

League of Legends Diamond Ranked Games (10 min) - DM Report PART-1

Introduction	1
Description of our dataset	1
Definitions / Glossary:	1
Data Mining goals and success criteria:	1
Characteristics of our data set	2
Description of the attributes	2
Exploratory data analysis (EDA)	3
Distribution and boxplot of our dataset	3
Distributions founded:	12
Correlation between attribute values (HeatMap):	14
Preliminary findings about the contents of the datasets:	16
Discussion of data quality:	16
Missing data:	16
Outliers:	16
Overall data integrity:	18
Entity integrity	18
Domain integrity	18
Values of unknown meaning:	18
Conclusion	18
Webgraphy / Bibliography	19
Appendix I - Python's code used.	20
Get our covariance matrix:	20
Get the boxplot for each attribute:	20
Get the histogram and displot (distribution plot):	20
Identify your data distribution with chi-square method	21

Introduction

In this report, we have tried to show the process using new technologies outside that are used in the business sector, like Python and their Machine Learning libraries. In this time, we tried to learn different ways to do the same things to enrich ourselves for our professional future.

Description of our dataset

The data set is related to the game called "League of Legends". League of Legends is a multiplayer online battle where 2 teams face off. There are 3 lanes, a jungle, and 5 roles. The goal is to take down the enemy Nexus to win the game.

The dataset contains the first 10 min of data in each game and also who won that game. Players have roughly the same level.

There are 19 features per team (38 in total) collected after 10min in-game. This includes kills, deaths, gold, experience, level...The target value is column `blueWins`. A value of 1 means the blue team has won, 0 otherwise.

Definitions / Glossary:

1. Warding totem: An item that a player can put on the map to reveal the nearby area. Very useful for map/objectives control.
2. Elite monsters: Monsters with high hp/damage that give a massive bonus (gold/XP/stats) when killed by a team.
3. Dragons: Elite monster which gives team bonus when killed. The 4th dragon killed by a team gives a massive stats bonus. The 5th dragon (Elder Dragon) offers a huge advantage to the team.
4. Herald: Elite monster which gives stats bonus when killed by the player. It helps to push a lane and destroys structures.
5. Towers: Structures you have to destroy to reach the enemy Nexus. They give gold.
6. Minions: NPC that belong to both teams. They give gold when killed by players.
7. Level: Champion level. Start at 1. Max is 18.
8. Jungle minions: NPC that belong to NO TEAM. They give gold and buffs when killed by players.

Data Mining goals and success criteria:

The primary goal are prediction of the target column, in this case `blueWins`, with the use of some variables or fields in the database to predict unknown or future values of other variables of interest. Also, describing the data set as a whole by determining global characteristics and dividing examples into groups.

In case of success criteria, relevancy of the data sources to avoid duplicates and unimportant results and completeness of the data to ensure all the essential information is covered.

Characteristics of our data set

The origin of the data set is from League of Legend, using Riot Games API. It was taken from the website [kaggle.com](https://www.kaggle.com). The dataset is stored in a .csv file which contains only one table with 9879 samples, every sample is from the first 10 minutes of a play. Also we have the team that finally won that match, the variable `blueWins`.

In our dataset we can use more columns using the Riot Games API, like for example the win streak ratio from everybody or also the Hero (champion selected) from each player, but the meta is changing every week, and some of the powerful Heroes today will be powerless tomorrow and vice versa. With the data set we are going to work with, we will be more independent of future metagame changes.

Description of the attributes

We have two types of attributes on our dataset, the first group is the type nominal, concretely booleans:

- **blueWins: The target column.** {1 if the blue team has won, 0 otherwise}

Target variables are the dependent variables in the model.

In [regression](#) problems, targets are continuous variables (power consumption, product quality...).

In [classification](#) problems, targets are binary (fault, churn...) or categorical (type of object, activity...). In this type of applications, targets are also-called categories or labels.

- **blueFirstBlood:** First kill of the game. {1 if the blue team did the first kill, 0 otherwise}

The rest of our attributes are numerical type, mostly of them Integer, except the following ones, that are decimal values:

- `redAvgLevel`, `redCSPerMin`, `redGoldPerMin`
`blueAvgLevel`, `blueCSPerMin`, `blueGoldPerMin`

Looking the attributes, we have to clarify the next concepts in case the reader has any doubts:

- **AvgLevel:** As we said before, level means the level of our champion in the game, it started at 1 and can be until 18 depending of the experience earned. The average consist in a simple average between the 5 members of the team.

- Gold: It is the money during the game. Killing minion or enemy players during the game the players will earn gold that they will use to buy equipment later to improve their champion during the game. The more money, the more items can be bought and the player will have a certain advantage. The unit used is as follows: "Blue team has 630 of money per minute".

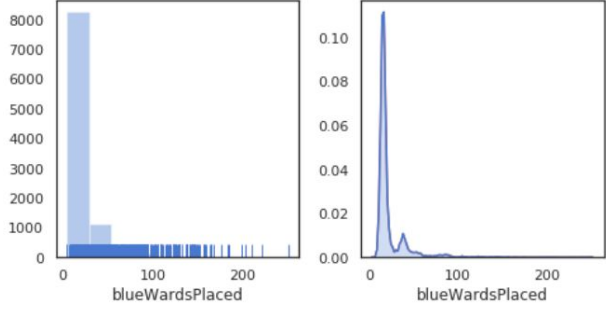
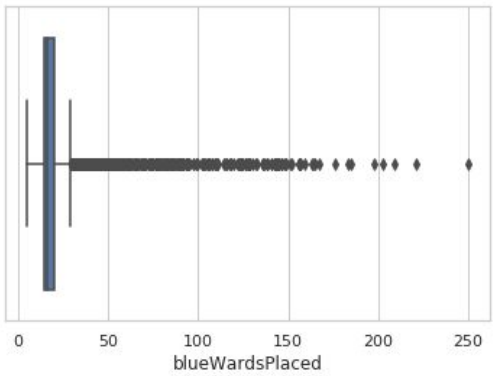
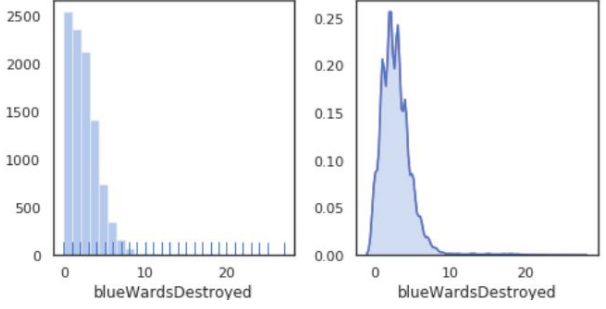
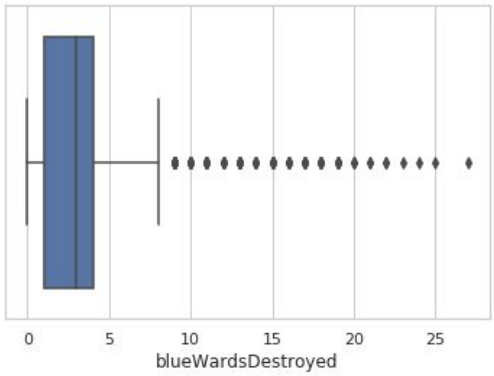
Exploratory data analysis (EDA)

Distribution and boxplot of our dataset

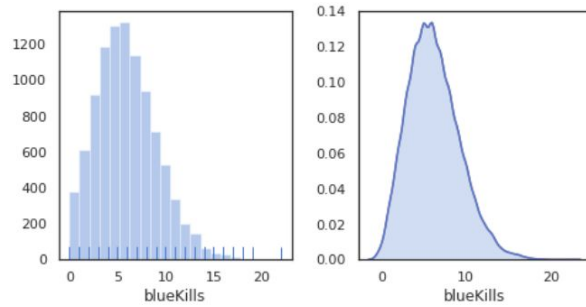
Distribution: show how the data is distributed over its entire range. If the data is very irregularly distributed, the resulting model will probably be of bad quality. Describes all the possible values and likelihoods that a random variable can take within a given range.

To complement our judgement we used an algorithm based on the chi-square method to try the best fit to the most famous distribution. [See appendix to watch it deeply]

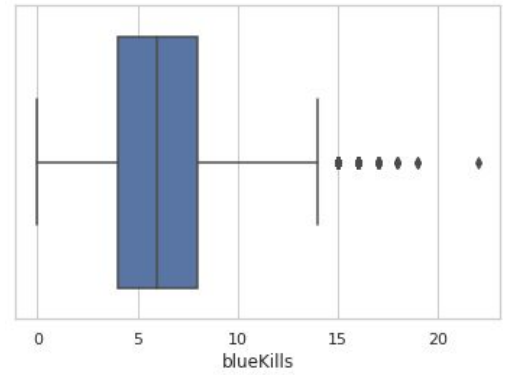
The outliers will be discussed later, but we saw convenient show them in the following table with the histogram. Also remember that both of them are a good way to see the outliers.

Attribute	Histogram and distribution	Boxplot with outliers values
blueWardsPlaced: Number of warding totems ¹ placed by the blue team on the map.	 <p>Distribution: Logarithmic / Inverse Gaussian The top values are next to each other in the range (10-20). After that they are very dispersed.</p>	
blueWardsDestroyed: Number of enemy warding totems the blue team has destroyed.	 <p>Distribution: Normal Most of the values are from 1 to 4. Since 20 until 27 they are one by one, with 26 with zero times.</p>	

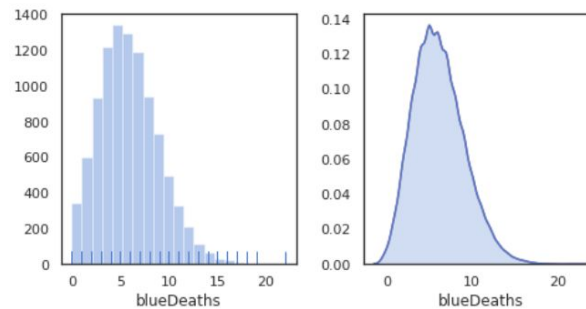
blueKills:
Number of enemies
killed by the blue team.



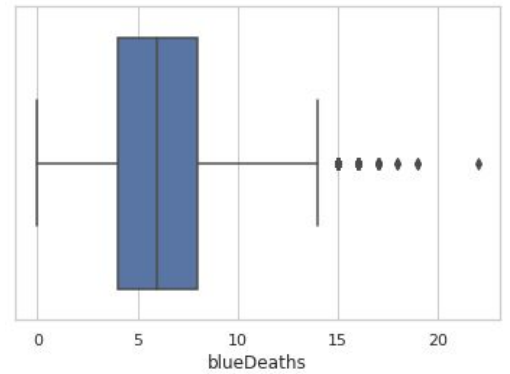
Distribution: Normal
4 to 7 are the most common values. 1 kill is
almost 5 times more probably than 0.



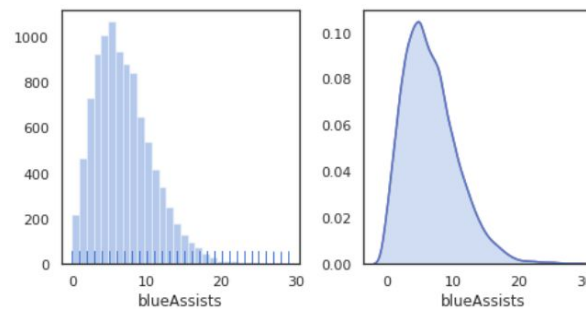
blueDeaths:
Number of deaths (blue
team).



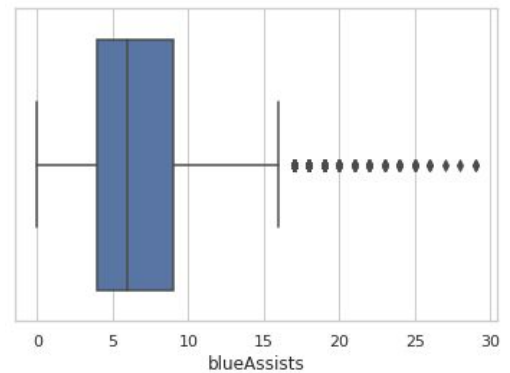
Distribution: Normal
Distribution very similar to blueKills



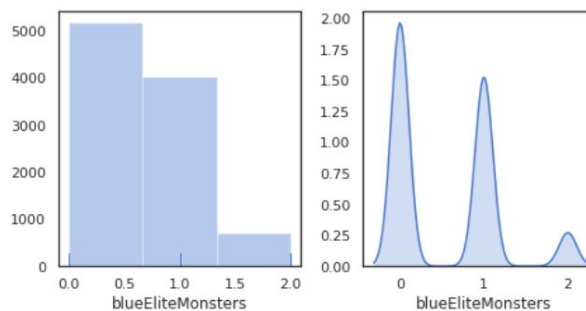
blueAssists:
Number of kill assists
(blue team).



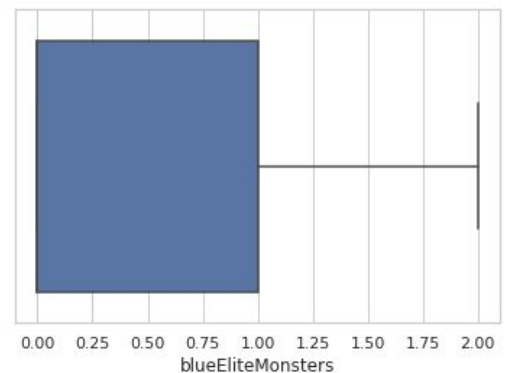
Distribution: weibull_max
This time 1 assist is double of probably than 0.



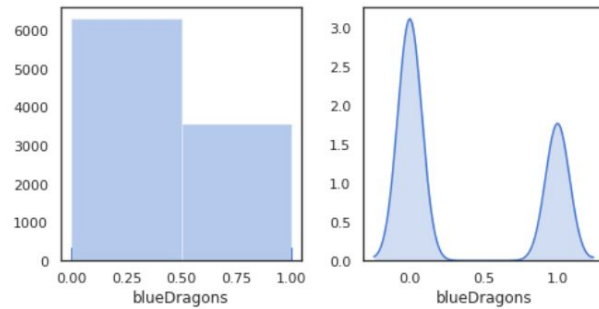
blueEliteMonsters:
Number of elite
monsters² killed by the
blue team (Dragons
and Heralds).



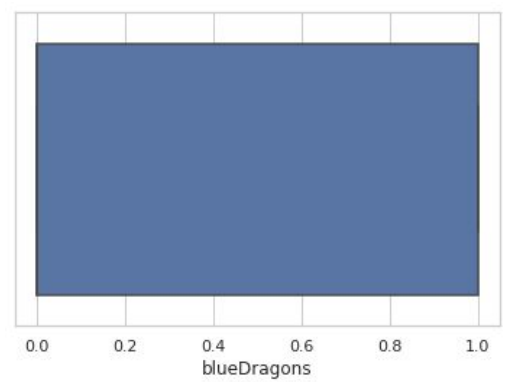
Distribution:
Kill all the monsters is 7 times than 0.



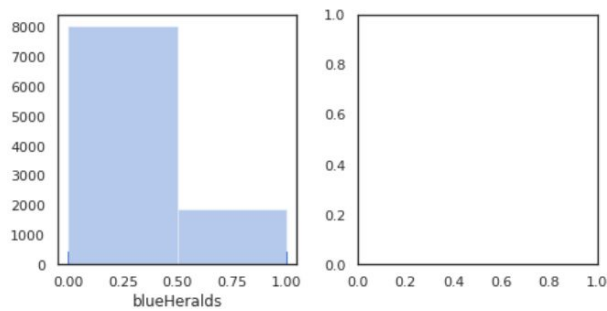
blueDragons:
Number of dragons³
killed by the blue team.



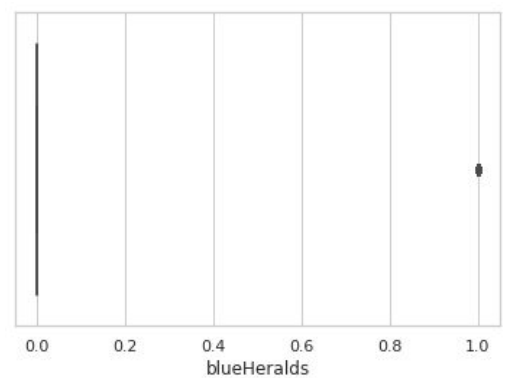
Distribution:
0 dragon is almost double of possibilities than 1 dragon.



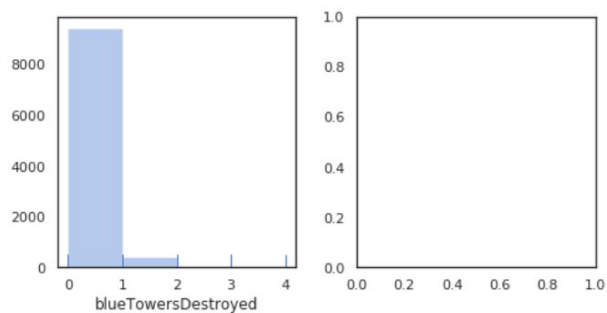
blueHeralds:
Number of heralds⁴
killed by the blue team.



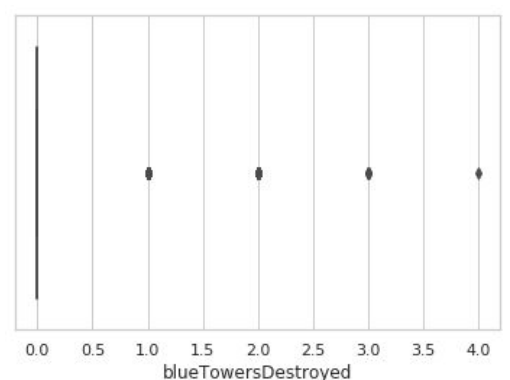
Distribution:
0 Heralds is 4 times more probably than 1. As we can see is very difficult to do it in the first 10 minutes, is a Monster very protected for the other team.



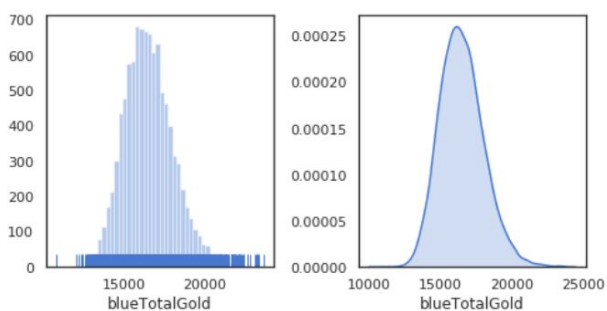
blueTowersDestroyed:
Number of structures
destroyed by the blue
team (towers⁵ ...).



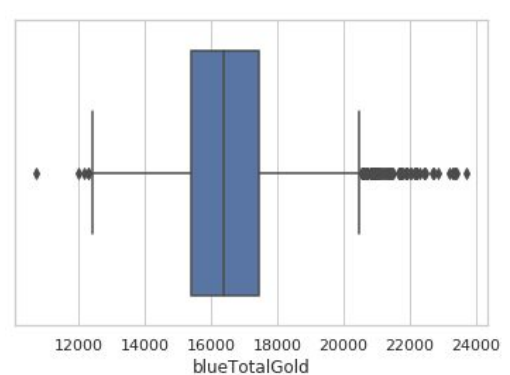
Distribution:
0 towers is almost 22 times more possible than 1, and 3 towers only happened 7 times.



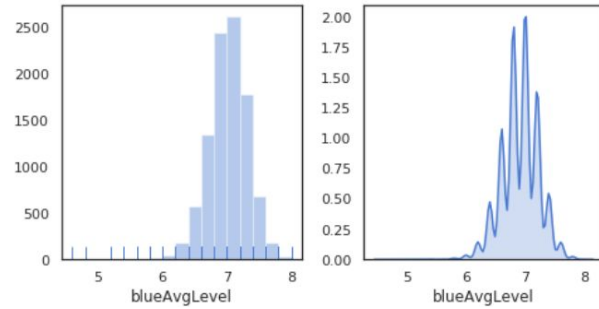
blueTotalGold:
Blue team total gold.



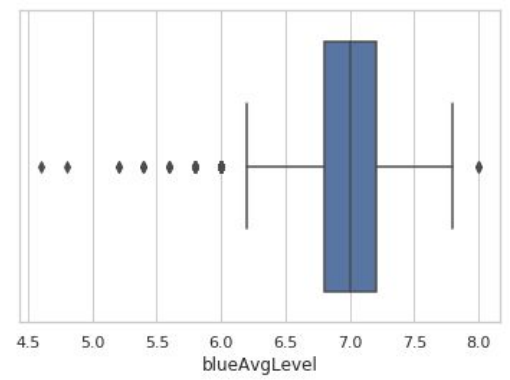
Distribution: Gamma/Pearson
Most of the values 15500 until 17000.



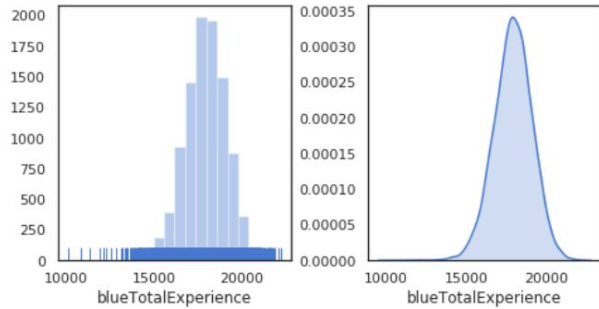
blueAvgLevel:
Blue team average
champion level.



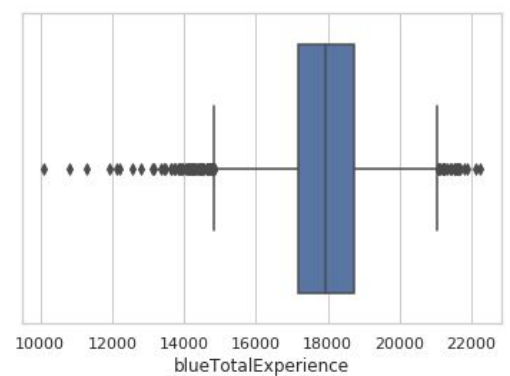
Distribution: Beta, weibull_min
Most of the values are very close to 7.



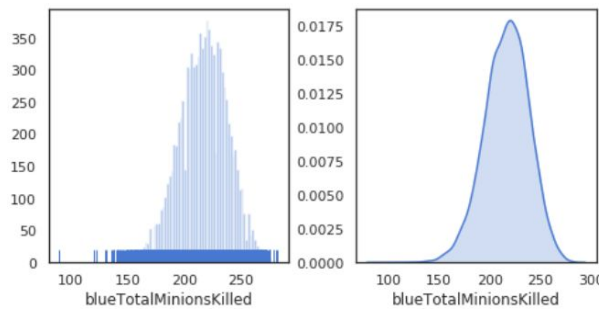
blueTotalExperience:
Blue team total
experience.



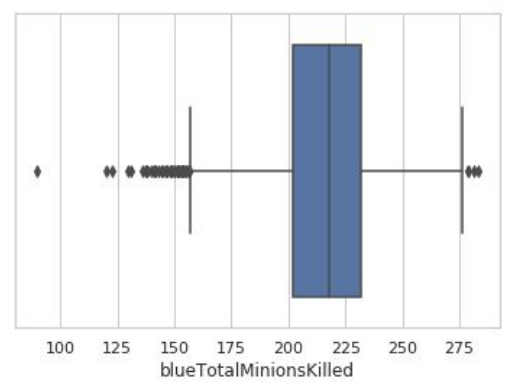
Distribution: Beta / Normal
Most of the values close to 18000.



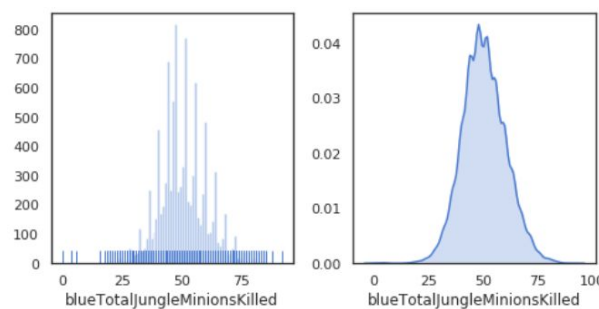
**blueTotalMinions
Killed:**
Blue team total
minions⁶ killed (CS).



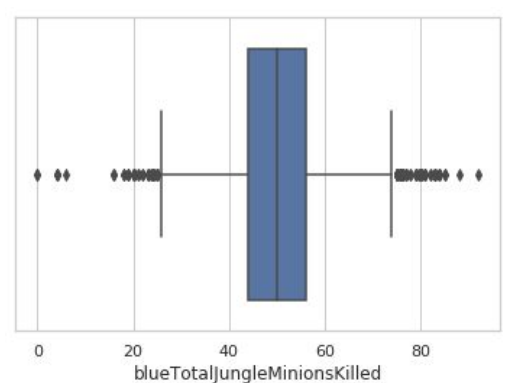
Distribution: Beta / Weibull_min
Most of the values between 228 to 222.



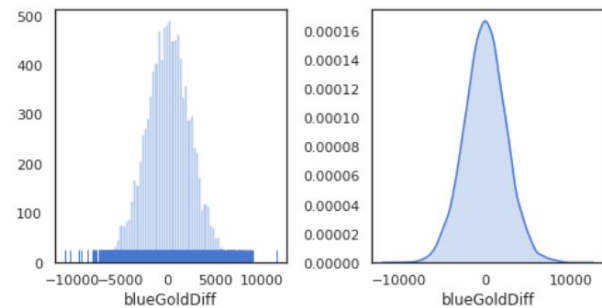
**blueTotalJungle
MinionsKilled:**
Blue team total jungle
monsters killed.



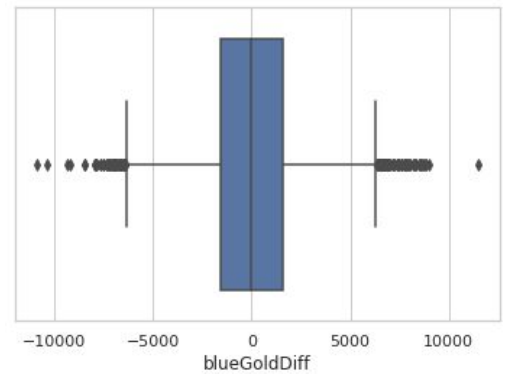
Distribution: Normal / Beta
The most probably value is 48 with 816 times, but
47 has more than 2 times less and 48 less than 3



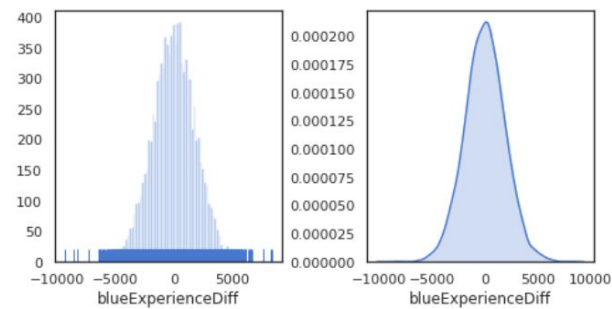
blueGoldDiff:
Blue team gold difference compared to the enemy team.



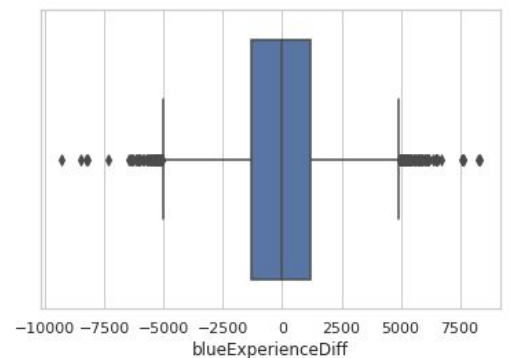
Distribution: Normal / Pearson3
Most of the values from between -1000 to +1000



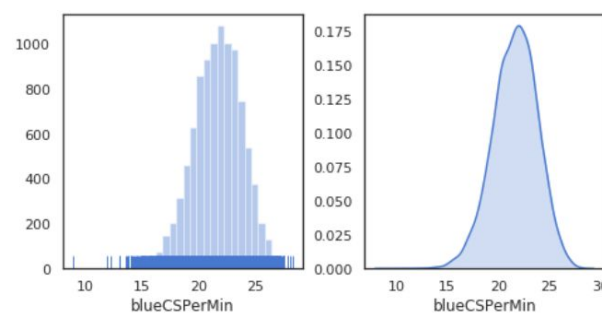
blueExperienceDiff:
Blue team experience difference compared to the enemy team.



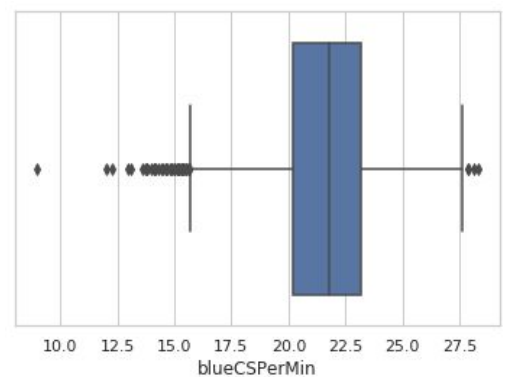
Distribution: Normal / Pearson3 / Gamma
Same as Gold difference, they follow the same distribution and are very correlated.



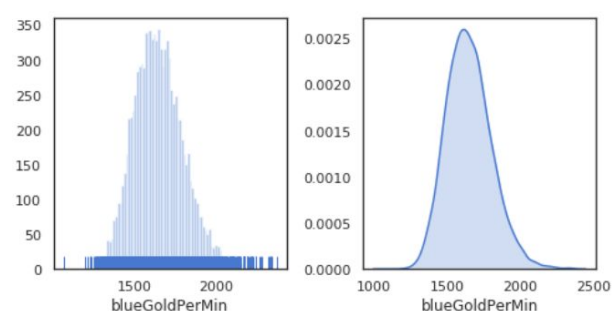
blueCSPerMin:
Blue team CS (minions) per minute.



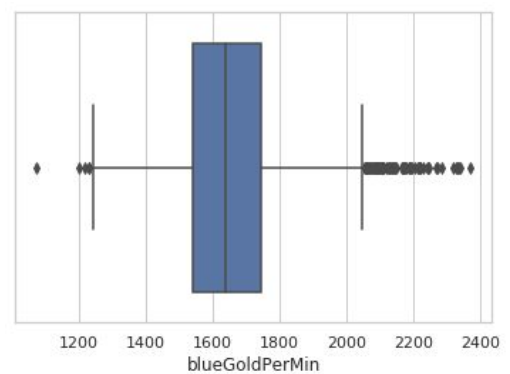
Distribution: Beta
Similar distribution of data than blueTotalMinionsKilled because is divided by 10.

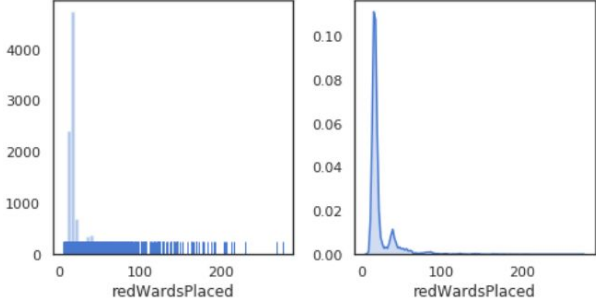
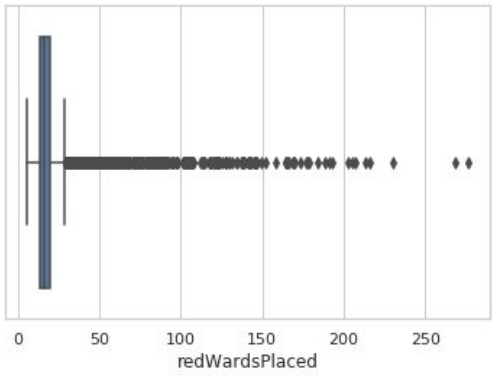
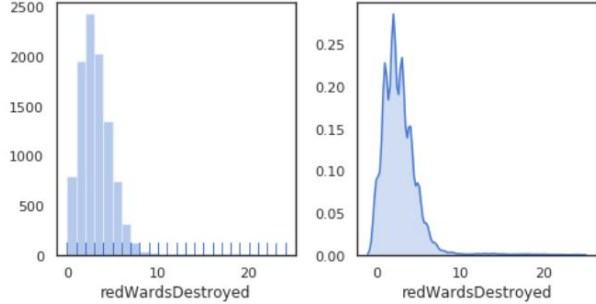
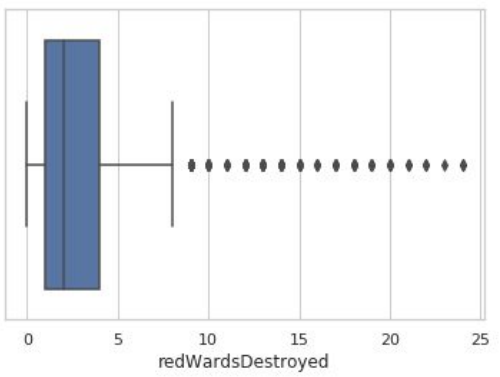
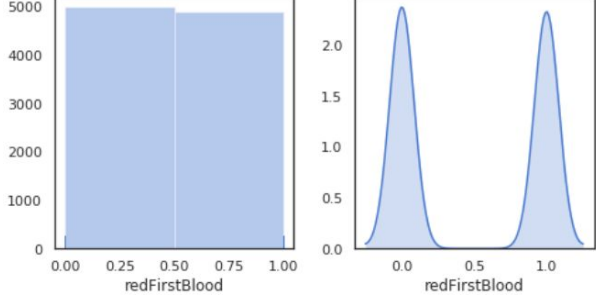
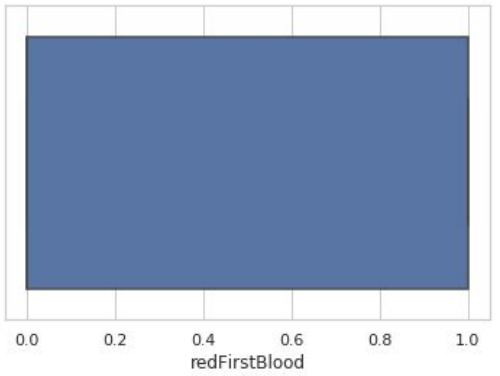
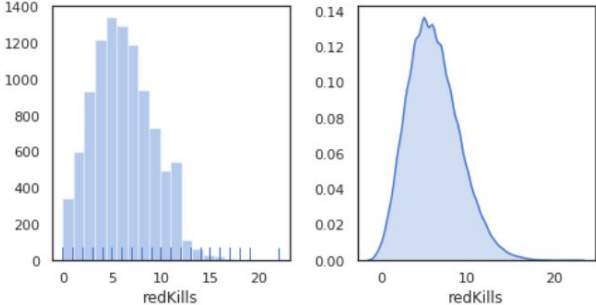
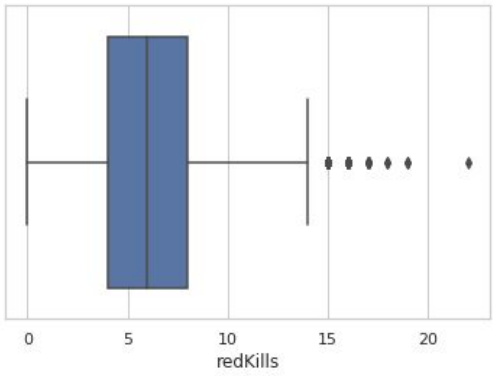


blueGoldPerMin:
Blue team gold per minute.

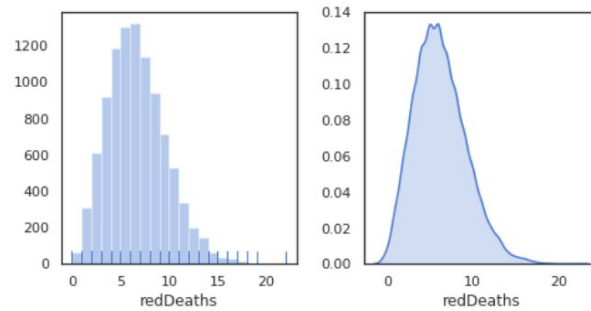


Distribution: Lognorm / Pearson
Similar distribution of data than blueTotalGold because is divided by 10.

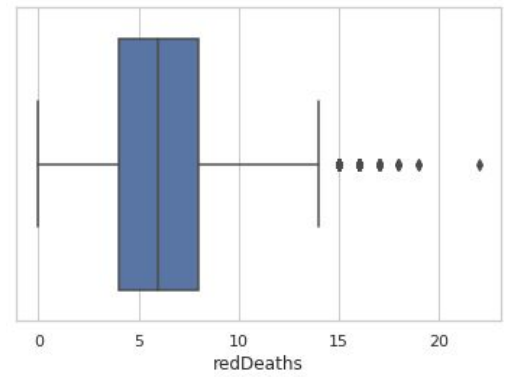


<p>redWardsPlaced: Number of warding totems placed by the red team on the map.</p>	 <p>Distribution: Lognorm / InvGauss The top values are next to each other in the range (10-20). After that they are very dispersed.</p>	
<p>redWardsDestroyed: Number of enemy warding totems the red team has destroyed.</p>	 <p>Distribution: Normal Most of the values are from 1 to 4. 23 is the only attribute with 1 occurrence.</p>	
<p>redFirstBlood: First kill of the game. 1 if the red team did the first kill, 0 otherwise.</p>	 <p>Distribution: Binomial</p>	
<p>redKills: Number of enemies killed by the red team.</p>	 <p>Distribution: Normal 4 to 5 are the most common values. 1 kill is almost 4 times more probably than 0.</p>	

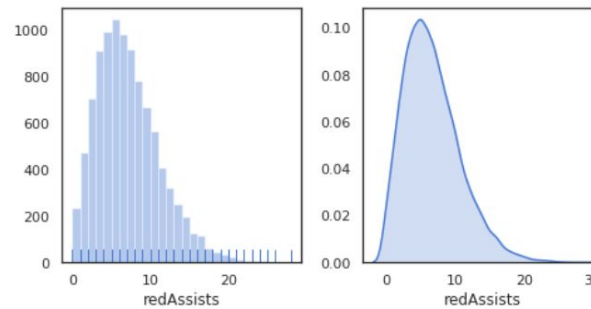
redDeaths:
Number of deaths (red team).



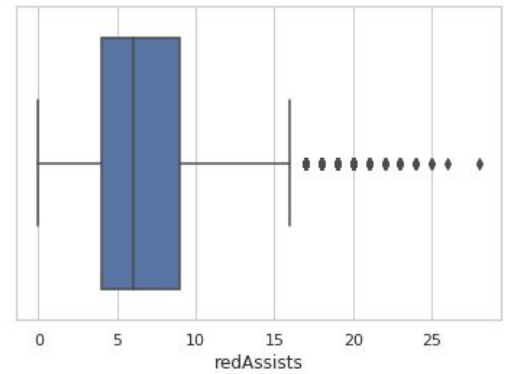
Distribution: Normal
Distribution very similar to redKills



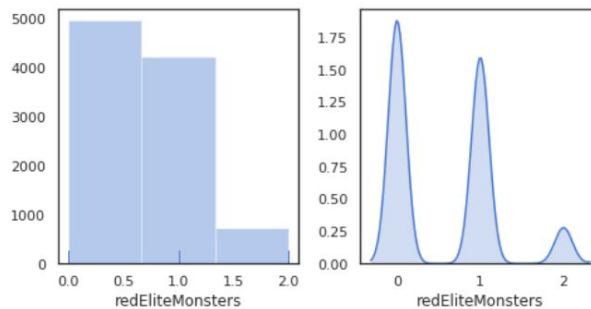
redAssists:
Number of kill assists (red team).



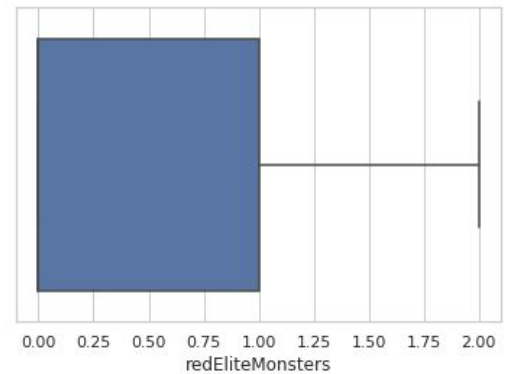
Distribution: Weibull_max / logarithm
This time 1 assist is double of probably than 0.



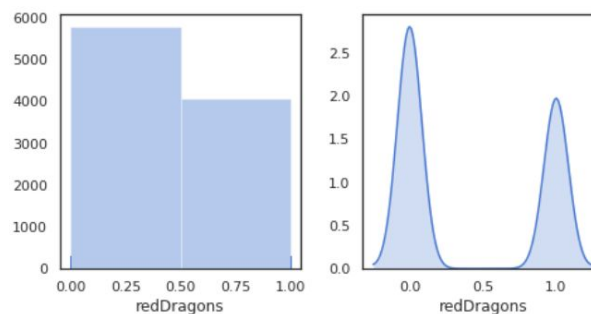
redEliteMonsters:
Number of elite monsters killed by the red team (Dragons and Heralds).



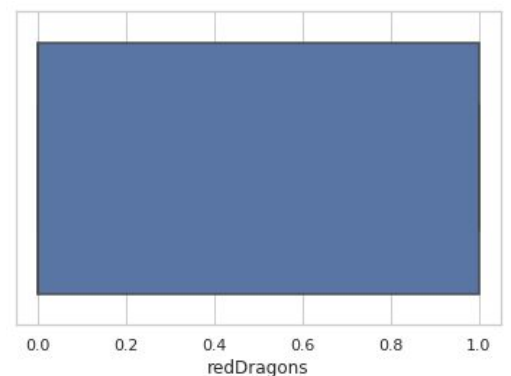
Distribution:
Kill all the monsters is 7 times than 0.

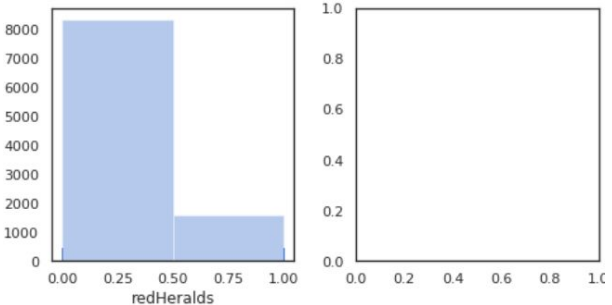
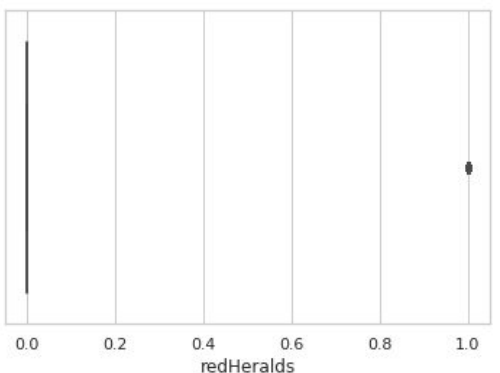
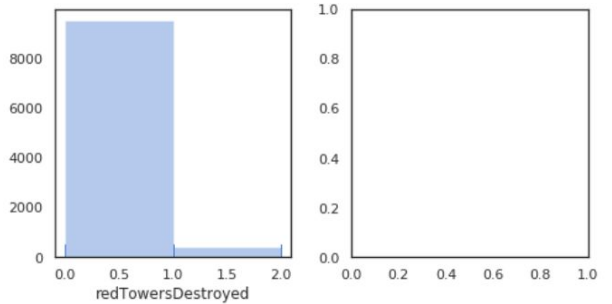
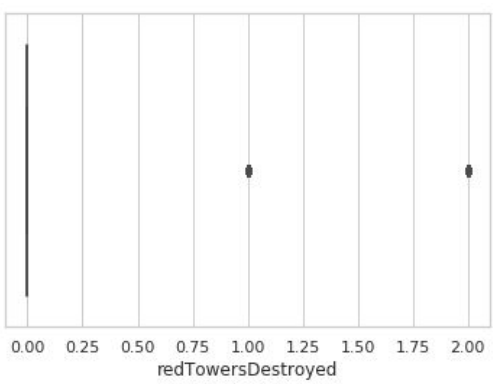
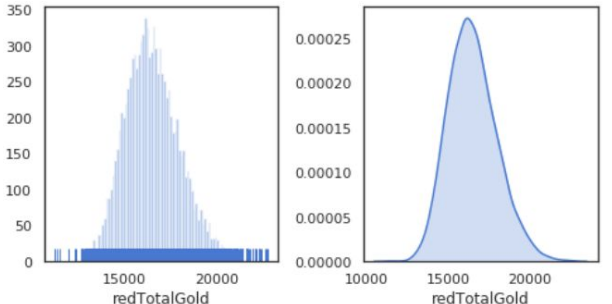
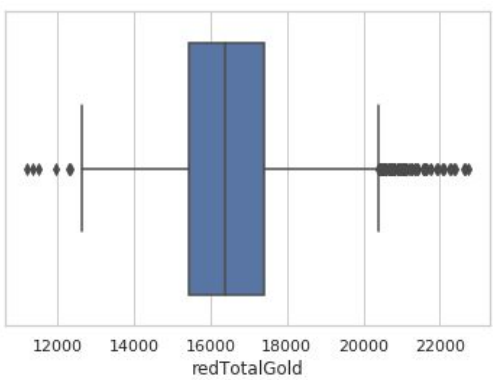
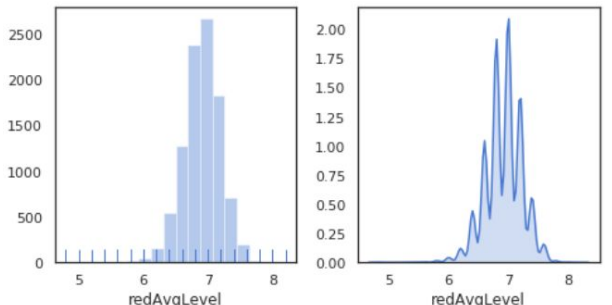
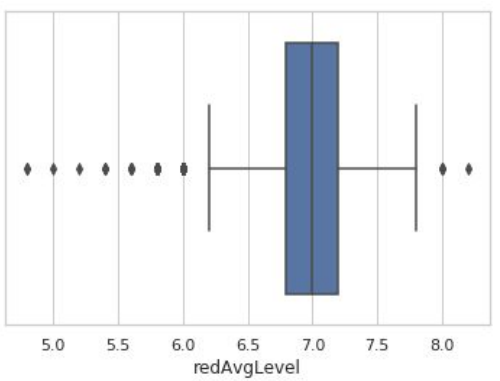


redDragons:
Number of dragons killed by the red team.

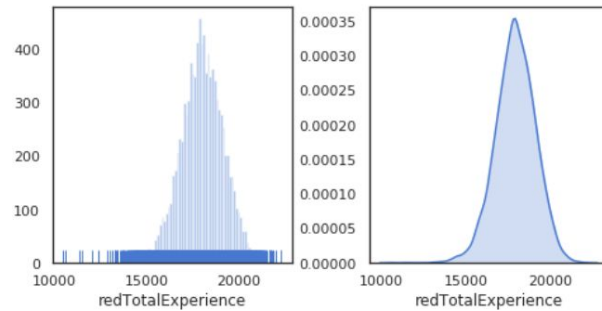


Distribution:
In this case 0 is not the almost of possibilities than 1(it happened in blueDragons), this time is around 40%

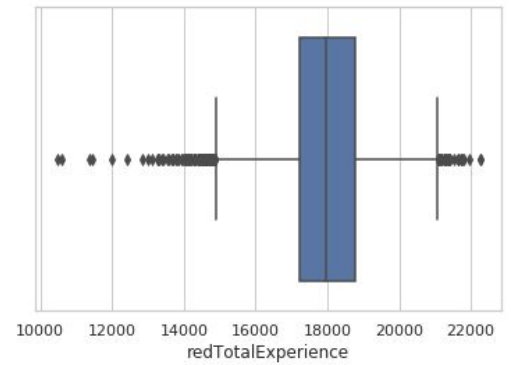


<p>redHeralds: Number of heralds killed by the red team.</p>	 <p>Distribution: 0 Heralds is 5 times more probably than 1. As we can see is very difficult to do it in the first 10 minutes, is a Monster very protected for the other team.</p>	
<p>redTowersDestroyed: Number of structures destroyed by the red team (towers...).</p>	 <p>Distribution: This time there is no 3 or 4 towers destroyed (it happened in blueTowersDestroyed)</p>	
<p>redTotalGold: Red team total gold.</p>	 <p>Distribution: Beta /Gamma/Pearson3 Most of the values 15500 until 17000.</p>	
<p>redAvgLevel: Red team average champion level⁷. The Champion level: Start at 1. Max is 18.</p>	 <p>Distribution: Beta Most of the values are very close to 7.</p>	

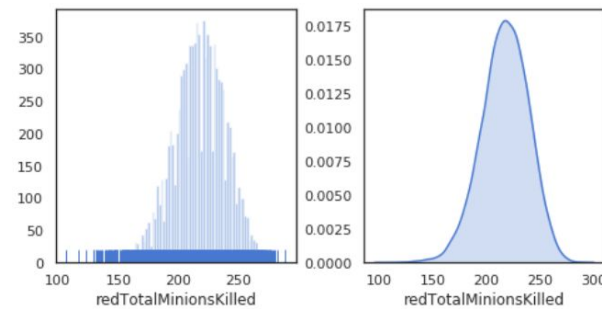
redTotalExperience:
Red team total
experience.



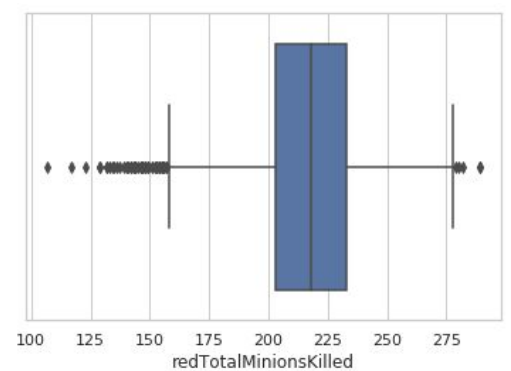
Distribution: Beta
Most of the values close to 18000.



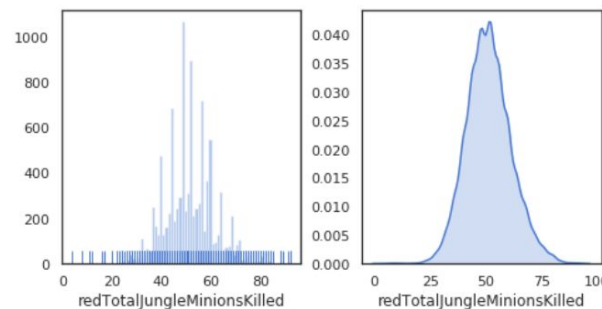
redTotalMinions
Killed: Red team
total minions killed
(CS).



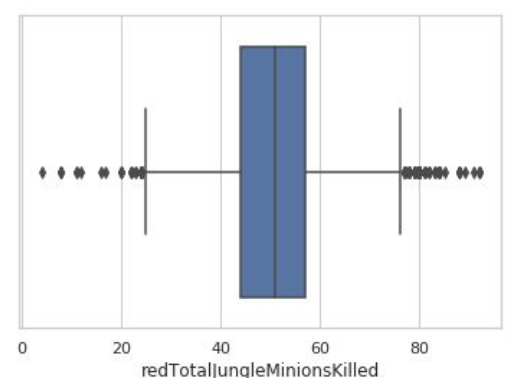
Distribution: Beta
Most of the values between 215 to 220, instead
of 228 to 222.



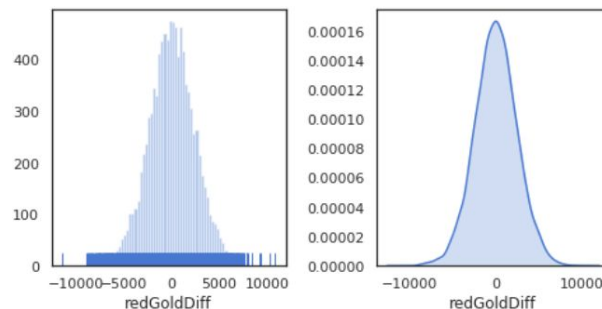
redTotalJungle
MinionsKilled:
Red team total jungle
monsters killed.



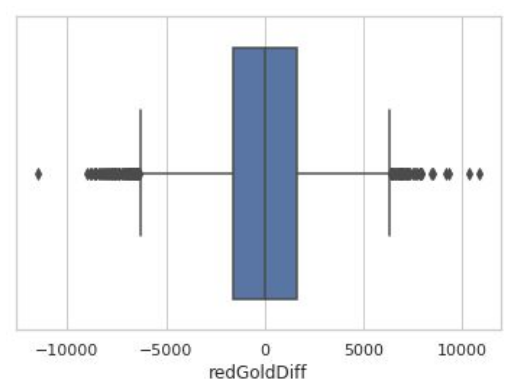
Distribution: Normal
The most probably value is 52 with 894 times, but
53 has more than 4 times less.

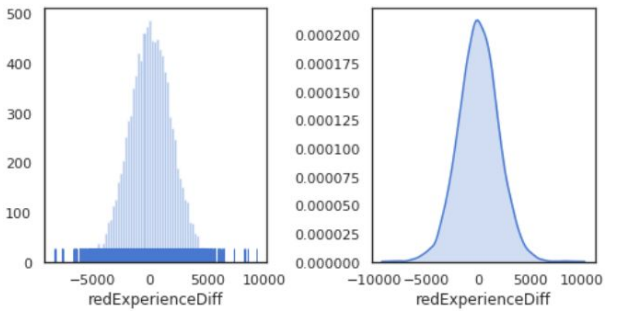
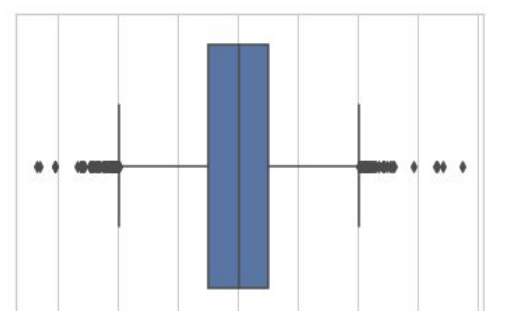
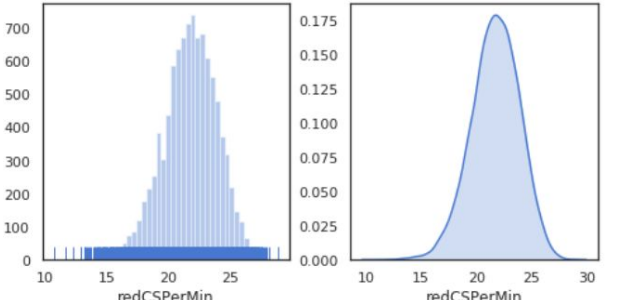
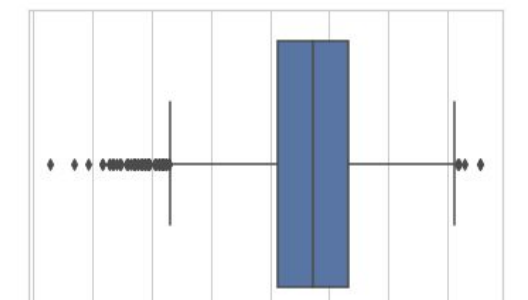
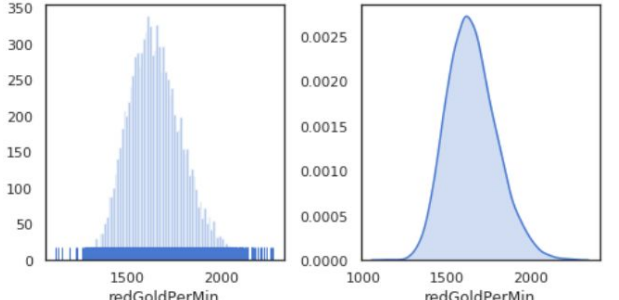
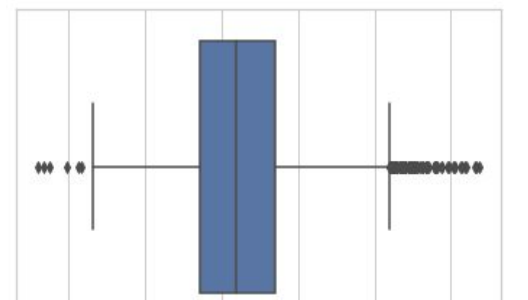


redGoldDiff:
Red team gold
difference compared to
the enemy team.



Distribution: Normal / Gamma
Most of the values from between -1000 to +1000



<p>redExperienceDiff: Red team experience difference compared to the enemy team.</p>	 <p>Distribution: Log / Beta / Normal Same as Gold difference, they follow the same distribution and are very correlated.</p>	
<p>redCSPerMin: Red team CS (minions) per minute.</p>	 <p>Distribution: Beta, Weibull_min Similar distribution of data than redTotalMinionsKilled because is divided by 10.</p>	
<p>redGoldPerMin: Red team gold per minute.</p>	 <p>Distribution: Lognorm / Beta / Pearson3 / Gamma Similar distribution of data than redTotalGold because is divided by 10.</p>	

Distributions founded:

Normal distribution, is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean. In graph form, normal distribution will appear as a bell curve.

Logarithmic distribution, is a discrete, positive distribution, peaking at $x = 1$, with one parameter and a long right tail. The logarithmic distribution is insurance for modeling a claim frequency. It has been used to describe.

Beta distribution, is a family of continuous probability distributions defined on the interval $[0, 1]$ parametrized by two positive shape parameters, denoted by α and β , that appear as exponents of the random variable and control the shape of the distribution.

Weibull max distribution, The Weibull distribution is one of the most widely used lifetime distributions in reliability engineering. It is a versatile distribution that can take on the characteristics of other types of distributions, based on the value of the shape parameter, β .

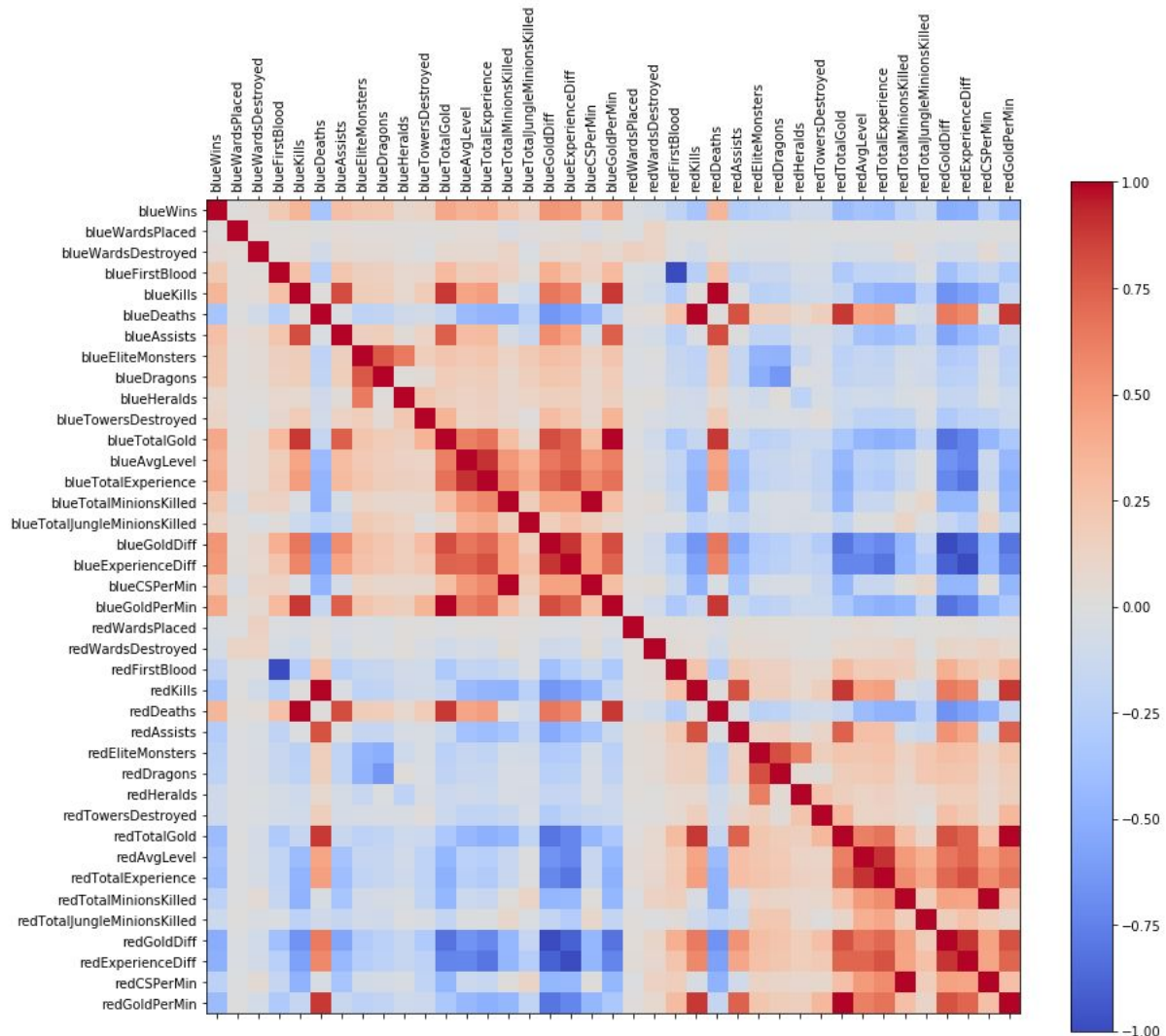
Binomial distribution, with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent experiments, each asking a yes–no question, and each with its own boolean-valued outcome: success/yes/true/one (with probability p) or failure/no/false/zero (with probability $q = 1 - p$).

To know the distribution of the variables we use the Histogram as an approximate representation of the distribution of numerical or categorical data.

A Histogram is a diagram showing the number of examples with the value of a given attribute contained in a certain class. A histogram is used for continuous data, where the bins represent ranges of data, while a bar chart is a plot of categorical variables. The histogram graphically shows the center of the data; spread of the data, presence of outliers, identifying the remote points.

The "torn" shape of the histogram sometimes makes it difficult to interpret the distribution. In our case we use the function [`seaborn.distplot`](#).

Correlation between attribute values (HeatMap):



The closer the correlation coefficient is for the value 1.0 means there is strong positive relationship between two variables. For example, in these dataset with the two variables `blueKills` and `redDeaths`, when for a positive increase in one variable, there is also a positive increase the second variable.

On the other hand when the the correlation coefficient value is closer to -1.0 means there is a strong-negative relationship between two variables. For a positive increase in one variable, there is a decrease in the second variable. In this dataset for example `blueFirstBlood` and `redFirstBlood`. Finally, if the correlation is next to 0, we can say that these variables are not correlated.

Strong-Positive correlation (near to 1.0)

```
blueKills - redDeaths
redKills - blueDeaths
blueDeaths - redKills
redDeaths - blueKills
blueTotalGold - blueGoldPerMin
```

redTotalGold - redGoldPerMin
blueTotalExperience - blueAvgLevel
redTotalExperience - redAvgLevel
blueTotalMinionsKilled - blueCSPerMin
redTotalMinionsKilled - redCSPerMin
blueGoldDiff - blueExperienceDiff
redGoldDiff - redExperienceDiff

Positive Correlation:

blueKills - blueAssists
redKills - redAssists
blueKills - blueTotalGold
redKills - redTotalGold
blueDeaths - redAssists
redDeaths - blueAssists
blueGoldPerMin - blueKills
redGoldPerMin - redKills

Strong-Negative correlation (near to -1.0):

blueFirstBlood - redFirstBlood
blueGoldDiff - redGoldDiff
blueExperienceDiff - redExperienceDiff
blueGoldDiff - redExperienceDiff
redGoldDiff - blueExperienceDiff

Negative Correlation:

redTotalGold - blueGoldDiff
redGoldDiff - blueTotalGold
redDragons - blueDragons
redTotalGold - blueExperienceDiff
redAvgLevel - blueExperienceDiff
blueAvgLevel - redExperienceDiff
redTotalExperience - blueExperienceDiff
redTotalExperience - blueGoldDiff
blueTotalExperience - redGoldDiff
blueTotalExperience - redExperienceDiff
redGoldPerMin - blueGoldDiff
blueGoldPerMin - redGoldDiff

Preliminary findings about the contents of the datasets:

Shortly we can see that the dataset obey our suspicions:

Kill towers → kill base → win game.

More experience → higher levels → easier killing.

More gold → better items → easier killing.

More wards → better vision → less deathing.

Discussion of data quality:

Missing data:

There is not missing values to discuss. So is a perfect quality. Seems that the Riot Games API do a very good work. We assume that the data was not corrected using some of the Replacement Policies methods to obtain missing data.

Outliers:

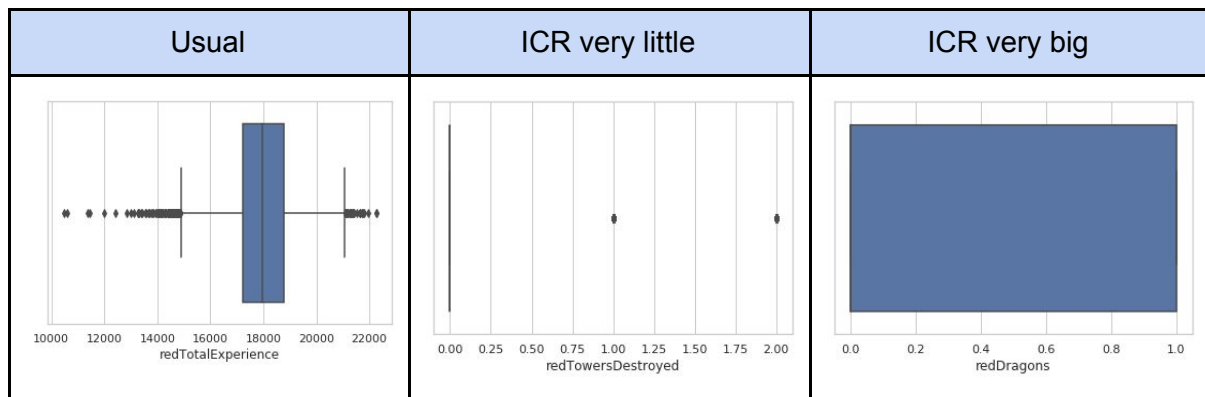
An outlier is an instance that is distant from other instances. They may be due to variability in the measurement or may indicate experimental errors.

An outlier is an observation that appears to deviate markedly from other observations in the sample. An outlier may indicate bad data. For example, the data may have been coded incorrectly or an experiment may not have been run correctly.

We assume that the data is okay because Riot Games API is very confident, but we can not be 100% sure considering that we did not collect the data by ourselves and also we did not play those games being able to have another way to measurement each game.

To find outliers in our dataset we used Box plots. Box plots are a method for detecting the presence of outliers, since they depict groups of data through their quartiles. The length of the box represents the interquartile range (IQR), which is the distance between the third quartile and the first quartile. The middle half of the data falls inside the interquartile range. The whisker below the box shows the minimum of the variable while the whisker above the box shows the maximum of the variable. Within the box, it will also be drawn a line which represents the median of the variable.

Watching our boxplots, we can divide them in three groups:



Also, we add the output of Knime to get supplementary information for our outliers watching them in a numeric table format.

Summary - 0:4 - Numeric Outliers

File Hilite Navigation View

Table "default" - Rows: 40 Spec - Columns: 5 Properties Flow Variables

Row ID	S Outlier column	I Membe...	I Outlier count	D Lower bound	D Upper bound
Row0	gameId	9879	154	4,425,625,377.25	4,579,396,145.25
Row1	blueWins	9879	0	-1.5	2.5
Row2	blueWardsPlaced	9879	1627	5	29
Row3	blueWardsDestroyed	9879	136	-3.5	8.5
Row4	blueFirstBlood	9879	0	-1.5	2.5
Row5	blueKills	9879	88	-2	14
Row6	blueDeaths	9879	65	-2	14
Row7	blueAssists	9879	209	-3.5	16.5
Row8	blueEliteMonsters	9879	0	-1.5	2.5
Row9	blueDragons	9879	0	-1.5	2.5
Row10	blueHeralds	9879	1857	0	0
Row11	blueTowersDestroyed	9879	464	0	0
Row12	blueTotalGold	9879	114	12,348.375	20,525.375
Row13	blueAvgLevel	9879	69	6.2	7.8
Row14	blueTotalExperience	9879	107	14,834	21,058
Row15	blueTotalMinionsKilled	9879	61	157	277
Row16	blueTotalJungleMinio...	9879	164	26	74
Row17	blueGoldDiff	9879	127	-6,358.875	6,368.125
Row18	blueExperienceDiff	9879	125	-5,048	4,968
Row19	blueCSPerMin	9879	61	15.7	27.7
Row20	blueGoldPerMin	9879	114	1,234.837	2,052.538
Row21	redWardsPlaced	9879	1667	5	29
Row22	redWardsDestroyed	9879	136	-3.5	8.5
Row23	redFirstBlood	9879	0	-1.5	2.5
Row24	redKills	9879	65	-2	14
Row25	redDeaths	9879	88	-2	14
Row26	redAssists	9879	205	-3.5	16.5
Row27	redEliteMonsters	9879	0	-1.5	2.5
Row28	redDragons	9879	0	-1.5	2.5
Row29	redHeralds	9879	1581	0	0
Row30	redTowersDestroyed	9879	396	0	0
Row31	redTotalGold	9879	99	12,440.125	20,405.125
Row32	redAvgLevel	9879	86	6.2	7.8
Row33	redTotalExperience	9879	134	14,876.125	21,097.125
Row34	redTotalMinionsKilled	9879	69	158	278
Row35	redTotalJungleMinion...	9879	131	24.5	76.5
Row36	redGoldDiff	9879	127	-6,371	6,359
Row37	redExperienceDiff	9879	126	-4,964.625	5,042.375
Row38	redCSPerMin	9879	69	15.8	27.8
Row39	redGoldPerMin	9879	99	1,244.013	2,040.513

Overall data integrity:

Come the data from a widely used official API, we can expect that data integrity is excellent. Of the 4 logical integrity, we are going to talk about the next two:

Entity integrity

Using the primary key `gameId`.

Domain integrity

Allowing only values `int64` or `float64` according to the attributes described above.

Values of unknown meaning:

No values in this area.

Conclusion

After this first report we can say that our dataset is really good, is not easy get a dataset with a good integrity, no missing values and no weird values.

In the next steps we will be able to classify data and use Classification, Regression, Decision trees etc. To be able to guess future team-winner in following matches only with the data of the first 10 minutes.

Webgraphy / Bibliography

[seaborn: statistical data visualization — seaborn 0.10.1 documentation](#)
[What is Data Integrity and Why Is It Important?](#)
[When is it justifiable to exclude 'outlier' data points from statistical analyses?](#)
[Process Mining Wil van der Aalst Data Science in Action Second Edition](#)
[How to Identify the Distribution of Your Data](#)
[Goodness-of-Fit Tests for Discrete Distributions](#)
[Weibull distribution - Wikipedia](#)
[Binomial distribution - Wikipedia](#)
[Dirichlet distribution - Wikipedia](#)
[Beta distribution - Wikipedia](#)
[kolmogorov smirnov - Goodness-of-Fit for continuous variables - Cross Validated](#)
[pandas.DataFrame.dtypes — pandas 1.0.3 documentation](#)
[Inverse Gaussian distribution - Wikipedia](#)
[Probability Distributions in Data Science - Towards Data Science](#)
[Logarithmic distribution - Wikipedia](#)
[Process Mining: Data science in Action - Inicio | Coursera](#)
[Riot Developer Portal](#)
[matplotlib.pyplot.subplots — Matplotlib 3.2.1 documentation](#)
[Scatterplot Matrix — seaborn 0.10.1 documentation](#)
[Tutorial on Outlier Detection in Python using the PyOD Library](#)
[CheatSheet-Python-7_-NumPy-1.pdf](#)
[pandas.DataFrame.hist — pandas 1.0.3 documentation](#)
[python - How can I hand a to bandwidth for the kde to seaborn's distplot? - Stack Overflow](#)
[La prueba de ji-cuadrado - Medwave](#)
[Prueba de chi-cuadrado \(\$\chi^2\$ \): qué es y cómo se usa en estadística](#)
[How To Make Histogram in Python with Pandas and Seaborn? - Python and R Tips](#)
[Identify your Data's Distribution](#)

Appendix I - Python's code used.

Get our covariance matrix:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data =
pd.read_csv('/kaggle/input/league-of-legends-diamond-ranked-games-10-min/high
_diamond_ranked_10min.csv', index_col=0)
corr = data.corr()
fig = plt.figure(figsize=(14,12))
ax = fig.add_subplot(111)
cax = ax.matshow(corr,cmap='coolwarm', vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,len(data.columns),1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
ax.set_yticks(ticks)
ax.set_xticklabels(data.columns)
ax.set_yticklabels(data.columns)
plt.show()
```

Get the boxplot for each attribute:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df =
pd.read_csv('/kaggle/input/league-of-legends-diamond-ranked-games-10-min/high
_diamond_ranked_10min.csv', index_col=0)

import seaborn as sns
sns.set(style="whitegrid")

for column in df:
    plt.figure()
    ax = sns.boxplot(x=df[column])
```

Get the histogram and displot (distribution plot):

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.set(style="white", palette="muted", color_codes=True)

#Set up the matplotlib figure
f, axes = plt.subplots(nrows=2, ncols=2, figsize=(7, 7), sharex=False)
#sns.despine(left=True) #if you want it True, just disable the commentary

grafica = df['blueKills'] #has to be changed to each attribute manually
d = grafica
# Plot a filled kernel density estimate with shade
sns.distplot(d, hist=False, color="b", kde_kws={"shade": True}, ax=axes[0,
1])

# Plot a kernel density estimate and rug plot
bins_needed = 10
sns.distplot(d, kde=False, rug=True, color="b", bins = bins_needed,
ax=axes[0, 0])

#plt.setp(axes, yticks=[])
plt.tight_layout()
```

Identify your data distribution with chi-square method

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
data =
pd.read_csv('/kaggle/input/league-of-legends-diamond-ranked-games-10-min/high
_diamond_ranked_10min.csv', index_col=0)

dist_names = ['weibull_min', 'norm', 'weibull_max', 'beta',
'invgauss', 'uniform', 'gamma', 'expon', 'lognorm', 'pearson3', 'triang']

y = data['redGoldPerMin']
y_std = data['redGoldPerMin']
size = 9879

chi_square_statistics = []
# 11 equi-distant bins of observed Data
percentile_bins = np.linspace(0,100,11)
percentile_cutoffs = np.percentile(y_std, percentile_bins)
observed_frequency, bins = (np.histogram(y_std, bins=percentile_cutoffs))
cum_observed_frequency = np.cumsum(observed_frequency)

# Loop through candidate distributions
for distribution in dist_names:
    # Set up distribution and get fitted distribution parameters
    dist = getattr(scipy.stats, distribution)
```

```
param = dist.fit(y_std)
print("{}\n{}\n".format(dist, param))

# Get expected counts in percentile bins
# cdf of fitted sistrinution across bins
cdf_fitted = dist.cdf(percentile_cutoffs, *param)
expected_frequency = []
for bin in range(len(percentile_bins)-1):
    expected_cdf_area = cdf_fitted[bin+1] - cdf_fitted[bin]
    expected_frequency.append(expected_cdf_area)

# Chi-square Statistics
expected_frequency = np.array(expected_frequency) * size
cum_expected_frequency = np.cumsum(expected_frequency)
ss = sum (((cum_expected_frequency - cum_observed_frequency) ** 2) /
cum_observed_frequency)
chi_square_statistics.append(ss)

#Sort by minimum ch-square statistics
results = pd.DataFrame()
results['Distribution'] = dist_names
results['chi_square'] = chi_square_statistics
results.sort_values(['chi_square'], inplace=True)

print ('\nDistributions listed by Betterment of fit:')
print ('.....')
print (results)
```