MACHINE LEARNING
FINAL ASSIGNMENT
COURSE 1

***The jupyter notebook with the code is at the end of the report**

I. INTRODUCTION

i. BACKGROUND

The thematic I want to analyze in this final assignment is the game of chess. Chess can be a competitive or recreational board game. It is a strategy game played on a square chessboard of 64 squares, half of them are white, half black and they are alternated along the board. It is played by two people where players are identified by two colors: white and black. White has the starting move every time. Each player start with the same pieces in their starting positions. There are six type of pieces each of which have their own movement rules. A game is won when the opponent resigns, runs out of time or his king is under attack and cant move nor defend. It also possible to draw a game. I will try to work on a dataset containing a list of chess games with various freatures. The final objective is to study and manipulate this dataset in order to train a machine learning model to predict the winner of a chess game.

II. THE DATA

i. DESCRIPTION OF THE DATASET

The dataset I will use is comprehensive of 20058 games collected from a random selection of users from the Lichess.org website. The dataset is in a csv format and contains 16 features:

1. ***Id*** **–** Unique identifier for every game.
2. ***Rated*** **–** List of boolean values (True or False) determining if the game was a rated game or unrated.
3. ***Created at*** **–** When the game started
4. ***Last Move at*** **–** When the game ended
5. ***Turns*** **–** Number of turns the game lasted
6. ***Victory Status*** **–** How the game ended
7. ***Winner*** **–** Who won the game (white or black)
8. ***Increment Code*** **–** Time mode for the game
9. ***White id*** **–** Id of the white player
10. ***White rating*** **–** Rating of the white player
11. ***Black Id*** **–** Id of the black player
12. ***Black Rating*** **–** Rating of the black player
13. ***Moves*** **–** List of all moves made in the game (chess notation)
14. ***Opening Eco*** **–** Standardized Code for any given opening
15. ***Opening Name*** **–** Name of the opening
16. ***Opening ply*** **–** Number of moves in the opening phase
17.

ii. INITIAL PLAN FOR DATA EXPLORATION

My initial plan is to take a lock at the data and find the most useful data that might help me with my task. Firstly I will clean the dataset and remove the unnecessary features present such as the starting time (created_at) and the end time (last_move_at) for each game. The rest of the dataset is pretty clean maybe we will need to do some features engineering. As for data exploration I will start by visualizing different features and the relations between them to understand the role that they can have on to find a solution to my problem.

iii. ACTIONS TAKEN FOR DATA CLEANING AND FEATURE ENGINEERING

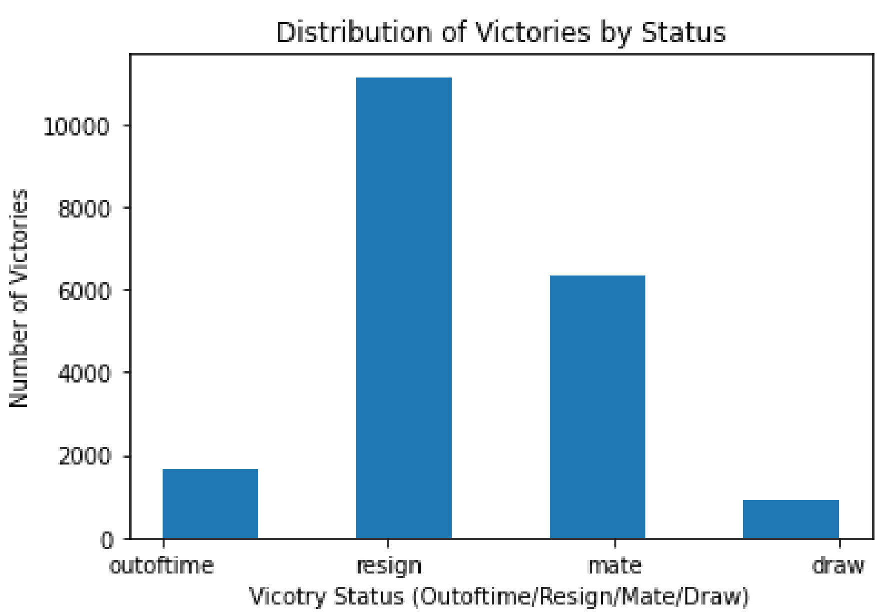
The dataset of choice is already well orginized. I will remove three columns, *created_at*, *"last_move_at" and "moves"* because they are not necessary for our purpose. In order to make some feature more convenient for future analysis I will do some feature engineering. I will create a new dataframe completely numerical by encoding the categorical features present. I will remove the columns "*opening_name*", "*id*", "*white_id*", "*black_id*" and "*opening_eco*". I will binary encode the "*rated*" (from True/False to 0 and 1), ordinal encode and "*winner*"columns and one-hot-encode the "*victory_status*" column. There is no need of any feature scaling since the data is not skewed and there are no outliers.

III. EXPLORATORY DATA ANALYSIS

After the exploratory analysis it is possible to infer some properties about the data. It is possible to see the distribution of the different winning scenarios, the distribution between white and black wins, how many turn in average are needed to end a game, the most common opening ('"*Van't Kruijs Opening") etc*. Lets see some of these features.

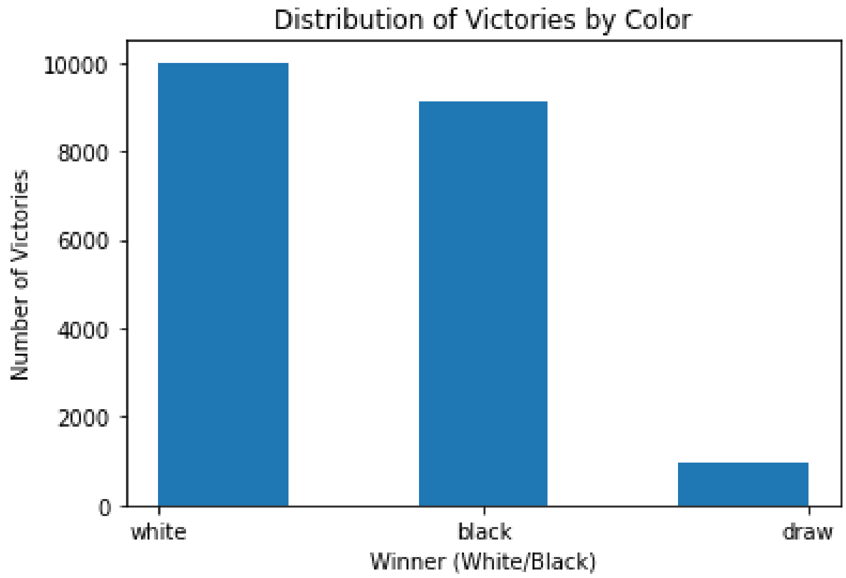
i. Distribution of Victories by Win Type

From the total of 20058 games 55% of the games ended due to resignation of the opponent, 31% ended by check-mate, 8% due to time running out and 4% were draws. So it is much more likely to end a game of chess by resignation. However to make this data more valuable we should intertwine it with other features such as openings of choice. In this way we might be able to see which opening is most likely to lead to a specific scenario.



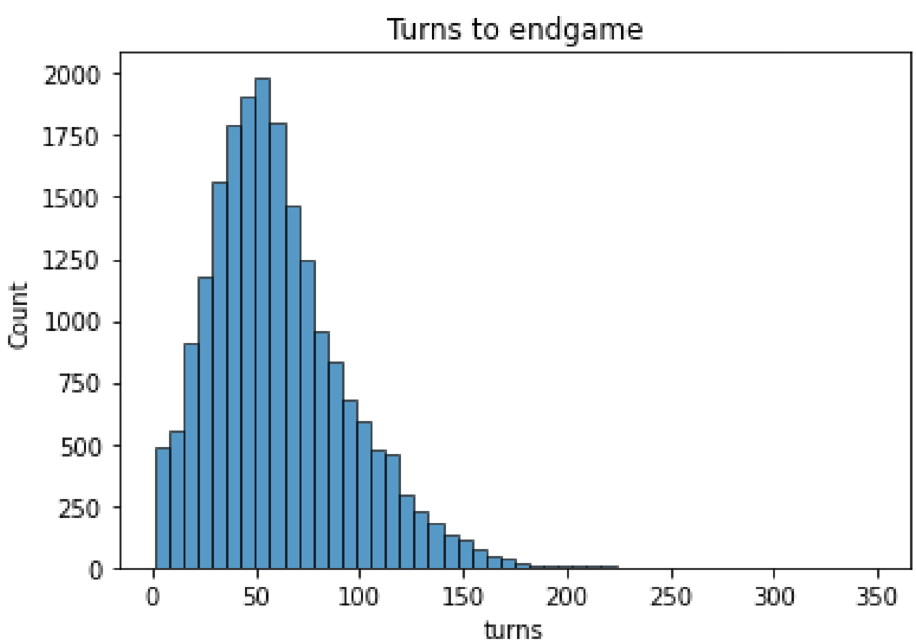
ii. Distribution of Victories by Color

From this histogram we can see that in the distribution of wins by color white is favored compared to black. This is as we expected to be since white always has the first move hence a potential one move advantage for the rest of the game.



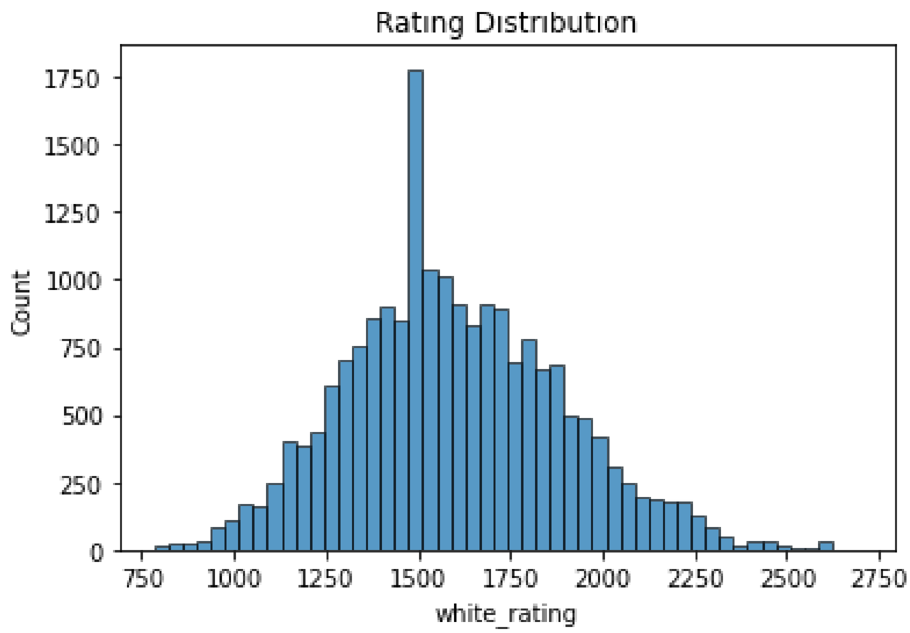
iii. Distribution of Turns to End a Game

From this plot we can see that the average turn number for a game is around 60. Past this mark the number of turns decrease exponentially. The games with 0 turns are those where the opponent left the game before before moving a piece.



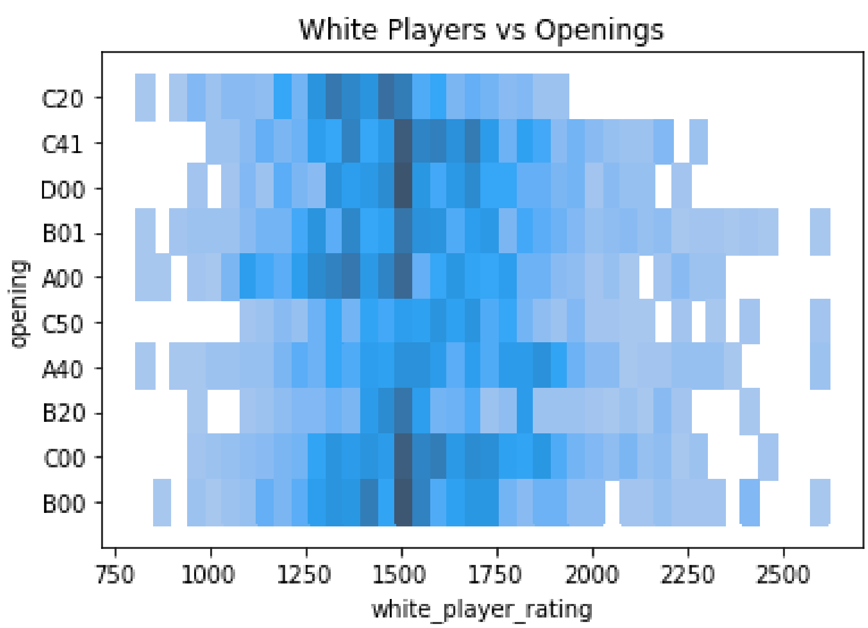
iv. Rating Distribution

This is the distribution of the players by rating. We can see that the players are normally distributed around the 1600 rating. The peak at 1500 is interesting. It is possible that there is a noticeable difference between players between 1500 and 1600.



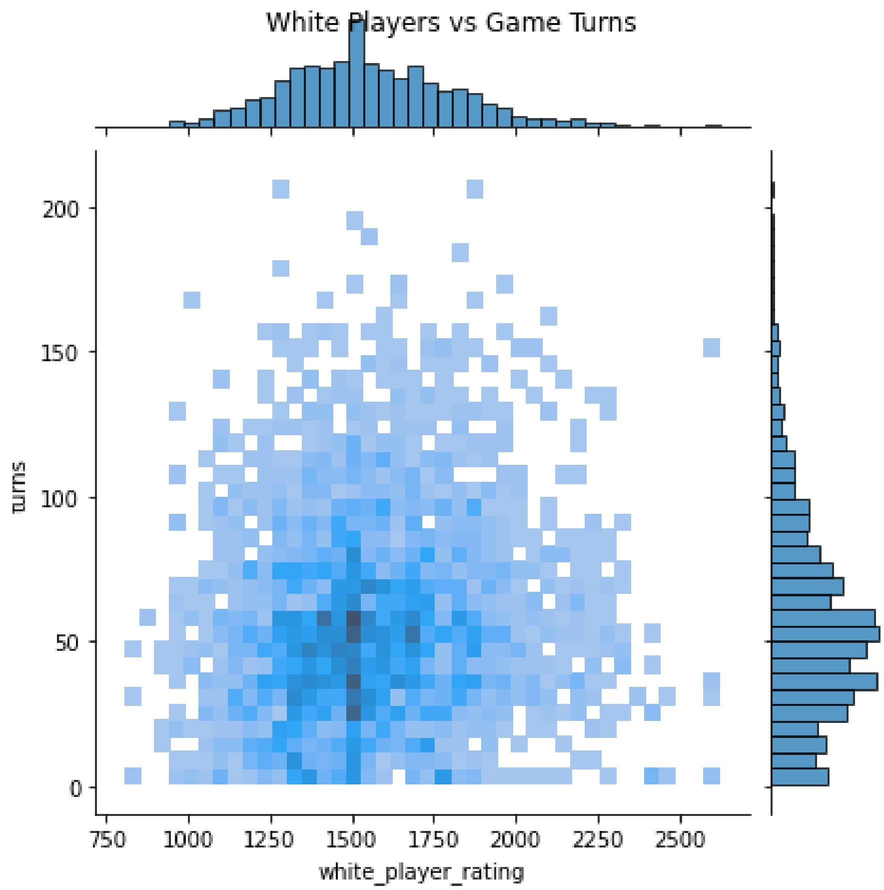
v. Players Rating and Openings Distribution

In this plot it is possible to see the different frequency of the top 10 openings used in the 20058 games in the dataset. The openings are identified by their standard codes. Of course the most significant area is between 1300 and 1800 rating since we got much more players data in this interval.



vi. Players vs Game Turns

In this plot is shown the relation between games played by players of different rating and the amount of turns need in order to complete the games. We can see that as rating goes higher the games tends to become shorter.



vii. Dataframe description (transposed)

We can see useful information such as the average turns per game, average white and black ratings, the rating of the highest rated player etc.

	count	mean	std	min	25%	50%	75%	max
turns	20058.0	60.465999	33.570585	1.0	37.0	55.0	79.0	349.0
white_rating	20058.0	1596.631868	291.253376	784.0	1398.0	1567.0	1793.0	2700.0
black_rating	20058.0	1588.831987	291.036126	789.0	1391.0	1562.0	1784.0	2723.0
opening_ply	20058.0	4.816981	2.797152	1.0	3.0	4.0	6.0	28.0
rated_enc	20058.0	0.805414	0.395891	0.0	1.0	1.0	1.0	1.0
winner_enc	20058.0	1.044571	0.975038	0.0	0.0	1.0	2.0	2.0
draw	20058.0	0.045169	0.207680	0.0	0.0	0.0	0.0	1.0
mate	20058.0	0.315336	0.464661	0.0	0.0	0.0	1.0	1.0
outoftime	20058.0	0.083757	0.277030	0.0	0.0	0.0	0.0	1.0
resign	20058.0	0.555738	0.496896	0.0	0.0	1.0	1.0	1.0

IV. HYPOTHESIS

Lets write down 3 possible hypothesis.

1. Every chess game has a 60% probability to end in less then 60 turns
2. Every chess game that started with a "*scotch game*" opening has been won by white 60% of the time
3. Every player under 1550 rating has won 70% of the time against a lower rated player

I will focus on the first hypothesis for which I will do a significant test. First we need to define a null hypothesis and an alternative:

i. **Null hypothesis:** 60% of the first 100 games in the dataframe ends in less the 60 turns.

ii. **Alternative hypothesis:** 70% of the first 100 games in the dataframe ends in less the 60 turns.

Lets find out what is the number of games that actually ended in less then 60 turns.

```
null_count=0
alt_count=0

for i in range(100):
    if data.turns.iloc[i]<=60:
        null_count=null_count+1
    else:
        alt_count=alt_count+1
```

```
print('Number or games ended in less then 60 turns:',null_count)
```

```
Number or games ended in less then 60 turns: 72
```

We will set the P-value cutoff at 5% as it was said it was common practice. Next we calculate the cumulative distribution function.

```
prob = 1 - binom.cdf(71, 100, 0.6)

print(str(round(prob*100, 1))+"%")
```

```
0.8%
```

The probability that there is a 60% chance that a chess game ends in less then 60 turns is 0.8%. This is under our cutoff of 5% which means that we should reject the null and conclude that the probability of chess games to end in less 60 turns is greater then 60%.

Since we decided on a p-value of 5% we have the other cutoff which is 95%.

```
print(binom.ppf(0.95,100,0.6)+1)
```

```
69.0
```

This means that the odds that 69 games or less ends in less then 60 turns, given a 60% probability of that happening, is 95% and that the odds that 69 games or more ends in less then 60 turns is 5%. If there was 69 games or less that ended in less the 60 turns we could accept the null hypothesis with a confidence level of 95%.

Lets see what are the odds to get the number of games that ended in less then 60 turns (72 games out of 100).

```
print (1-binom.cdf(72, 100, 0.6))
print (binom.cdf(72, 100, 0.7))
```

```
0.004600434301985534
0.7036338394422568
```

There is 5% chance to make a Type I Error (60% of chess games end in less then 60 turns but we say they don't) and 70% chance to make a Type II Error (more then 70% of chess games ends in less the 60 turns and we say they don't. The first will almost fail our 5% threshold and the second will fail it.

To reduce both those error we have the possibility to increase the sample size. From 100 to 10000. Lets see what would be the odds then.

```
print (binom.ppf(0.95,10000,0.6))
print (binom.ppf(0.05,10000,0.7))
```

```
6081.0
6925.0
```

We need a value higher then 6081 in order to say that 60% of the games ends in less then 60 turns. A value lower then 6925 means that 70% of the games don't end in less then 60 turns.

```
print (1-binom.cdf(6500, 10000, 0.6))
print (binom.cdf(6500, 10000, 0.7))
```

```
1.1102230246251565e-16
3.133038425550557e-27
```

Taking a arbitrary half point of 6500 games.The probability of having a 60% chance for game that 6500 games ends in less then 60 turns is very low. Similarly it is also very unlikely that 6500 games ends in less then 60 turns when there is 70% chance per game of this happening.

V. SUGGESTIONS

An interesting idea would be to group all games by their game time format (*increment_code*) and see how different players with different rating perform when there is more ore less time for a game. As subsequent steps for analyzing the data would be to try to apply some models to investigate the relation between different features (eg. relation between player rating and time to finish a game) with different tools such as: simple linear regression, polynomial linear regressions. It would also be interesting to implement a multiple linear regression model.

VI. DATA SET SUMMARY

This dataset has some pros and some cons. The pros are that this dataset is pretty big with a lot of entries, all the features are of similar scale (beside the *created_at* and *last_move_at* which are not but I removed those columns) and the data is not skewed. However I feel like there are missing some interesting features such as the overall player precision for white and black in each game, number of mistakes made and average time per move.

Final_course1

January 16, 2021

1 Machine Learning Final Exercise

1.1 Working with chess games

Importing necessary libraries

[1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from scipy.stats import binom
import seaborn as sns
import matplotlib.pyplot as plt
%pylab inline
%matplotlib inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Loading the csv file from my local machine and reading it to a pandas dataframe

[2]:
```
filepath = "/home/Enry/Course1_final_test/games.csv"
data = pd.read_csv(filepath)
data.head()
```

[2]:
```
         id  rated    created_at  last_move_at  turns victory_status winner  \
0  TZJHLljE  False  1.504210e+12  1.504210e+12     13      outoftime  white
1  l1NXvwaE   True  1.504130e+12  1.504130e+12     16         resign  black
2  mIICvQHh   True  1.504130e+12  1.504130e+12     61           mate  white
3  kWKvrqYL   True  1.504110e+12  1.504110e+12     61           mate  white
4  9tXo1AUZ   True  1.504030e+12  1.504030e+12     95           mate  white

  increment_code       white_id  white_rating      black_id  black_rating  \
0           15+2       bourgris          1500          a-00          1191
1           5+10           a-00          1322     skinnerua          1261
2           5+10         ischia          1496          a-00          1500
3           20+0  daniamurashov          1439  adivanov2009          1454
4           30+3      nik221107          1523  adivanov2009          1469

                                               moves opening_eco  \
0  d4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+ Nc3 Ba5…         D10
```

```
1  d4 Nc6 e4 e5 f4 f6 dxe5 fxe5 fxe5 Nxe5 Qd4 Nc6…         B00
2  e4 e5 d3 d6 Be3 c6 Be2 b5 Nd2 a5 a4 c5 axb5 Nc…         C20
3  d4 d5 Nf3 Bf5 Nc3 Nf6 Bf4 Ng4 e3 Nc6 Be2 Qd7 O…         D02
4  e4 e5 Nf3 d6 d4 Nc6 d5 Nb4 a3 Na6 Nc3 Be7 b4 N…         C41

                             opening_name  opening_ply
0        Slav Defense: Exchange Variation            5
1  Nimzowitsch Defense: Kennedy Variation            4
2   King's Pawn Game: Leonardis Variation            3
3  Queen's Pawn Game: Zukertort Variation            3
4                        Philidor Defense            5
```

[3]:
```
data.white_id.value_counts()
```

[3]:
```
taranga          72
chess-brahs      53
a_p_t_e_m_u_u    49
ssf7             48
bleda            48
                 ..
mahdi1365         1
boratkazak        1
peter-br          1
scorpiopawn       1
hasan123          1
Name: white_id, Length: 9438, dtype: int64
```

Observe the dataframe structure

[4]:
```
print('Column Names')
print(data.columns.tolist())

print('\n Number of rows')
print(data.shape[0])

print('\n Number of columns')
print(data.shape[1])

print('\n Data type per column')
print(data.dtypes)
```

```
Column Names
['id', 'rated', 'created_at', 'last_move_at', 'turns', 'victory_status',
'winner', 'increment_code', 'white_id', 'white_rating', 'black_id',
'black_rating', 'moves', 'opening_eco', 'opening_name', 'opening_ply']

 Number of rows
20058
```

```
 Number of columns
16

 Data type per column
id                  object
rated                 bool
created_at         float64
last_move_at       float64
turns                int64
victory_status      object
winner              object
increment_code      object
white_id            object
white_rating         int64
black_id            object
black_rating         int64
moves               object
opening_eco         object
opening_name        object
opening_ply          int64
dtype: object
```

Create a copy of the dataframe

[5]:
```python
data_copy = data.copy()
```

1.2 Data Cleaning

Removing the white_id and black_id columns since we are not interested in how well a particular player did so it is useless information for us. Also lets remove the created_at and last_move_at since we dont need to know when the games were made.

[6]:
```python
data = data.drop(['created_at', 'last_move_at' , 'moves'], axis=1)
```

[7]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20058 entries, 0 to 20057
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              20058 non-null  object
 1   rated           20058 non-null  bool
 2   turns           20058 non-null  int64
 3   victory_status  20058 non-null  object
 4   winner          20058 non-null  object
 5   increment_code  20058 non-null  object
 6   white_id        20058 non-null  object
```

```
 7   white_rating    20058 non-null  int64
 8   black_id        20058 non-null  object
 9   black_rating    20058 non-null  int64
 10  opening_eco     20058 non-null  object
 11  opening_name    20058 non-null  object
 12  opening_ply     20058 non-null  int64
dtypes: bool(1), int64(4), object(8)
memory usage: 1.9+ MB
```

[8]:
```
data.head()
```

[8]:
```
         id  rated  turns victory_status winner increment_code       white_id  \
0  TZJHLljE  False     13      outoftime  white           15+2       bourgris
1  l1NXvwaE   True     16         resign  black           5+10           a-00
2  mIICvQHh   True     61           mate  white           5+10         ischia
3  kWKvrqYL   True     61           mate  white           20+0  daniamurashov
4  9tXo1AUZ   True     95           mate  white           30+3      nik221107

   white_rating      black_id  black_rating opening_eco  \
0          1500          a-00          1191         D10
1          1322     skinnerua          1261         B00
2          1496          a-00          1500         C20
3          1439  adivanov2009          1454         D02
4          1523  adivanov2009          1469         C41

                            opening_name  opening_ply
0        Slav Defense: Exchange Variation            5
1  Nimzowitsch Defense: Kennedy Variation            4
2   King's Pawn Game: Leonardis Variation            3
3  Queen's Pawn Game: Zukertort Variation            3
4                        Philidor Defense            5
```

Making a copy of the dataset for future usage

[9]:
```
data_enc=data.copy()
```

First lets remove the game, white and black ids

[10]:
```
data_enc = data_enc.drop(['id', 'white_id' , 'black_id' , 'opening_name' ,␣
 ↪'opening_eco'], axis=1)
```

1.3 Feature Engineering

1.3.1 Creating a completely numberical dataset

Lets encode all properties. Defining an econder

[11]:
```
enc = preprocessing.LabelEncoder()
```

Binary encoding of the rated and winner columns

[12]:
```
data_enc['rated_enc'] = enc.fit_transform(data_enc.rated)
print(data_enc[['rated_enc', 'rated']].head())
```

```
   rated_enc  rated
0          0  False
1          1   True
2          1   True
3          1   True
4          1   True
```

[13]:
```
data_enc['winner_enc'] = enc.fit_transform(data_enc.winner)
print(data_enc[['winner_enc', 'winner']].head())
```

```
   winner_enc winner
0           2  white
1           0  black
2           2  white
3           2  white
4           2  white
```

Lastrly lest do a one-hot-encoding of the victory_status column

[14]:
```
ohe=pd.get_dummies(data_enc.victory_status)
data_enc=pd.concat([data_enc, ohe], axis=1)
```

[15]:
```
data_enc.describe().T
```

[15]:
```
                 count         mean         std    min     25%     50%     75%  \
turns          20058.0    60.465999   33.570585    1.0    37.0    55.0    79.0
white_rating   20058.0  1596.631868  291.253376  784.0  1398.0  1567.0  1793.0
black_rating   20058.0  1588.831987  291.036126  789.0  1391.0  1562.0  1784.0
opening_ply    20058.0     4.816981    2.797152    1.0     3.0     4.0     6.0
rated_enc      20058.0     0.805414    0.395891    0.0     1.0     1.0     1.0
winner_enc     20058.0     1.044571    0.975038    0.0     0.0     1.0     2.0
draw           20058.0     0.045169    0.207680    0.0     0.0     0.0     0.0
mate           20058.0     0.315336    0.464661    0.0     0.0     0.0     1.0
outoftime      20058.0     0.083757    0.277030    0.0     0.0     0.0     0.0
resign         20058.0     0.555738    0.496896    0.0     0.0     1.0     1.0

                  max
turns           349.0
white_rating   2700.0
black_rating   2723.0
opening_ply      28.0
rated_enc         1.0
winner_enc        2.0
```

```
draw             1.0
mate             1.0
outoftime        1.0
resign           1.0
```

1.4 Exploratory Data Analysis

Lets take a look at the number of victory of each type

[16]:
```
data.victory_status.value_counts()
```

[16]:
```
resign       11147
mate          6325
outoftime     1680
draw           906
Name: victory_status, dtype: int64
```

And the number of total wins for white and for black

[17]:
```
data.winner.value_counts()
```

[17]:
```
white    10001
black     9107
draw       950
Name: winner, dtype: int64
```

Lets see if there are some players that many games in this dataset. For white and for black

[18]:
```
data.white_id.value_counts()
```

[18]:
```
taranga          72
chess-brahs      53
a_p_t_e_m_u_u    49
ssf7             48
bleda            48
                 ..
mahdi1365         1
boratkazak        1
peter-br          1
scorpiopawn       1
hasan123          1
Name: white_id, Length: 9438, dtype: int64
```

[19]:
```
data.black_id.value_counts()
```

[19]:
```
taranga               82
vladimir-kramnik-1    60
a_p_t_e_m_u_u         47
```

```
docboss               44
king5891              44
                      ..
allmight87             1
stigdagerman           1
meddico                1
omaolmhuaidh           1
radchenko1939          1
Name: black_id, Length: 9331, dtype: int64
```

This can be intereting as we now have seen that some players have indeed many games. This means that there is the possibility to specifically analyze these players since there is many data for them.

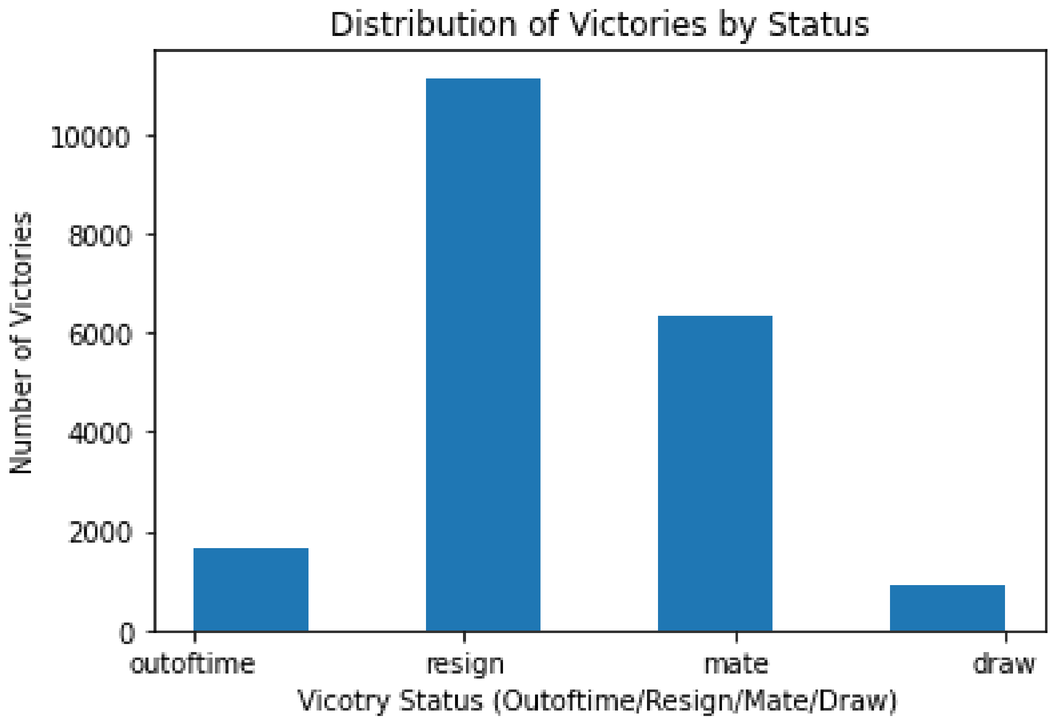
Lets see the most popular openings

[20]:
```
data.opening_name.value_counts()
```

[20]:
```
Van't Kruijs Opening
368
Sicilian Defense
358
Sicilian Defense: Bowdler Attack
296
Scotch Game
271
French Defense: Knight Variation
271
      …
English Opening: Anglo-Indian Defense |  Queen's Indian Formation
1
Ruy Lopez: Exchange |  Alekhine Variation
1
Semi-Slav Defense: Marshall Gambit |  Main Line
1
Queen's Gambit Declined: Janowski Variation
1
English Opening: King's English Variation |  Four Knights Variation |  Korchnoi
Line      1
Name: opening_name, Length: 1477, dtype: int64
```

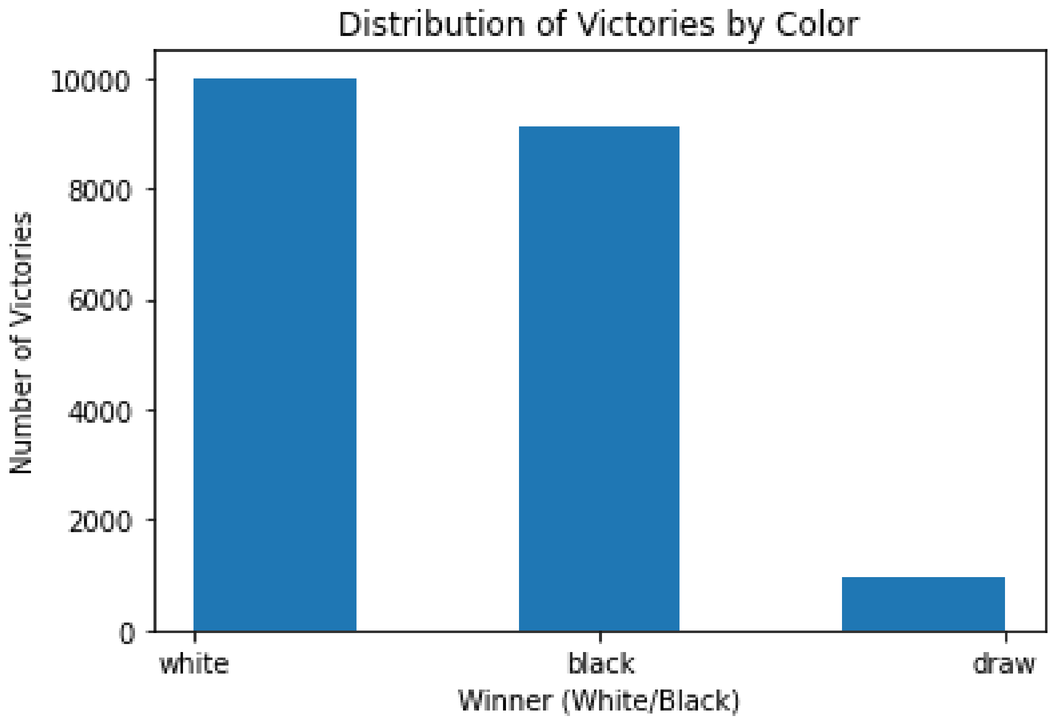
Lets plot an histogram visually showing the relative distribution of all the possible victory fashions

[21]:
```
ax = plt.axes()
ax.hist(data.victory_status, bins=7);

ax.set(xlabel='Vicotry Status (Outoftime/Resign/Mate/Draw)',
       ylabel='Number of Victories',
       title='Distribution of Victories by Status');
```



Number of victories by color

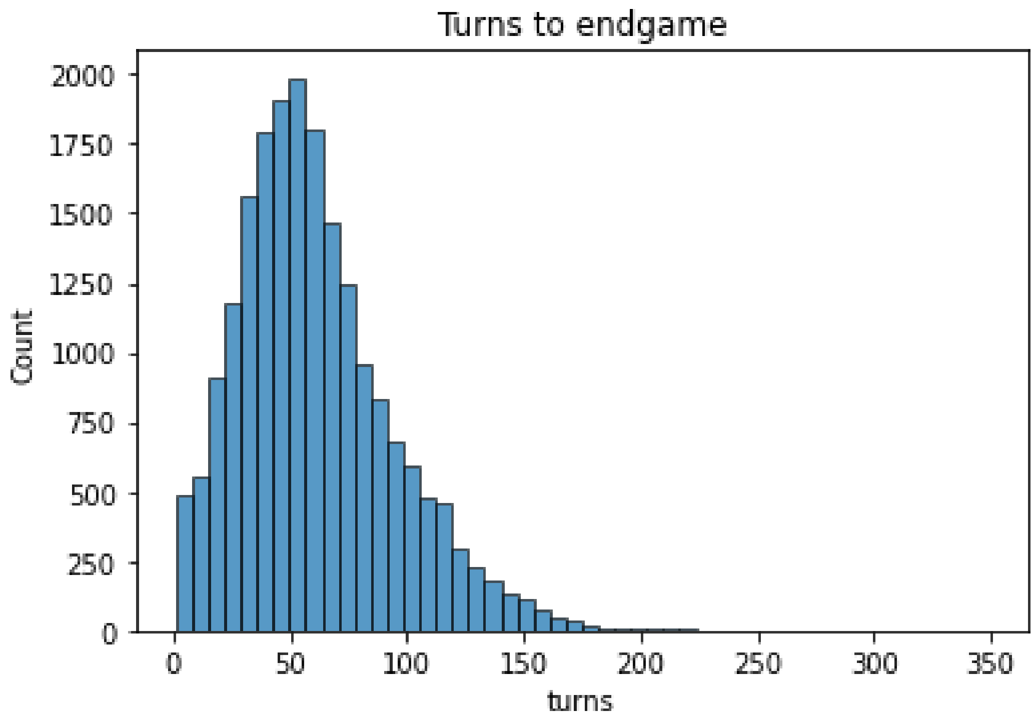
[22]:
```
ax = plt.axes()
ax.hist(data.winner, bins=5);

ax.set(xlabel='Winner (White/Black)',
       ylabel='Number of Victories',
       title='Distribution of Victories by Color');
```



Distribution of games by their number of turns per game

[23]:
```
sns.histplot(data['turns'], bins=50).set_title('Turns to endgame')
```

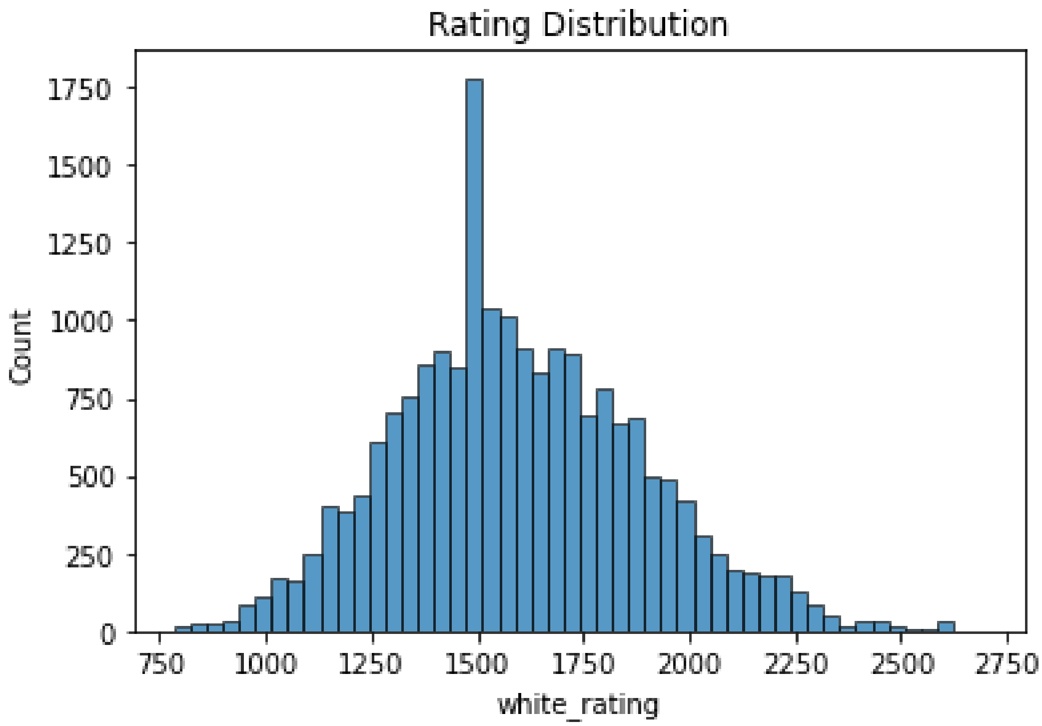
[23]: Text(0.5, 1.0, 'Turns to endgame')



Distribution of the rating for players

[24]:
```
sns.histplot(data['white_rating'], bins=50).set_title('Rating Distribution')
```

[24]: Text(0.5, 1.0, 'Rating Distribution')



Lets explore the relation between openings and number of turns when white wins

Finding the top 10 most common openings (Standardized Code)

[25]:
```
openings_data=data.opening_eco.value_counts()
openings_data.index
```

[25]:
```
Index(['A00', 'C00', 'D00', 'B01', 'C41', 'C20', 'A40', 'B00', 'B20', 'C50',
       …
       'E47', 'C90', 'D82', 'D81', 'C75', 'E48', 'A99', 'A89', 'D59', 'A71'],
      dtype='object', length=365)
```

[26]:
```
openings_list=openings_data.index.to_list()
common_openings=openings_list[:10]
#common_openings
```

[27]:
```
common_openings
```

[27]:
```
['A00', 'C00', 'D00', 'B01', 'C41', 'C20', 'A40', 'B00', 'B20', 'C50']
```

[28]:
```
white_player_rating=[]
black_player_rating=[]
opening=[]
turns=[]
```

```
for i in range(len(data)):
    for j in range(len(common_openings)):
        if data.opening_eco.iloc[i] is common_openings[j] and data.winner.
 ↪iloc[i] == 'white':
            white_player_rating.append(data.white_rating.iloc[i])
            black_player_rating.append(data.black_rating.iloc[i])
            opening.append(data.opening_eco.iloc[i])
            turns.append(data.turns.iloc[i])
```

[29]:
```
data_common_openings=pd.DataFrame(columns=['white_player_rating',␣
 ↪'black_player_rating', 'opening' , 'turns'])
```

[30]:
```
data_common_openings.white_player_rating=white_player_rating
data_common_openings.black_player_rating=black_player_rating
data_common_openings.opening=opening
data_common_openings.turns=turns
```

[31]:
```
data_common_openings
```

[31]:

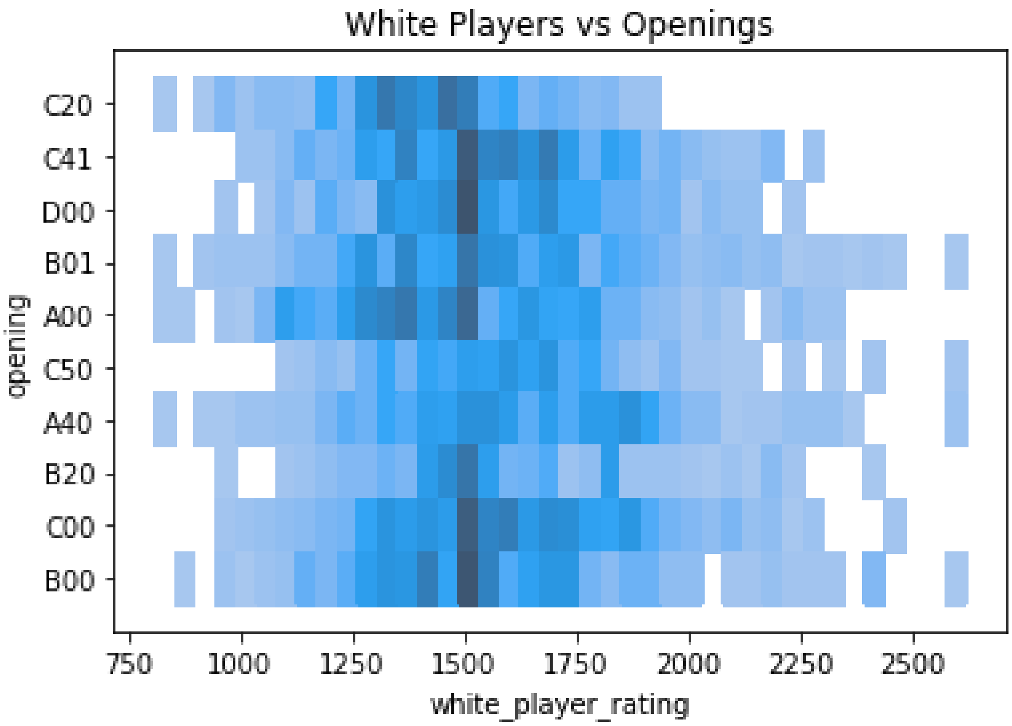
	white_player_rating	black_player_rating	opening	turns
0	1496	1500	C20	61
1	1523	1469	C41	95
2	1520	1423	D00	33
3	1381	1209	B01	119
4	1381	1272	A00	39
…	…	…	…	…
3370	1817	2050	A00	119
3371	2255	2008	C50	22
3372	1328	1252	C00	43
3373	1219	1250	A40	37
3374	1219	1286	D00	35

[3375 rows x 4 columns]

Lets see the distribution of openings per players by rating

[32]:
```
sns.histplot(x=data_common_openings['white_player_rating'],␣
 ↪y=data_common_openings['opening']).set_title('White Players vs Openings')
```

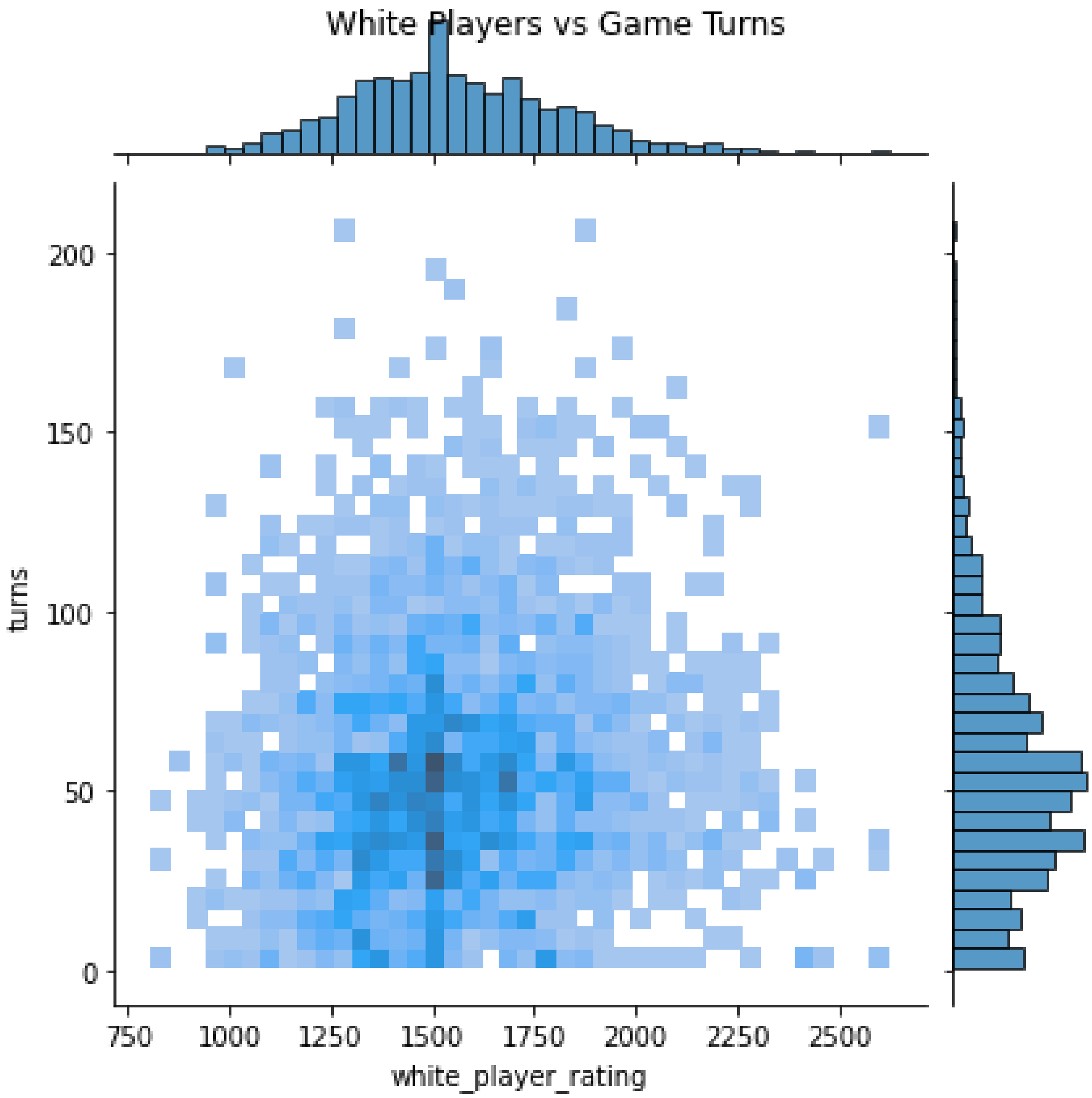
[32]: Text(0.5, 1.0, 'White Players vs Openings')



Lets see what is, on average the number of turns that a white player needs to win in relation to his rating.

[33]:
```
fig=sns.jointplot(data=data_common_openings, x="white_player_rating",␣
↪y="turns", kind='hist')
fig.fig.suptitle('White Players vs Game Turns')
```

[33]: Text(0.5, 0.98, 'White Players vs Game Turns')



1.5 Hypotetsis Testing

[34]:
```
data_enc.describe().T
```

[34]:

	count	mean	std	min	25%	50%	75%	\
turns	20058.0	60.465999	33.570585	1.0	37.0	55.0	79.0	
white_rating	20058.0	1596.631868	291.253376	784.0	1398.0	1567.0	1793.0	
black_rating	20058.0	1588.831987	291.036126	789.0	1391.0	1562.0	1784.0	
opening_ply	20058.0	4.816981	2.797152	1.0	3.0	4.0	6.0	
rated_enc	20058.0	0.805414	0.395891	0.0	1.0	1.0	1.0	
winner_enc	20058.0	1.044571	0.975038	0.0	0.0	1.0	2.0	
draw	20058.0	0.045169	0.207680	0.0	0.0	0.0	0.0	
mate	20058.0	0.315336	0.464661	0.0	0.0	0.0	1.0	

```
outoftime     20058.0    0.083757    0.277030    0.0    0.0    0.0    0.0
resign        20058.0    0.555738    0.496896    0.0    0.0    1.0    1.0

                 max
turns          349.0
white_rating  2700.0
black_rating  2723.0
opening_ply     28.0
rated_enc        1.0
winner_enc       2.0
draw             1.0
mate             1.0
outoftime        1.0
resign           1.0
```

1. First Hypothesis study

Null hypothesis: 60% of the first 100 games in the dataframe ends in less the 60 turns

Alternative hypothesis: 70% of the first 100 games in the dataframe ends in less the 60 turns

[35]:
```
null_count=0
alt_count=0

for i in range(100):
    if data.turns.iloc[i]<=60:
        null_count=null_count+1
    else:
        alt_count=alt_count+1
```

[36]:
```
print('Number or games ended in less then 60 turns:',null_count)
```

```
Number or games ended in less then 60 turns: 72
```

[37]:
```
print('Number or games ended in more then 60 turns:', alt_count)
```

```
Number or games ended in more then 60 turns: 28
```

P-value cutoff set at 5%

Cumulative distribution function

[38]:
```
prob = 1 - binom.cdf(71, 100, 0.6)

print(str(round(prob*100, 1))+"%")
```

```
0.8%
```

The probability that there is a 60% chance that a chess game ends in less then 60 turns is 0.8%. This is under our cutoff of 5% which means that we should reject the null and conclude that the probability for chess games to end in less 60 turns is greater then 60%

Getting the 95% cutoff

[39]:
```
print(binom.ppf(0.95,100,0.6)+1)
```

```
69.0
```

Which means that the odds that 69 games or less ends in less then 60 turns is 95% and that the odds that 69 games or more ends in less then 60 turns is 5%. If there was 69 games or less that ended in less the 60 turns we could accept the null hypotesis with a confidence level of 95%.

lets see what are the odds to get the number of games that ended in less then 60 turns (72 games out of 100).

[40]:
```
print (1-binom.cdf(72, 100, 0.6))
print (binom.cdf(72, 100, 0.7))
```

```
0.004600434301985534
0.7036338394422568
```

There is 5% chance that 60% of chess games end in less then 60 turns but we say they dont and 70% chance that more then 70% of chess games ends in less the 60 turns and we say they dont. So both will fail our 5% treshold

Lets increase the sample size

[41]:
```
print (binom.ppf(0.95,10000,0.6))
print (binom.ppf(0.05,10000,0.7))
```

```
6081.0
6925.0
```

1. We need a value higher ther 625 in order to say that 60% of the games ends in less then 60 turns
2. A value lower then 676 means that 70% of the games dont end in less then 60 turns

[42]:
```
print (1-binom.cdf(6500, 10000, 0.6))
print (binom.cdf(6500, 10000, 0.7))
```

```
1.1102230246251565e-16
3.133038425550557e-27
```

The probability of having a 60% chance for game that 650 games ends in less then 60 turns is very low. Similarly it is also very unlikely that 6500 games ends in less then 60 turns when there is 70% chance per game of this happening.