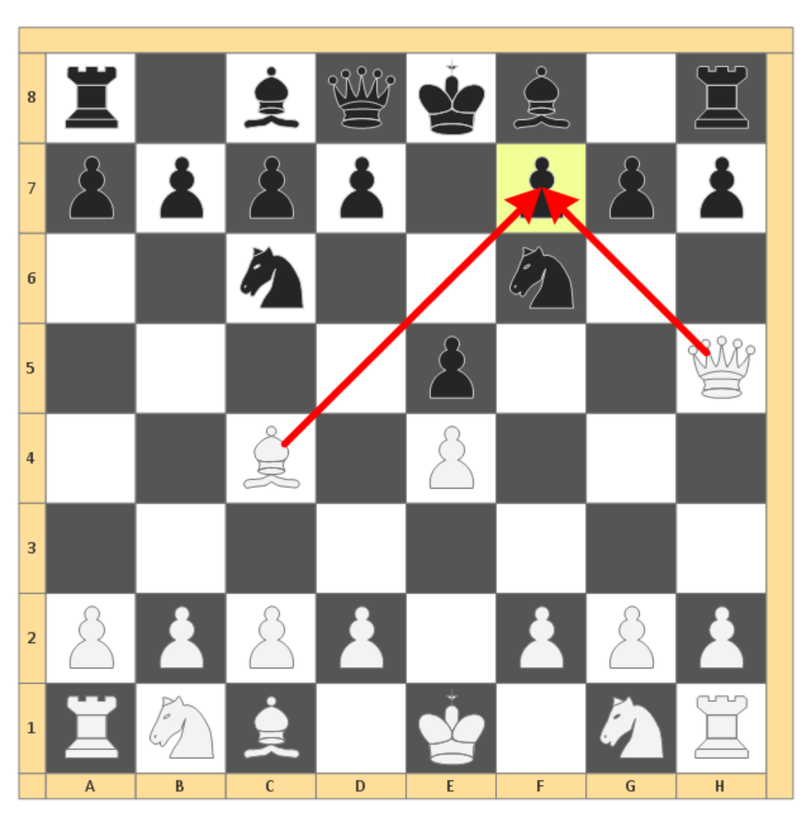
How do knights move?

David Na

EN.685.648 FA21
Lab Group 4

Problem Statement

Can we predict a victory for white?



Target Variable

$$\hat{y} = \begin{cases} 0, & \text{if white did not win} \\ 1, & \text{if white wins} \end{cases}$$

$$\hat{y} = \beta_0 + \beta_1 x$$

$$logistic(z) = logit^{-1}(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

The main target variable will be whether white wins or not.

We must one hot encode our target variable

The logistic model will output how the probability of y changes as our x changes.

In order to keep our variables to the range (0,1) we transform the variable using the logistic function.

(Module 9-10 Content)

Get

The Database: ChessDB.db

TABLES:

countries	: Maps country to Two letter code
users	: List of all scraped users
users_selected	: List of all the users I select to scrape from
all_games_raw	: List of all available games
all_games_country	: List of all games with the country filled
all_games_country_selected	: List of all games selected with country
all_games_countries_winstreak	: NOT INCLUDED IN DATA

Extracting the data: Step 1

Extract Users from a Team: (Only able to query by username for games)

```python
from Extractor import *
API_URL = 'https://lichess.org/'
AccessToken = "lip_FdeTEN7DhU3ge3Eg8Shu"
API = LichessRequestor(Token = AccessToken, base_url = API_URL)
```

```python
path = f'api/team/agadmators-team/users'
data = API.get(path=path)
```

```python
users_raw = LichessRequestor.parse(data)
```

```python
user_list = LichessRequestor.parse_team(users_raw)
```

```sql
query = """
Select t1.id, t1.blitz_games, t1.blitz_rating, t1.rapid_games, t1.rapid_rating, t1.bullet_games,
t1.bullet_rating, t1.country
FROM
(SELECT
    id, blitz_games, blitz_rating, rapid_games, rapid_rating, bullet_games, bullet_rating,
    (CASE country
        WHEN "GB-ENG" THEN "UK"
        WHEN "GB-NIR" THEN "UK"
        WHEN "GB-SCT" THEN "UK"
        WHEN "GB-WLS" THEN "UK"
        WHEN "_united-nations" THEN "UK"
        WHEN "ES-CT" THEN "ES"
        WHEN "CA-QC" THEN "CA"
        ELSE country
    END) as country
FROM users
where
country not in ("No Country", "_east-turkestan", "_adygea", "_earth", "_belarus-wrw",
"_lichess", "_pirate", "_rainbow", "east-turkestan")
AND blitz_games > 100
AND rapid_games > 100
AND bullet_games > 100) t1

"""
```

Using the User list created above, I run a SQL query to "clean" the data. I also choose to only include users who have played at least 100 games in the three types of games I query for

Extracting the data: Step 2

```
## Random sampling from the user dataset
## Querying all the data would take 60 hours...
userlist_cleaned = []
AllUsers = pd.DataFrame()
Ratingless800 = users[users['blitz_rating'] <= 800]
Rating800to1000 = users[(users['blitz_rating'] > 800) & (users['blitz_rating'] <= 1000)]
Rating1000to1200 = users[(users['blitz_rating'] > 1000) & (users['blitz_rating'] <= 1200)].sample(100)
Rating1200to1400 = users[(users['blitz_rating'] > 1000) & (users['blitz_rating'] <= 1200)].sample(100)
Rating1400to1600 = users[(users['blitz_rating'] > 1400) & (users['blitz_rating'] <= 1600)].sample(250)
Rating1600to1800 = users[(users['blitz_rating'] > 1600) & (users['blitz_rating'] <= 1800)].sample(250)
Rating1800to2000 = users[(users['blitz_rating'] > 1800) & (users['blitz_rating'] <= 2000)].sample(250)
Rating2000to2200 = users[(users['blitz_rating'] > 2000) & (users['blitz_rating'] <= 2200)].sample(250)
RatingAbove2200 = users[(users['blitz_rating'] > 2200)].sample(250)
userlist_cleaned = [Ratingless800, Rating800to1000, Rating1000to1200, Rating1200to1400, Rating1200to1400
         ,Rating1400to1600, Rating1600to1800, Rating1800to2000, Rating2000to2200, RatingAbove2200]
UsersSelected = pd.concat(userlist_cleaned, ignore_index = True)
```

```
UsersSelected.to_csv("Users_Selected.csv")
```

Based on the cleaned set of users, I sample the dataset by rating bracket. I take users from every increment of 200 rating from 800 to 2200.

The lower ratings had less users – sample less

```
## Query games from user database
all_games = []
all_games_df = pd.DataFrame()
for i in range(len(UsersSelected)):
    print(i)
    params = {
        'max': 15,
        'rated': True,
        'perfType': 'blitz,rapid,bullet',
        'analysed': True,
        'evals': True,
        'opening': True,
    }
    username = UsersSelected['id'][i]
    path = f'api/games/user/{username}'
    games = API.get(path=path, params = params)
    games_data = LichessRequestor.parse(games, convert=False)
    data = LichessRequestor.parse_games(games_data, LichessRequestor)
    all_games.append(data)
all_games_df = pd.concat(all_games, ignore_index = True)
```

```
all_games_df.to_csv("all_games.csv")
```

I run a for loop for each user in the user selected list provided above

I pull 15 games of type: blitz, rapid, bullet for each user

Data Dictionary

Lichess.org

gameid	The unique game id
white_id	The unique player id for white
white_rating	The rating for white for specific game type
white_country	The country of origin for white
white_games	Then number of games played for white (of type = game_type)
white_win_last_10	How many wins for white in last 10 games (omitted)
white_inaccuracies	Number of inaccuracies
white_mistakes	Number of mistakes
white_blunder	Number of blunders
white_acpl	Adjusted score of evaluation – Average Centipawn Loss
game_type	The type of the game: Blitz, Rapid, Bullet
opening	The opening played by white
winner	The winner (white or black or draw)
win_by	Outcome of the game (resign, draw, stalemate, etc.)

The variables for white are also provided for black

Extracting the data: Step 3

Lichess.org

```
all_games_dict = all_games_df.to_dict()
all_games_df_with_countries = LichessRequestor.parse_countries(all_games_dict)
```

```
all_games_df_with_countries['black_country'].replace({"GB-ENG": "UK", "GB-NIR": "UK", "GB-SCT": "UK",
                                                      "GB-WLS": "UK", "_united-nations" : "UN", "ES-CT": "ES",
                                                      "CA-QC": "CA"}, inplace = True)
all_games_df_with_countries['white_country'].replace({"GB-ENG": "UK", "GB-NIR": "UK", "GB-SCT": "UK",
                                                      "GB-WLS": "UK", "_united-nations" : "UN", "ES-CT": "ES",
                                                      "CA-QC": "CA"}, inplace = True)
```

```
## White details
api = LichessRequestor(Token = AccessToken, base_url = API_URL)
white_id = games_data['white_id'][index]
path_white = f'api/user/{white_id}'
white = LichessRequestor.parse(api.get(path=path_white))
if 'profile' in white[0]:
    if 'country' in white[0]['profile']:
        games_data['white_country'][index] = (white[0]['profile']['country'])
    else:
        games_data['white_country'][index] = ("No Country")
else:
    games_data['white_country'][index] = "No Country"

try:
    games_data['white_games'][index] = white[0]['perfs'][game_type]['games']
except:
    games_data['white_games'][index] = -1
    pass
```

Done for Black and White

For every game that we pulled, we run another API call step to pull the number of games played as well as the country of origin for the player

The API get request does not pull profile information with the game data.

Extracting the data: Step 4

```python
## White details
white_id = games_data_list['white_id'][index]
path_wg = f'api/games/user/{white_id}'
api = LichessRequestor(Token = AccessToken, base_url = API_URL)

## Some JSON Decode errors because of API
try:
    white_games = LichessRequestor2.parse(api.get(path=path_wg, params=params))
    wins = 0
    for g in white_games:
        if 'winner' in g:
            winner = g['winner']
            if (white_id == g['players'][winner]['user']['id']):
                wins += 1
    games_data_list['white_win_last_10'][index] = wins
except ValueError:
    games_data_list['white_win_last_10'][index] = -1
    pass

api.session.close()
```

```python
params = {
    'max': 10,
    'until': game_time,
    'rated': True,
    'perfType': game_type,
    'tags': False,
    'moves': False,
    'clock': False
}
```

The next step of the API was to call the requestor again and parse the last ten games of the same game type played by the user.

We can pull the user games as we recorded each users' id

This step was taking over 24 hours to complete. I was able to run it successfully once, but then my python kernel died. I lost the metadata and was not able to rerun this step.

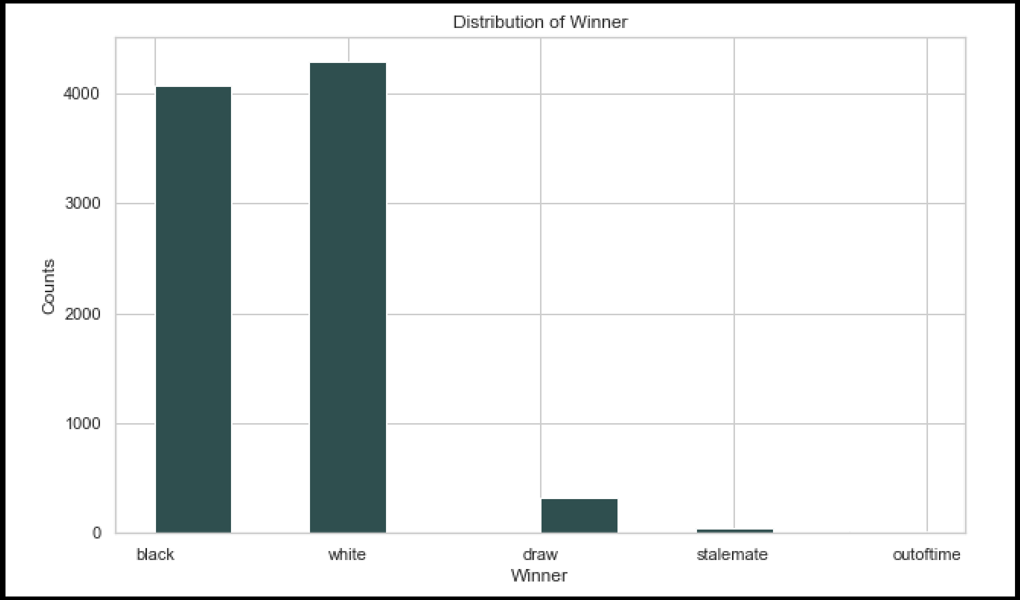
The Database: ChessDB.db

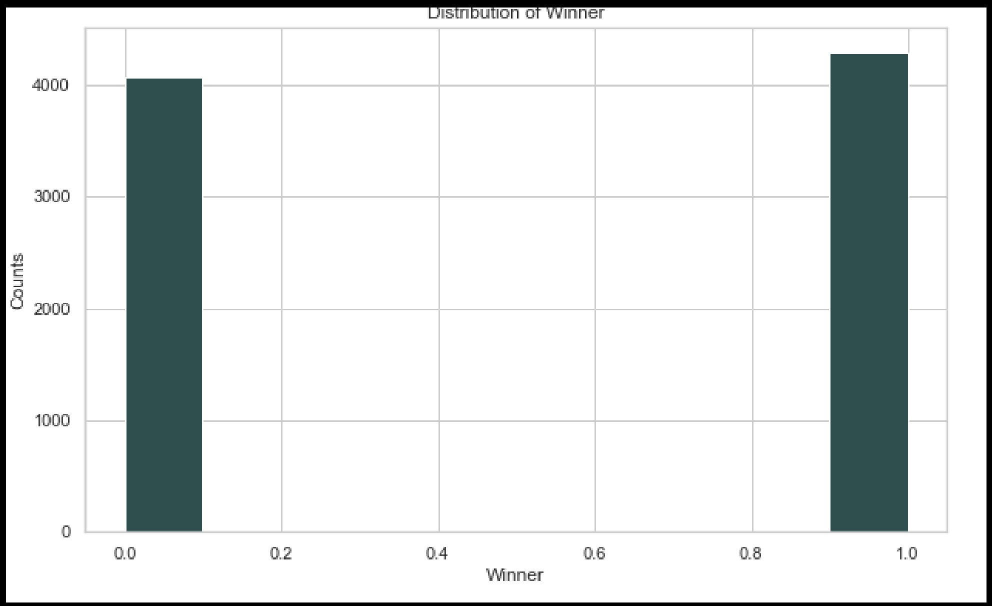
TABLES:

countries	: Maps country to Two letter code
users	: List of all scraped users
users_selected	: List of all the users I select to scrape from
all_games_raw	: List of all available games
all_games_country	: List of all games with the country filled
all_games_country_selected	: List of all games selected with country
all_games_countries_winstreak	: NOT INCLUDED IN DATA

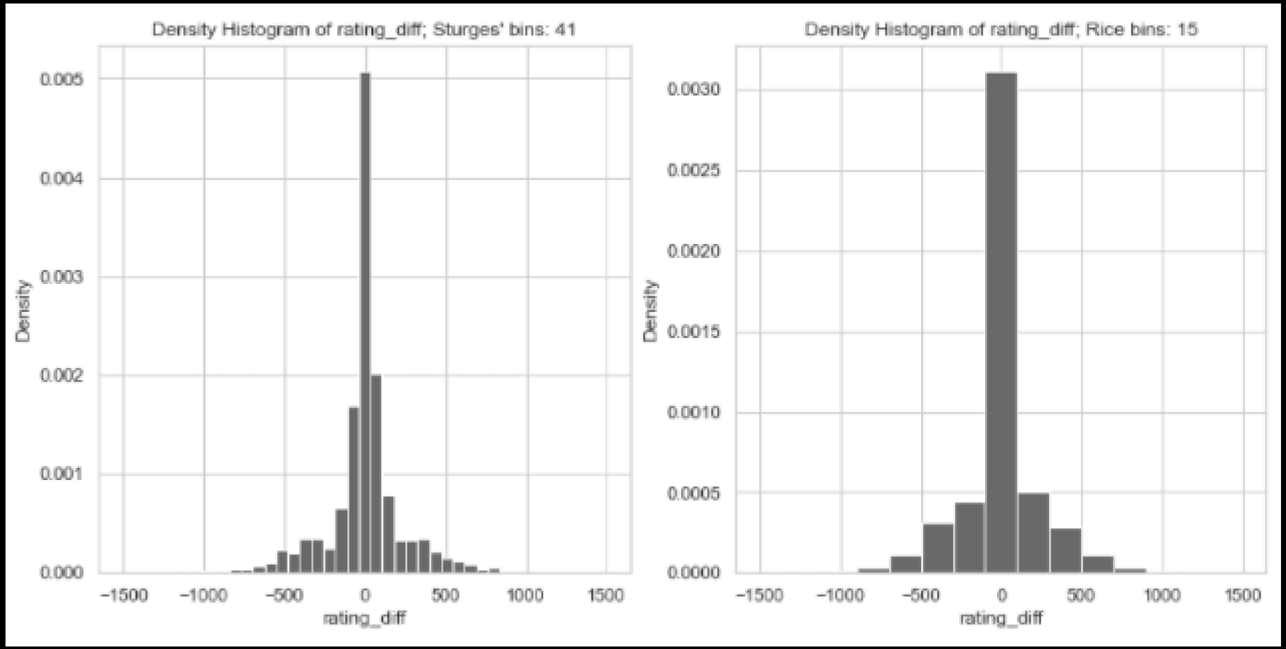
Explore

Target Variable: Winner





Rating – Taking the Difference

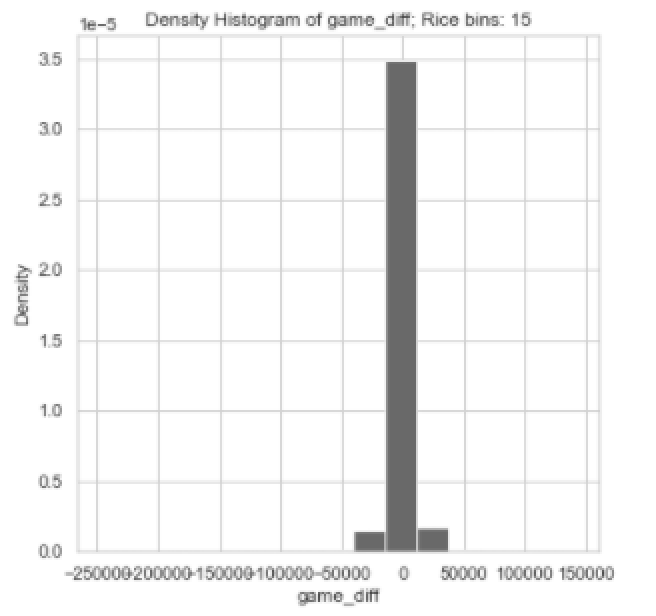



```
When Rating Diff is 0
1984
2274
When Rating Diff is 100
686
900
When Rating Diff is 200
476
525
When Rating Diff is 300
409
367
When Rating Diff is 400
193
240
```

```
When Rating Diff is 500
96
169
When Rating Diff is 600
43
104
When Rating Diff is 700
13
62
When Rating Diff is 800
5
39
When Rating Diff is 900
2
24
```

White wins with the higher rating
White wins with the lower rating

Number of Games - Difference

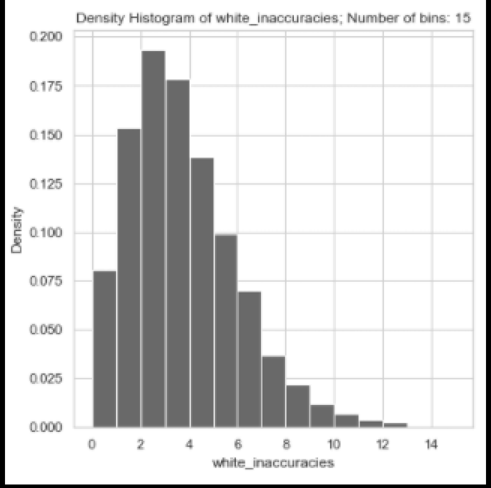


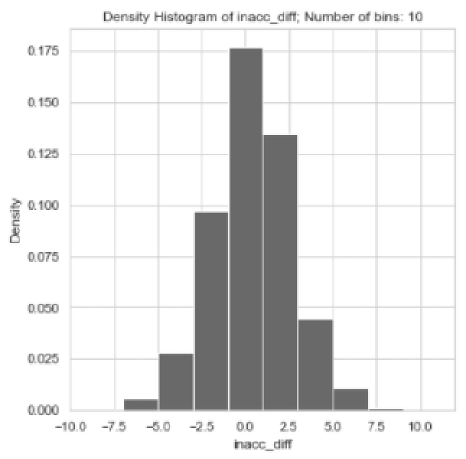
Still disparity in win probability despite high differences in game counts.

	white_games	black_games	winner
2135	61.0	103123.0	1
2136	103123.0	52.0	0
2137	22.0	103123.0	1
2138	247.0	103123.0	1
2139	416.0	103123.0	1
2140	103123.0	496.0	1
2377	127085.0	765.0	0
2564	2978.0	116036.0	1
3010	2044.0	115046.0	1
3569	24001.0	270321.0	1
3583	1657.0	105070.0	0
3638	116752.0	956.0	1
3952	118440.0	4089.0	1
5038	2694.0	128454.0	0
7241	146377.0	5598.0	1
8148	118977.0	2603.0	1

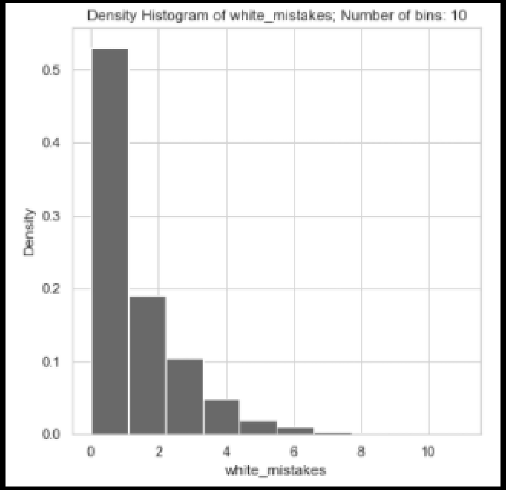
Inaccuracies, mistakes and blunders density histograms

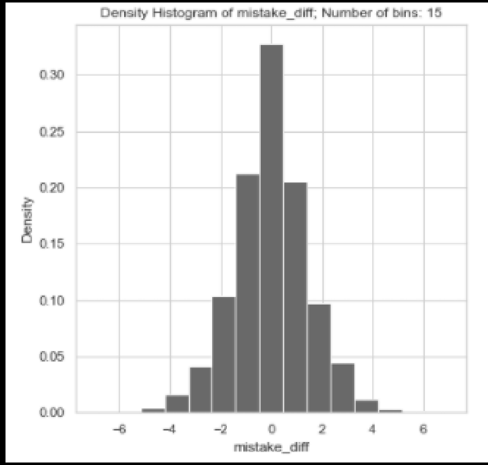
Inaccuracies



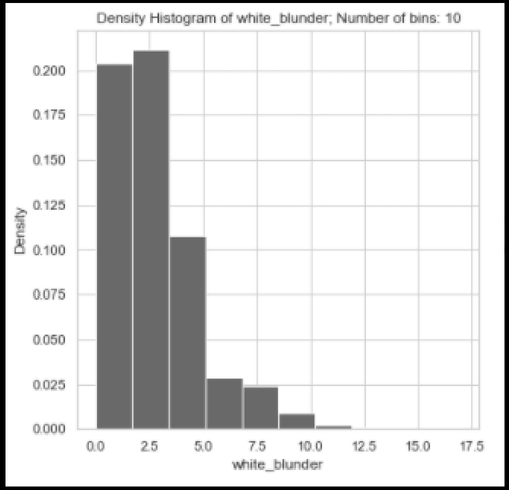


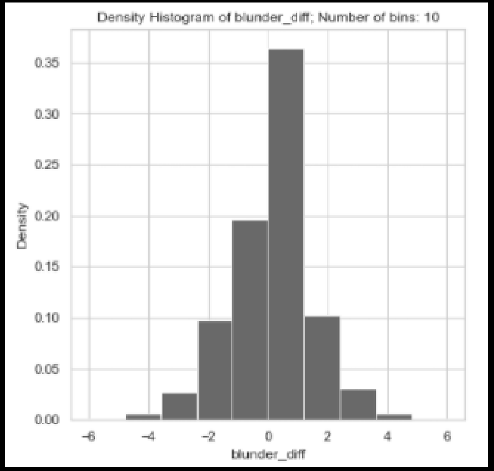
Mistakes





Blunders



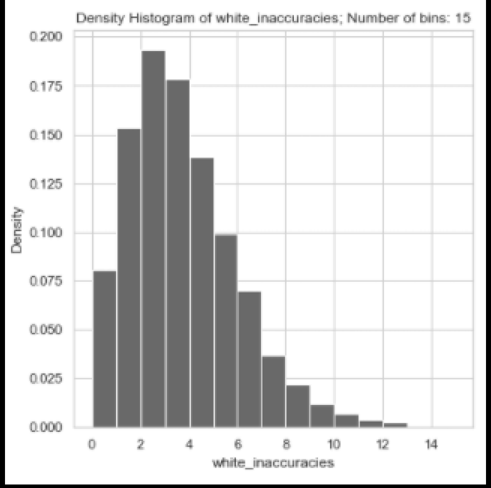


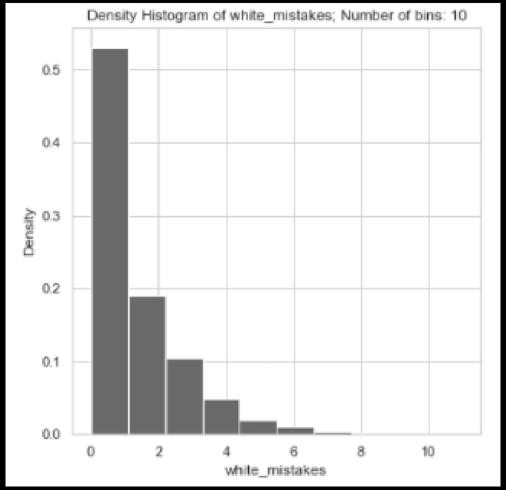
Inaccuracies, mistakes and blunders density histograms

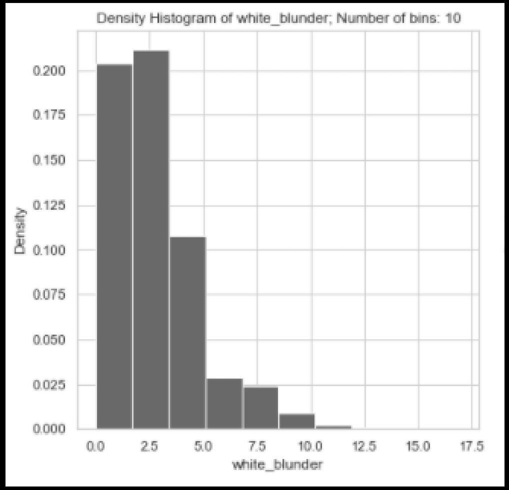
Inaccuracies

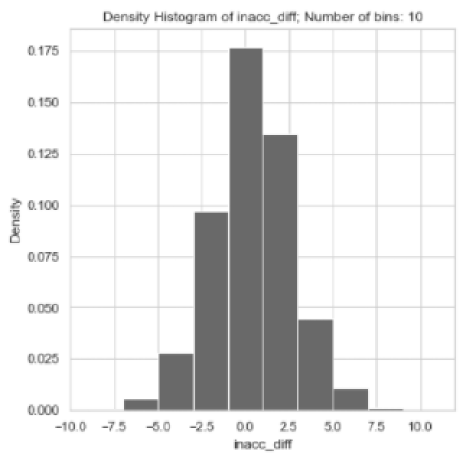
Mistakes

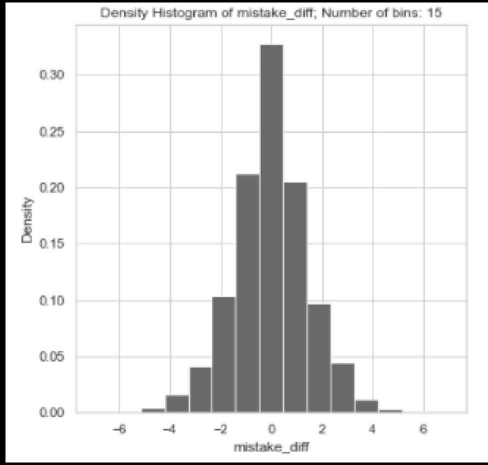
Blunders

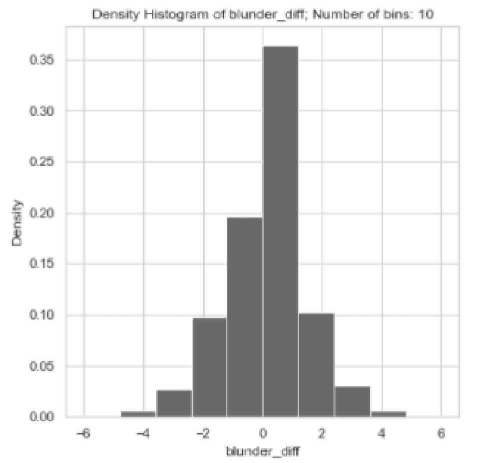




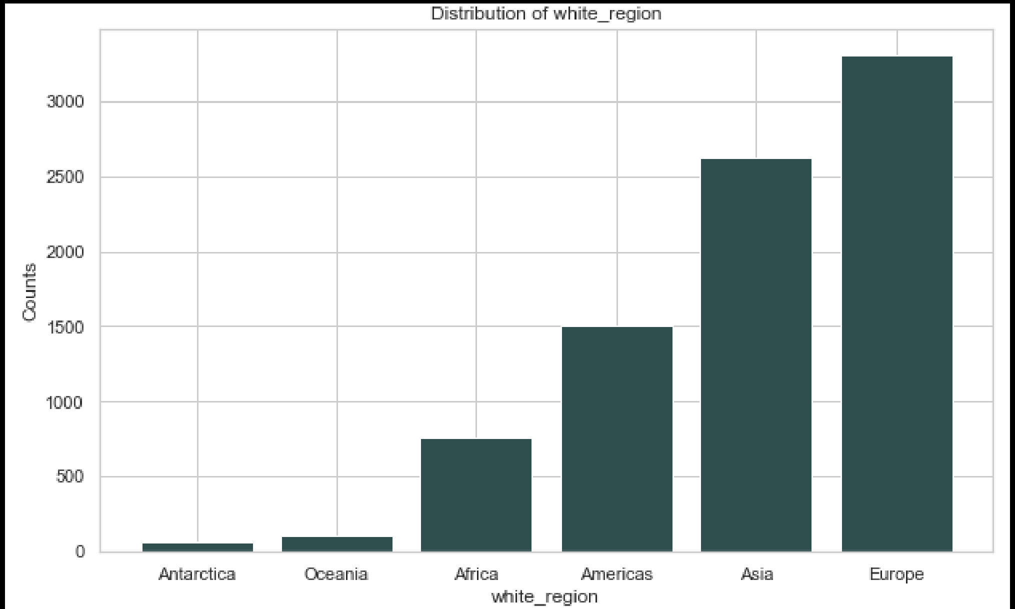


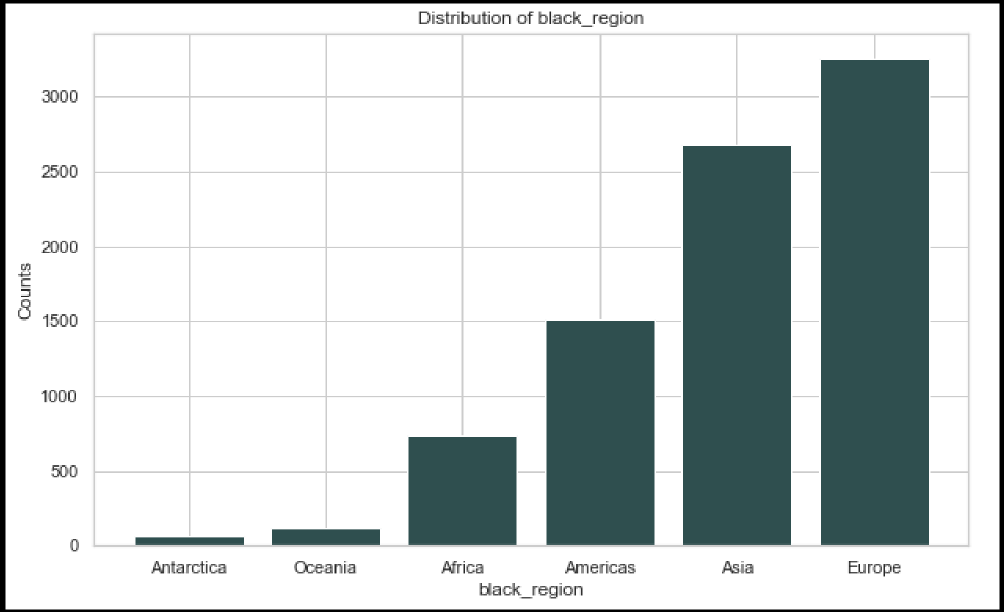




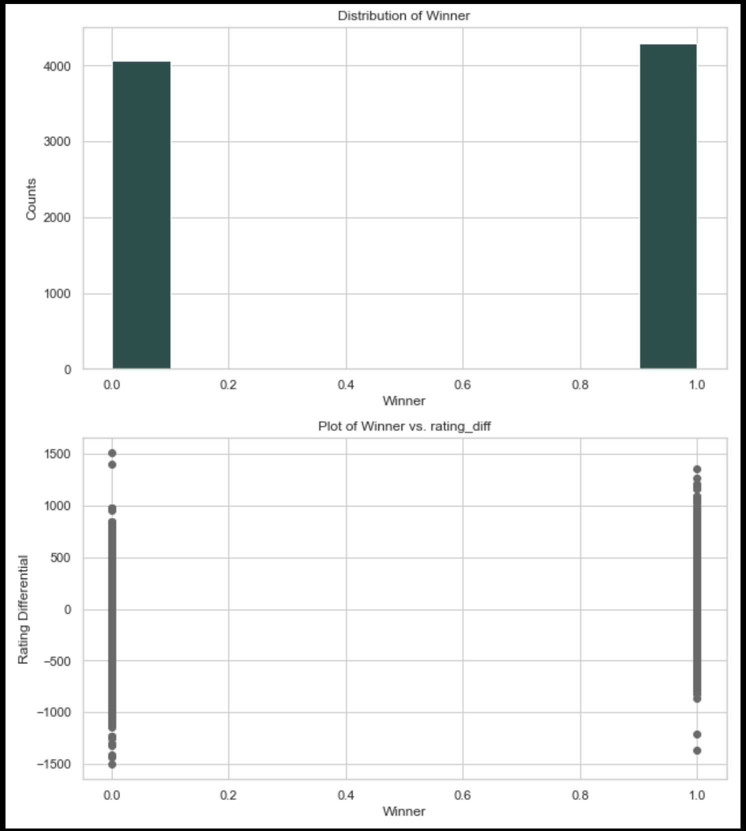


Country to Continent





Winner vs. Rating



Data seems to be relatively uniform between wins and losses

There seems to be some change in the win rate as the rating differential increases.

Winner vs. Inaccuracies, mistakes, and blunders

winner	0	1
count	4066.000000	4295.000000
mean	0.382932	-0.654482
std	2.266759	2.239671
min	-9.000000	-9.000000
25%	-1.000000	-2.000000
50%	0.000000	-1.000000
75%	2.000000	1.000000
max	9.000000	11.000000

Delta Inaccuracies

winner	0	1
count	4066.000000	4295.000000
mean	0.228234	-0.269616
std	1.605663	1.549057
min	-6.000000	-7.000000
25%	-1.000000	-1.000000
50%	0.000000	0.000000
75%	1.000000	1.000000
max	7.000000	6.000000

Delta Mistakes

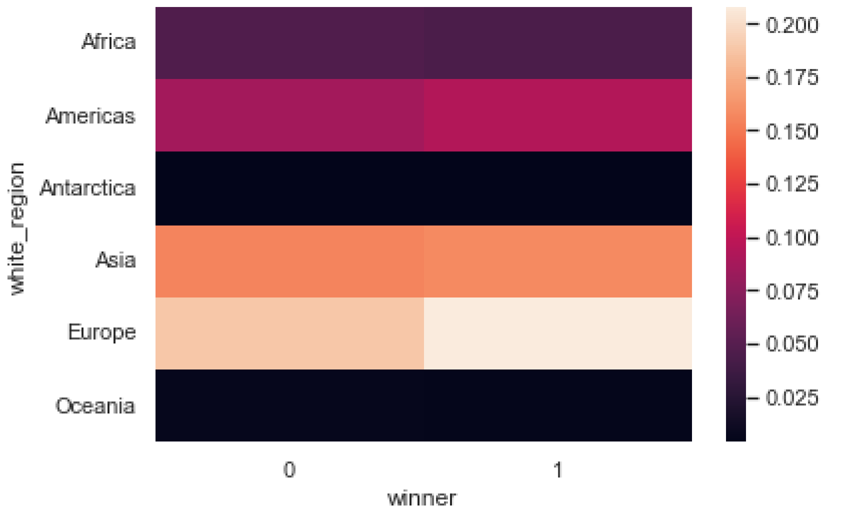
winner	0	1
count	4066.000000	4295.000000
mean	0.933104	-0.864261
std	1.285725	1.222734
min	-6.000000	-6.000000
25%	0.000000	-2.000000
50%	1.000000	-1.000000
75%	2.000000	0.000000
max	6.000000	5.000000

Delta Blunders

Winner vs. Region Black and White

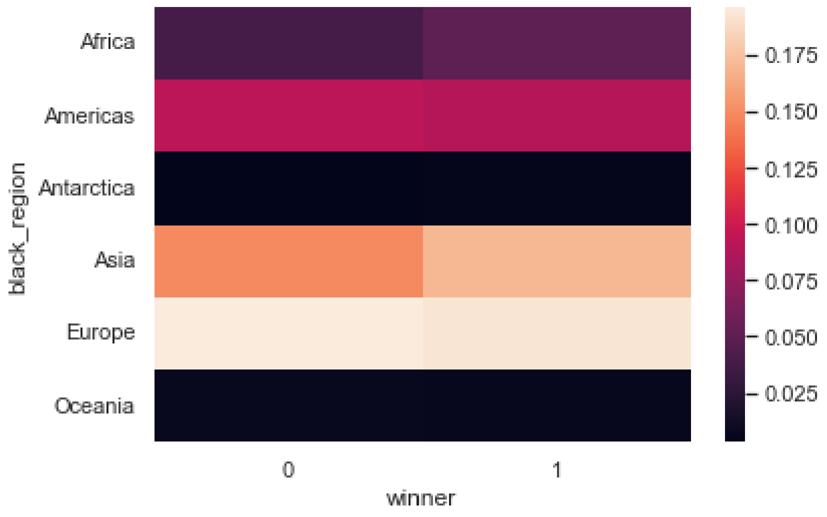
```
winner              0         1
white_region
Africa       0.046286  0.044253
Americas     0.086114  0.093649
Antarctica   0.004066  0.003708
Asia         0.155364  0.158235
Europe       0.188016  0.207870
Oceania      0.006459  0.005980

<AxesSubplot:xlabel='winner', ylabel='white_region'>
```




```
winner              0         1
black_region
Africa       0.038034  0.049874
Americas     0.092333  0.088626
Antarctica   0.002990  0.004904
Asia         0.149504  0.170434
Europe       0.196508  0.193159
Oceania      0.006937  0.006698

<AxesSubplot:xlabel='winner', ylabel='black_region'>
```



Winner vs. Same region, Same Country

```
0    0.603158
1    0.396842
Name: same_region, dtype: float64
```

```
0    0.862935
1    0.137065
Name: same_country, dtype: float64
```

```
winner              0         1
same_region
0            0.299725  0.303433
1            0.186581  0.210262
```

```
winner              0         1
same_country
0            0.423155  0.439780
1            0.063150  0.073915
```

Results seem to indicate that there may be some change caused by two players being from the same region but not from the same country

Model

Null Model

Baseline Model: Bernoulli Distribution

For the Bernoulli model, we learned that we could predict with a single parameter.

$p = the\ probability\ of\ "success“$

$p = The\ probability\ of\ white\ winning\ the\ next\ game$

Given:

$$\hat{y} = \begin{cases} 0, & \text{if white did not win} \\ 1, & \text{if white loses} \end{cases}$$

Our Null Model:

$p = mean(\hat{y})$
$p = 51.37\%$

There is a 51.37% chance that white wins the next chess match.

Logistic Regression

Model With All Variables

			95% BCI		
Coefficients		Mean	Lo	Hi	P(y=1)
	β_0	-0.00	-0.05	-0.00	0.50
white_rating	β_1	-0.00	-0.00	-0.00	-0.00
white_games	β_2	0.00	-0.00	0.00	0.00
rating_diff	β_3	0.00	0.00	0.00	0.00
game_diff	β_4	0.00	0.00	0.00	0.00
inacc_diff	β_5	0.00	-0.01	0.09	0.00
mistake_diff	β_6	0.00	-0.01	0.14	0.00
blunder_diff	β_7	0.00	0.00	0.30	0.00
acpl_diff	β_8	0.10	0.09	0.11	0.03
same_region_yes	β_9	0.00	-0.00	0.08	0.00
blitz	β_{10}	0.00	-0.07	0.00	0.00
rapid	β_{11}	-0.00	-0.01	0.02	-0.00

Metrics	Mean	Lo	Hi
Error (%)	7.99	7.22	8.60
Efron's R^2	0.74	0.72	0.76

Model With All Variables (3 sd)

			95% BCI		
Coefficients		Mean	Lo	Hi	P(y=1)
	β_0	-0.000	-0.047	-0.000	0.500
white_rating	β_1	-0.000	-0.000	-0.000	-0.000
white_games	β_2	0.000	0.000	0.000	0.000
rating_diff	β_3	0.001	0.001	0.001	0.000
game_diff	β_4	0.000	0.000	0.000	0.000
inacc_diff	β_5	0.002	-0.006	0.090	0.000
mistake_diff	β_6	0.001	-0.009	0.184	0.000
blunder_diff	β_7	0.003	0.003	0.297	0.001
acpl_diff	β_8	0.104	0.089	0.108	0.026
same_region_yes	β_9	0.000	0.000	0.082	0.000
blitz	β_{10}	0.000	-0.056	0.000	0.000
rapid	β_{11}	-0.000	-0.007	0.030	-0.000

Metrics	Mean	Lo	Hi
Error (%)	7.989	7.401	8.519
Efron's R^2	0.739	0.723	0.758

Mean Centering

Mean centering helps to scale our data. We want 0 values to be meaningful

```
white_rating_centered
Unit Difference: -0.0198
white_games_centered
Unit Difference: -0.0001
rating_diff_centered
Unit Difference: -0.0
game_diff_centered
Unit Difference: -0.0003
inacc_diff_centered
Unit Difference: -0.0
mistake_diff_centered
Unit Difference: -0.0182
blunder_diff_centered
Unit Difference: -0.0055
acpl_diff_centered
Unit Difference: -0.0397
same_region_yes
Unit Difference: 0.0242
blitz
Unit Difference: 0.0176
rapid
Unit Difference: 0.0043
```

Mean-Centered Model (3 sd)

			95% BCI		
Coefficients		**Mean**	**Lo**	**Hi**	**P(y=1)**
	β_0	0.080	0.000	0.121	0.520
white_rating_centered	β_1	0.000	0.000	0.001	0.000
white_games_centered	β_2	0.000	-0.000	0.000	0.000
rating_diff_centered	β_3	0.001	0.001	0.002	0.000
game_diff_centered	β_4	0.000	0.000	0.000	0.000
inacc_diff_centered	β_5	0.073	0.002	0.091	0.018
mistake_diff_centered	β_6	0.022	-0.017	0.111	0.005
blunder_diff_centered	β_7	0.160	0.003	0.276	0.040
acpl_diff_centered	β_8	0.097	0.092	0.109	0.024
same_region_yes	β_9	0.071	0.000	0.106	0.018
blitz	β_{10}	0.017	-0.019	0.045	0.004
rapid	β_{11}	0.032	-0.000	0.056	0.008

Metrics	**Mean**	**Lo**	**Hi**
Error (%)	7.954	7.349	8.482
Efron's R^2	0.740	0.721	0.757

Unit Change for the initial model

As we know that a unit change may necessarily be different per variable, we change the per unit change for the variables to get a more accurate picture.

```
white_rating_centered100
Unit Difference: -0.0265
white_games_centered1000
Unit Difference: -0.0091
rating_diff_centered100
Unit Difference: -0.003
game_diff_centered1000
Unit Difference: -0.0291
inacc_diff_centered
Unit Difference: -0.0073
mistake_diff_centered
Unit Difference: -0.0135
blunder_diff_centered
Unit Difference: -0.0222
acpl_diff_centered10
Unit Difference: -0.0561
same_region_yes
Unit Difference: 0.2312
blitz
Unit Difference: 0.062
rapid
Unit Difference: -0.0375
```

			95% BCI		
Coefficients		Mean	Lo	Hi	P(y=1)
	β_0	0.080	0.000	0.121	0.520
white_rating_centered	β_1	0.000	0.000	0.001	0.000
white_games_centered	β_2	0.000	-0.000	0.000	0.000
rating_diff_centered	β_3	0.001	0.001	0.002	0.000
game_diff_centered	β_4	0.000	0.000	0.000	0.000
inacc_diff_centered	β_5	0.073	0.002	0.091	0.018
mistake_diff_centered	β_6	0.022	-0.017	0.111	0.005
blunder_diff_centered	β_7	0.160	0.003	0.276	0.040
acpl_diff_centered	β_8	0.097	0.092	0.109	0.024
same_region_yes	β_9	0.071	0.000	0.106	0.018
blitz	β_{10}	0.017	-0.019	0.045	0.004
rapid	β_{11}	0.032	-0.000	0.056	0.008

Metrics	Mean	Lo	Hi
Error (%)	7.954	7.349	8.482
Efron's R^2	0.740	0.721	0.757

Something fishy...

		95% BCI			
Coefficients		Mean	Lo	Hi	P(y=1)
	β_0	0.129	0.054	0.206	0.532
inacc_diff_centered	β_1	0.044	0.014	0.071	0.011
mistake_diff_centered	β_2	0.078	0.026	0.136	0.020
blunder_diff_centered	β_3	0.210	0.116	0.297	0.053
acpl_diff_centered10	β_4	0.913	0.854	0.960	0.228

Metrics	Mean	Lo	Hi
Error (%)	7.954	7.401	8.566
Efron's R^2	0.737	0.715	0.755

I knew it...

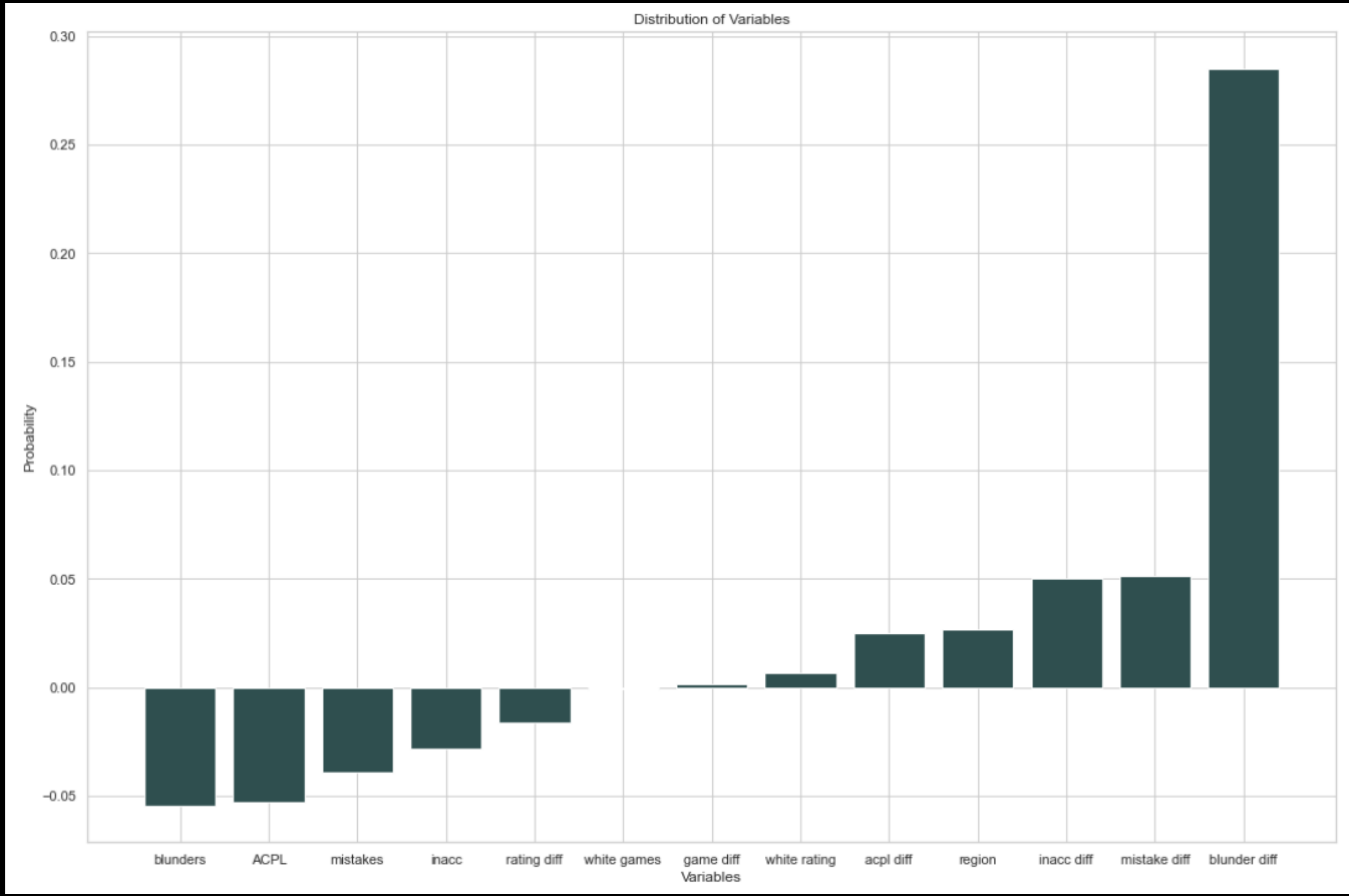
There was something off about the model. It seemed that the probability was being predicted solely by these four variables.

Now we know if we can predict the future and know exactly the moves white would make, we could predict his win...

Obviously, this isn't helpful for our model answering the problem...

Backtracking...

Its time to look at each of the variables



blunder_diff_centered	0.285229
inacc_diff	0.051539
mistake_diff_centered	0.050369
same_region_yes	0.026748
acpl_diff_centered	0.025284
white_rating_centered100	0.006442
game_diff_centered1000	0.001465
white_games_centered1000	-0.000677
rating_diff_centered100	-0.016399
white_inaccuracies_centered	-0.028104
white_mistakes_centered	-0.038923
white_acpl_centered10	-0.052568
white_blunder_centered	-0.054338

Our bad model

			95% BCI		
Coefficients		Mean	Lo	Hi	P(y=1)
	β_0	0.031	-0.046	0.097	0.508
game_diff_centered1000	β_1	0.010	0.005	0.016	0.003
white_rating_centered100	β_2	0.017	0.003	0.033	0.004
blunder_diff_centered	β_3	1.143	1.089	1.194	0.286
same_region_yes	β_4	0.117	0.033	0.211	0.029

Metrics	Mean	Lo	Hi
Error (%)	21.373	20.510	22.325
Efron's R^2	0.385	0.366	0.408

```
game_diff_centered1000
Unit Difference: -0.0078
white_rating_centered100
Unit Difference: -0.0026
blunder_diff_centered
Unit Difference: -0.0043
same_region_yes
Unit Difference: 0.2776
```

Validation

Validation Metrics:

Accuracy

```
Accuracy 95% CI:  [0.53635766 0.6005622 ]
Mean:  0.5662390267511943
Null Mean:  0.5136945341466331
```

Error Rate

```
Accuracy 95% CI:  [0.3994378  0.46364234]
Error Rate:  0.43376097324880575
Null Error:  0.48630546585336687
```

Compared to our null model, our model performed slightly better.

Future Additions

- Different type of model
- Additional Variables/More robust data to train on
- More time to make more modified API calls to get better data