Scientific Programming with Python - Final Project

intro:

The subject of my project is the online computer game League of Legends.
In the game, two teams of five players battle in player versus player combat, with each team occupying and defending their own half of the map.
Each of the ten players controls a character, known as a "champion", with unique abilities and differing styles of play. During a match, champions collect experience points to gain levels and purchase items in order to defeat the opposing team.
In the game's main mode, Summoner's Rift, a team wins by pushing through to the enemy base and destroying their "nexus", a large structure located within it.
During the game, the players also fight AI controlled minions, place vision wards to deal with fog of war, and support each other with healing and shielding abilities.
The main goal of this project is to predict which team will win based on the dataset provided.
Dataset characteristics:

1. The dataset contains 65896 rows divided into 17 columns .

```
RangeIndex: 65896 entries, 0 to 65895
Data columns (total 17 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   gameId                 65896 non-null  int64
 1   gameDuraton            63247 non-null  float64
 2   blueWardPlaced         63298 non-null  float64
 3   blueWardkills          65896 non-null  int64
 4   blueTotalMinionKills   54138 non-null  float64
 5   blueJungleMinionKills  65896 non-null  int64
 6   blueTotalHeal          63266 non-null  float64
 7   redWardPlaced          59234 non-null  float64
 8   redWardkills           65896 non-null  int64
 9   redTotalMinionKills    50118 non-null  float64
 10  redJungleMinionKills   63203 non-null  float64
 11  redTotalHeal           59283 non-null  float64
 12  win                    65896 non-null  object
 13  FirstBlood             63265 non-null  object
 14  FirstTower             65896 non-null  object
 15  FirstBaron             65896 non-null  object
 16  FirstDragon            65896 non-null  object
```

2. The dataset consists of 5 categorial features and 12 numeric continuous features.
   Features:

   Categorical - Nominal: possible values- (Red,Blue)

   1. Win: which team won the game.
   2. FirstDragon: which team killed a dragon first.
   3. FirstBaron: which team first killed “Baron Nashor”.
   4. FirstTower: which team first destroyed an enemy tower.
   5. FirstBlood: which team first killed an enemy champion.

   Numeric:

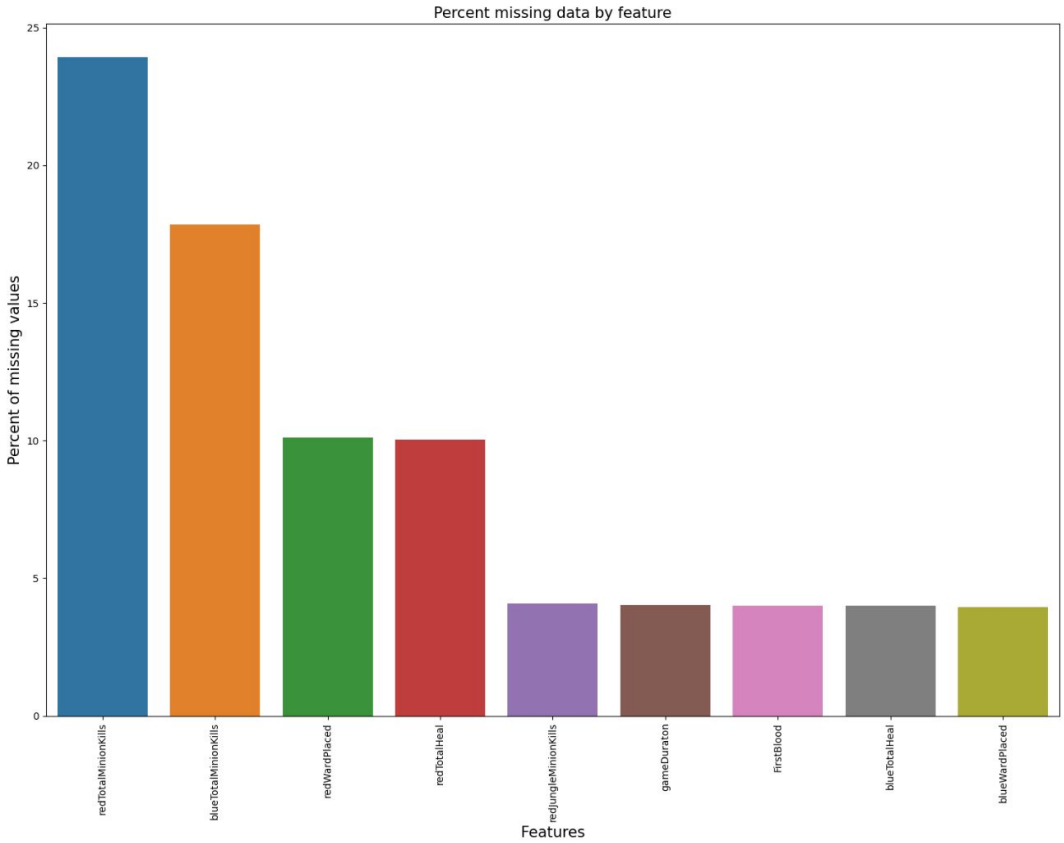
   6. gameId: Unique game ID.
   7. gameDuraton: Game Duration(seconds)
   8. blueWardPlaced: number of blue team ward placed
   9. blueWardkills: number of blue words killed.
   10. blueTotalMinionKills: number of minions killed by the blue team.
   11. blueJungleMinionKills: number of jungle minions killed by the blue team.
   12. blueTotalHeal: amount of blue team heals.
   13. redWardPlaced: number of red team ward placed
   14. redWardkills: number of red words killed.
   15. redTotalMinionKills: number of minions killed by the red team.
   16. redJungleMinionKills: number of jungle minions killed by the red team.
   17. redTotalHeal: amount of red team heals.

3. Statistical analysis of the features:
   It is possible to see based on the min/max that there are probably many outliers at the dataset. We will deal with that later.

	gameId	gameDuraton	blueWardPlaced	blueWardkills	blueTotalMinionKills	blueJungleMinionKills	blueTotalHeal	redWardPlaced
count	65896	63247	63298	65896	54138	65896	63266	59234
unique								
top								
freq								
mean	4172853622	1427.689867	53.61599103	19.81989802	498.9246001	122.692303	24487.68154	53.76393625
std	78892906.26	432.2107279	31.2496922	15.00322678	179.5331271	67.14307104	15160.87987	31.47986211
min	3191955256	-1702	0	0	0	0	0	0
25%	4144504756	1121	33	8	379	81	13442	33
50%	4194611108	1414	54	18	515	126	21469	54
75%	4224057038	1724	74	29	623	169	32203.75	74
max	4257021673	3301	230	118	1315	400	261707	226

	redWardkills	redTotalMinionKills	redJungleMinionKills	redTotalHeal	win	FirstBlood	FirstTower	FirstBaron	FirstDragon
count	65896	50118	63203	59283	65896	63265	65896	65896	65896
unique					2	2	2	2	2
top					Red	Blue	Blue	Red	Red
freq					33237	32042	33994	50450	39847
mean	19.4883301	504.3887226	124.3248422	24815.26085					
std	14.77295967	183.994081	67.97596492	15320.54813					
min	0	0	0	0					
25%	8	381	82	13684.5					
50%	18	522	128	21840					
75%	29	632	171	32650					
max	117	1234	417	206758					

Initial Data Analysis:




```
gameId - 0%
gameDuraton - 4%
blueWardPlaced - 4%
blueWardkills - 0%
blueTotalMinionKills - 18%
blueJungleMinionKills - 0%
blueTotalHeal - 4%
redWardPlaced - 10%
redWardkills - 0%
redTotalMinionKills - 24%
redJungleMinionKills - 4%
redTotalHeal - 10%
win - 0%
FirstBlood - 4%
FirstTower - 0%
FirstBaron - 0%
FirstDragon - 0%
```

In order to perform data cleansing first I wanted to understand the amount of missing values and their locations.

In these snippets it's clear that most of the missing data Is located at the “blueTotalMinionKills” column And the “redTotalMinionKills” but not exclusively.
In order to clean the data set, two main methods were applied.

1. Rows containing missing categorical values (in our case 4% of the dataset that are missing the “FirstBlood” feature) will be removed.
2. Rows containing missing numeric values will be fixed by populating the missing values with values derived from other columns with some relation between these two features. But in order to do so, first I needed to fix the incorrect data if any at the different features.

Fixing outliers:

Since all categorical columns contain only 2 unique values, there are no outliers we can detect in these columns.

Note: at all plots presented in this section below each bar represents 1% of the range available values.

Example: if a column has values in the range of 0-100 , each bar represents a diff of 1 in the value scale.

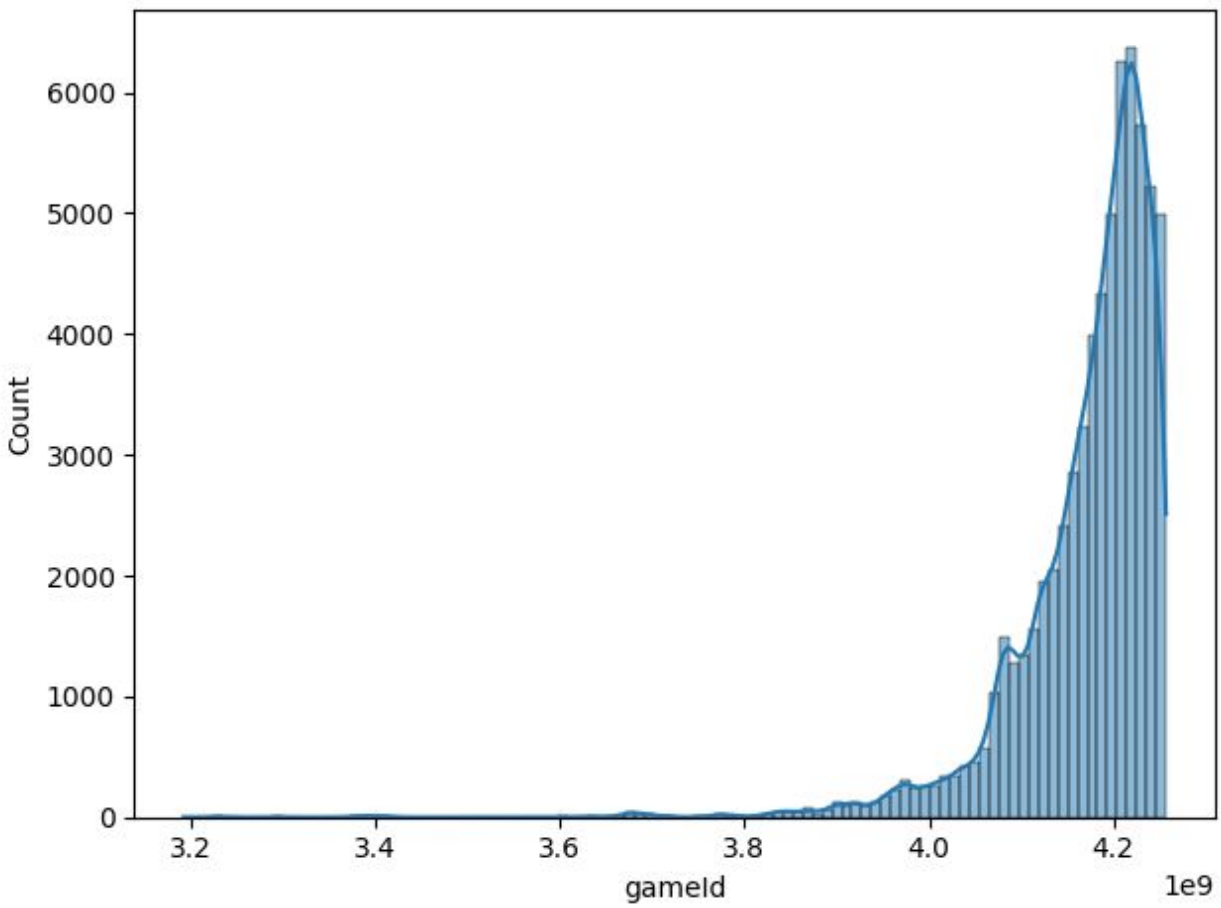
1. <u>Checking gameId:</u>
   First, I need to drop all duplicate values in this column since each game must be unique. That leads to removing 20 rows out of the dataset.
   Before:

```
gameId                 65896 non-null  int64
```

   After:

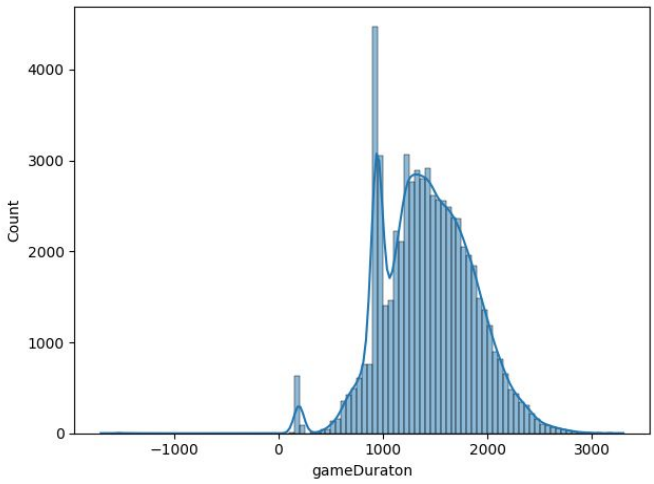
```
gameId                 65876 non-null  int64
```

   As seen below, all values in this column are larger than 0 and therefore are valid game Id.

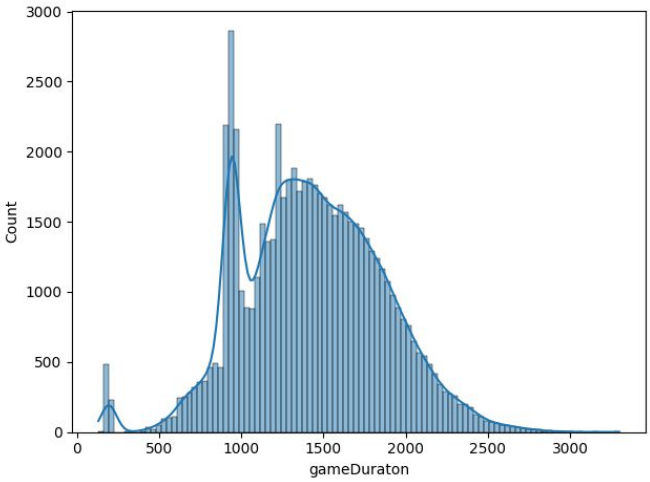


2. <u>Checking gameDuraton:</u>
   As seen below there are several rows containing a negative game duration which is not valid. Additionally, the lower hill on the left side may be incorrect but since game duration can't affect the winning team which is my prediction goal I will leave it this way. Additionally, the spike around 900 secondes is valid since in this game you can surrender after 15 minute (900 secondes) which explains the spike around that value.
   Source: https://leagueoflegends.fandom.com/wiki/Surrendering#Conditions
   All values below zero will have the mean of the column.

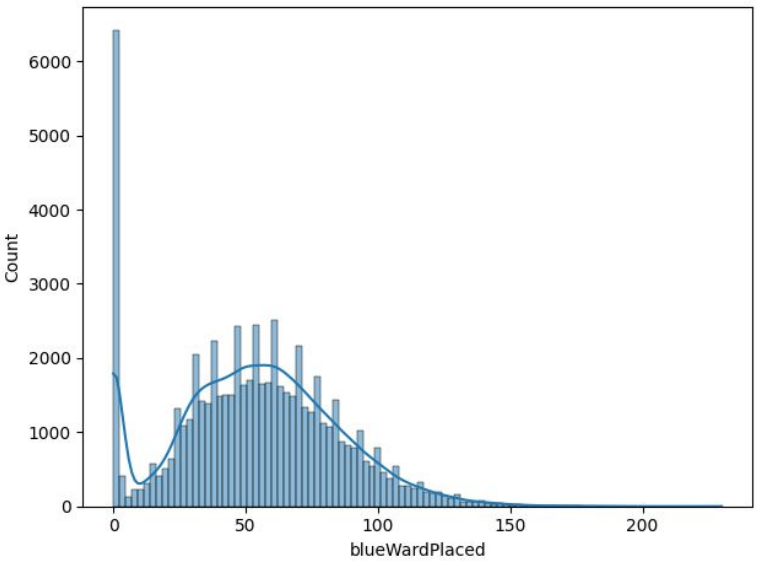
Before:

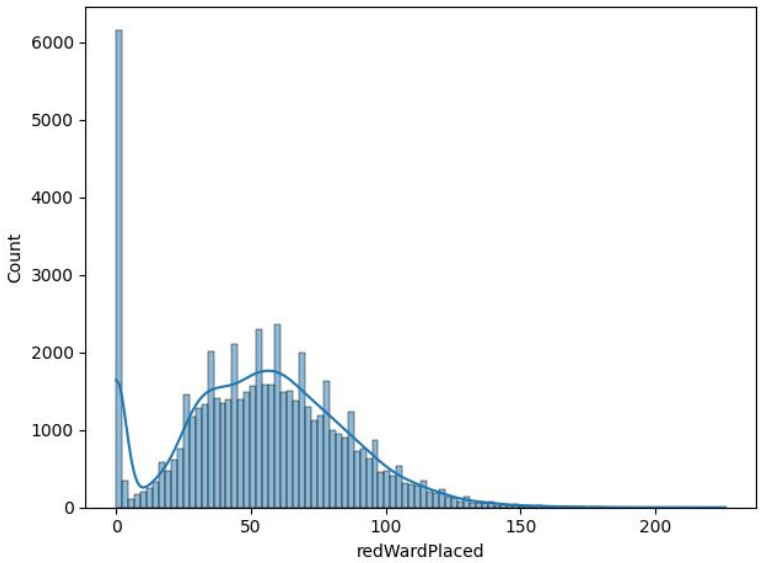


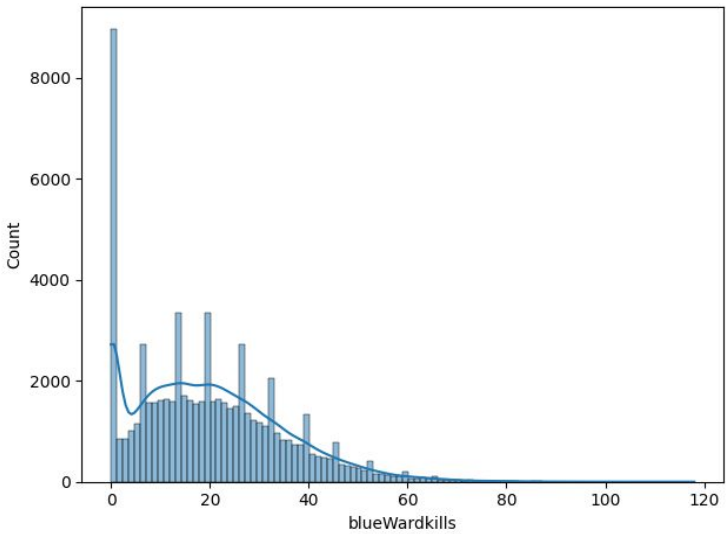
After:

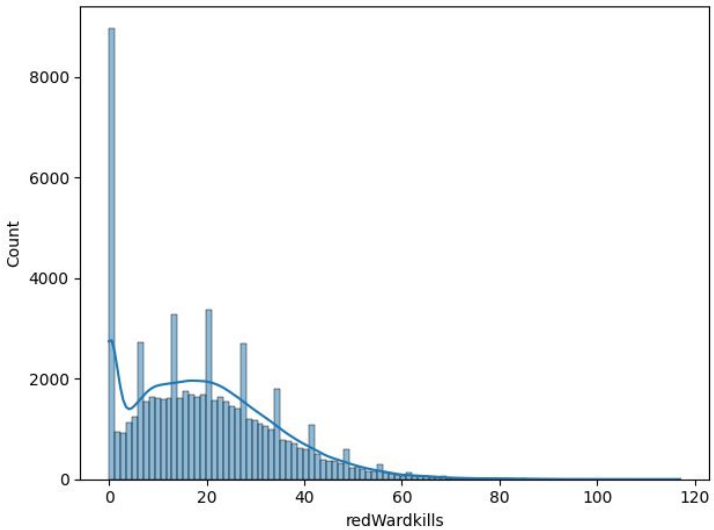


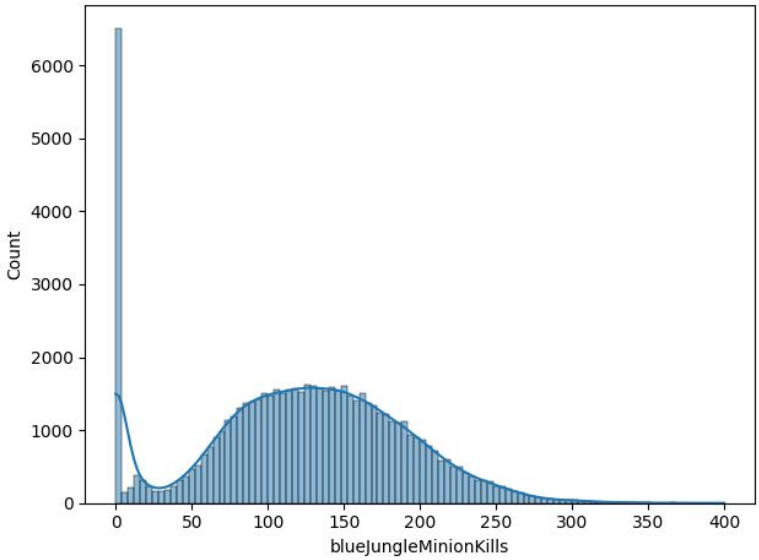
3. When considering what to do with the other features there were some significant errors related to zero values as seen below.

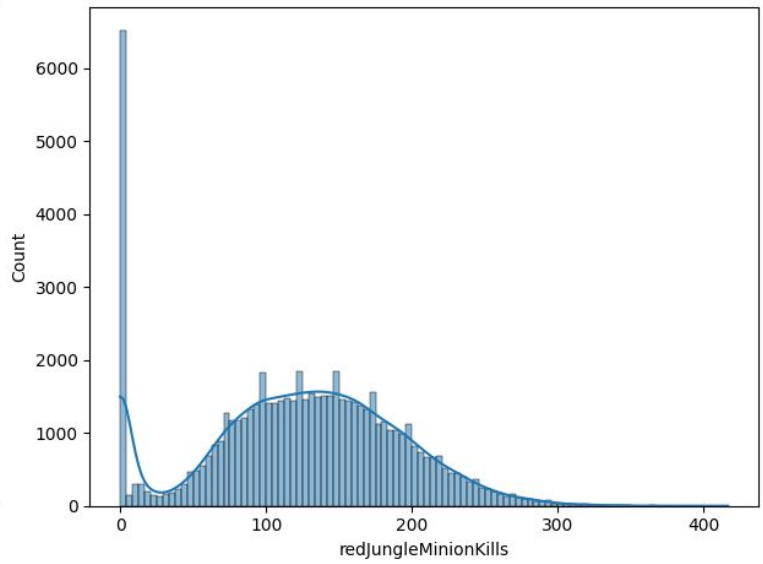




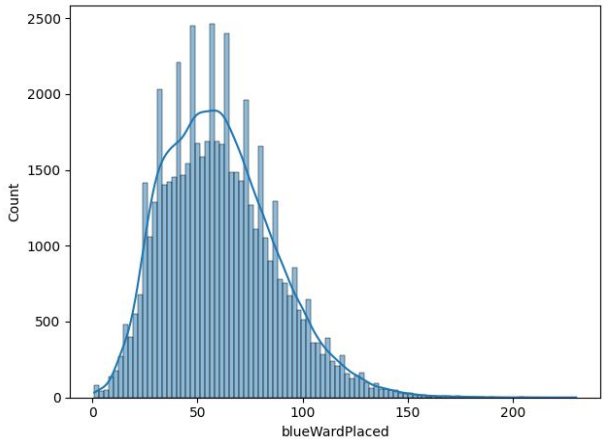


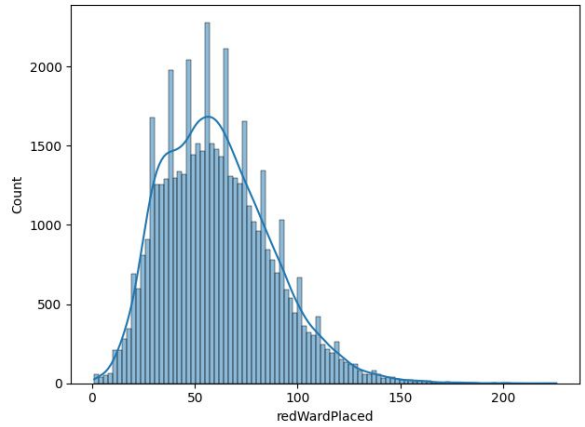


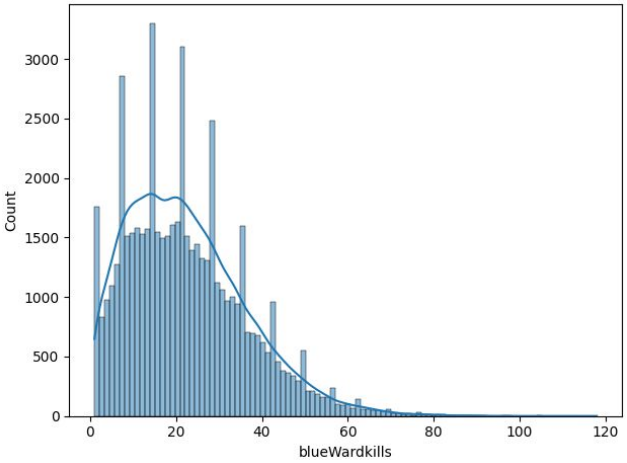


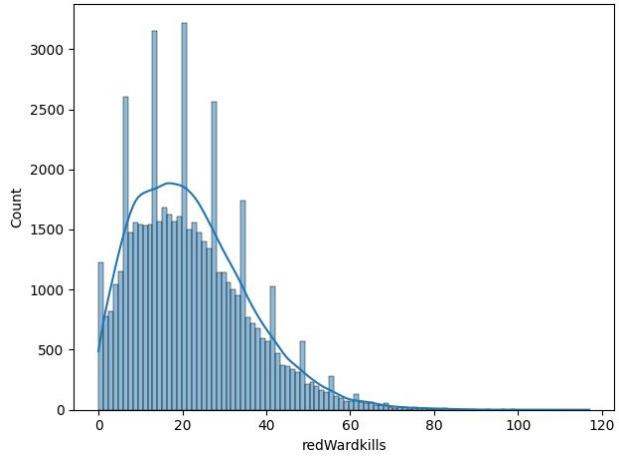


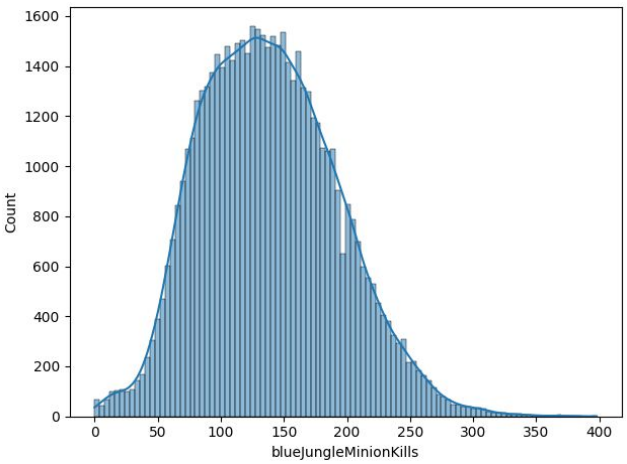
After noticing this i wanted to check how correlated are these columns and by that understand if maybe the rows containing zeros in one column are actually the same rows with zeros at other columns. If that's the case I would rather delete these rows since 6 corrupted values at a single row is not a reliable row. After removing all rows containing zero in the ‘blueWardPlaced’ column and the ‘blueWardkills’ column i received a data set containing 53,349 rows which is approximately 81% of the original dataset which actually means these are all the rows containing zeros at the other features.
Features distribution after removing the rows:

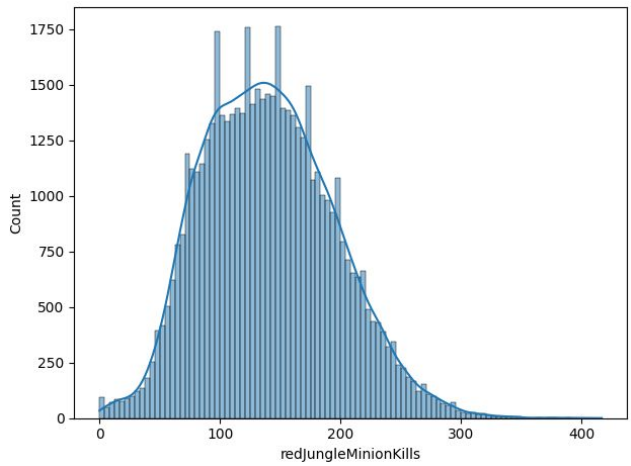






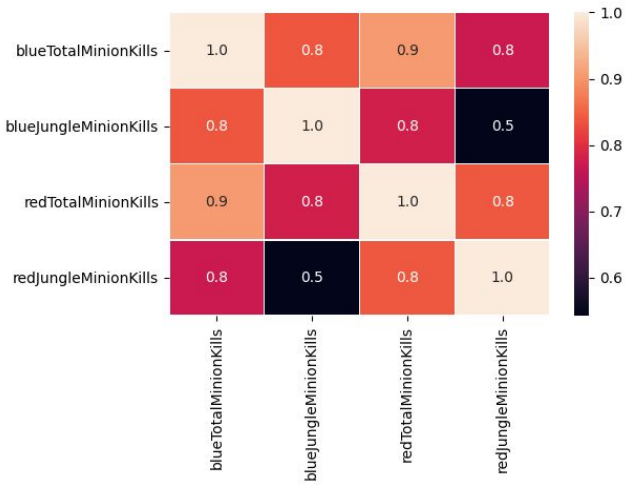


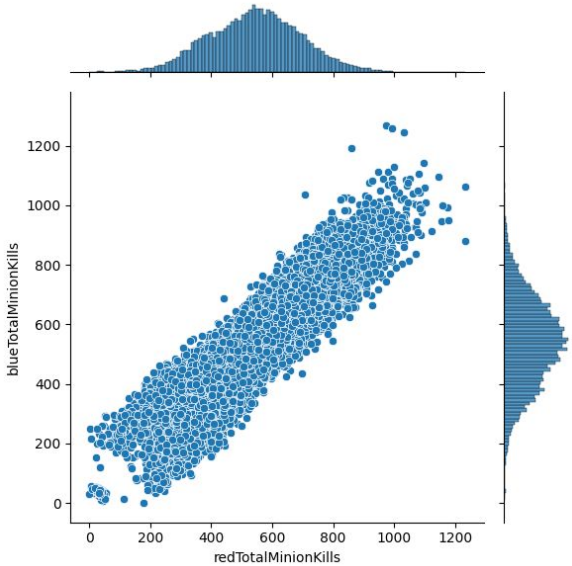


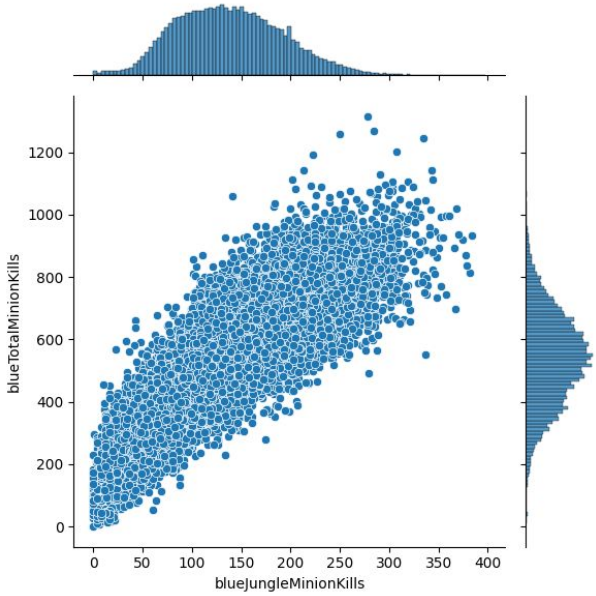


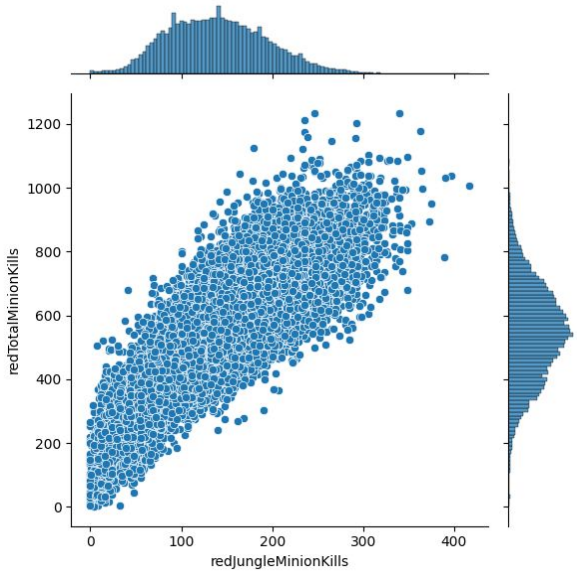
The reason this happened is that all of these six columns are highly correlated to each other. (i will check that in the next section)
As seen above, the distribution of the features is much better after removing these rows.
As noted above, rows with six inaccurate and estimated values (35% of columns) are better to be dropped then fixed.

After fixing or removing all the related outliers I can now fix all rows with missing values. As stated before, most of the missing data is located in the ‘blueTotalMinionKills’ and the ‘redTotalMinionKills’ columns.
These columns are highly correlated to the ‘blueJungleMinionKills’ column and the ‘redJungleMinionKills’ accordingly.









As seen in the plots above, these columns are highly correlated, but more interesting is that the ‘redTotalMinionKills’ and the ‘blueTotalMinionKills’ have the highest correlation.
In order to fix the missing values at the ‘redTotalMinionKills’ and ‘blueTotalMinionKills’ i will use 3 values in order to achieve the most accurate value possible.

1. Calculate the relation between every one of ‘redTotalMinionKills’ and ‘blueTotalMinionKills’ and keep the fraction representing the relation between them.
   Name: total_ratio

2. Calculate the relation between every one of ‘redTotalMinionKills’ and ‘redJungleMinionKills’ and keep the fraction representing the relation between them.
   And same for the second combination (‘blueTotalMinionKills’, ‘blueJungleMinionKills’).
   Name: total_jungle_ratio

3. Calculate the mean of the columns ‘redTotalMinionKills’ and ‘blueTotalMinionKills’ and save it.
   Name: total_mean

When reaching a missing value i will try to use one of these 2 calculations if possible:
1. Total_ratio * other total column if present.
2. Total_jungle_ratio * jungle if present.
3. Assign the total_mean

- The calculated value of total_ratio is 1.0212, in cases that first calculation is possible I will place the same value.
- The calculated value of the total_jungle_ratio is 0.2536 and 0.2549 (red,blue) . I will use 4 times the value present in the jungle column.

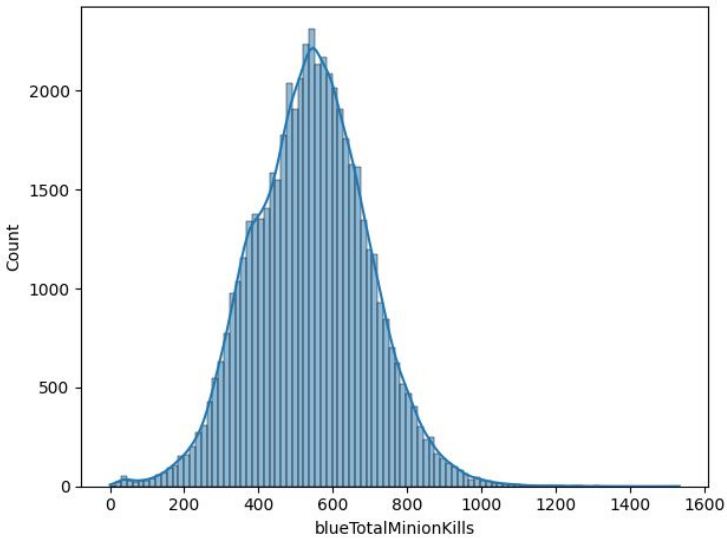
Final answer = first available option by order.

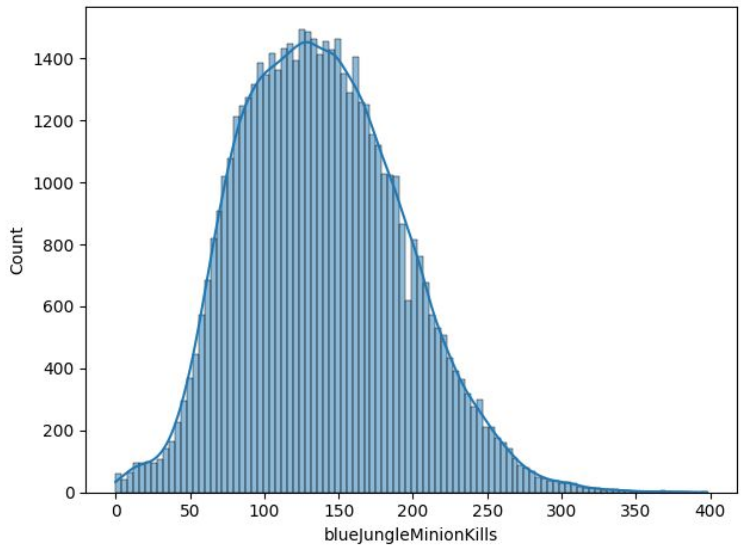
Additionally, when fixing missing values at the column ‘redJungleMinionKills’, i will use the same calculations used in explanations above.

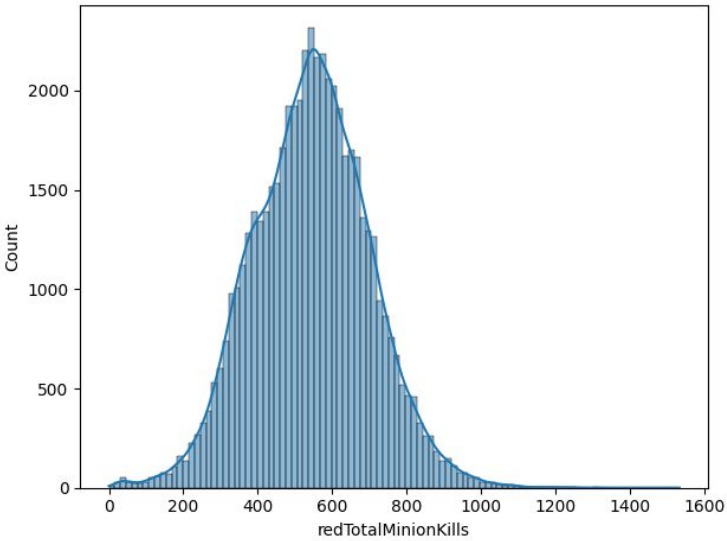
1. Missing value = ‘redTotalMinionKills’ / 4
2. Missing value = column mean.

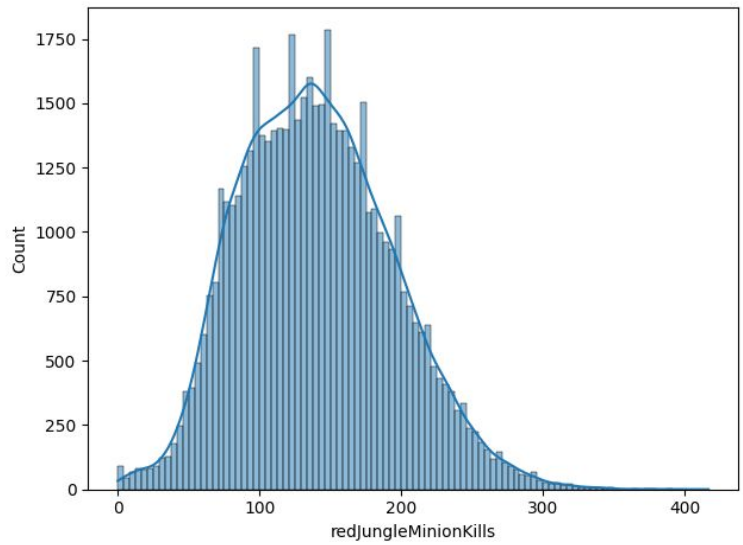
Final answer = first available option by order.

After fixing the missing values at the columns: “redTotalMinionKills”, “redJungleMinionKills” and “blueTotalMinionKills”. These are the new plots and data distribution:









Fixing the ‘redTotalHeal’ and ‘blueTotalHeal’ columns would be by populating the missing values with the mean of three values.
Since the correlation of this column is 0.7 with the ‘gameDuraton’, ‘blueTotalMinionKills’ and ‘blueJungleMinionKills’ columns, I will use this correlation to better populate the fields.

First value = the relation between ‘blueTotalHeal’ and ‘gameDuraton’ (the mean of all relations upon the column).

Second value = the relation between ‘blueTotalHeal’ and ‘blueTotalMinionKills’ (the mean of all relations upon the column).

Third value = the relation between ‘blueTotalHeal’ and ‘blueJungleMinionKills’ (the mean of all relations upon the column).

Each missing value will be populated by the mean of these 3 values.

Fixing the missing values in the ‘gameDuraton’ column will be by the same method mentioned above using the ‘redTotalMinionKills’ and ‘blueTotalMinionKills’

Fixing the missing values in the ‘redWardPlaced’ column will be by the same method mentioned above using the blueWardPlaced’ column

Finally, all remaining outliers based on the numeric columns that are at the top and bottom 0.13%. Leaving the dataset at size of 50,933 which is approximately 77.2 % of the original dataset.

Formatting:

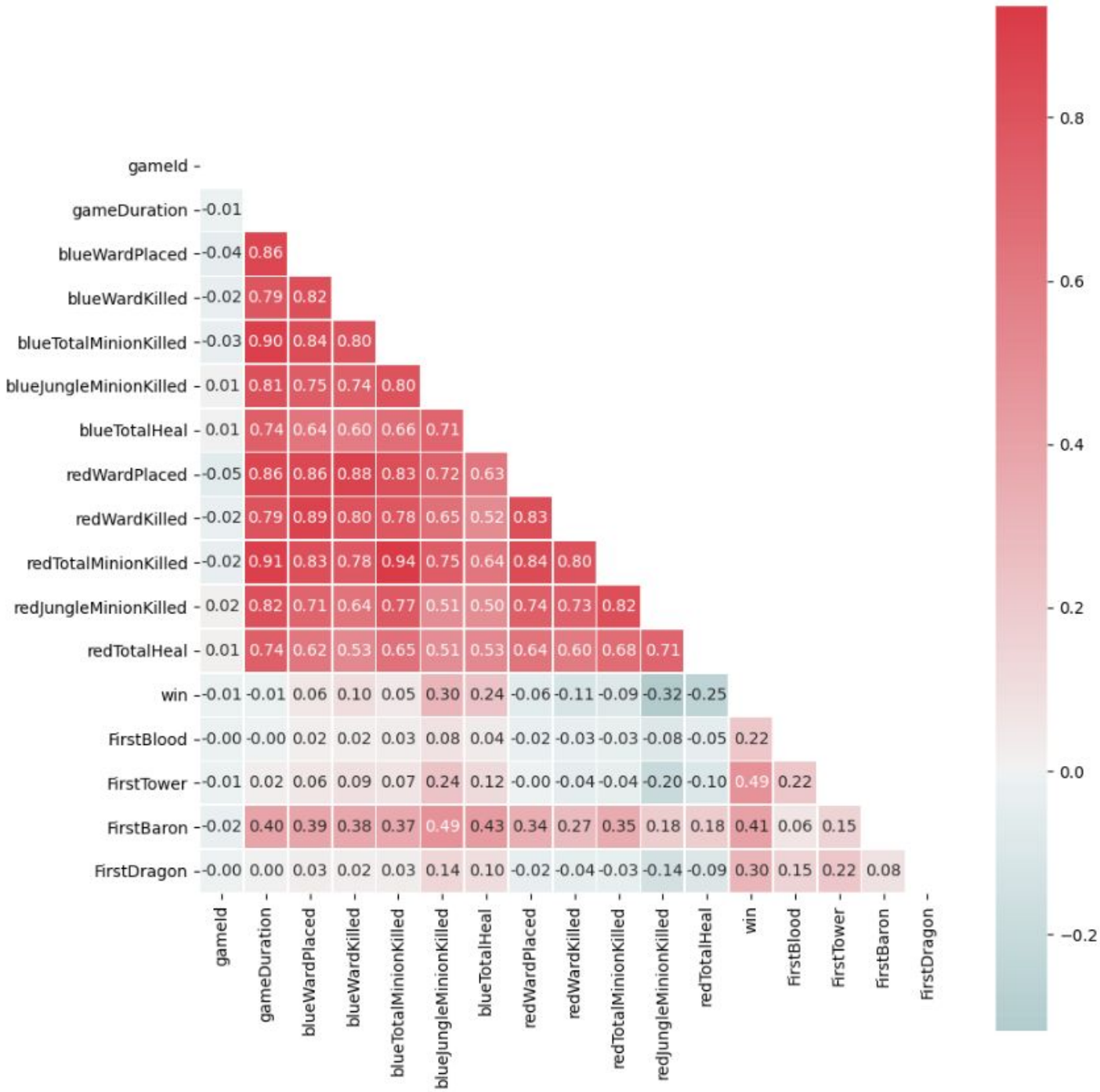
1. gameDuraton = gameDuration
2. blueWardkills = blueWardKilled
3. blueTotalMinionKills = blueTotalMinionKilled
4. blueJungleMinionKills = blueJungleMinionKilled
5. redWardkills = redWardKilled
6. redTotalMinionKills = redTotalMinionKilled
7. redJungleMinionKills = redJungleMinionKilled

Stage summary:

1. Remove all rows with missing categorical values.
2. Remove all duplicates at unique columns (gameId)
3. Fix blatant outliers - zero values, that affects the data cleansing by removing the rows with outliers at six or more columns.
4. Fixing all missing data by using strong correlated columns in order to populate the missing values in a precise manner.
5. Remove all remaining outliers (top and bottom 0.13%) of each column.

Exploratory Data Analysis:

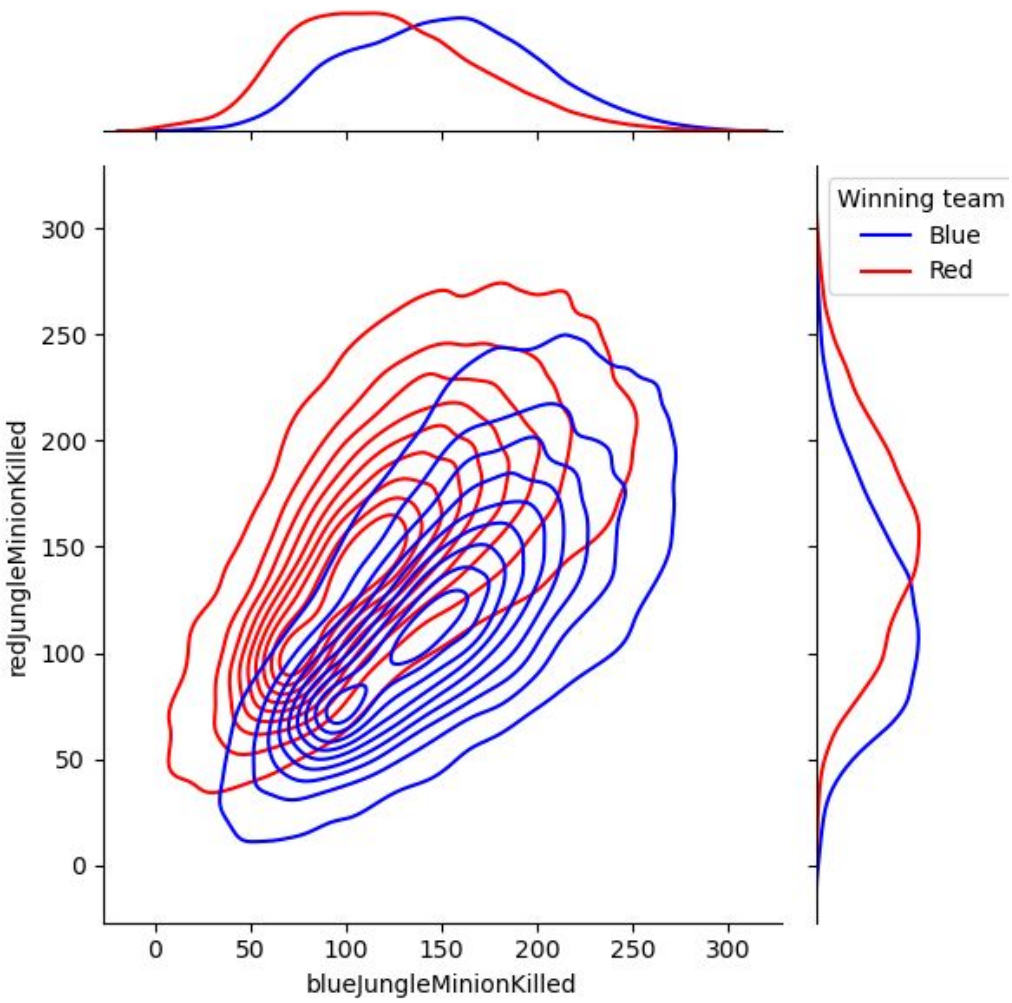
1. General heatmap:



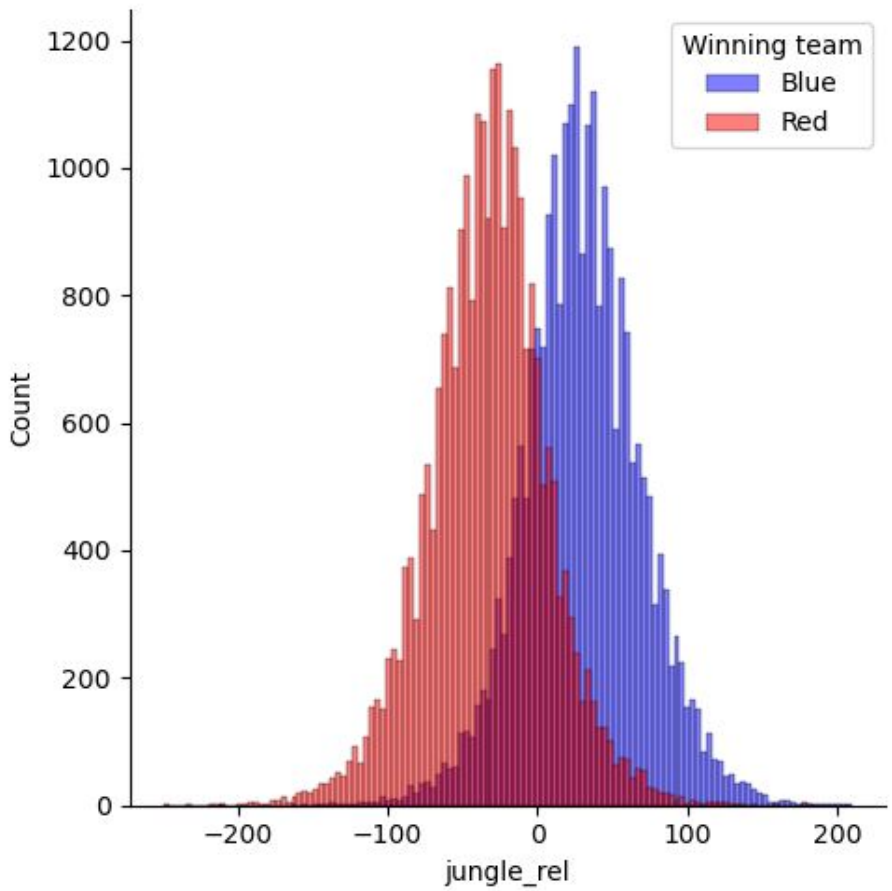
From this heatmap it's clear that there are some columns that are fairly correlated with the ‘win’ column.
The correlated features are:

1. FirstTower
2. FirstBaron
3. FirstDragon
4. FirstBlood
5. redJungleMinionKilled
6. blueJungleMinionKilled
7. blueTotalHeal
8. redTotalHeal

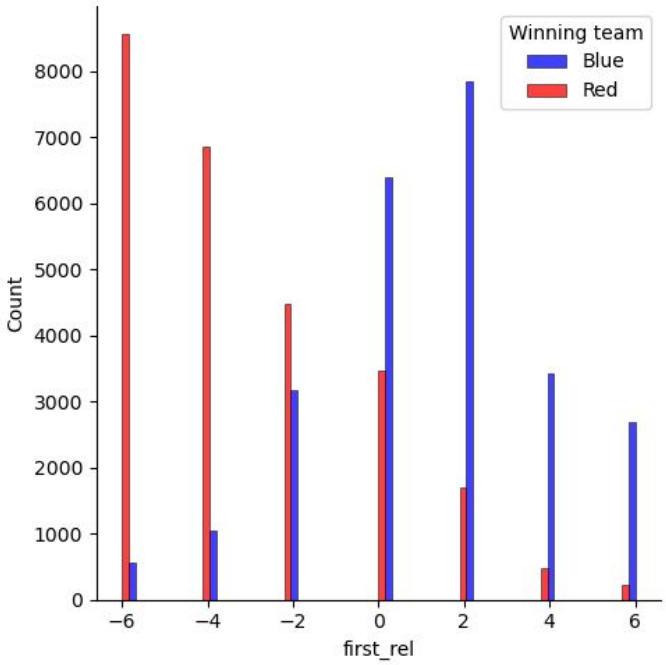
Since both ‘redJungleMinionKilled’ and ‘blueJungleMinionKilled’ are highly correlated to each other and also correlated to the win column i will first check the relation between them.

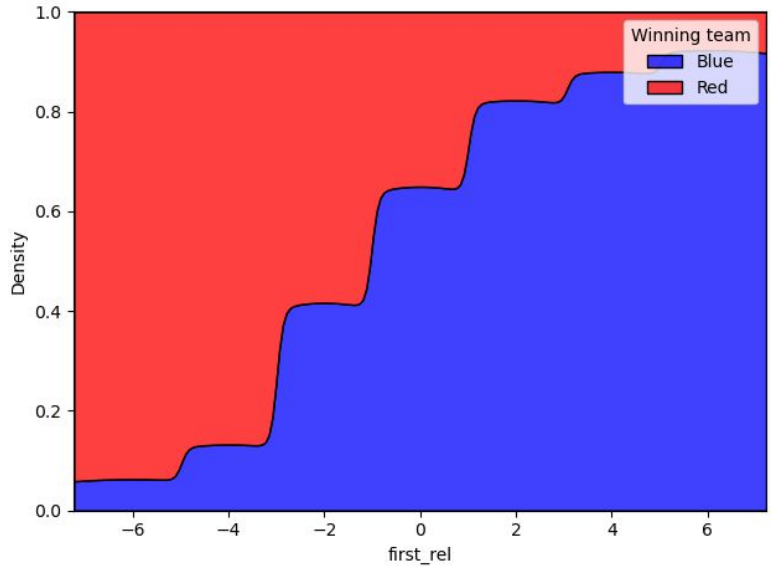


In this plot it's clear that there are some regions where ‘win’ is highly correlated to the ratio between these features. In order to take advantage of this property I created a new column called jungle_rel.
Junge_rel = ‘blueJungleMinionKilled’ - ‘redJungleMinionKilled’.
This way, if the value is positive it is in favor of the blue team and if it's negative it's in favor of the red team.

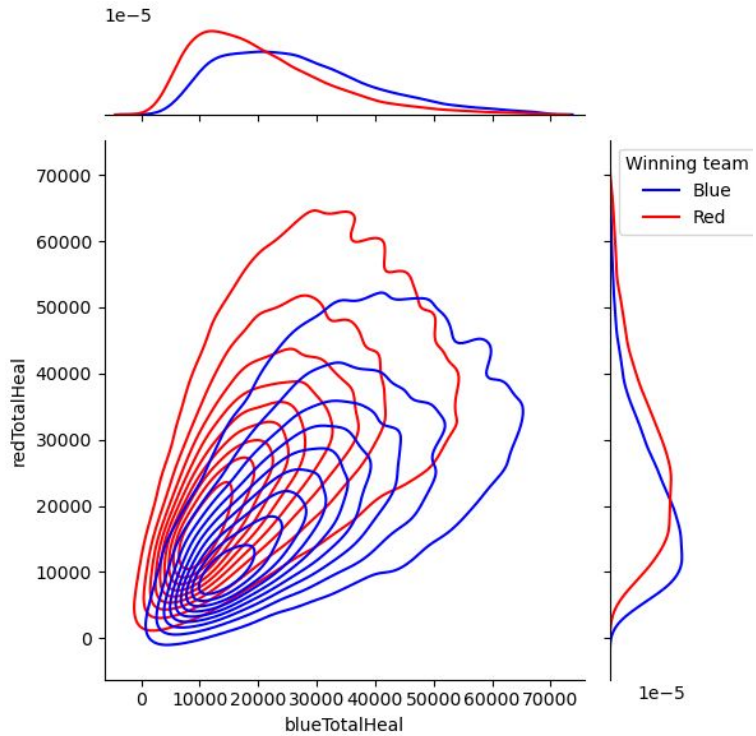


Additionally, as seen at the heatmap above, the four categorical features - ‘FirstBlood’ , ‘FirstTower’ , ‘FirstBaron’ and ‘FirstDragon’ are highly correlated with win. So I created a new column combining them all.
The new column called ‘first_rel’ is calculated like this: sum all four columns by the value they hold, if the value is red add the agreed value with a negative sign, if the value is blue, add the agreed value with a positive sign.

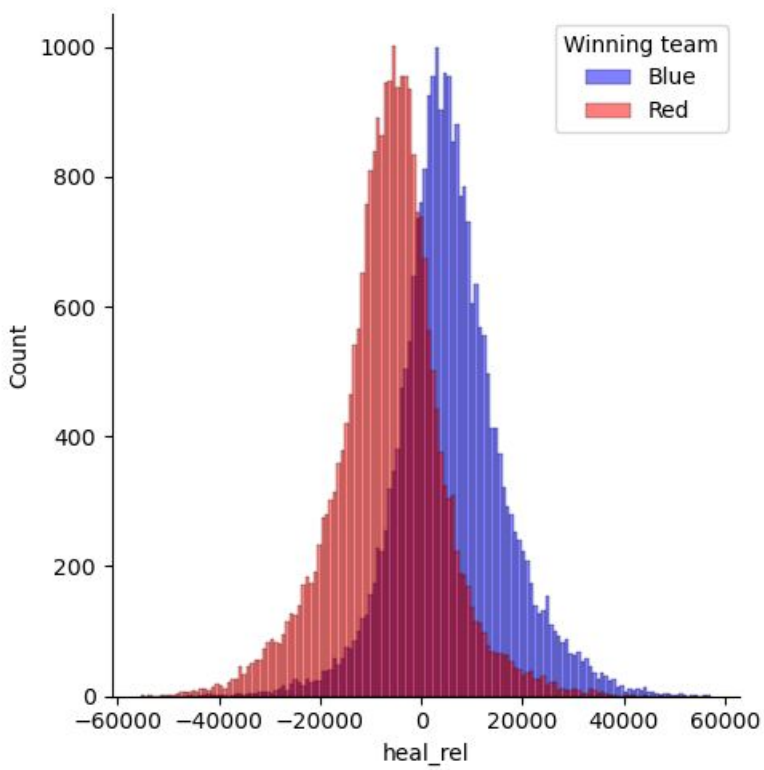




From these plots it's clear that this column splits fairly good the data based on the sum of the four categorical columns. The plot on the left displays the amount of samples divided by the winning team at each value of the ‘first_rel’ column.
It seems that the dataset is not distributed evenly, but when looking at the plot on the right it's easy to see that the distribution of samples is evenly spread upon all fields maintaining a symmetric ratio.

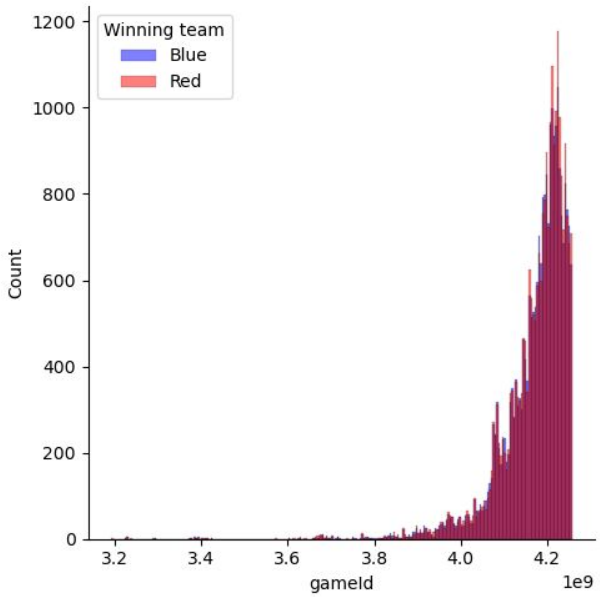


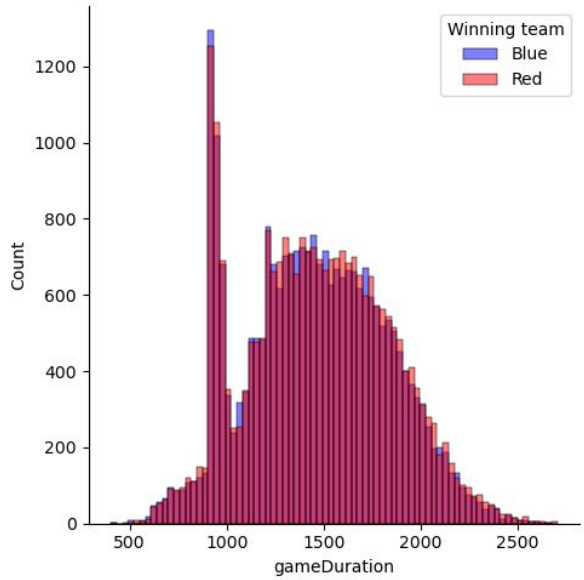
As for ‘redTotalHeal’ and ‘blueTotalHeal’ which are also strongly correlated to each other and are relatively strongly correlated to the ‘win’ column i will perform the same actions performed for the jungle_rel and will call the new column ‘heal_rel’.
heal_rel = ‘blueTotalHeal’ - ‘redTotalHeal’.

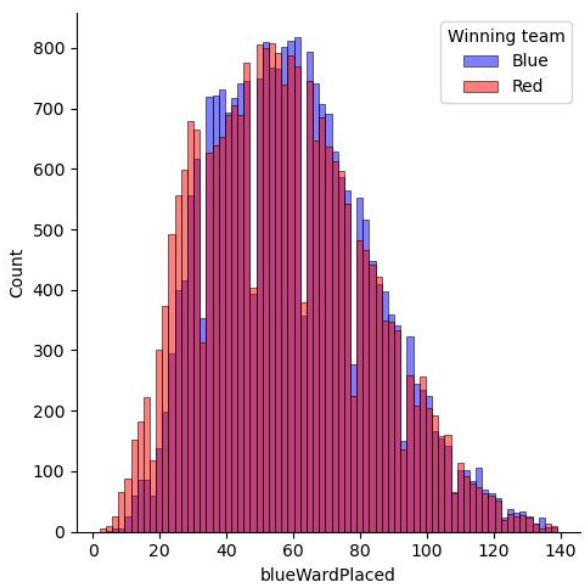


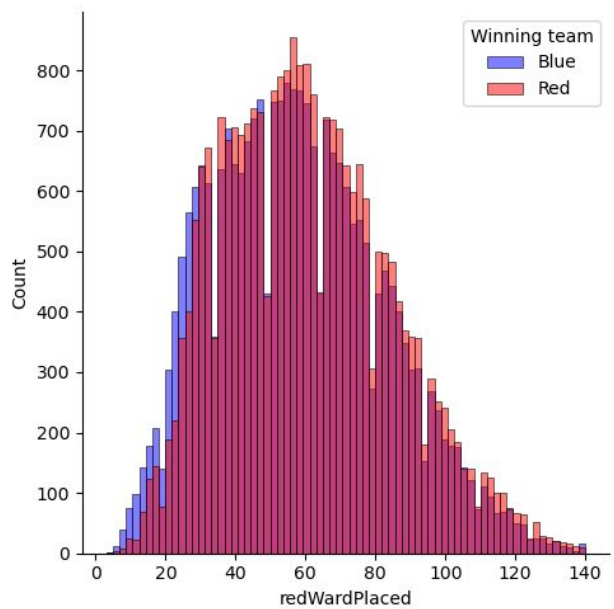
Eventually, after creating the ‘jungle_rel’, ‘heal_rel’ and ‘ward_rel’ (created the same way as the previous two) columns, I merged them into one containing them all.

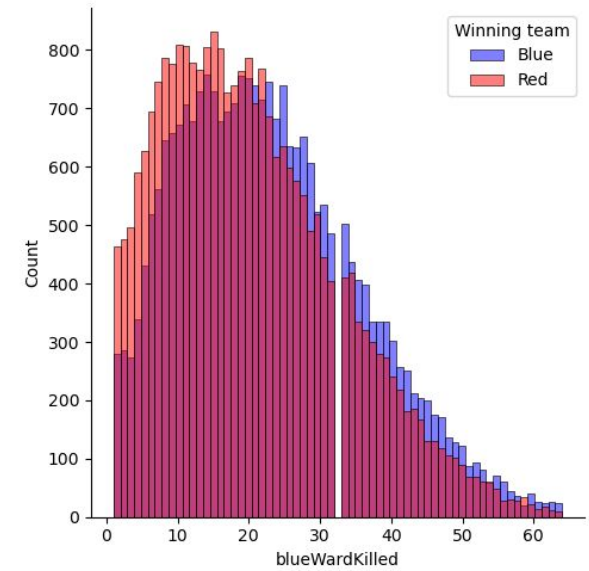
In order for me to be sure that i took advantage of all relevant columns, i created for each feature a distribution plot that shows if there is any diverging i might be able to use.

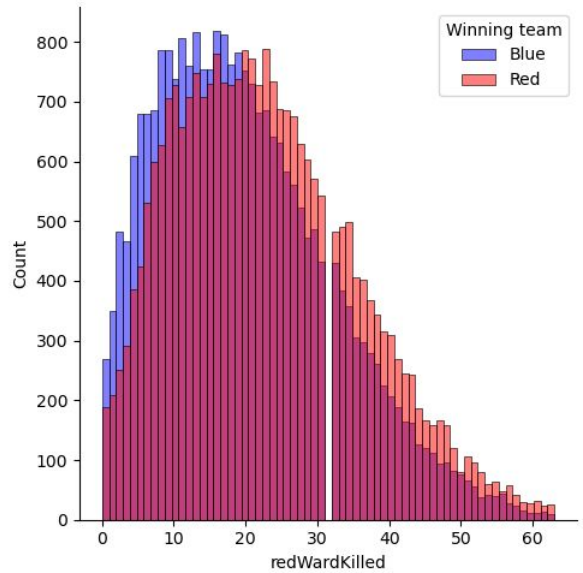


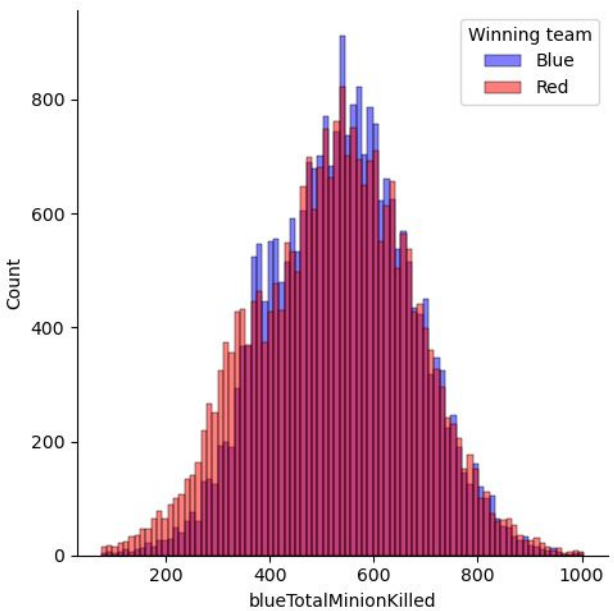


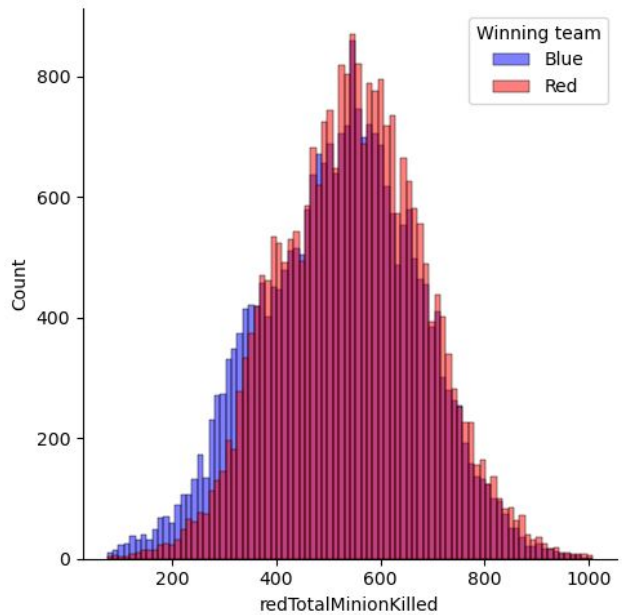


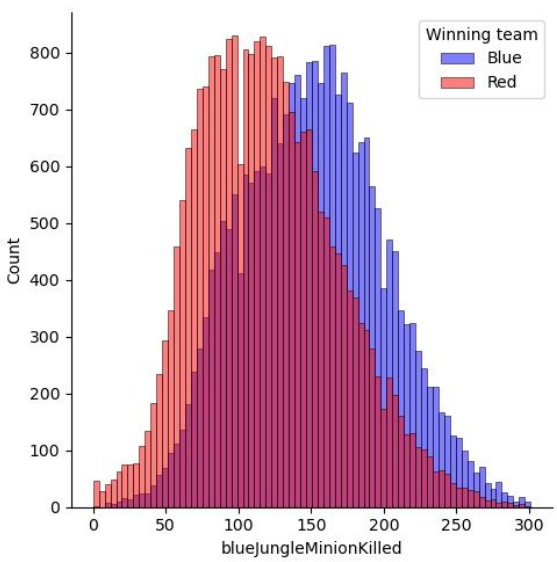


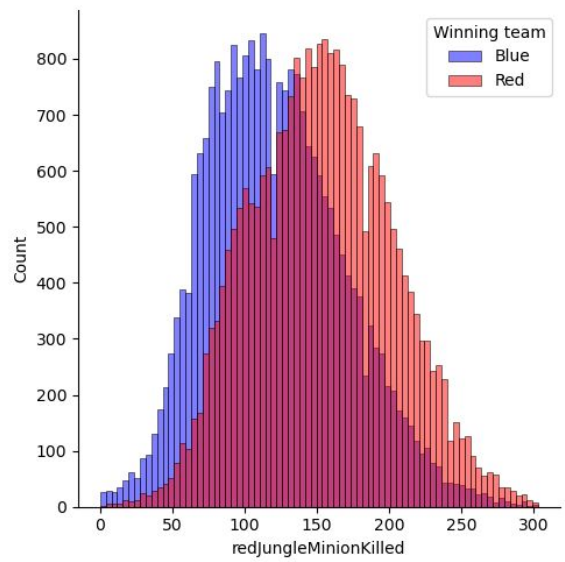

Winning team
Blue
Red
Count
blueWardKilled

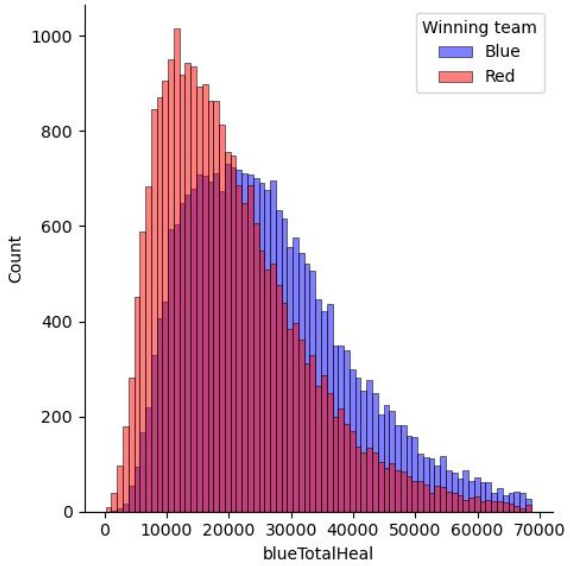

Winning team
Blue
Red
Count
redWardKilled

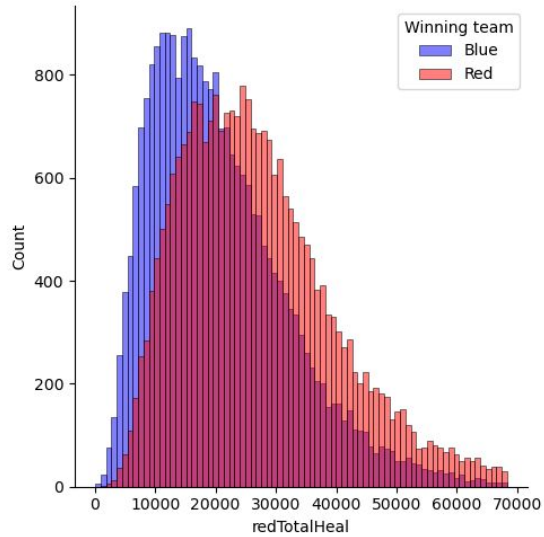

Winning team
Blue
Red
Count
blueTotalMinionKilled

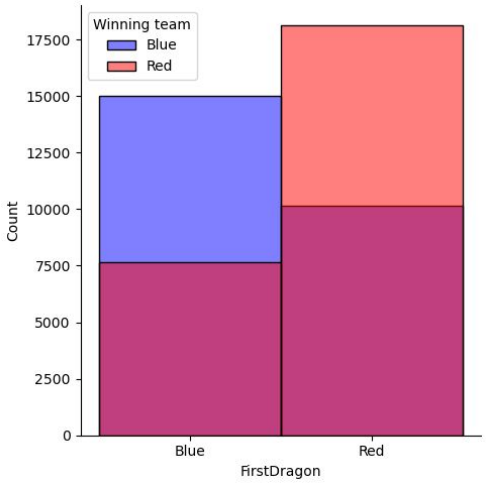

Winning team
Blue
Red
Count
redTotalMinionKilled

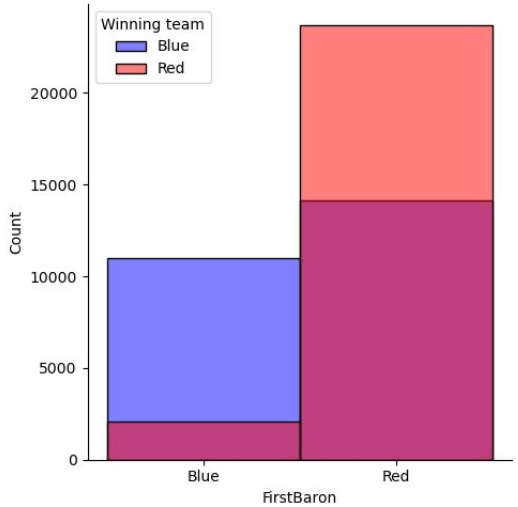

Winning team
Blue
Red
Count
blueJungleMinionKilled

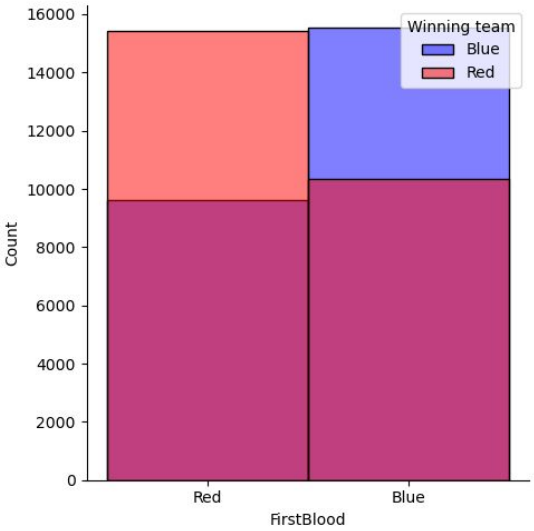

Winning team
Blue
Red
Count
redJungleMinionKilled

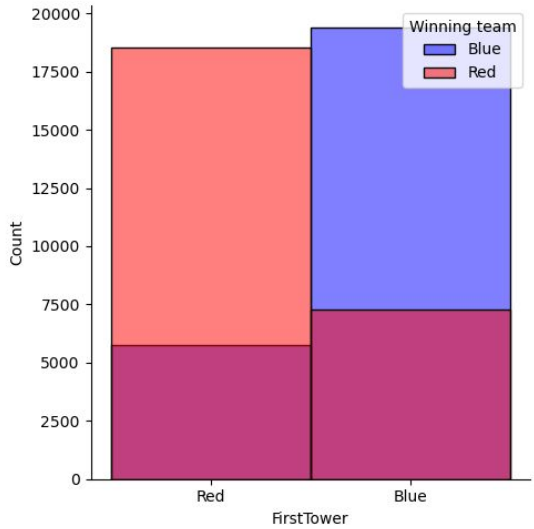

Winning team
Blue
Red
Count
blueTotalHeal

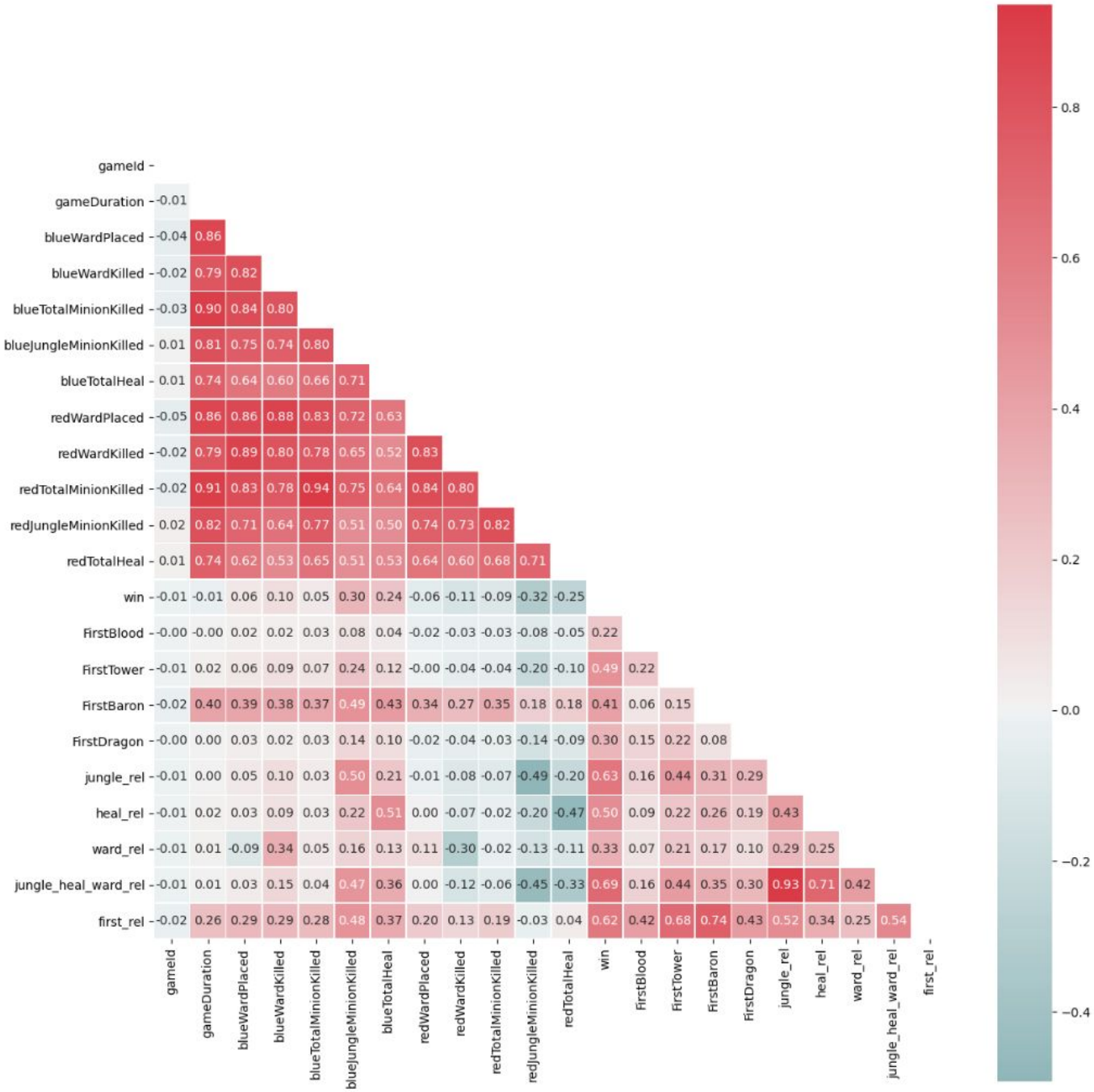

Winning team
Blue
Red
Count
redTotalHeal


Winning team
Blue
Red
Count
Blue
Red
FirstDragon


Winning team
Blue
Red
Count
Blue
Red
FirstBaron


Winning team
Blue
Red
Count
Red
Blue
FirstBlood


Winning team
Blue
Red
Count
Red
Blue
FirstTower

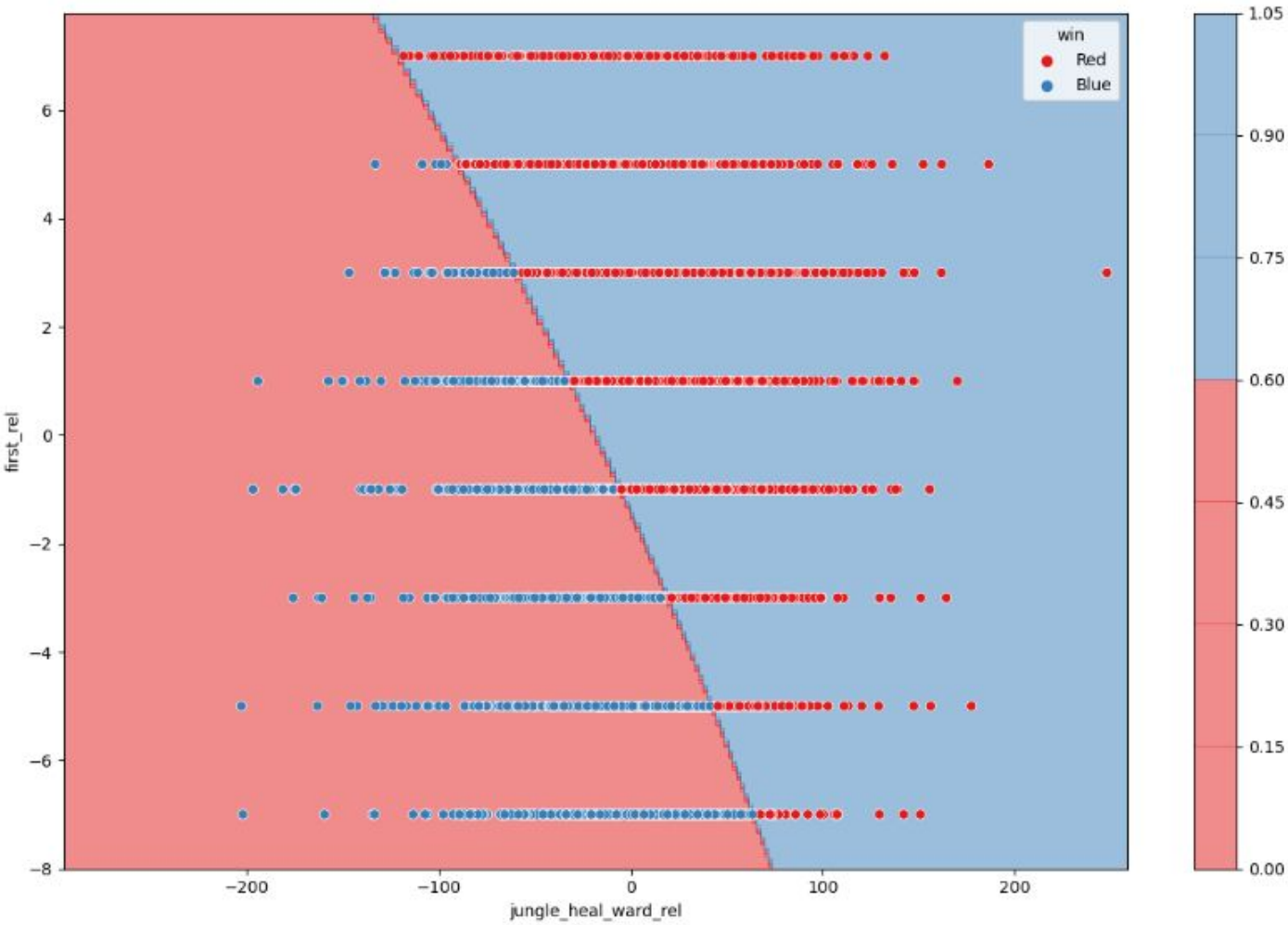


As shown above, all columns which could have been used in order to create better correlations have been used. As well, now we can see that the new columns have an even higher correlation with the predicted feature ‘win’ which is exactly what we wanted.

Classification:

1. The two features I choose to work with are ‘jungle_heal_ward_rel’ and ‘first_rel’ which are new columns I created.
   These features are most correlated with the ‘win’ column. Additionally, all other features have provided less accurate results.
   The naive bayes model is trained by 80% of the data which is approximately 41k rows.
   The accuracy score of these model using the two features stated about 88%.

```
jungle_heal_ward_rel & first_rel:
accuracy:  0.882104643172671
```



Displaying only wrong predictions as a scatter plot upon the contour plot displaying the decision regions.
It's interesting to see that the amount of miss predicted values for each ‘first_rel’ value is about the same, with a difference in the percentage of miss predictions between (red / blue).

2. A. **baseline decision tree**:
   When creating the base line tree classifier, I used the original dataset and dropped all missing values. (training 80%, test 20%)

	precision	recall	f1-score	support
Red	0.82	0.83	0.83	2709
Blue	0.83	0.82	0.82	2714
accuracy			0.82	5423
macro avg	0.82	0.82	0.82	5423
weighted avg	0.82	0.82	0.82	5423

   The score above was achieved without limiting the tree's depth.
   When I tried to limit the tree to various maximum depths, I found out that the best accuracy was achieved at max depth of 6 - 11, in this case the accuracy was 85%.

	precision	recall	f1-score	support
Red	0.84	0.85	0.85	2709
Blue	0.85	0.84	0.84	2714
accuracy			0.85	5423
macro avg	0.85	0.85	0.85	5423
weighted avg	0.85	0.85	0.85	5423

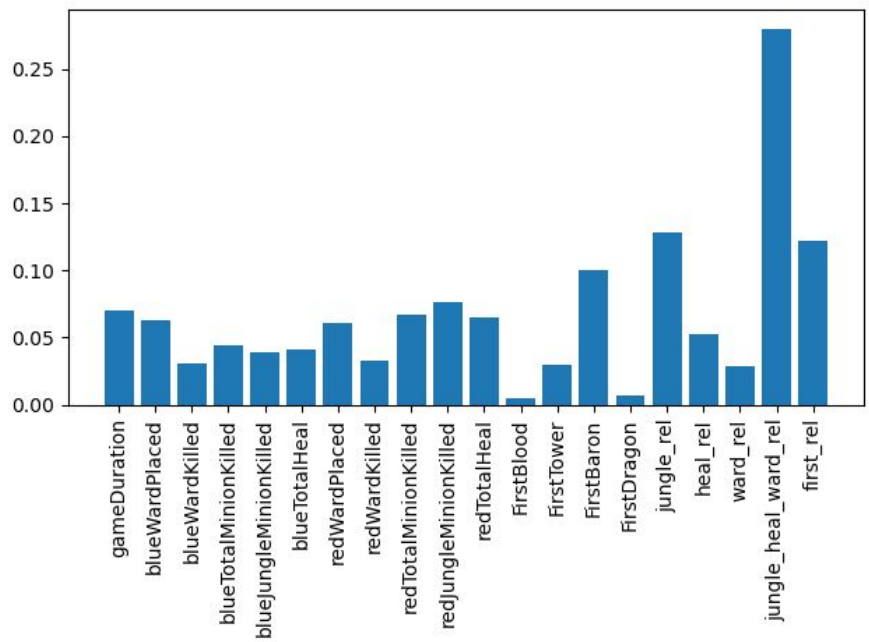
B. **manipulated data decision tree:**

In order to pick the most relevant features, i will first try to build a tree based on all of the features,then i will use the permutation importance in order to decide which features will be used at the final tree.

(training 80%, test 20%)

	precision	recall	f1-score	support
Red	0.85	0.85	0.85	5178
Blue	0.84	0.84	0.84	5009
accuracy			0.85	10187
macro avg	0.85	0.85	0.85	10187
weighted avg	0.85	0.85	0.85	10187

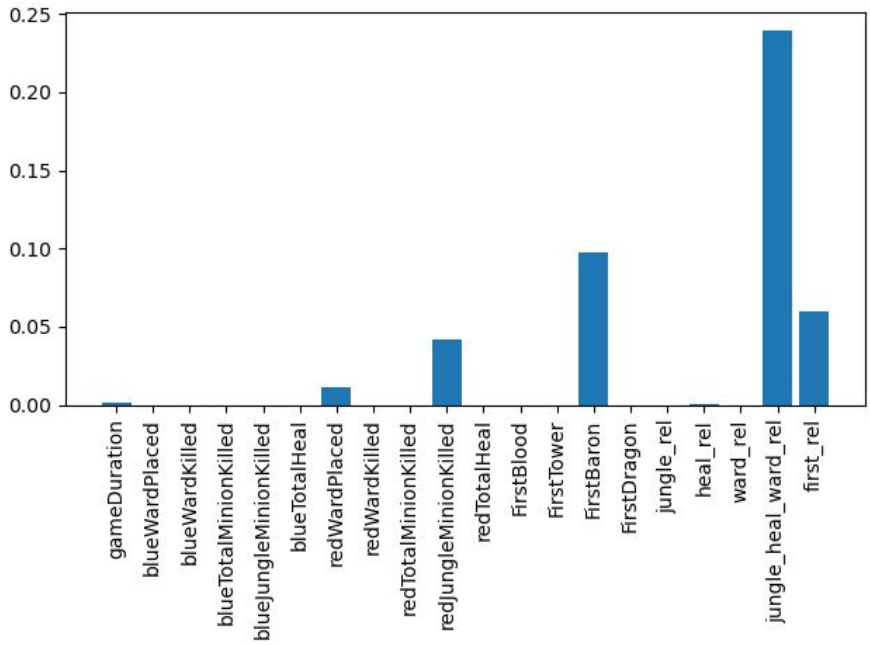
When creating a tree classifier based on the manipulated data without any limitations i received 85% accuracy. Next , i will decide which columns will be used.



From this plot, i cant really decide which are the top 3-4 features i want to use. So first i will limit the trees depth to 5, this way, only highly important features will be used.

After limiting the max depth to 5 i received a new tree with accuracy score of 89%.

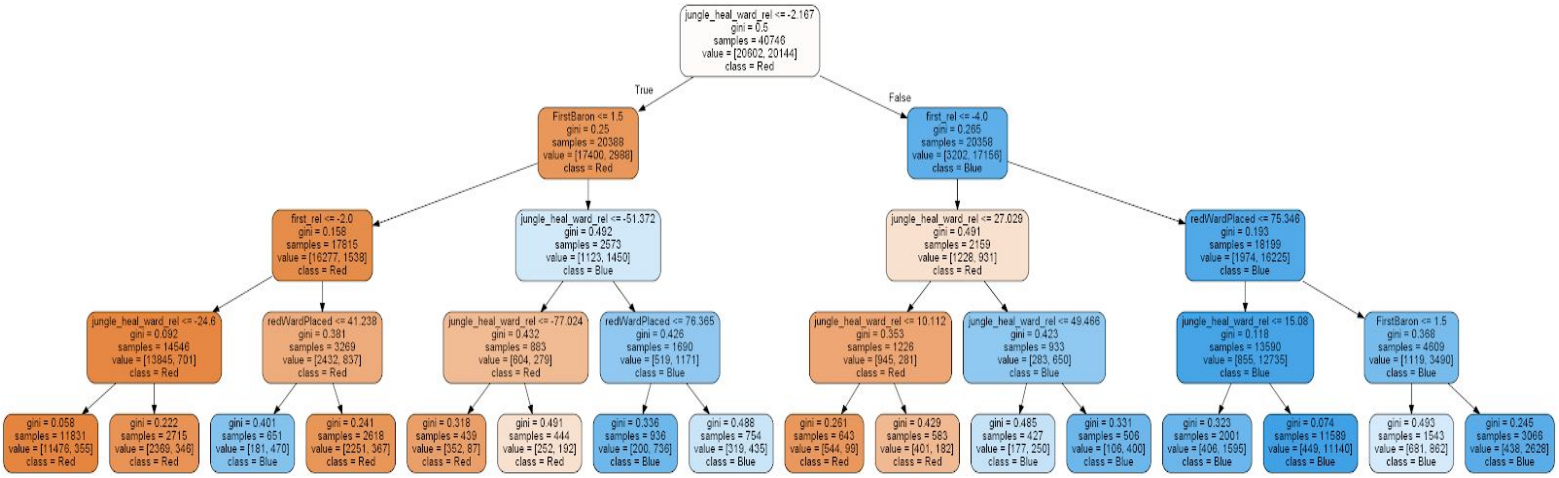
	precision	recall	f1-score	support
Red	0.90	0.89	0.90	5178
Blue	0.89	0.90	0.89	5009
accuracy			0.89	10187
macro avg	0.89	0.89	0.89	10187
weighted avg	0.89	0.89	0.89	10187



From this plot it's clear that most relevant features are ‘jungle_heal_ward_rel’, ‘first_rel’ ‘FirstBaron’ and ‘redWardPlaced’. Additionally, since ‘redJungleMinionKilled’ is part of the relation columns so i decided not to use it.
Sure enough we are able to get great results based only on four columns and with the max depth of 4.
Tree visualization:

	precision	recall	f1-score	support
Red	0.90	0.89	0.90	5178
Blue	0.89	0.90	0.89	5009
accuracy			0.89	10187
macro avg	0.89	0.89	0.89	10187
weighted avg	0.89	0.89	0.89	10187

The final tree classifier accuracy is 89% using 4 features and the max depth of 4. In this case its an improvement at both tree size and accuracy compared to the original baseline tree.



```
|--- jungle_heal_ward_rel <= -2.17
|   |--- FirstBaron <= 1.50
|   |   |--- first_rel <= -2.00
|   |   |   |--- jungle_heal_ward_rel <= -24.60
|   |   |   |   |--- class: Red
|   |   |   |--- jungle_heal_ward_rel >  -24.60
|   |   |   |   |--- class: Red
|   |   |--- first_rel >  -2.00
|   |   |   |--- redWardPlaced <= 41.24
|   |   |   |   |--- class: Blue
|   |   |   |--- redWardPlaced >  41.24
|   |   |   |   |--- class: Red
|   |--- FirstBaron >  1.50
|   |   |--- jungle_heal_ward_rel <= -51.37
|   |   |   |--- jungle_heal_ward_rel <= -77.02
|   |   |   |   |--- class: Red
|   |   |   |--- jungle_heal_ward_rel >  -77.02
|   |   |   |   |--- class: Red
|   |   |--- jungle_heal_ward_rel >  -51.37
|   |   |   |--- redWardPlaced <= 76.36
|   |   |   |   |--- class: Blue
|   |   |   |--- redWardPlaced >  76.36
|   |   |   |   |--- class: Blue
|--- jungle_heal_ward_rel >  -2.17
|   |--- first_rel <= -4.00
|   |   |--- jungle_heal_ward_rel <= 27.03
|   |   |   |--- jungle_heal_ward_rel <= 10.11
|   |   |   |   |--- class: Red
|   |   |   |--- jungle_heal_ward_rel >  10.11
|   |   |   |   |--- class: Red
|   |   |--- jungle_heal_ward_rel >  27.03
|   |   |   |--- jungle_heal_ward_rel <= 49.47
|   |   |   |   |--- class: Blue
|   |   |   |--- jungle_heal_ward_rel >  49.47
|   |   |   |   |--- class: Blue
|   |--- first_rel >  -4.00
|   |   |--- redWardPlaced <= 75.35
|   |   |   |--- jungle_heal_ward_rel <= 15.08
|   |   |   |   |--- class: Blue
|   |   |   |--- jungle_heal_ward_rel >  15.08
|   |   |   |   |--- class: Blue
|   |   |--- redWardPlaced >  75.35
|   |   |   |--- FirstBaron <= 1.50
|   |   |   |   |--- class: Blue
|   |   |   |--- FirstBaron >  1.50
|   |   |   |   |--- class: Blue
```