tornejos.

IV. Apertures i respostes

En una partida d'escacs hi ha 20 possibles primers moviments (20 per a blanques i 20 per a negres). Tot i això, un cop han passat 3 moviments ja en tenim 9 milions. Després de 4 ja en tenim 288 bilions.
Davant de l'absurda quantitat de possibilitats a la que ens estem enfrontant estudiarem els primer moviment, per a blanques i per a negres.

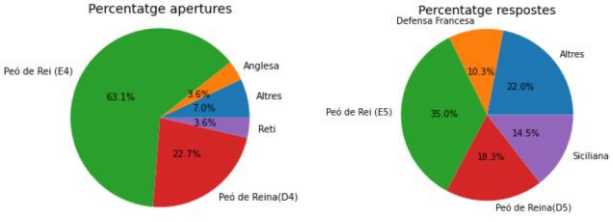


Fig. 4. A l'esquerra tenim el moviment més comú per a blanques i la dreta el moviment més comú per a negres.

En observar les gràfiques anteriors es pot clarament veure com la majoria de jugadors decideixen optar per obrir amb un peó de Rei o de Reina i que la resposta sol ser un peó de Rei o de Reina. En aquí es va intentar utilitzar algoritmes d'apriori per poder obtenir les seqüencies de moviments més comunes però la partició no va poder ser assolida de manera apropiada i acabàvem repetint normes o unificant moviments de diverses partides per lo que es va decidir per obviar-ho.

També es van realitzar diverses gràfiques per veure quin percentatge de victòries havien obtingut les apertures i respostes anteriors:

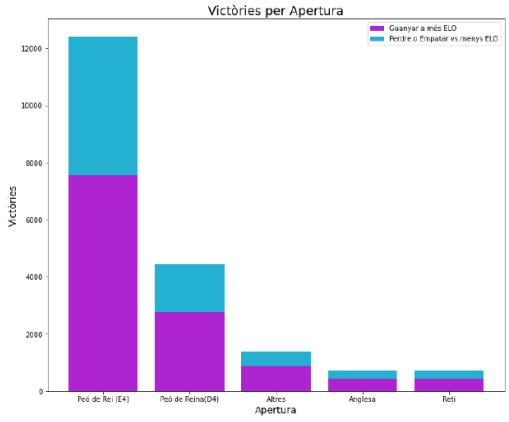


Fig. 5. Rendiment d'apertures

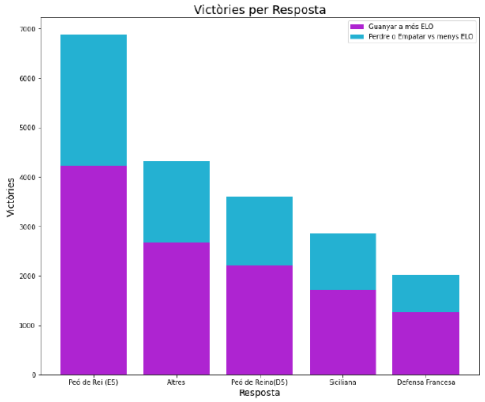


Fig. 6. Rendiment de respostes

Davant d'un moviment, com ja hem mencionat abans, es pot respondre de diferents maneres. És per això que també podem veure les diferents respostes i quin resultat han obtingut.

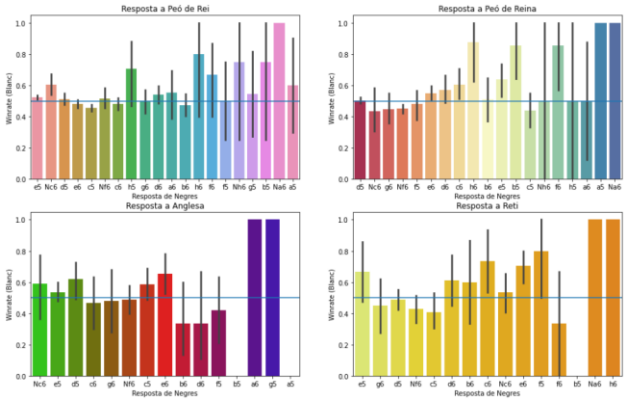


Fig. 7. Respostes a les apertures

Cal mencionar que en aquestes gràfiques hi ha resultats que poden portar a conclusions errònies. Això es deu a que, per exemple en la segona gràfica podem observar com respondre a D4 (Peó de Reina) amb Na6 té un 100% de winrate però resulta un moviment anecdòtic i que ningú jugaria. Que es donés en una ocasió i acabés guanyant no implica que sigui la millor resposta. De fet, si introduïm aquesta seqüència de moviments a Stockfish:

Fig. 8. Tauler representatiu amb Stockfish evaluant

La barra de l'esquerra indica l'avantatge. En una partida completament igualada (al principi d'una partida, abans del primer moviment per exemple) obtenim 0.0. En aquesta situació però podem veure que instantàniament les blanques tenen avantatge ja que el moviment de les negres no acaba de ser un bon moviment.

V. Usuaris

Un cop analitzats els primers moviments mirarem els jugadors del nostre DataSet. Per això mirarem els winrates i intentarem veure perquè certs usuaris han obtingut tantes victòries i perquè altres han perdut tants cops. Cal indicar que fer trampes no sempre vol dir guanyar, potser també vol dir perdre a propòsit.

Endreçant-los per winrate, de major a menor obtenim els següents resultats:

white_id	white_victories	white_games	black_games	black_victories	victories	games_played	win_pct
siindbad	13	13	13	13	26	26	1
chesscarl	18	18	28	27	45	46	0.978261
lzchips	12	13	13	13	25	26	0.961538
nitsua49	1	20	26	4	5	46	0.10869
kylecuver1	0	12	15	2	2	27	0.07407
andreschil	1	14	18	1	2	32	0.0625

Fig. 9. Usuaris amb més winrate i amb menor

Com és possible que alguns usaris hagin guanyat 26 de 26 o 45 de 46 i altres hagin guanyat només 5 de 46 o 2 de 32?

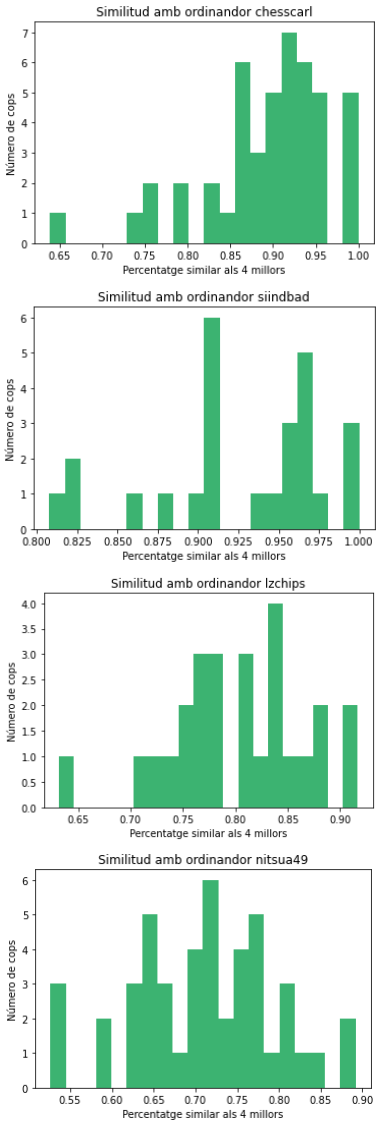
Per tal de poder analitzar aquests usuaris en profunditat primer mirarem el nivell dels seus rivals i posteriorment passarem totes les seves partides per Stockfish i compararem els seus moviments amb els 4 millors moviments.

```
Siindbad is a higher rating in 17 of their 26 games.
Average rating above opponents when higher-rated: 262.11764705882354
Average rating below opponents when lower-rated: 67.0
Chesscarl is a higher rating in 43 of their 46 games.
Average rating above opponents when higher-rated: 355.48837209302326
Average rating below opponents when lower-rated: 68.0
Lzchips is a higher rating in 26 of their 26 games.
Average rating above opponents when higher-rated: 664.1153846153846
Average rating below opponents when lower-rated: nan
Andreschil is a higher rating in 1 of their 33 games.
Average rating above opponents when higher-rated: 86.0
Average rating below opponents when lower-rated: 354.96875
Nitsua49 is a higher rating in 0 of their 46 games.
Average rating above opponents when higher-rated: nan
Average rating below opponents when lower-rated: 299.19565217391306
```

Fig. 10. Comparatives d'Elo entre els jugadors amb més winrate i menor.

En la figura anterior podem observar com els jugadors que més winrate tenien solien tenir més de 250 punts de diferència amb el seu rival i com ja hem vist a la Figura 2 té lògica. El contrari pels jugadors amb menys winrate, que quan jugaven contra algú més fort que ells sempre tenien una gran diferència.

Finalment compararem les seves precisions, per veure en què es poden diferenciar aquests jugadors. Aquesta comparativa comporta un alt cos computacional, fent que cada gràfica trigués uns 45 minuts com a mínim per obtenir-se.



Figures 11-14: Precisió de 4 dels 6 usuaris prèviament observats

Podem observar com les precisions dels jugadors amb més Elo estan centrades en valors més grans, propiciant que guanyessin més. Així doncs podem concloure que aquests jugadors no estaven fent trampes, simplement jugaven contra algú més fluix.

VI. Dificultats i possibles millores

Al llarg del projecte van trobar-se dificultats que van afectar a com el projecte va realitzar-se. La principal de totes va ser l'apriori, que com s'ha mencionat no va poder-se aplicar de la manera que es volia.

```
Rule White: d4 c4 cxd5d4 e4 f4  -> d4 Nf3 Nc3e4 Nf3 d4
Support White: 0.10489936979060785
Confidence White: 0.41222288795686046
Lift White: 3.3851826141231998
====================================
Rule White: d4 c4 cxd5d4 e4 f4  -> d4 Nf3 Nc3e4 Nf3 d4
Support White: 0.10489936979060785
Confidence White: 0.41222288795686046
Lift White: 3.3851826141231998
====================================
Rule White: d4 c4 cxd5d4 e4 f4  -> d4 Nf3 Nc3e4 Nf3 d4
Support White: 0.10367960967676357
Confidence White: 0.4074295985620132
Lift White: 3.392545400468122
====================================
Rule White: d4 c4 cxd5d4 e4 f4  -> d4 Nf3 Nc3e4 Nf3 d4
Support White: 0.10367960967676357
Confidence White: 0.4074295985620132
Lift White: 3.392545400468122
====================================
```

Fig. 15. Errors en l'apriori

Les possibles millores del projecte serien, entre d'altres, la correcta implementació de l'apriori, reduir el cost computacional de les gràfiques i intentar aconseguir alguna manera d'avaluar les partides mentre s'estan jugant i no un cop ja han sigut jugades ja que això podria permetre a algú que faci trampes jugar un cert número de partides.

VII. Codi

Finalment mostrarem el codi emprat per a la obtenció de tots els resultats obtinguts. En el codi es poden observar comentaris per tal d'entendre què es realitza en cada part i en cada moment, així com de solucions a errors que han anat sorgint.

```
# In[1]:

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np # linear algebra
import matplotlib.pyplot as plt
import seaborn as sns
#from apyori import apriori

from stockfish import Stockfish
import chess
import chess.engine
import math
from math import sqrt

#On tinc Stockfish al meu ordinador
#S'ha de posar r davant de la ubicació perquè sinó sortia un Unicode Error al llegir-ho.

engine = chess.engine.SimpleEngine.popen_uci(r"C:\Users\Usuario\Desktop\Bigdata\Chess\stockfish_15
stockfish = Stockfish(r'C:\Users\Usuario\Desktop\Bigdata\Chess\stockfish_15_win_x64_avx2\stockfish_
stockfish.set_elo_rating(2600)
```

```
# In[2]:

#Llegim el dataset d'escacs

df = pd.read_csv('chess.csv')
```

```
# In[3]: Clustering


X = df[["white_rating", "black_rating"]]
K = 5;
Centroids = (X.sample(n=K))
plt.scatter(X["white_rating"], X["black_rating"], c = "green")
plt.scatter(Centroids["white_rating"], Centroids["black_rating"], c = "black")
plt.xlabel("White rating")
plt.ylabel("Black rating")
plt.show()
```

```
# In[4]: Clustering adaptant-se.
# Cel·la lenta, triga certs minuts a executar-se però funciona completament.

pd.options.mode.chained_assignment = None #Sortia un warning molest que no afectava i així e

diff = 1
j=0

while(diff!=0):
    XD=X
    i=1
    for index1, row_c in Centroids.iterrows():
        ED=[]
        for index2, row_d in XD.iterrows():
            d1 = (row_c.loc["white_rating"]-row_d.loc["white_rating"])**2
            d2 = (row_c.loc["black_rating"]-row_d.loc["black_rating"])**2
            d = sqrt(d1+d2)
            ED.append(d)
        X[i] = ED
        i = i+1

    C = []
    for index, row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos = i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new = X.groupby(["Cluster"]).mean()[["white_rating", "black_rating"]]
    if j == 0:
        diff = 1
        j = j+1
    else:
        diff = (Centroids_new['white_rating'] - Centroids['white_rating']).sum() + (Centroid

    Centroids = X.groupby(["Cluster"]).mean()[["white_rating","black_rating"]]
```

```
color=['blue','green','cyan','red','yellow']
for k in range(K):
    data=X[X["Cluster"]==k+1]
    plt.scatter(data["white_rating"],data["black_rating"],c=color[k])
plt.scatter(Centroids["white_rating"],Centroids["black_rating"],c='black')
plt.xlabel('White Rating')
plt.ylabel('Black Rating')
# plt.show()
```

```
# In[5]:

#Partides amb pocs moviments
df = df[df.turns >= 6]


# Creem una nova sèrie amb tots el smoviments, com a string ens serveix per a poc

df['moves_list'] = df.moves.apply(lambda x: x.split())


#Creem noves columnes, amb el primer, segon, tercer, quart, cinquè i sisè moviment

df['Apertura'] = df.moves_list.apply(lambda x: x[0])
df['response'] = df.moves_list.apply(lambda x: x[1])
df['Apertura1'] = df.moves_list.apply(lambda x: x[2])
df['response1'] = df.moves_list.apply(lambda x: x[3])
df['Apertura2'] = df.moves_list.apply(lambda x: x[4])
df['response2'] = df.moves_list.apply(lambda x: x[5])

#Combinem els primers tres moviments per al blanc i per al negre

df['3W'] = df['Apertura'] + ' ' + df['Apertura1'] + ' ' + df['Apertura2']
df['3B'] = df['response'] + ' ' + df['response1'] + ' ' + df['response2']

df['3First'] = df['Apertura'] + ' ' + df['response'] + ' ' + df['Apertura1'] + ' ' + df['r

#print(df['3W'])
#print(df['3B'])
```

```
# In[6]: Masses errors, per això ho he deixat comentat.


# association_rules = apriori(df['3W'], min_support=0.1, min_confidence=0.2, min_lift=3.39, m

# for i in range (0, len(df['3W'])):
#         pair = df['3W']
#         pair[i] = df['3W']

# for item in association_rules:
#         items = [x for x in pair]
#         print("Rule White: " + items[0] + items[1] + items[2]+ items[3] + items[4] + items[
#         print("Support White: " + str(item[1]))
#         print("Confidence White: " + str(item[2][0][2]))
#         print("Lift White: " + str(item[2][0][3]))
#         print("====================================")




# association_rules_black = apriori(df['3B'], min_support=0.01, min_confidence=0.2, min_lift=

# for item in association_rules_black:
#         pair = df['3B']
#         items = [x for x in pair]
#         print("Rule Black " + (items[0])) # items[1] + items[2])) #+ #items[3]) + (items[4]
#         print("Support Black: " + str(item[1]))
#         print("Confidence Black: " + str(item[2][0][2]))
#         print("Lift Black: " + str(item[2][0][3]))
#         print("====================================")



# association_rules_first = apriori(df['3First'], min_support=0.075, min_confidence=0.2, min_
```

```
# In[7]:


#Nova columna anomentant les obertures
df['opening_name'] = df.moves_list.apply(lambda x: 'Peó de Rei (E4)' if x[0] == 'e4' else ('Peó de Reina(D4
df['response_name'] = df.moves_list.apply(lambda x: 'Peó de Rei (E5)' if x[1] == 'e5' else ('Peó de Reina(D

#Diferència de rating
df['rating_gap'] = abs(df['white_rating'] - df['black_rating'])

#Ha guanyat el que més ELO té?
df['higher_rated_victory'] = np.where((df['winner'] == 'White') & (df['white_rating'] > df['black_rating']) |

#Classifiquem la diferència d'ELO.
df['rating_gap_class'] = df.rating_gap.apply(lambda x: '0-50' if (x <= 50) else ('51-100' if (x > 50 and x

#Ha guanyat el blanc?
df['white_victory'] = np.where(df['winner'] == 'White', 1, 0)

#Ha guanyat negres?

df['black_victory'] = np.where(df['winner'] == 'Black', 0, 1)

#5 apertures per a blanc

english = df[df.Apertura == 'c4']
queens_pawn = df[df.Apertura == 'd4']
kings_pawn = df[df.Apertura == 'e4']
reti = df[df.Apertura == 'Nf3']
fianchetto = df[df.Apertura == 'g3']

#4 apertures per a les negres
kings_pawn1=df[df.response == 'e5']
queens_pawn1= df[df.response == 'd5']
french= df[df.response == 'e6']
siciliana= df[df.response == 'c5']


#Apertures més emprades pels jugadors

opening_data = df.groupby('opening_name')['game_id'].count()
plt.pie(x=opening_data, autopct='%.1f%%', labels=opening_data.keys(), pctdistance=0.5)
plt.title('Percentatge apertures', fontsize=14)
plt.axis('equal') #Per a que sigui rodona
plt.show()

#Respostes més emprades

response_data = df.groupby('response_name')['game_id'].count()
plt.pie(x=response_data, autopct='%.1f%%', labels=response_data.keys(), pctdistance=0.5)
plt.title('Percentatge respostes', fontsize=14)
plt.axis('equal') #Per a que sigui rodona
plt.show()


# Creem un nou data frame per veure qui va guanyar
english_winner_data = english.groupby('winner')['game_id'].count()
queen_winner_data = queens_pawn.groupby('winner')['game_id'].count()
king_winner_data = kings_pawn.groupby('winner')['game_id'].count()
reti_winner_data = reti.groupby('winner')['game_id'].count()
fianchetto_winner_data = fianchetto.groupby('winner')['game_id'].count()


#Creem un nou data frame per veure qui va guanyar

french_winner_data = french.groupby('winner')['game_id'].count()
queen1_winner_data = queens_pawn1.groupby('winner')['game_id'].count()
king1_winner_data = kings_pawn1.groupby('winner')['game_id'].count()
siciliana_winner_data = siciliana.groupby('winner')['game_id'].count()
```

```
pie, axs = plt.subplots(2,2, figsize=[10,10])

plt.subplot(2,2,1)
plt.pie(x=english_winner_data, autopct="%.1f%%", labels=english_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar obrint amb C4", fontsize=8)
plt.axis('equal')


plt.subplot(2,2,2)
plt.pie(x=queen_winner_data, autopct="%.1f%%", labels=queen_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar obrint amb D4", fontsize=8)
plt.axis('equal')

plt.subplot(2,2,3)
plt.pie(x=king_winner_data, autopct="%.1f%%", labels=king_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar obrint amb E4", fontsize=8)
plt.axis('equal')

plt.subplot(2,2,4)
plt.pie(x=reti_winner_data, autopct="%.1f%%", labels=reti_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar obrint amb NF3", fontsize=8)
plt.axis('equal')

plt.show()

plt.subplot(3,3,5)
plt.pie(x=fianchetto_winner_data, autopct="%.1f%%", labels=fianchetto_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyadors després d'obrir amb G6", fontsize=7)
plt.axis('equal')
plt.show()
```

```
pie, axs = plt.subplots(2,2, figsize=[10,10])

plt.subplot(2,2,1)
plt.pie(x=french_winner_data, autopct="%.1f%%", labels=french_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar defensant amb E6", fontsize=8)
plt.axis('equal')


plt.subplot(2,2,2)
plt.pie(x=queen1_winner_data, autopct="%.1f%%", labels=queen1_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar defensant amb D5", fontsize=8)
plt.axis('equal')

plt.subplot(2,2,3)
plt.pie(x=king1_winner_data, autopct="%.1f%%", labels=king1_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar defensant amb E5", fontsize=8)
plt.axis('equal')

plt.subplot(2,2,4)
plt.pie(x=siciliana_winner_data, autopct="%.1f%%", labels=siciliana_winner_data.keys(), pctdistance=0.5)
plt.title("Guanyar defensant amb C5", fontsize=8)
plt.axis('equal')

plt.show()
```

```
# In[8]:


#Total de victòries de blanc agrupades per apertura

df_grouped_ratings = df.groupby('opening_name')['higher_rated_victory'].sum()
df_grouped_ratings = df_grouped_ratings.to_frame() #Convertim de sèries a dataframe.

#Total de victòries de negres agrupades per resposta

df_grouped_ratings1 = df.groupby('response_name')['higher_rated_victory'].sum()
df_grouped_ratings1 = df_grouped_ratings1.to_frame() #Convertim de sèries a dataframe.


#Nova columna amb el número de partides totals

df_grouped_ratings['totals'] = df.groupby('opening_name')['higher_rated_victory'].count()


#Nova columna amb derrotes/empats

df_grouped_ratings['losses_or_draws'] = df_grouped_ratings['totals'] - df_grouped_ratings['higher_rated_victory']
print(df_grouped_ratings.head(10))


#Afegim una columna amb el número total

df_grouped_ratings1['totals'] = df.groupby('response_name')['higher_rated_victory'].count()

#Afegim una columna per a derrotes/empats

df_grouped_ratings1['losses_or_draws'] = df_grouped_ratings1['totals'] - df_grouped_ratings1['higher_rated_victory']
print(df_grouped_ratings1.head(10))
```

```
df_grouped_ratings = df_grouped_ratings.sort_values('totals', ascending=False)
fig, ax = plt.subplots(1, figsize=(12,10))
ax.bar([0,1,2,3,4], df_grouped_ratings['higher_rated_victory'], label='Guanyar a més ELO', color='#ae24d1', tick_label=df_grouped_ratings
ax.bar([0,1,2,3,4], df_grouped_ratings['losses_or_draws'], label='Perdre o Empatar vs menys ELO', bottom=df_grouped_ratings['higher_rated
ax.set_ylabel('Victòries', fontsize=14)
ax.set_xlabel('Apertura', fontsize=14)
ax.set_title('Victòries per Apertura', fontsize=18)
ax.legend()
plt.show()

#Quines respostes dominen quin ELO

df_grouped_ratings1 = df_grouped_ratings1.sort_values('totals', ascending=False)
fig, ax = plt.subplots(1, figsize=(12,10))
ax.bar([0,1,2,3,4], df_grouped_ratings1['higher_rated_victory'], label='Guanyar a més ELO', color='#ae24d1', tick_label=df_grouped_rating
ax.bar([0,1,2,3,4], df_grouped_ratings1['losses_or_draws'], label='Perdre o Empatar vs menys ELO', bottom=df_grouped_ratings1['higher_rat
ax.set_ylabel('Victòries', fontsize=14)
ax.set_xlabel('Resposta', fontsize=14)
ax.set_title('Victòries per Resposta', fontsize=18)
ax.legend()
plt.show()


pie, axs = plt.subplots(2,2, figsize=[16,10])

plt.subplot(2,2,1)
sns.barplot(
    data=kings_pawn,
    x='response',
    y='white_victory',

)
plt.title('Resposta a Peó de Rei')
plt.ylabel('Winrate (Blanc)')
plt.xlabel('Resposta de Negres')
plt.axhline(0.5)
```

```
plt.subplot(2,2,2)
sns.barplot(
    data=queens_pawn,
    x='response',
    y='white_victory',
    palette='Spectral'
)
plt.title('Resposta a Peó de Reina')
plt.ylabel('Winrate (Blanc)')
plt.xlabel('Resposta de Negres')
plt.axhline(0.5)

plt.subplot(2,2,3)
sns.barplot(
    data=english,
    x='response',
    y='white_victory',
    palette='brg_r'
)
plt.title('Resposta a Anglesa')
plt.ylabel('Winrate (Blanc)')
plt.xlabel('Resposta de Negres')
plt.axhline(0.5)

plt.subplot(2,2,4)
sns.barplot(
    data=reti,
    x='response',
    y='white_victory',
    palette='Wistia'
)
plt.title('Resposta a Reti')
plt.ylabel('Winrate (Blanc)')
plt.xlabel('Resposta de Negres')
plt.axhline(0.5)

plt.show()
```

```
# In[9]:

#Dataframe sota cert elo

num = 1200
num1=1600
num2=2000
num3=2400
num4=2800


df_under_num = df[(df.white_rating < num) & (df.black_rating < num)]

df_under_num_winners = df_under_num.groupby('winner')['game_id'].count()
df_under_num_winners


df_under_num1 = df[(df.white_rating < num1) & (df.black_rating < num1)]

df_under_num1_winners = df_under_num1.groupby('winner')['game_id'].count()
df_under_num1_winners

df_under_num2 = df[(df.white_rating < num2) & (df.black_rating < num2)]

df_under_num2_winners = df_under_num2.groupby('winner')['game_id'].count()
df_under_num2_winners

df_under_num3 = df[(df.white_rating < num3) & (df.black_rating < num3)]

df_under_num3_winners = df_under_num3.groupby('winner')['game_id'].count()
df_under_num3_winners

df_under_num4 = df[(df.white_rating < num4) & (df.black_rating < num4)]

df_under_num4_winners = df_under_num4.groupby('winner')['game_id'].count()
df_under_num4_winners
```

```
pie, axs = plt.subplots(5,1, figsize=[18,28])

plt.subplot(5,1,1)
plt.pie(x=df_under_num_winners, autopct="%.1f%%", labels=df_under_num_winners.keys(), pctdistance=0.5)
plt.title("Winrate per sota {} ELO".format(num), fontsize=10)
plt.axis('equal')


plt.subplot(5,1,2)
plt.pie(x=df_under_num1_winners, autopct="%.1f%%", labels=df_under_num1_winners.keys(), pctdistance=0.5)
plt.title("Winrate per sota {} ELO".format(num1), fontsize=10)
plt.axis('equal')


plt.subplot(5,1,3)
plt.pie(x=df_under_num2_winners, autopct="%.1f%%", labels=df_under_num2_winners.keys(), pctdistance=0.5)
plt.title("Winrate per sota {} ELO".format(num2), fontsize=10)
plt.axis('equal')

plt.subplot(5,1,4)
plt.pie(x=df_under_num3_winners, autopct="%.1f%%", labels=df_under_num3_winners.keys(), pctdistance=0.5)
plt.title("Winrate per sota {} ELO".format(num3), fontsize=10)
plt.axis('equal')

plt.subplot(5,1,5)
plt.pie(x=df_under_num4_winners, autopct="%.1f%%", labels=df_under_num4_winners.keys(), pctdistance=0.5)
plt.title("Winrate per sota {} ELO".format(num4), fontsize=10)
plt.axis('equal')

plt.show()

sns.barplot(
    data=df,
    x='rating_gap_class',
    y='higher_rated_victory',
    order=['0-50', '51-100', '101-150', '151-200', '201-250', '>250'],

)
plt.xlabel('Diferència ELO', fontsize=16)
plt.ylabel('Win Rate per més ELO', fontsize=16)
plt.title('Win Rate per diferència ELO', fontsize=16)
plt.show()
```

```
# In[10]:

#Usuaris amb blanques i quantes victòries van tenir

df_users_white = df.groupby('white_id',as_index=False).agg({'white_victory':'sum', 'game_id':'count'})
df_users_white.rename(columns={'white_victory':'white_victories', 'game_id':'white_games'}, inplace=True)
df_users_white = df_users_white.set_index('white_id')

#Usuaris amb negres i quants cops van perdre contra les blanques

df_users_black = df.groupby('black_id',as_index=False).agg({'white_victory':'sum', 'game_id':'count'})
df_users_black.rename(columns={'white_victory':'white_victories', 'game_id':'black_games'}, inplace=True)
df_users_black = df_users_black.set_index('black_id')

#Només coneixem les victòries amb blanques així que hem d'extreure les victòries amb negres

df_users_black['black_victories'] = df_users_black.black_games - df_users_black.white_victories
df_users_black.drop('white_victories', axis=1, inplace=True)

#Unim els dos dataframes per usuari. Alguns tindran NaN per la manera en que els unim.
#Els substituïrem per 0's
df_users = df_users_white.join(df_users_black)
df_users = df_users.fillna(0) #Tots els NaN es converteixen en 0
```

```
#Columnes amb total de victòries i de partides jugades

df_users['victories'] = df_users.white_victories + df_users.black_victories
df_users['games_played'] = df_users.white_games + df_users.black_games

#Percentatge de victòries

df_users['win_pct'] = df_users.victories / df_users.games_played



#Jugadors amb més winrate
df_users_sorted = df_users.sort_values(by=['win_pct'], ascending=False)
df_users_sorted.head(15)
```

```
# In[11]:

#Molts de la llista anterior no tenen suficients partides per lo que filtrarem
#els usuaris que tinguin poques partides per poder observar les dades més clarament

game_threshold = 25
df_users_many_games = df_users[(df_users.games_played >= game_threshold)]
df_users_many_games_sorted = df_users_many_games.sort_values(by=['win_pct'], ascending=False)
df_users_many_games_sorted.head(12)

print(df_users_many_games_sorted)

#Un cop filtrat podem observar certs usuaris amb moltíssim winrate.
#Analitzarem els 3 primers de la llista. Han fet trampes?
#Després mirarem els que més han perdut per poder veure perquè han perdut tant.
```

```
# In[12]:

    # Siindbad, 100% de winrate en 26 partides, 13 amb blanques i 13 amb negres:

df1 = pd.read_csv('chess.csv') #He hagut de tornar a llegir el csv perquè tenia molts problem
#si no ho feia creant el nou data frame.

df1 = df1.loc[(df1.white_id == 'siindbad') | (df1.black_id == 'siindbad')]

df1.rated.unique()


df1['rating_gap'] = abs(df1['white_rating'] - df1['black_rating'])

df1['siindbad_higher'] = np.where((df1['white_id'] == 'siindbad') & (df1['white_rating'] > df



df1[['white_id', 'white_rating', 'black_id', 'black_rating', 'siindbad_higher', 'rating_gap']

#Obtenim les partides on tenia més ELO amb la funció .sum()

num_siindbad_higher = df1.siindbad_higher.sum()

print('Siindbad is a higher rating in', num_siindbad_higher, 'of their', len(df1) , 'games.')

df1_siindbad_is_higher = df1[(df1.siindbad_higher == 1)]
df1_siindbad_is_lower = df1[(df1.siindbad_higher == 0)]

print('Average rating above opponents when higher-rated:', df1_siindbad_is_higher.rating_gap.
print('Average rating below opponents when lower-rated:', df1_siindbad_is_lower.rating_gap.me

#Podem observar com també té criteris similars a l'usuari anterior ja que també
#ha jugat moltes partides per sobre del seu rival.
```

```
# In[13]:

    #Chesscarl, 97.82% de winrate, 18/18 amb blanques i 27/28 amb negres:

#Crearem un nou dataframe per a aquest usuari
#Requereix l'ús de .loc ja que sinó apareix un error


df = df.loc[(df.white_id == 'chesscarl') | (df.black_id == 'chesscarl')]
df.rated.unique()


#Mirem si primer juga contra diverses persones:

#print(df.white_id.unique())
#print(df.black_id.unique())

#És possible que sempre jugués contra persones amb menys ELO?

df['carl_higher'] = np.where((df['white_id'] == 'chesscarl') & (df['white_rating'] > df['blac

df[['white_id', 'white_rating', 'black_id', 'black_rating', 'carl_higher', 'rating_gap']].head

#Obtenim les partides on tenia més ELO amb la funció .sum()

num_carl_higher = df.carl_higher.sum()
print('Chesscarl is a higher rating in', num_carl_higher, 'of their', len(df), 'games.')

df_carl_is_higher = df[(df.carl_higher == 1)]
df_carl_is_lower = df[(df.carl_higher == 0)]

print('Average rating above opponents when higher-rated:', df_carl_is_higher.rating_gap.mean()
print('Average rating below opponents when lower-rated:', df_carl_is_lower.rating_gap.mean())

#Podem observar com era millor jugador en 43 de 46 partides. En aquestes 43 partides tenia de
#mitjana 355.488 punts per sobre del seu rival. Si mirem a les gràfiques prèvies podem veure
#amb una diferència per sobre de 250 punts es té un winrate de més del 80% per lo que podem de
#que no estava fent trampes
```

```
# In[14]:

    #Lzchips, 96.15% de winrate, 12/13 amb blanques i 13/13 amb negres:

df2 = pd.read_csv('chess.csv') #He hagut de tornar a llegir el csv perquè tenia molts problemes
#si no ho feia creant el nou data frame.

df2 = df2.loc[(df2.white_id == 'lzchips') | (df2.black_id == 'lzchips')]

df2.rated.unique()

df2['rating_gap'] = abs(df2['white_rating'] - df2['black_rating'])

df2['lzchips_higher'] = np.where((df2['white_id'] == 'lzchips') & (df2['white_rating'] > df2['black_rating'

df2[['white_id', 'white_rating', 'black_id', 'black_rating', 'lzchips_higher', 'rating_gap']].head(10)

#Obtenim les partides on tenia més ELO amb la funció .sum()

num_lzchips_higher = df2.lzchips_higher.sum()

print('Lzchips is a higher rating in', num_lzchips_higher, 'of their', len(df2) , 'games.')

df2_lzchips_is_higher = df2[(df2.lzchips_higher == 1)]
df2_lzchips_is_lower = df2[(df2.lzchips_higher == 0)]

print('Average rating above opponents when higher-rated:', df2_lzchips_is_higher.rating_gap.mean())
print('Average rating below opponents when lower-rated:', df2_lzchips_is_lower.rating_gap.mean())

#Aquest usuari només ha jugat amb gent molt per sota del seu ELO. Si observem les seves partides
#podem veure que només va jugar partides en les que no es guanyava ni perdia ELO per lo que és encara
#més entendible el seu winrate.
```

```
# In[15]: I la gent que ha perdut més partides?

#Mirant la taula de la cel·la 7 podem trobar els usuaris amb menys winrate.

#Ara podem aplicar la lògica inversa en els usuaris que més partides han perdut.
#Fer trampes no vol dir sempre guanyar, també pot ser perdre per donar puntuació
#a un altre jugador. Aquests jugadors ja sortien a la llista abans però els he tornat
#a imprimir i endreçar
```

```
# In[16]:

    #andreschil, 6.25% de winrate 1/14 amb blanques i 1/18 amb negres:

df3 = pd.read_csv('chess.csv') #He hagut de tornar a llegir el csv perquè tenia molts problemes
#si no ho feia creant el nou data frame.

df3 = df3.loc[(df3.white_id == 'andreschil') | (df3.black_id == 'andreschil')]

df3.rated.unique()


df3['rating_gap'] = abs(df3['white_rating'] - df3['black_rating'])

df3['andreschil_higher'] = np.where((df3['white_id'] == 'andreschil') & (df3['white_rating'] > df3['black_r

df3[['white_id', 'white_rating', 'black_id', 'black_rating', 'andreschil_higher', 'rating_gap']].head(10)

#Obtenim les partides on tenia més ELO amb la funció .sum()

num_andreschil_higher = df3.andreschil_higher.sum()

print('Andreschil is a higher rating in', num_andreschil_higher, 'of their', len(df3) , 'games.')

df3_andreschil_is_higher = df3[(df3.andreschil_higher == 1)]
df3_andreschil_is_lower = df3[(df3.andreschil_higher == 0)]

print('Average rating above opponents when higher-rated:', df3_andreschil_is_higher.rating_gap.mean())
print('Average rating below opponents when lower-rated:', df3_andreschil_is_lower.rating_gap.mean())

#Podem veure que aquest jugador ha jugat un 96% de partides contra usuaris que en mitjana tenien gairebé
#355 punts per sobre, sent entendible el número de derrotes que ha acumulat.

#Per acabar aquest anàlisi agafarem un usuari amb moltes partides i que n'hagi perdut moltes.
#L'usuari nitsua49 té 46 partides i és el 3r amb menys winrate.
```

```
# In[17]:

    #Nitsua49, 10.87% de winrate, 1/20 amb blanques i 4/26 amb negres:

df4 = pd.read_csv('chess.csv') #He hagut de tornar a llegir el csv perquè tenia molts problemes
#si no ho feia creant el nou data frame.

df4 = df4.loc[(df4.white_id == 'nitsua49') | (df4.black_id == 'nitsua49')]

df4.rated.unique()


df4['rating_gap'] = abs(df4['white_rating'] - df4['black_rating'])

df4['nitsua49_higher'] = np.where((df4['white_id'] == 'nitsua49') & (df4['white_rating'] > df4['black_ratin

df4[['white_id', 'white_rating', 'black_id', 'black_rating', 'nitsua49_higher', 'rating_gap']].head(10)

#Obtenim les partides on tenia més ELO amb la funció .sum()

num_nitsua49_higher = df4.nitsua49_higher.sum()

print('Nitsua49 is a higher rating in', num_nitsua49_higher, 'of their', len(df4) , 'games.')

df4_nitsua49_is_higher = df4[(df4.nitsua49_higher == 1)]
df4_nitsua49_is_lower = df4[(df4.nitsua49_higher == 0)]

print('Average rating above opponents when higher-rated:', df4_nitsua49_is_higher.rating_gap.mean())
print('Average rating below opponents when lower-rated:', df4_nitsua49_is_lower.rating_gap.mean())


#Igual que l'anterior usuari, sempre ha jugat contra gent molt més bona.
```

```
# In[18]: Precisió dels jugadors



def get_engine_similarity(df, username, top_moves_num=3):
    #Llista per poder comparar els resultats del jugador amb els de l'ordinador
    engine_pct = []
    df_white = df[(df.white_id == username)]
    df_black = df[(df.black_id == username)]

    #Bucle de moviments blancs i comparació amb els de l'ordinador.
    for index, row in df_white.iterrows():
        similar_moves = 0
        moves_list = row['moves_list'] #Moviments de l'usuari
        board = chess.Board()
        for i in range(len(moves_list)):
            if i % 2 == 0:
                #On tinc Stockfish al meu ordinador
                stockfish = Stockfish(r'C:\Users\Usuario\Desktop\Bigdata\Chess\stockfish_15_w
                stockfish.set_fen_position(board.fen()) #Actualització dels moviments de l'or
                engine_moves = stockfish.get_top_moves(top_moves_num) #Llista de diccionaris
                 #Notació del moviment de l'usuari
                board.push_san(moves_list[i])
                fen1 = board.fen()
                board.pop()
                #Notació del moviment de l'ordinador i el comparem amb el de l'usuari
                for dic in engine_moves:
                    board.push_san(dic['Move'])
                    fen2 = board.fen()
                    board.pop()
                    if fen1 == fen2:
                        similar_moves = similar_moves + 1
                board.push_san(moves_list[i])
            else:
                board.push_san(moves_list[i])
        similarity_pct = similar_moves / math.ceil(len(moves_list)/2)
        engine_pct.append(similarity_pct)

    for index, row in df_black.iterrows():
        similar_moves = 0
        moves_list = row['moves_list']
        board = chess.Board()
        for i in range(len(moves_list)):
            if i % 2 == 1:
                #On tinc Stockfish al meu ordinador
                stockfish = Stockfish(r'C:\Users\Usuario\Desktop\Bigdata\Chess\stockfish_15_
                stockfish.set_fen_position(board.fen())
                engine_moves = stockfish.get_top_moves(top_moves_num)
                board.push_san(moves_list[i])
                fen1 = board.fen()
                board.pop()
                for dic in engine_moves:
                    board.push_san(dic['Move'])
                    fen2 = board.fen()
                    board.pop()
                    if fen1 == fen2:
                        similar_moves = similar_moves + 1
                board.push_san(moves_list[i])
            else:
                board.push_san(moves_list[i])
        similarity_pct = similar_moves / math.floor(len(moves_list)/2)
        engine_pct.append(similarity_pct)

    return engine_pct
```

```
# In[19]: #Aquesta cel·la en particular és extremadament lenta
#degut al increïble cost computacional que requereix comparar
#cada moviment de cada pertida als 4 millors de l'ordinador.

chesscarl_game_engine_analysis = get_engine_similarity(df, 'chesscarl', 4)
siindbad_game_engine_analysis = get_engine_similarity(df, 'siindbad', 4)
lzchips_game_engine_analysis = get_engine_similarity(df, 'lzchips', 4)

nitsua49_game_engine_analysis = get_engine_similarity(df, 'nitsua49', 4)

print(chesscarl_game_engine_analysis)
print(siindbad_game_engine_analysis)
print(nitsua49_game_engine_analysis)
print(lzchips_game_engine_analysis)

plt.hist(chesscarl_game_engine_analysis, bins=20, color='mediumseagreen')
plt.xlabel('Percentatge similar als 4 millors')
plt.ylabel('Número de cops')
plt.title('Similitud amb ordinandor (Chesscarl)')
plt.show()

plt.hist(siindbad_game_engine_analysis, bins=20, color='mediumseagreen')
plt.xlabel('Percentatge similar als 4 millors')
plt.ylabel('Número de cops')
plt.title('Similitud amb ordinandor (Siindbad)')
plt.show()

plt.hist(lzchips_game_engine_analysis, bins=20, color='mediumseagreen')
plt.xlabel('Percentatge similar als 4 millors')
plt.ylabel('Número de cops')
plt.title('Similitud amb ordinandor (Lzchips)')
plt.show()

plt.hist(nitsua49_game_engine_analysis, bins=20, color='mediumseagreen')
plt.xlabel('Percentatge similar als 4 millors')
plt.ylabel('Número de cops')
plt.title('Similitud amb ordinandor (Nitsua49)')
plt.show()
```