League of Legend

Name(s): Ziheng Tang

Website Link: (your website link)

In [684…
```
import pandas as pd
import numpy as np
from pathlib import Path

import plotly.express as px
pd.options.plotting.backend = 'plotly'

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import Binarizer, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier

# from dsc80_utils import * # Feel free to uncomment and use this.
```

Step 1: Introduction

Quick view of the dataset

In [685…
```
full_dataset = pd.read_csv('2025_LoL_esports_match_data_from_OraclesElixir.csv')
```

```
C:\Users\tangz\AppData\Local\Temp\ipykernel_6608\149352961.py:1: DtypeWarning:

Columns (2) have mixed types. Specify dtype option on import or set low_memory=Fals
e.
```

In [686…
```
pd.set_option('display.max_columns', None)
# pd.reset_option('display.max_columns')
full_dataset.head()
```

Out[686…

	gameid	datacompleteness	url	league	year	split	playoffs	date	g
0	LOLTMNT03_179647	complete	NaN	LFL2	2025	Winter	0	2025-01-11 11:11:24	
1	LOLTMNT03_179647	complete	NaN	LFL2	2025	Winter	0	2025-01-11 11:11:24	
2	LOLTMNT03_179647	complete	NaN	LFL2	2025	Winter	0	2025-01-11 11:11:24	
3	LOLTMNT03_179647	complete	NaN	LFL2	2025	Winter	0	2025-01-11 11:11:24	
4	LOLTMNT03_179647	complete	NaN	LFL2	2025	Winter	0	2025-01-11 11:11:24	

In [687…
```
full_dataset.shape[0]
```

Out[687… 118932

This dataset contains the information similar to the Results Screen after each game. I have split the data in to sections for better understanding:

1. **global identification data**: gameid, url, league, year, split, playoffs, date, game, patch, participantid, playername, playerid, teamname, teamid
2. **pre-game data**: side, position, champion, ban1, ban2, ban3, ban4, ban5, pick1, pick2, pick3, pick4, pick5
3. **end results**: gamelength, result, kills, deaths, assists, teamkills, teamdeaths, doublekills, triplekills, quadrakills, pentakills, firstblood, firstbloodkill, firstbloodassist, firstbloodvictim, team kpm, ckpm, firstdragon, dragons, opp_dragons, elementaldrakes, opp_elementaldrakes, infernals, mountains, clouds, oceans, chemtechs, hextechs, dragons (type unknown), elders, opp_elders, firstherald, heralds, opp_heralds, void_grubs, opp_void_grubs, firstbaron, barons, opp_barons, atakhans, opp_atakhans, firsttower, towers, opp_towers, firstmidtower, firsttothreetowers, turretplates, opp_turretplates, inhibitors, opp_inhibitors, damagetochampions, dpm, damageshare, damagetakenperminute, damagemitigatedperminute, damagetotowers, wardsplaced, wpm, wardskilled, wcpm, controlwardsbought, visionscore, vspm, totalgold, earnedgold, earned gpm, earnedgoldshare, goldspent, gspd, gpr, total cs, minionkills, monsterkills, monsterkillsownjungle, monsterkillsenemyjungle, cspm, goldat10, xpat10, csat10, opp_goldat10, opp_xpat10, opp_csat10, golddiffat10, xpdiffat10, csdiffat10, killsat10,

assistsat10, deathsat10, opp_killsat10, opp_assistsat10, opp_deathsat10, goldat15, xpat15, csat15, opp_goldat15, opp_xpat15, opp_csat15, golddiffat15, xpdiffat15, csdiffat15, killsat15, assistsat15, deathsat15, opp_killsat15, opp_assistsat15, opp_deathsat15, goldat20, xpat20, csat20, opp_goldat20, opp_xpat20, opp_csat20, golddiffat20, xpdiffat20, csdiffat20, killsat20, assistsat20, deathsat20, opp_killsat20, opp_assistsat20, opp_deathsat20, goldat25, xpat25, csat25, opp_goldat25, opp_xpat25, opp_csat25, golddiffat25, xpdiffat25, csdiffat25, killsat25, assistsat25, deathsat25, opp_killsat25, opp_assistsat25, opp_deathsat25

4. datacompleteness

Questions to explore:

1. What are some interesting traits of the data for the large neutral monsters (atakhans, barons, dragons, and such)?
2. Can we predict a winning team just by looking at some combinations of the large neutral monsters data?
3. Which large neutral monsters contribute the most to the win rate?

This project will focus on question 2, but we will be exploring related questions such as question 1 & 3 as we procceed.

Step 2: Data Cleaning and Exploratory Data Analysis

2.1 Getting what we need

Since we are only looking at the win result and data for neutral monsters, lets clean the data so that we only have what we need.

We will be needing:

- for easier to identify each game: `gameid`
- win result : `result`
- neutral monsters : `firstdragon` , `dragons` , `opp_dragons` , `elementaldrakes` , `opp_elementaldrakes` , `infernals` , `mountains` , `clouds` , `oceans` , `chemtechs` , `hextechs` , `dragons (type unknown)` , `elders` , `opp_elders` , `firstherald` , `heralds` , `opp_heralds` , `void_grubs` , `opp_void_grubs` , `firstbaron` , `barons` , `opp_barons` , `atakhans` , `opp_atakhans`
- to make missingness easier: `datacompleteness`
- to know if it is team data or single player data: `position`
- to identify which side is the team on: `side`

In [688…

```
neutral_monsters_dataset = full_dataset[['gameid', 'position', 'side', 'datacomplet
neutral_monsters_dataset.shape[0]
```

Out[688… 118932

2.2 Spliting the data

The dataset contains single player data - data for an individual player, and team data - data for a team. This is distinguished by the value in the `position` column. Team data has 'team' for that column, and any other value suggests that it is a single player data.

Since we wnat to predict whether a team wins or not, we would probably want to only look at team data and not single player data. But we will not delete them for now, let's confirm that single player data provides us no required information. For better visualization, lets's split the dataset into team data and single player data.

In [689…
```
single_player_df = neutral_monsters_dataset[neutral_monsters_dataset['position'] !=
single_player_df.shape[0]
```

Out[689… 99110

In [690…
```
team_df = neutral_monsters_dataset[neutral_monsters_dataset['position'] == 'team']
team_df.shape[0]
```

Out[690… 19822

2.3 Familiarizing at all columns

Let's looks at statisitcs for all the columns and figue ut what they mean

In [691…
```
def print_info(col: str):
    print(f'team data for {col}:')
    print(team_df[col].unique())

    print(f'\n\nsingle player data for {col}:')
    print(single_player_df[col].unique())
```

`datacompleteness`

In [692…
```
print_info('datacompleteness')
```

```
team data for datacompleteness:
['complete' 'partial']


single player data for datacompleteness:
['complete' 'partial']
```

Rows that are 'partial' have missing data. Same for both team data and single player data

`firstdragon`

In [693…
```
print_info('firstdragon')
```

```
team data for firstdragon:
[0. 1.]


single player data for firstdragon:
[nan]
```

Whether the team takes the first dragon or not, 1 means yes, 0 means no. This data is only present in team data.

In [694…
```
neutral_monsters_dataset['firstdragon'].dtype
```

Out[694…
```
dtype('float64')
```

`dragons` and `opp_dragons`

In [695…
```
print_info('dragons')
print()
print_info('opp_dragons')
```

```
team data for dragons:
[0. 2. 3. 4. 1. 5. 7. 6.]


single player data for dragons:
[nan]

team data for opp_dragons:
[2. 0. 3. 4. 1. 5. 7. 6.]


single player data for opp_dragons:
[nan]
```

How many dragons the team and the opponent team killed. Dragons contains all elemental drakes and the elder. This data is only present in team data.

In [696…
```
team_df.sample(5)[['dragons', 'elementaldrakes', 'elders']]
```

Out[696…

	dragons	elementaldrakes	elders
83351	3.0	3.0	0.0
21586	3.0	2.0	1.0
118259	1.0	1.0	0.0
32938	3.0	3.0	0.0
53482	0.0	0.0	0.0

In [697…
```
team_df[(team_df['elementaldrakes'] > 0) & (team_df['elders'] > 0)].sample(5)[['dra
```

Out[697…

	dragons	elementaldrakes	elders
20878	4.0	3.0	1.0
11650	2.0	1.0	1.0
80699	6.0	4.0	2.0
21298	5.0	4.0	1.0
3551	3.0	2.0	1.0

From this we can conclude that by 'dragons', the dataset means all 'elementaldrakes' plus 'elders'.

In [698…
```
neutral_monsters_dataset['dragons'].dtype, neutral_monsters_dataset['opp_dragons'].
```

Out[698…
```
(dtype('float64'), dtype('float64'))
```

`elementaldrakes` & `opp_elementaldrakes`

In [699…
```
print_info('elementaldrakes')
print()
print_info('opp_elementaldrakes')
```

```
team data for elementaldrakes:
[ 0.  2.  3.  4.  1. nan]


single player data for elementaldrakes:
[nan]

team data for opp_elementaldrakes:
[ 2.  0.  3.  4.  1. nan]


single player data for opp_elementaldrakes:
[nan]

[ 0.  2.  3.  4.  1. nan]


single player data for elementaldrakes:
[nan]

team data for opp_elementaldrakes:
[ 2.  0.  3.  4.  1. nan]


single player data for opp_elementaldrakes:
[nan]
```

In [700…
```
team_df[team_df['elementaldrakes'].isna() & (team_df['datacompleteness'] !=  'parti
```

Out[700…
```
0
```

In [701…
```
team_df[team_df['opp_elementaldrakes'].isna() & (team_df['datacompleteness'] !=  'p
```

Out[701…
```
0
```

How many elemental drakes the team and the opponent team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

infernals

In [702…
```
print_info('infernals')
```

```
team data for infernals:
[ 0.  1. nan  2.  3.  4.]


single player data for infernals:
[nan]
```

In [703…
```
team_df[team_df['infernals'].isna() & (team_df['datacompleteness'] !=  'partial')].
```

Out[703…
```
0
```

How many infernal drakes the team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

mountains

In [704…
```
print_info('mountains')
```

```
team data for mountains:
[ 0.  3. nan  1.  2.  4.]


single player data for mountains:
[nan]
```

In [705…
```
team_df[team_df['mountains'].isna() & (team_df['datacompleteness'] !=  'partial')].
```

Out[705…
```
0
```

How many mountain drakes the team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

clouds

In [706…
```
print_info('clouds')
```

```
team data for clouds:
[ 0.  1.  2. nan  3.  4.]


single player data for clouds:
[nan]
```

In [707…
```
team_df[team_df['clouds'].isna() & (team_df['datacompleteness'] !=  'partial')].sha
```

Out[707…
```
0
```

How many cloud drakes the team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

oceans

In [708…
```
print_info('oceans')
```

```
team data for oceans:
[ 0.  1. nan  2.  3.  4.]


single player data for oceans:
[nan]
```

In [709…
```
team_df[team_df['oceans'].isna() & (team_df['datacompleteness'] !=  'partial')].sha
```

Out[709…
```
0
```

How many ocean drakes the team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

chemtechs

In [710…
```
print_info('chemtechs')
```

```
team data for chemtechs:
[ 0. nan  1.  2.  3.  4.]


single player data for chemtechs:
[nan]
```

In [711…
```
team_df[team_df['chemtechs'].isna() & (team_df['datacompleteness'] !=  'partial')].
```

Out[711…
```
0
```

How many chemtech drakes the team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

hextechs

In [712…
```
print_info('hextechs')
```

```
team data for hextechs:
[ 0.  1.  2. nan  3.  4.]


single player data for hextechs:
[nan]
```

In [713…
```
team_df[team_df['hextechs'].isna() & (team_df['datacompleteness'] !=  'partial')].s
```

Out[713… 0

How many hextech drakes the team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

`dragons (type unknown)`

In [714…
```
print_info('dragons (type unknown)')
```

```
team data for dragons (type unknown):
[nan  4.  1.  2.  3.  0.]


single player data for dragons (type unknown):
[nan]
```

In [715…
```
team_df[team_df['dragons (type unknown)'].isna() & (team_df['datacompleteness'] !=
```

Out[715… 18188

In [716…
```
team_df[team_df['dragons (type unknown)'].notna() & (team_df['datacompleteness'] !=
```

Out[716… 0

This column is special. It is not nan only when `datacompleteness` is 'partial'. Hence this column represents the total number of elemental drakes the team killed, when the data is incomplete. We can also conclude that this column is probabily a main reason why `datacompleteness` is 'partial'. This data is only present in team data.

`elders` & `opp_elders`

In [717…
```
print_info('elders')
print()
print_info('opp_elders')
```

```
team data for elders:
[ 0. nan  1.  3.  2.]


single player data for elders:
[nan]

team data for opp_elders:
[ 0. nan  1.  3.  2.]


single player data for opp_elders:
[nan]
```

In [718…
```
team_df[team_df['elders'].isna() & (team_df['datacompleteness'] !=  'partial')].sha
```

Out[718…
```
0
```

In [719…
```
team_df[team_df['opp_elders'].isna() & (team_df['datacompleteness'] !=  'partial')]
```

Out[719…
```
0
```

How many elder dragons the team and the opponent team killed. This data is only present in team data. Data is nan only when the data is considered 'partial'.

`firstherald`

In [720…
```
print_info('firstherald')
```

```
team data for firstherald:
[ 0.  1. nan]


single player data for firstherald:
[nan]
```

In [721…
```
team_df[team_df['firstherald'].isna() & (team_df['datacompleteness'] !=  'partial')
```

Out[721…
```
0
```

Whether the team takes the first herald or not, 1 means yes, 0 means no. This data is only present in team data. Data is nan only when `datacompleteness` is 'partial'.

`heralds` & `opp_heralds`

In [722…
```
print_info('heralds')
print()
print_info('opp_heralds')
```

```
team data for heralds:
[0. 1.]


single player data for heralds:
[nan]

team data for opp_heralds:
[1. 0.]


single player data for opp_heralds:
[nan]
```

How many heralds the team and the opponent team killed. This data is only present in team data.

`void_grubs` & `opp_void_grubs`

In [723…
```
print_info('void_grubs')
print()
print_info('opp_void_grubs')
```

```
team data for void_grubs:
[0. 6. 2. 4. 3. 5. 1.]


single player data for void_grubs:
[nan]

team data for opp_void_grubs:
[6. 0. 4. 2. 3. 1. 5.]


single player data for opp_void_grubs:
[nan]
```

How many void_grubs the team and the opponent team killed. This data is only present in team data.

`firstbaron`

In [724…
```
print_info('firstbaron')
```

```
team data for firstbaron:
[ 0.  1. nan]


single player data for firstbaron:
[nan]
```

In [725…
```
team_df[team_df['firstbaron'].isna() & (team_df['datacompleteness'] !=  'partial')]
```

Out[725… 0

Whether the team takes the first baron or not, 1 means yes, 0 means no. This data is only present in team data.

`barons` & `opp_barons`

In [726…
```
print_info('barons')
print()
print_info('opp_barons')
```

```
team data for barons:
[0. 1. 2. 3. 4.]


single player data for barons:
[ 0.  1. nan  2.  3.  4.]

team data for opp_barons:
[1. 0. 2. 3. 4.]


single player data for opp_barons:
[ 0.  1. nan  2.  3.  4.]
```

In [727…
```
single_player_df[single_player_df['barons'].isna() & (single_player_df['datacomplet
```

Out[727… 0

In [728…
```
single_player_df[single_player_df['opp_barons'].isna() & (single_player_df['datacom
```

Out[728… 0

In [729…
```
random_game_id = np.random.choice(neutral_monsters_dataset[neutral_monsters_dataset
random_game = neutral_monsters_dataset[neutral_monsters_dataset['gameid'] == random
random_game[['gameid', 'position', 'side',  'barons', 'opp_barons']]
```

Out[729…

	gameid	position	side	barons	opp_barons
23592	LOLTMNT01_218987	top	Blue	0.0	1.0
23593	LOLTMNT01_218987	jng	Blue	0.0	0.0
23594	LOLTMNT01_218987	mid	Blue	0.0	0.0
23595	LOLTMNT01_218987	bot	Blue	0.0	0.0
23596	LOLTMNT01_218987	sup	Blue	0.0	0.0
23597	LOLTMNT01_218987	top	Red	1.0	0.0
23598	LOLTMNT01_218987	jng	Red	0.0	0.0
23599	LOLTMNT01_218987	mid	Red	0.0	0.0
23600	LOLTMNT01_218987	bot	Red	0.0	0.0
23601	LOLTMNT01_218987	sup	Red	0.0	0.0
23602	LOLTMNT01_218987	team	Blue	0.0	1.0
23603	LOLTMNT01_218987	team	Red	1.0	0.0

How many barons the team and the opponent team killed. Data is nan for single player data only when the data is considered 'partial'. Single player data only helps to identify who was the one to slay the baron.

`atakhans` & `opp_atakhans`

In [730…

```
print_info('atakhans')
print()
print_info('opp_atakhans')
```

```
team data for atakhans:
[ 0.  1. nan]


single player data for atakhans:
[nan]

team data for opp_atakhans:
[ 1.  0. nan]


single player data for opp_atakhans:
[nan]

[ 0.  1. nan]


single player data for atakhans:
[nan]

team data for opp_atakhans:
[ 1.  0. nan]


single player data for opp_atakhans:
[nan]
```

In [731…
```
team_df[team_df['atakhans'].isna()].shape[0]
```

Out[731…
```
1686
```

In [732…
```
team_df[team_df['opp_atakhans'].isna()].shape[0]
```

Out[732…
```
1686
```

In [733…
```
team_df[team_df['atakhans'].isna() & (team_df['datacompleteness'] !=  'partial')].s
```

Out[733…
```
52
```

In [734…
```
team_df[team_df['opp_atakhans'].isna() & (team_df['datacompleteness'] !=  'partial'
```

Out[734…
```
52
```

In [735…
```
team_df[(team_df['opp_atakhans'].isna()) & (team_df['datacompleteness'] !=  'partia
```

Out[735…

	gameid	position	side	datacompleteness	firstdragon	dragons	opp_dra
4894	LOLTMNT01_194596	team	Blue	complete	1.0	1.0	
7546	LOLTMNT01_203165	team	Blue	complete	0.0	0.0	
9826	LOLTMNT01_205363	team	Blue	complete	0.0	2.0	
9851	LOLTMNT01_205378	team	Red	complete	0.0	1.0	
9827	LOLTMNT01_205363	team	Red	complete	1.0	1.0	
8207	LOLTMNT04_96183	team	Red	complete	1.0	3.0	

How many atakhans the team and the opponent team killed. This data is only valid for team data. Data is nan when the data is considered 'partial'. But it is also nan when both teams do not kill the atakhan for the whole game.

`result`

In [736…
```
print_info('result')
```

```
team data for result:
[0 1]


single player data for result:
[0 1]
```

Whether the team or the player won or not.

We are done familiarizing with the dataset, and we can conclude that single player data is useless for our analysis. Hence we will only focus on team data and we can drop the `position` column, due to it being all 'team' for team data.

We can also drop the `firstherald` column, since there is only one herald to take. If a team killed a herald, that mean they must have gotten the first and only herald.

In [737…
```
final_team_df = team_df.drop(columns=['position', 'firstherald'])
final_team_df.head()
```

Out[737...

	gameid	side	datacompleteness	firstdragon	dragons	opp_dragons	eleme
10	LOLTMNT03_179647	Blue	complete	0.0	0.0	2.0	
11	LOLTMNT03_179647	Red	complete	1.0	2.0	0.0	
22	LOLTMNT06_96134	Blue	complete	0.0	3.0	2.0	
23	LOLTMNT06_96134	Red	complete	1.0	2.0	3.0	
34	LOLTMNT06_95160	Blue	complete	0.0	0.0	4.0	

Conclusion for 2.3:

- `gameid`, `side` are used to identify whether the teams are in the same game and what side they are on.
- `result` is whether the team won or not, and is what we want to predict in the end of the day.
- `firstdragon`, `firstherald`, `firstbaron` are identifiers for whether the team obtained the first corresponding large neutral monster. 1 means yes and 0 means no. We will keep them as binarys since we can use them for the later prediction tasks.
- `dragons` contains the total number of `elementaldrakes` + `elders` that the team has killed.
- `elementaldrakes` is the total number of elemental drakes killed by the team, which should be the sum of `infernals`, `mountains`, `clouds`, `oceans`, `chemtechs`, and `hextechs` of the same row. **The maximum number for these columns is 4, this is because killing more than 4 elemental drakes will not provide your team new buffs after 4. Only the first 4 kills per team are recorded**.
- `elementaldrakes`, `infernals`, `mountains`, `clouds`, `oceans`, `chemtechs`, `hextechs`, `elders`, `firstherald`, `firstbaron`, `atakhans`, and **another column** are the **ONLY** columns affected by `datacompleteness`. If `datacompleteness` = 'partial', they will be `np.nan`. If `datacompleteness` = 'complete', they will be the total number of corresponding large neutral monsters the team has killed, with the exception of `atakhans`. `atakhans` will also be `np.nan` if both teams in the game failed to kill the Atakhan.
- `dragons (type unknown)` is the other column that is also affected by `datacompleteness`, but in the opposite way. It will be an integer between 0 and 4, inclusive, if and only if `datacompleteness` is 'partial'.

- `heralds`, `void_grubs`, `barons` are just the all total numbers of the corresponding large neutral monster that the team has killed.
- `opp_dragons`, `opp_elementaldrakes`, `opp_elders`, `opp_heralds`, `opp_void_grubs`, `opp_barons`, `opp_atakhans` are data for the opposite team. They have the same traits as their non-opp version.

In [738…
```
print(final_team_df.head().to_markdown(index=False))
```

gameid	side	datacompleteness	firstdragon	dragons	opp_dragons	elementaldrakes	opp_elementaldrakes	infernals	mountains	clouds	oceans	chemtechs	hextechs	dragons (type unknown)	elders	opp_elders	heralds	opp_heralds	void_grubs	opp_void_grubs	firstbaron	barons	opp_barons	atakhans	opp_atakhans	result
LOLTMNT03_179647	Blue	complete	0	0	2	0	0	0	0	0	0	0	nan	0	0	0	0	0	6	0	0	1	0	1	0	1
LOLTMNT03_179647	Red	complete	1	2	0	1	0	2	0	0	0	0	nan	0	0	1	0	1	0	6	0	1	1	0	1	0
LOLTMNT06_96134	Blue	complete	0	3	2	0	3	0	0	0	0	0	nan	0	0	1	0	1	0	6	0	1	1	0	1	0
LOLTMNT06_96134	Red	complete	1	2	3	0	0	0	1	0	0	0	nan	0	0	0	1	0	6	0	0	1	0	1	0	1
LOLTMNT06_95160	Blue	complete	0	0	4	0	0	0	0	0	0	0	nan	0	0	0	1	0	2	4	0	0	1	0	1	0

2.4 Univariate Analysis

Graph some important features selected.

Let's take a look at the distribution for some of column by using histograms.

In [739…
```
fig = px.histogram(
    final_team_df,
```

```
    x="dragons",
    histnorm="percent",
    title="Percent of Games for Sum of Total Elder Dragon and Total Elemental Drake
)
fig.update_traces(marker_line_color="white", marker_line_width=2)
fig.show()
```

In [740…
```
fig.write_html('assets/dragons.html', include_plotlyjs='cdn')
```

In [741…
```
fig = px.histogram(
    final_team_df,
    x="elementaldrakes",
    histnorm="percent",
    title="Percent of Games for Total Elemental Drakes Taken by a Single Team"
)
fig.update_traces(marker_line_color="white", marker_line_width=2)
fig.show()
```

In [742…
```
fig.write_html('assets/elementaldrakes.html', include_plotlyjs='cdn')
```

In [743…
```
fig = px.histogram(
    final_team_df,
    x="elders",
    histnorm="percent",
    title="Percent of Games for Total Elder Drakes Taken by a Single Team"
)
fig.update_traces(marker_line_color="white", marker_line_width=2)
fig.show()
```

In [744…
```
fig.write_html('assets/elders.html', include_plotlyjs='cdn')
```

In [745…
```
fig = px.histogram(
    final_team_df,
    x="heralds",
    histnorm="percent",
    title="Percent of Games for Total Heralds Taken by a Single Team"
)
fig.update_traces(marker_line_color="white", marker_line_width=2)
fig.show()
```

In [746…
```
fig = px.histogram(
    final_team_df,
    x="void_grubs",
    histnorm="percent",
    title="Percent of Games for Total Void Grubs Taken by a Single Team"
)
fig.update_traces(marker_line_color="white", marker_line_width=2)
fig.show()
```

In [747…
```
fig = px.histogram(
    final_team_df,
    x="barons",
```

```
    histnorm="percent",
    title="Percent of Games for Total Barons Taken by a Single Team"
)
fig.update_traces(marker_line_color="white", marker_line_width=2)
fig.show()
```

In [748…

```
fig = px.histogram(
    final_team_df,
    x="atakhans",
    histnorm="percent",
    title="Percent of Games for Total Atakhans Taken by a Single Team"
)
fig.update_traces(marker_line_color="white", marker_line_width=2)
fig.show()
```

2.5 Bivariate Analysis

`firstdragon`, `heralds` (which has same information as `firstherald`), and `firstbaron`

Which one of these, or what combination of these will give higher win rates?

In [749…

```
firstheral_firstdragon_win_rate = final_team_df.pivot_table(
    index="firstdragon",
    columns="heralds",
    values="result",
    aggfunc="mean"
)
firstheral_firstdragon_win_rate
```

Out[749…

heralds	0.0	1.0
firstdragon		
0.0	0.267442	0.589389
1.0	0.411898	0.732858

In [750…

```
firstheral_firstbaron_win_rate = final_team_df.pivot_table(
    index="firstbaron",
    columns="heralds",
    values="result",
    aggfunc="mean"
)
firstheral_firstbaron_win_rate
```

Out[750…

heralds	0.0	1.0
firstbaron		
0.0	0.123632	0.429590
1.0	0.786970	0.897101

In [751…

```
firstdragon_firstbaron_win_rate = final_team_df.pivot_table(
    index="firstbaron",
    columns="firstdragon",
    values="result",
    aggfunc="mean"
)
firstdragon_firstbaron_win_rate
```

Out[751…

firstdragon	0.0	1.0
firstbaron		
0.0	0.190768	0.322224
1.0	0.819580	0.883126

In [752…

```
pivot_first_kills = final_team_df.pivot_table(
    index=["firstdragon", "heralds"],
    columns="firstbaron",
    values="result",
    aggfunc="mean"
)
pivot_first_kills
```

Out[752…

firstbaron		0.0	1.0
firstdragon	**heralds**		
0.0	**0.0**	0.090270	0.736926
	1.0	0.339402	0.868806
1.0	**0.0**	0.164286	0.825503
	1.0	0.524989	0.921394

In [753…

```
print(pivot_first_kills.to_markdown())
```

```
|            |       0.0 |      1.0 |
|:-----------|----------:|---------:|
| (0.0, 0.0) | 0.0902696 | 0.736926 |
| (0.0, 1.0) | 0.339402  | 0.868806 |
| (1.0, 0.0) | 0.164286  | 0.825503 |
| (1.0, 1.0) | 0.524989  | 0.921394 |
```

These pivot tables clearly shows that getting the first Baron is a lot more essential than getting the first kills for the other two.

Here is a ranking of how important a large neutral monster is to win the game, from my understanding of the game:

1. The Elder Drake and the Baron
2. Elemental Drakes: Infernal, Ocean, and Hextech
3. Elemental Drakes: Mountain, Chemtech, and Cloud
4. Atakhan
5. Herald
6. Void Grubs

Lets check if my understanding of the game is valid.

First, simplify our dataset. we will create new columns for dragons, elementaldrakes, elders, heralds, void grubs, barons, and atakhans, so that the new columns will contain -1, 0, or 1 for killing less of the corresponding monsters than the opponent, killing the same amount, and killing more.

In [754…
```
diff_team_df = final_team_df.copy()
diff_team_df['diff_dragons'] = final_team_df['dragons'] - final_team_df['opp_dragon
diff_team_df['diff_elementaldrakes'] = final_team_df['elementaldrakes'] - final_tea
diff_team_df['diff_elders'] = final_team_df['elders'] - final_team_df['opp_elders']
diff_team_df['diff_heralds'] = final_team_df['heralds'] - final_team_df['opp_herald
diff_team_df['diff_void_grubs'] = final_team_df['void_grubs'] - final_team_df['opp_
diff_team_df['diff_barons'] = final_team_df['barons'] - final_team_df['opp_barons']
diff_team_df['diff_atakhans'] = final_team_df['atakhans'] - final_team_df['opp_atak

def assign_he_zero_pos(num):
    if num > 0:
        return 'killed more'
    elif num == 0:
        return 'killed the same'
    else:
        return 'killed less'

diff_team_df['diff_dragons'] = diff_team_df['diff_dragons'].apply(assign_he_zero_po
diff_team_df['diff_elementaldrakes'] = diff_team_df['diff_elementaldrakes'].apply(a
diff_team_df['diff_elders'] = diff_team_df['diff_elders'].apply(assign_he_zero_pos)
diff_team_df['diff_heralds'] = diff_team_df['diff_heralds'].apply(assign_he_zero_po
diff_team_df['diff_void_grubs'] = diff_team_df['diff_void_grubs'].apply(assign_he_z
diff_team_df['diff_barons'] = diff_team_df['diff_barons'].apply(assign_he_zero_pos)
diff_team_df['diff_atakhans'] = diff_team_df['diff_atakhans'].apply(assign_he_zero_
```

In [822…
```
dragons = diff_team_df.groupby('diff_dragons')[['result']].sum().rename(columns={'r

elementaldrakes = diff_team_df.groupby('diff_elementaldrakes')[['result']].sum().re

elders = diff_team_df.groupby('diff_elders')[['result']].sum().rename(columns={'res

heralds = diff_team_df.groupby('diff_heralds')[['result']].sum().rename(columns={'r

void_grubs = diff_team_df.groupby('diff_void_grubs')[['result']].sum().rename(colum
```

```
atakhans = diff_team_df.groupby('diff_atakhans')[['result']].sum().rename(columns={

barons = diff_team_df.groupby('diff_barons')[['result']].sum().rename(columns={'res

monster_kill_wins = pd.concat([dragons, elementaldrakes, elders, heralds, void_grub
monster_kill_wins
```

Out[822…

	dragons win count	elementaldrakes win count	elders win count	heralds win count	void_grubs win count	atakhans win count	barons win count
killed less	1439	2161	906	3337	3556	2646	722
killed more	7022	6436	598	6553	5320	7002	6884
killed the same	1450	1314	8407	21	1035	263	2305

In [823…

```
dragons = diff_team_df.groupby('diff_dragons')[['result']].sum()
dragons['result'] = dragons['result'] / dragons['result'].sum()
dragons = dragons.rename(columns={'result':'dragons win rate'})

elementaldrakes = diff_team_df.groupby('diff_elementaldrakes')[['result']].sum()
elementaldrakes['result'] = elementaldrakes['result'] / elementaldrakes['result'].s
elementaldrakes = elementaldrakes.rename(columns={'result':'elementaldrakes win rat

elders = diff_team_df.groupby('diff_elders')[['result']].sum()
elders['result'] = elders['result'] / elders['result'].sum()
elders = elders.rename(columns={'result':'elders win rate'})

heralds = diff_team_df.groupby('diff_heralds')[['result']].sum()
heralds['result'] = heralds['result'] / heralds['result'].sum()
heralds = heralds.rename(columns={'result':'heralds win rate'})

void_grubs = diff_team_df.groupby('diff_void_grubs')[['result']].sum()
void_grubs['result'] = void_grubs['result'] / void_grubs['result'].sum()
void_grubs = void_grubs.rename(columns={'result':'void_grubs win rate'})

atakhans = diff_team_df.groupby('diff_atakhans')[['result']].sum()
atakhans['result'] = atakhans['result'] / atakhans['result'].sum()
atakhans = atakhans.rename(columns={'result':'atakhans win rate'})

barons = diff_team_df.groupby('diff_barons')[['result']].sum()
barons['result'] = barons['result'] / barons['result'].sum()
barons = barons.rename(columns={'result':'barons win rate'})

monster_kill_ratio = pd.concat([dragons, elementaldrakes, elders, heralds, void_gru
monster_kill_ratio
```

Out[823…

	dragons win rate	elementaldrakes win rate	elders win rate	heralds win rate	void_grubs win rate	atakhans win rate	barons win rate
killed less	0.145192	0.218041	0.091414	0.336697	0.358793	0.266976	0.072848
killed more	0.708506	0.649379	0.060337	0.661185	0.536777	0.706488	0.694582
killed the same	0.146302	0.132580	0.848249	0.002119	0.104429	0.026536	0.232570

In [826…

```
print(monster_kill_ratio.to_markdown())
```

```
|                |   dragons win rate |   elementaldrakes win rate |   elders win r
ate |   heralds win rate |   void_grubs win rate |   atakhans win rate |   barons wi
n rate |
|:---------------|-------------------:|---------------------------:|---------------
---:|-------------------:|----------------------:|--------------------:|------------
------:|
| killed less    |           0.145192 |                   0.218041 |         0.0914
136 |         0.336697   |              0.358793 |           0.266976  |         0.0
728484 |
| killed more    |           0.708506 |                   0.649379 |         0.0603
37  |         0.661185   |              0.536777 |           0.706488  |         0.6
94582  |
| killed the same |          0.146302 |                   0.13258  |         0.8482
49  |         0.00211886 |              0.104429 |           0.0265362 |         0.2
3257   |
```

In [824…

```
df_plot = monster_kill_wins.reset_index().rename(columns={'index': 'Kill Status'})
df_long = pd.melt(
    df_plot,
    id_vars='Kill Status',
    var_name='Monster Type',
    value_name='Win Count'
)

fig = px.bar(
    df_long,
    x='Monster Type',
    y='Win Count',
    color='Kill Status',
    barmode='group',
    title='Win Counts by Monster Kill Status',
    labels={
        'Monster Type': 'Monster Type',
        'Win Count': 'Total Win Count'
    }
)

fig.update_layout(
    xaxis_title_text='Monster Type',
    yaxis_title_text='Total Win Count',
    title_font_size=20,
```

```
    legend_title_text='Kill Status'
)
```

In [825…
```
fig.write_html('assets/monsters.html', include_plotlyjs='cdn')
```

We can imediately tell something is wrong with my assumption because the win count for elder dragon kill status tells us most games are won without any team killing more elder dragons. In fact, more games are lost when the team kills more elder dragons. But this can be explained due to the game mechanic of how the first elder dragon is spawned. The elder dragon will start spawning only when 4 elemental drakes are killed. Which makes it harder than any other large neutral monsters because all others spawn according to the match time. However, this does not explain why killing more elder dragons tend to lose the game.

Another reason why this elder drake data is different from my assumption may be because a lot of the jungle players in this dataset are unskilled. Let's make a scatter plot of the gained_gold at the end of the match of each lane compared to the total gained_gold of the whole team. We will be needing the original full dataset for this. We will randomly sample 1000 games because there are 19822 games in this dataset, the graph may look very messy if we graph all games.

In [759…
```
random_game_ids = np.random.choice(full_dataset['gameid'], 1000)

player_df_renamed = full_dataset[(full_dataset['position'] != 'team') & (full_datas
    columns={'earned gpm': 'Player GPM'}
)

team_df_renamed = full_dataset[(full_dataset['position'] == 'team') & (full_dataset
    columns={'earned gpm': 'Team GPM'}
)

# Merge the two DataFrames on the shared 'gameid' column.
merged_df = pd.merge(
    player_df_renamed,
    team_df_renamed[['gameid', 'Team GPM']],
    on='gameid',
    how='inner'
)

fig = px.scatter(
    merged_df,
    x='Player GPM',
    y='Team GPM',
    color='position',
    hover_data=['gameid'],
    title='Player GPM vs. Team GPM, Grouped by Position',
    labels={
        'Team GPM': 'Team Earned Gold Per Minute',
        'Player GPM': 'Player Earned Gold Per Minute'
    },
    height=800,
    width=1000
```

```
)

fig.update_layout(
    title_font_size=20,
    legend_title_text='Player Position'
)

fig.update_traces(marker=dict(size=5, opacity=0.5))
```

In [760…
```
fig.write_html('assets/jungles.html', include_plotlyjs='cdn')
```

This graph proves that the jungle players in this dataset is fine and not below average. It is a little bit hard to tell but jungle players are bunched up in the mid left of the graph, which is normal because usually the econ goes more to the mid, top, and bot.

Moving on to other columns. Since there are up to 4 Elemental Drakes, 6 Void Grubs, multiply Barons, and multiple Elder Dragon to kill for each game, we would like to see the graph for the win and lose count for each number of kills.

In [761…
```
drake_outcome_counts = final_team_df.groupby(['elementaldrakes', 'result']).size().
drake_outcome_counts['Game Outcome'] = drake_outcome_counts['result'].apply(
    lambda x: 'Win' if x == 1 else 'Loss'
)

fig = px.bar(
    drake_outcome_counts,
    x='elementaldrakes',
    y='Total Games',
    color='Game Outcome',
    barmode='group',
    title='Game Count Distribution by Elemental Drakes Kill Count (Win vs Loss)',
)

fig.update_layout(
    xaxis_title_text='Elemental Drakes Kill Count',
    yaxis_title_text='Total Games',
    title_font_size=20
)
```

In [762…
```
void_grubs_outcome_counts = final_team_df.groupby(['void_grubs', 'result']).size().
void_grubs_outcome_counts['Game Outcome'] = void_grubs_outcome_counts['result'].app
    lambda x: 'Win' if x == 1 else 'Loss'
)

fig = px.bar(
    void_grubs_outcome_counts,
    x='void_grubs',
    y='Total Games',
    color='Game Outcome',
    barmode='group',
    title='Game Count Distribution by Void Grubs Kill Count (Win vs Loss)',
)
```

```
fig.update_layout(
    xaxis_title_text='Void Grubs Kill Count',
    yaxis_title_text='Total Games',
    title_font_size=20
)
```

In [763…
```
fig.write_html('assets/void_grubs.html', include_plotlyjs='cdn')
```

In [764…
```
barons_outcome_counts = final_team_df.groupby(['barons', 'result']).size().reset_in
barons_outcome_counts['Game Outcome'] = barons_outcome_counts['result'].apply(
    lambda x: 'Win' if x == 1 else 'Loss'
)

fig = px.bar(
    barons_outcome_counts,
    x='barons',
    y='Total Games',
    color='Game Outcome',
    barmode='group',
    title='Game Count Distribution by Barons Kill Count (Win vs Loss)',
)

fig.update_layout(
    xaxis_title_text='Barons Kill Count',
    yaxis_title_text='Total Games',
    title_font_size=20
)
```

In [765…
```
elders_outcome_counts = final_team_df.groupby(['elders', 'result']).size().reset_in
elders_outcome_counts['Game Outcome'] = elders_outcome_counts['result'].apply(
    lambda x: 'Win' if x == 1 else 'Loss'
)

fig = px.bar(
    elders_outcome_counts,
    x='elders',
    y='Total Games',
    color='Game Outcome',
    barmode='group',
    title='Game Count Distribution by Elder Dragon Kill Count (Win vs Loss)',
)

fig.update_layout(
    xaxis_title_text='Elder Dragon Kill Count',
    yaxis_title_text='Total Games',
    title_font_size=20
)
```

2.6 Missing Values

All columns have been checked above and they all have expected values apart from `Nan`.

Step 3: Assessment of Missingness

Permutation Tests

We claimed above that `Nan` of all the columns have a very strong relationship to the `'datacompleteness'` column. Now let's conduct some purmutation tests to see if this is true.

In [766…
```
final_team_df['datacompleteness'].unique()
```

Out[766…
```
array(['complete', 'partial'], dtype=object)
```

Let's set `'datacompleteness'` to binary

In [767…
```
final_team_df['datacompleteness'] = final_team_df['datacompleteness'] == 'complete'
```

There are columns with `Nan` values:

In [768…
```
final_team_df.columns[final_team_df.isna().any()]
```

Out[768…
```
Index(['elementaldrakes', 'opp_elementaldrakes', 'infernals', 'mountains',
       'clouds', 'oceans', 'chemtechs', 'hextechs', 'dragons (type unknown)',
       'elders', 'opp_elders', 'firstbaron', 'atakhans', 'opp_atakhans'],
      dtype='object')
```

In [769…
```
final_team_df['elementaldrakes_is_missing'] = final_team_df['elementaldrakes'].isna
```

In [770…
```
fig = px.histogram(
    final_team_df,
    x='datacompleteness',
    facet_col='elementaldrakes_is_missing',
    histnorm='percent',
    category_orders={"datacompleteness": [True, False]},
    title='Data Completeness Distribution by Elementaldrakes Missingness',
    labels={
        'datacompleteness': 'Data Complete (Y)',
        'elementaldrakes_is_missing': 'Elementaldrakes Missing',
        'count': 'Percentage of Games'
    },
    height=450
)

fig.for_each_annotation(lambda a: a.update(text=a.text.replace("elementaldrakes_is_

fig.update_xaxes(
    tickvals=[True, False],
    ticktext=['Complete', 'Partial'],
    showgrid=False,
    title_text='Data Completeness Status'
)
```

```
fig.update_yaxes(
    title_text='Percentage of Games (%)',
)

fig.update_layout(
    font=dict(family='Inter', size=12, color='#333'),
)
```

In [771…
```
fig.write_html('assets/datacompletenes_elementaldrakes_missingness.html', include_p
```

In [772…
```
def run_permutation_test(df, missing_col, test_col, n_permutations=5000):
    info_df = df[[test_col]].copy()
    info_df['isnan'] = df[missing_col].isna()

    trials = []

    observed_ts = np.abs(info_df.loc[info_df['isnan'], test_col].mean() - info_df.l

    for trial in range(n_permutations):
        shuffled = info_df.copy()
        shuffled[test_col] = np.random.permutation(shuffled[test_col])
        ts = np.abs(shuffled.loc[shuffled['isnan'], test_col].mean() - shuffled.loc
        trials.append(ts)

    p_value = (trials >= observed_ts).mean()

    return p_value, observed_ts
```

In [773…
```
for missing_col_name in final_team_df.columns[final_team_df.isna().any()].tolist():
    permutation_results_string = f"Permutation Test Results ('datacompleteness' vs.
    p_val, obs_diff = run_permutation_test(final_team_df, missing_col_name, 'dataco

    sig_message = "NOT statistically significant"
    if p_val < 0.05:
        sig_message = "STATISTICALLY significant"

    permutation_results_string += (
        f"  Observed Mean Diff (Missing Group - Observed Group): {obs_diff:.4f}\n"
        f"  P-value: {p_val:.4f} (Alpha=0.05)\n"
        f"  Conclusion: The relationship is {sig_message}\n"
    )
    print(permutation_results_string)
```

```
Permutation Test Results ('datacompleteness' vs. elementaldrakes):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. opp_elementaldrakes):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. infernals):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. mountains):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. clouds):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. oceans):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. chemtechs):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. hextechs):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. dragons (type unknown)):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. elders):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. opp_elders):
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. firstbaron):
```

```
  Observed Mean Diff (Missing Group - Observed Group): 1.0000
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. atakhans):
  Observed Mean Diff (Missing Group - Observed Group): 0.9692
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant

Permutation Test Results ('datacompleteness' vs. opp_atakhans):
  Observed Mean Diff (Missing Group - Observed Group): 0.9692
  P-value: 0.0000 (Alpha=0.05)
  Conclusion: The relationship is STATISTICALLY significant
```

These statistics proves that all columns with `Nan` values are related to `'datacompleteness'`. If `'datacompleteness'` is not 'complete', then these column values will be `Nan`. Therefore, all missingness are MAR, and is related to `'datacompleteness'` values.

Let's also test to see if this missingness is related to winning the game or not:

In [774…

```python
for missing_col_name in final_team_df.columns[final_team_df.isna().any()].tolist():
    permutation_results_string = f"Permutation Test Results ('result' vs. {missing_
    p_val, obs_diff = run_permutation_test(final_team_df, missing_col_name, 'result

    sig_message = "NOT statistically significant"
    if p_val < 0.05:
        sig_message = "STATISTICALLY significant"

    permutation_results_string += (
        f"  Observed Mean Diff (Missing Group - Observed Group): {obs_diff:.4f}\n"
        f"  P-value: {p_val:.4f} (Alpha=0.05)\n"
        f"  Conclusion: The relationship is {sig_message}\n"
    )
    print(permutation_results_string)
```

```
Permutation Test Results ('result' vs. elementaldrakes):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. opp_elementaldrakes):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. infernals):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. mountains):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. clouds):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. oceans):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. chemtechs):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. hextechs):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. dragons (type unknown)):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. elders):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. opp_elders):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. firstbaron):
```

```
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. atakhans):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant

Permutation Test Results ('result' vs. opp_atakhans):
  Observed Mean Diff (Missing Group - Observed Group): 0.0000
  P-value: 1.0000 (Alpha=0.05)
  Conclusion: The relationship is NOT statistically significant
```

We are safe to say that it is very likely that the missingness is not related to the `'result'` column.

Filling in `Nan` Values

We do know that our missingness depends completely on the column `datacompleteness`, but that tells us little for how to fill in the missing values. `elementaldrakes`, `infernals`, `mountains`, `clouds`, `oceans`, `chemtechs`, and `hextechs` have missing values related to the values in `dragons (type unknown)`, but `elders`, `firstbaron`, `atakhans` are not related to `dragons (type unknown)`.

1. First, we will deal with the easy one: `atakhans` and `opp_atakhans`. Fill in some of the Nan for `atakhans` and `opp_atakhans` with 0s, the analysis above proves that this column has Nan values when both teams do not kill the atakhans. And fill in values for `opp_atakhans`, which should have 0 if `atakhans` is 1 and vice verse.

In [775…
```
condition = final_team_df['atakhans'].isna() & final_team_df['opp_atakhans'].isna()
columns_to_impute = ['atakhans', 'opp_atakhans']
final_team_df.loc[condition, columns_to_impute] = 0
```

In [776…
```
final_team_df[final_team_df['atakhans'].isna() & (final_team_df['datacompleteness']
```

Out[776…
```
0
```

In [777…
```
final_team_df[final_team_df['opp_atakhans'].isna() & (final_team_df['datacompletene
```

Out[777…
```
0
```

2. Next, we will look at all the elemental dragons. These columns all have missing values, but they will be extremely hard to fill in due to the fact that there is an additional rule for having only a maximum of 4 elemental dragons killed per team. This means that if we want to fill in the missing values for any of these elemental dragons with statistics like the mean or the median or even random numbers extracted from a distribution, we

may find games having more than 4 elemental dragons killed for a single team, making this invalid. Hence, I have decided to drop all the types of elemental dragons and just keep `elementaldrakes` and `opp_elementaldrakes`. We will also be needing to fill in the missing values for these 2 columns because we know the missing data from `dragons (type unknown)`.

In [778…
```
final_team_df = final_team_df.drop(columns=['infernals', 'mountains', 'clouds', 'oc
```

In [779…
```
final_team_df.loc[~final_team_df['datacompleteness'], 'elementaldrakes'] = final_te
```

In [780…
```
final_team_df[final_team_df['elementaldrakes'].isna()].shape[0]
```

Out[780… `0`

In [781…
```
final_team_df['opp_elementaldrakes'] = final_team_df['elementaldrakes']
final_team_df['opp_elementaldrakes'] = final_team_df.groupby('gameid')['opp_element
final_team_df[final_team_df['opp_elementaldrakes'].isna()].shape[0]
```

```
C:\Users\tangz\AppData\Local\Temp\ipykernel_6608\417669497.py:2: FutureWarning:

The provided callable <built-in function sum> is currently using SeriesGroupBy.sum.
In a future version of pandas, the provided callable will be used directly. To keep
current behavior pass the string "sum" instead.
```

Out[781… `0`

We can now drop `'dragons (type unknown)'` since we have no use of it anymore.

In [782…
```
final_team_df = final_team_df.drop(columns=['dragons (type unknown)'])
```

In [783…
```
final_team_df.columns[final_team_df.isna().any()]
```

Out[783…
```
Index(['elders', 'opp_elders', 'firstbaron', 'atakhans', 'opp_atakhans'], dtype='o
bject')
```

The remaining columns with missing values are `'elders'`, `'opp_elders'`, `'atakhans'`, `'opp_atakhans'` and `'firstbaron'`. But since these missingness are related to `'datacompleteness'`, yet all are `Nan` when `'datacompleteness'` is 'partial', we are not able to fill in the values based on that column. Instead, we will look at whether the these 3 columns' true values have anything to do with the `'result'` column.

In [784…
```
nanless_df = final_team_df[final_team_df['datacompleteness']]
proportion_pivot_table = pd.crosstab(
    nanless_df['atakhans'],
    nanless_df['result'],
    normalize='columns',
    rownames=['Atakhan Taken'],
    colnames=['Game Result (0=Loss, 1=Win)']
)
proportion_pivot_table
```

Out[784...

Game Result (0=Loss, 1=Win)	0	1
Atakhan Taken		
0.0	0.801737	0.230042
1.0	0.198263	0.769958

In [785...

```
nanless_df = final_team_df[final_team_df['datacompleteness']]
proportion_pivot_table = pd.crosstab(
    nanless_df['elders'],
    nanless_df['result'],
    normalize='columns',
    rownames=['Elder Dragons Taken'],
    colnames=['Game Result (0=Loss, 1=Win)']
)
proportion_pivot_table
```

Out[785...

Game Result (0=Loss, 1=Win)	0	1
Elder Dragons Taken		
0.0	0.984605	0.928964
1.0	0.014625	0.064988
2.0	0.000770	0.005388
3.0	0.000000	0.000660

In [786...

```
nanless_df = final_team_df[final_team_df['datacompleteness']]
proportion_pivot_table = pd.crosstab(
    nanless_df['firstbaron'],
    nanless_df['result'],
    normalize='columns',
    rownames=['First Baron Taken'],
    colnames=['Game Result (0=Loss, 1=Win)']
)
proportion_pivot_table
```

Out[786...

Game Result (0=Loss, 1=Win)	0	1
First Baron Taken		
0.0	0.880141	0.296459
1.0	0.119859	0.703541

We will use these distributions to fill in the missing values.

In [787...

```
def impute_group_by_distribution(group_series):
    observed = group_series.dropna()
    nan_indices = group_series[group_series.isna()].index
```

```
    imputed_values = np.random.choice(
        observed.values.astype(int),
        size=len(nan_indices),
        replace=True
    )
    return pd.Series(imputed_values, index=nan_indices)

imputed_elders_series = final_team_df.groupby('result')['elders'].apply(impute_grou
imputed_elders_series.index = imputed_elders_series.index.droplevel(0)

imputed_atakhan_series = final_team_df.groupby('result')['atakhans'].apply(impute_g
imputed_atakhan_series.index = imputed_atakhan_series.index.droplevel(0)

imputed_firstbaron_series = final_team_df.groupby('result')['firstbaron'].apply(imp
imputed_firstbaron_series.index = imputed_firstbaron_series.index.droplevel(0)

final_team_df.loc[imputed_elders_series.index, 'elders'] = imputed_elders_series.va
final_team_df.loc[imputed_firstbaron_series.index, 'firstbaron'] = imputed_firstbar
final_team_df.loc[imputed_atakhan_series.index, 'atakhans'] = imputed_atakhan_serie
```

In [788…

```
final_team_df['opp_elders'] = final_team_df['elders']
final_team_df['opp_elders'] = final_team_df.groupby('gameid')['elders'].transform(s
final_team_df[final_team_df['opp_elders'].isna()].shape[0]
```

```
C:\Users\tangz\AppData\Local\Temp\ipykernel_6608\930611178.py:2: FutureWarning:

The provided callable <built-in function sum> is currently using SeriesGroupBy.sum.
In a future version of pandas, the provided callable will be used directly. To keep
current behavior pass the string "sum" instead.
```

Out[788…

```
0
```

In [789…

```
final_team_df['opp_atakhans'] = final_team_df['atakhans']
final_team_df['opp_atakhans'] = final_team_df.groupby('gameid')['atakhans'].transfo
final_team_df[final_team_df['opp_atakhans'].isna()].shape[0]
```

```
C:\Users\tangz\AppData\Local\Temp\ipykernel_6608\3737007830.py:2: FutureWarning:

The provided callable <built-in function sum> is currently using SeriesGroupBy.sum.
In a future version of pandas, the provided callable will be used directly. To keep
current behavior pass the string "sum" instead.
```

Out[789…

```
0
```

In [790…

```
final_team_df.columns[final_team_df.isna().any()]
```

Out[790…

```
Index([], dtype='object')
```

In [791…

```
proportion_pivot_table = pd.crosstab(
    final_team_df['elders'],
    final_team_df['result'],
    normalize='columns',
    rownames=['Elder Dragons Taken'],
    colnames=['Game Result (0=Loss, 1=Win)']
```

```
)
proportion_pivot_table
```

Out[791…

Game Result (0=Loss, 1=Win)	0	1
Elder Dragons Taken		
0.0	0.984462	0.928362
1.0	0.014832	0.065382
2.0	0.000706	0.005348
3.0	0.000000	0.000908

In [792…

```
proportion_pivot_table = pd.crosstab(
    final_team_df['firstbaron'],
    final_team_df['result'],
    normalize='columns',
    rownames=['First Baron Taken'],
    colnames=['Game Result (0=Loss, 1=Win)']
)
proportion_pivot_table
```

Out[792…

Game Result (0=Loss, 1=Win)	0	1
First Baron Taken		
0.0	0.881142	0.295228
1.0	0.118858	0.704772

All missing values are all filled in and the distribution are still similar.

Step 4: Hypothesis Testing

4.1 Hypothesis Test I for `'elders'`

When analyzing the dataset, we saw that barely any team killed any elder dragon. As I have pointed out in previous steps, elder dragon kills are much rarer than any other large neutral monster kills due to it having very strict spawn rules. According to my asumption back in step 2.5, my ranking for the importance of killing a elder dragon is very high. ChatGPT agrees with me and says that any elder dragon kills for a team will increase their win rate up to 70%. But I think the win rate is even higher. Let's conduct a hypothesis test to reject ChatGPT.

- `Null Hypothesis`: Given that any elder dragons were killed by a team, the team's win rate is 70%

- `Alternative Hypothesis`: Given that any elder dragons were killed by a team, the team's win rate is more than 70%
- `Statistic`: Win rate given more than one elder dragon was killed by the team

In [793…
```
elder_kills_df = final_team_df[final_team_df['elders'] > 0]
observed_ts = elder_kills_df['result'].mean()
trial_size = elder_kills_df.shape[0]
p = 0.7
N = 5000

sims = np.random.binomial(trial_size, p, size=N) / trial_size

p_val = np.mean(sims >= observed_ts)
print(f'The win rate for at least 1 elder dragon kill is {observed_ts}')
if p_val < 0.05:
    print(f'We are safe to Reject the Null, the p-value is {p_val}. ChatGPT is wron
else:
    print(f'We failed to reject the Null, the p-value is {p_val}. ChatGPT may be ri
```

```
The win rate for at least 1 elder dragon kill is 0.8217592592592593
We are safe to Reject the Null, the p-value is 0.0. ChatGPT is wrong
```

This proves my assumption that the win rate is larger than 70%

4.2 Hypothesis Test II for `'elders'`

Now let's see how likely is it for any team to kill the elder dragon. As I have said before, it is rare for a game to have either team killing the elder dragon. ChatGPT says there are only around 10% of the games will have any elder dragon kills. Let's conduct a hypothesis test to prove this.

- `Null Hypothesis`: Around 10% of all games will have at least 1 elder dragon kills from either team
- `Alternative Hypothesis`: Games that have at least 1 elder dragon kills from either team is not around 10%
- `Statistic`: abs((percentage of games where at least 1 elder dragon was killed out of all games) - 0.1)

In [794…
```
any_elder_kills_df = final_team_df[(final_team_df['elders'] > 0) | (final_team_df['
p = 0.1
observed_ts = abs(any_elder_kills_df.shape[0] / final_team_df.shape[0] - p)
trial_size = final_team_df.shape[0]
N = 5000

sims = abs(np.random.binomial(trial_size, p, size=N) / trial_size - p)

p_val = np.mean(sims >= observed_ts)
print(f'The chance of killing any elder dragon per game is {any_elder_kills_df.shap
```

```
if p_val < 0.05:
    print(f'We are safe to Reject the Null, the p-value is {p_val}. ChatGPT is wron
else:
    print(f'We failed to reject the Null, the p-value is {p_val}. ChatGPT may be ri
```

```
The chance of killing any elder dragon per game is 0.0816264756331349
We are safe to Reject the Null, the p-value is 0.0. ChatGPT is wrong
```

This proves that this dataset shows a different chance for either team to kill any elder dragons than 0.1

4.3 Hypothesis Test for `'side'`

A question that has bugged me for some time is whether the side (Blue or Red) influences the win rate. This is supposed to be a 5v5 fair game, but it would be interesting if being on one side would increase the chance of winning. Let's do a hypothesis test to check.

- `Null Hypothesis`: Being on the Blue side has 50% chance of winning.
- `Alternative Hypothesis`: Being on the Blue side does not have a 50% chance of winning.
- `Statistic`: abs((percentage of games where Blue won) - 0.5)

In [815…

```
blue_team_df = final_team_df[final_team_df['side'] == 'Blue']
p = 0.5
observed_ts = abs(blue_team_df['result'].mean() - p)
trial_size = blue_team_df.shape[0]
N = 5000

sims = abs(np.random.binomial(trial_size, p, size=N) / trial_size - p)

p_val = np.mean(sims >= observed_ts)
print(f'The chance of Blue winning is {blue_team_df['result'].mean()}')
if p_val < 0.05:
    print(f'We are safe to Reject the Null, the p-value is {p_val}. The chance of B
else:
    print(f'We failed to reject the Null, the p-value is {p_val}. The chance of Blu
```

```
The chance of Blue winning is 0.5338512763596004
We are safe to Reject the Null, the p-value is 0.0. The chance of Blue winning is no
t 50%
```

It seems that the game is not balanced by sides.

Step 5: Framing a Prediction Problem

The final Prediction problem for this project is simple:

Given statistics of the large neutral monster kills of a team, predict whether the team will eventually win the game or not.

- `Inputs`: combinations of the game metadata: `'firstdragon'`, `'dragons'`, `'opp_dragons'`, `'elementaldrakes'`, `'opp_elementaldrakes'`, `'elders'`, `'opp_elders'`, `'heralds'`, `'opp_heralds'`, `'void_grubs'`, `'opp_void_grubs'`, `'firstbaron'`, `'barons'`, `'opp_barons'`, `'atakhans'`, and `'opp_atakhans'`
- `Output`: prediction of `'result'`, binary

Final dataframe to use for this task: `neutral_monster_metadata_df`, which only contains the columns above, with an additional `'side'` used later for fairness testing.

In [796…
```
neutral_monster_metadata_df = final_team_df.drop(columns = ['gameid', 'datacomplete
```

In [797…
```
neutral_monster_metadata_df.head()
```

Out[797…

	side	firstdragon	dragons	opp_dragons	elementaldrakes	opp_elementaldrakes	elder
10	Blue	0.0	0.0	2.0	0.0	2.0	0.(
11	Red	1.0	2.0	0.0	2.0	0.0	0.(
22	Blue	0.0	3.0	2.0	3.0	2.0	0.(
23	Red	1.0	2.0	3.0	2.0	3.0	0.(
34	Blue	0.0	0.0	4.0	0.0	4.0	0.(

In [820…
```
neutral_monster_metadata_df[neutral_monster_metadata_df['result']==1].shape[0]
```

Out[820… 9911

In [821…
```
neutral_monster_metadata_df[neutral_monster_metadata_df['result']==0].shape[0]
```

Out[821… 9911

Step 6: Baseline Model

My baseline model would be a decision tree that only considers how many more `'barons'` and `'elders'` a team has killed than the opponent team. That is, create 2 new features: 1. `'barons_diff'` = `'barons'` - `'opp_barons'`; 2. `'elders_diff'` = `'elders'` - `'opp_elders'`. Use these 2 column values to determine whether the team wins or not. There is no need for any transformers because both these columns are linear and their scales would not matter for a decision tree.

I will use GridSearchCV to test out the best hyperparameters.

The evaluation metric would be accuracy because data was collected from both blue team and red team of a game. The percentage of wins of this dataset is the same as the

percentage of loses: 50%. Hence there is no unbalanced data and accuracy is a valid metric.

In [836…
```
feature_cols_all = [
    'firstdragon', 'elementaldrakes', 'opp_elementaldrakes',
    'elders', 'opp_elders', 'heralds', 'opp_heralds', 'void_grubs',
    'opp_void_grubs', 'firstbaron', 'barons', 'opp_barons', 'atakhans', 'opp_atakha
]
```

In [837…
```
# Create the new features:
neutral_monster_metadata_df['barons_diff'] = neutral_monster_metadata_df['barons']
neutral_monster_metadata_df['elders_diff'] = neutral_monster_metadata_df['elders']

feature_cols_base = ['barons_diff', 'elders_diff']
```

Split the data into outer train and outer test in this way so that all models use the same training and testing data.

In [838…
```
X_all = neutral_monster_metadata_df[feature_cols_all]
y = neutral_monster_metadata_df['result']

X_all_train, X_all_test, y_train, y_test = train_test_split(
    X_all, y, test_size=0.1
)

X_train = neutral_monster_metadata_df.loc[X_all_train.index, feature_cols_base]
X_test = neutral_monster_metadata_df.loc[X_all_test.index, feature_cols_base]
```

In [839…
```
hyperparams = {
    'max_depth' : [2, 4, 8, 16],
    'min_samples_split' : [100, 1000, 2000],
    'criterion' : ['gini', 'entropy']
}
```

In [840…
```
base_classifier = DecisionTreeClassifier()
grid_search = GridSearchCV(
    estimator=base_classifier,
    param_grid=hyperparams,
    cv=5,
    scoring='accuracy'
)

grid_search.fit(X_train, y_train)

print("\nGrid Search Results:")
print(f"Best Parameters Found: {grid_search.best_params_}")
print(f"Best Cross-Validation Accuracy: {grid_search.best_score_:.4f}")
```

```
Grid Search Results:
Best Parameters Found: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_split':
1000}
Best Cross-Validation Accuracy: 0.8182
```

In [841…
```
best_dt_classifier = grid_search.best_estimator_
y_pred = best_dt_classifier.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)

print(f"\nOptimal Decision Tree Evaluation:")
print(f"Model Accuracy on Test Set: {accuracy:.4f}")
print(f"Interpretation: The best Decision Tree model correctly predicts the game re
```

```
Optimal Decision Tree Evaluation:
Model Accuracy on Test Set: 0.8185
Interpretation: The best Decision Tree model correctly predicts the game result 81.8
5% of the time.
```

Step 7: Final Model

Model One: A decision tree that takes in all features of the final dataset

Features: `'firstdragon'`, `'dragons'`, `'opp_dragons'`, `'elementaldrakes'`, `'opp_elementaldrakes'`, `'elders'`, `'opp_elders'`, `'heralds'`, `'opp_heralds'`, `'void_grubs'`, `'opp_void_grubs'`, `'firstbaron'`, `'barons'`, `'opp_barons'`, `'atakhans'`, and `'opp_atakhans'`

I will keep using GridSearchCV to find the best hyperparameters and accuracy for evaluation.

I am binarizing `'void_grubs'` and `'opp_void_grubs'` because the gaming community commonly agrees that killing 1 void grub is extremely useless and would be the same as killing none. And killing 2 or above is considered useful but there is no significant difference after 2.

I am standardizing all quantitative columns just to make the final parameters more interpretable. I will skip quantitative columns that can only be 0 and 1 because these columns are more like binary columns.

In [842…
```
BINARIZE_COLS = ['void_grubs', 'opp_void_grubs']
SCALE_COLS = ['elementaldrakes', 'opp_elementaldrakes', 'elders', 'opp_elders', 'ba
```

In [843…
```
preprocessor = ColumnTransformer(
    transformers=[
        ('binarize_grubs', Binarizer(threshold=1), BINARIZE_COLS),
        ('scale_counts', StandardScaler(), SCALE_COLS),
    ],
    remainder='passthrough'
)

full_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('dt', DecisionTreeClassifier())
])
```

In [844…
```
hyperparams = {
    'dt__max_depth' : [2, 4, 8, 16, 32, 64],
    'dt__min_samples_split' : [5, 10, 100, 1000],
```

```
    'dt__criterion' : ['gini', 'entropy']
}
```

In [845…
```
print("\nAll Features Decision Tree Stats:")

grid_search = GridSearchCV(
    estimator=full_pipeline,
    param_grid=hyperparams,
    cv=5,
    scoring='accuracy'
)

grid_search.fit(X_all_train, y_train)

print("\nGrid Search Results:")
print(f"Best Parameters Found: {grid_search.best_params_}")
print(f"Best Cross-Validation Accuracy: {grid_search.best_score_:.4f}")
```

```
All Features Decision Tree Stats:

Grid Search Results:
Best Parameters Found: {'dt__criterion': 'gini', 'dt__max_depth': 8, 'dt__min_sample
s_split': 10}
Best Cross-Validation Accuracy: 0.9212
```

In [846…
```
best_dt_pipeline = grid_search.best_estimator_
y_pred = best_dt_pipeline.predict(X_all_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"\nOptimal All Features Decision Tree Evaluation:")
print(f"Model Accuracy on Test Set: {accuracy:.4f}")
print(f"Interpretation: The best All Features Decision Tree model correctly predict
```

```
Optimal All Features Decision Tree Evaluation:
Model Accuracy on Test Set: 0.9218
Interpretation: The best All Features Decision Tree model correctly predicts the gam
e result 92.18% of the time.
```

Now lets transform this tree into a random forest classifier

In [847…
```
full_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('rf', RandomForestClassifier(random_state=42))
])
```

In [851…
```
hyperparams = {
    'rf__n_estimators': [10, 50, 100],
    'rf__max_depth' : [4, 8, 16, 32],
    'rf__min_samples_split' : [5, 10, 100, 1000],
    'rf__criterion' : ['gini', 'entropy']
}
```

In [852…
```
grid_search = GridSearchCV(
    estimator=full_pipeline,
    param_grid=hyperparams,
```

```
    cv=5,
    scoring='accuracy'
)

grid_search.fit(X_all_train, y_train)

print("\nGrid Search Results:")
print(f"Best Parameters Found: {grid_search.best_params_}")
print(f"Best Cross-Validation Score: {grid_search.best_score_:.4f}")
```

```
Grid Search Results:
Best Parameters Found: {'rf__criterion': 'entropy', 'rf__max_depth': 16, 'rf__min_sa
mples_split': 10, 'rf__n_estimators': 50}
Best Cross-Validation Score: 0.9277
```

In [853…

```
best_rf_pipeline = grid_search.best_estimator_
y_pred = best_rf_pipeline.predict(X_all_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Optimal All Features Random Forest Evaluation:")
print(f"Model Accuracy on Test Set: {accuracy:.4f}")
print(f"Interpretation: The best All Features Random Forest model correctly predict
```

```
Optimal All Features Random Forest Evaluation:
Model Accuracy on Test Set: 0.9314
Interpretation: The best All Features Random Forest model correctly predicts the gam
e result 93.14% of the time.
```

Conclusion:

The All Features Random Forest model is the one with the highest accuracy hence my final model. The hyperparameters are `{'rf__criterion': 'entropy', 'rf__max_depth': 16, 'rf__min_samples_split': 10, 'rf__n_estimators': 100}`

Step 8: Fairness Analysis

As we proved in step 4.3, the data hints that blue and red teams do not have a even 50/50 chance of winning. Let's see if my All Features Random Forest model is fair enough to have equally accurate predictions for both teams. We will check this by using a permutation test to see if they are from the same distribution.

- **Null Hypothesis**: Our model is fair. The accuracy for predicting the right result for blue team and red team are roughly the same, and any differences are due to random chance.
- **Alternative Hypothesis**: Our model is unfair. The accuracy for predicting the right result for blue team and red team are not the same
- **Statistic**: abs((percentage of games where Blue was predicted correctly) - (percentage of games where Red was predicted correctly))

In [813…
```
test_data = neutral_monster_metadata_df.loc[X_all_test.index, 'side']

test_results_df = pd.DataFrame({
    'True_Result': y_test,
    'Predicted_Result': y_pred,
    'side': test_data
})

test_results_df['correctly_predicted'] = test_results_df['True_Result'] == test_res

test_results_df.shape[0]
```

Out[813… 1983

Sanity check: there are 19822 data points in neutral_monster_metadata_df and we split 10% of the data for testing. Hence 1983 is a very reasonable size for test_results_df.

In [862…
```
blue_team_df = test_results_df[test_results_df['side'] == 'Blue']
red_team_df = test_results_df[test_results_df['side'] == 'Red']
observed_ts = abs(blue_team_df['correctly_predicted'].mean() - red_team_df['correct
trial_size = test_results_df.shape[0]
N = 5000

sims = []

for _ in range(N):
    shuffled = test_results_df.copy()
    shuffled['shuffled_side'] = np.random.permutation(shuffled['side'])
    shuffled_blue_team_df = shuffled[shuffled['shuffled_side'] == 'Blue']
    shuffled_red_team_df = shuffled[shuffled['shuffled_side'] == 'Red']
    ts = abs(shuffled_blue_team_df['correctly_predicted'].mean() - shuffled_red_tea
    sims.append(ts)

p_val = np.mean(sims >= observed_ts)
print(f'The accuracy of Blue predictions from the model is {blue_team_df['correctly
print(f'The accuracy of Red predictions from the model is {red_team_df['correctly_p
```

```
if p_val < 0.05:
    print(f'We are safe to Reject the Null, the p-value is {p_val}. The accuracy of
else:
    print(f'We failed to reject the Null, the p-value is {p_val}. The accuracy of b
```

```
The accuracy of Blue predictions from the model is 0.9324191968658179
The accuracy of Red predictions from the model is 0.9106029106029107
We failed to reject the Null, the p-value is 0.0774. The accuracy of blue and red wi
n predictions are very similar. The model is very likely to be fair.
```

In [866…
```
blue_team_df.shape[0]
```

Out[866…
```
1021
```

In [865…
```
blue_team_df[blue_team_df['True_Result'] == 1].shape[0]
```

Out[865…
```
541
```

In [867…
```
red_team_df.shape[0]
```

Out[867…
```
962
```

In [868…
```
red_team_df[red_team_df['True_Result'] == 1].shape[0]
```

Out[868…
```
438
```

Conclusion: The final All Features Random Forest model acheives accuracy parity.