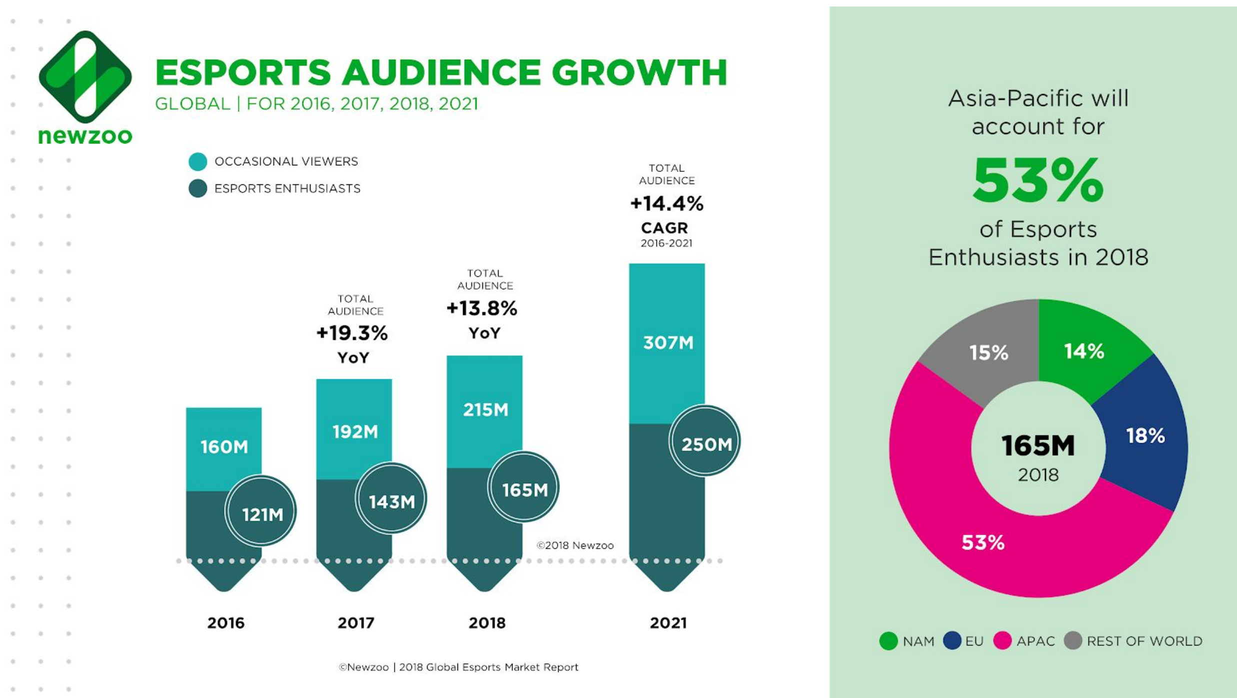
Capstone Project #1: BF4 Player Performance Analysis

Capstone Report #1 Data Collection, Wrangling and EDA

Date: 9/30/2019

Introduction and Objective

Esports has grown in the past decade or so into a million dollar industry with the development of competitive gaming and faster internet. It is projected that in 2020, esports will exceed $1 billion in revenue generated.



Just as any sports competition, players and coaches are taking note of how data can be used to monitor the performance and status of a player. Stats are collected to see how effective players are in different aspects of the game. On the other side, game designers are collecting data from their games so that they can analyze the competitiveness and balanceness of their games. A good example is Overwatch, a competitive online shooter created by Activision Blizzard, since it's launch in 2016 has been issued more than 100 patches in the course of 3 years so that the game is better balanced.

We gathered BF4 data and analyzed it using statistical analysis and data visualization. The reason BattleField 4 was chosen was mainly it's a popular first person shooter game that has a large player base and a variety of weapons along with different vehicles which allows variability in playstyle. There has also been rumors amongst players that certain play styles are overpowered, we will use data to verify whether this is true.

In summary, the objective of the capstone is to use BF4 as an example to explore what kinds of analysis is suitable for analyzing online competitive games.

Dataset Acquisition

The player data collected for BF4 was acquired via web scraping and API calls. While there were no obvious API limitations, the data collection still took a considerable amount of time due to the size of the entire data set exceeding 30GB, coupled with unstable internet connections and server response errors.

The approach to acquiring and storing the data is as follows:

- All player rankings are stored on the webpage of the site, using beautifulsoup, a python html parser, we are able to go through player rankings page by page and acquire their ranking information
- API data can be called using the player id obtained from the web scrape mentioned above and used to pull in individual player data and have it stored somewhere

(Player Ranking) Dataset #1: We chose the console platform XOne and found the pages containing player name and ranking, I had my code go through each page individually and use beautifulsoup to parse the data. Each page contained 50 player entries, and there were a total of 600 pages. We were able to collect rankings for about 35,000+ players.

- Estimated time required: 20 minutes
- Actual time: 45 minutes

(Performance Data) Dataset #2: Before collecting this data, we did 5 api calls and came back with about 4MB of data, meaning 1 call would generate 1 player information which would be about 1MB; This meant that our data would be about 35GB in size. Since the data is so big, we decided to store the collected data in a google cloud bucket as google provides great services in regards to processing big data. In the end, we had each individual player data stored in google cloud services as a pickle file with the entire dataset having a total size of 38GB.

- Estimated time required: 72 hours (3 days)
- Actual time: 130 hours (5+ days)

Data Wrangling

The data we collected are individual player files. In order to convert this data to a workable dataframe, we have to first choose which columns we'd like in our dataframe. After reviewing the data, we found that some crucial features existed under subcategories. For instance, if you were to convert "vehicleCategory" into a dataframe, you'd find a table where a number of stats such as "kpm", "spm", "name", "destroys", "kills", "score" etc. expressed as the x-axis, and the names of the type of weapons such as "assault rifle", "LMG", "sidearm" expressed as y-axis.

Weapon categories

Name	Score	Time	↓Kills	Headshots	HK %	Kpm	Shots	Hits	Accuracy %
ASSAULT RIFLE	121,219	23:38:47	757	153	20.21%	0.53	44,371	5,132	11.57%
LMG	83,523	12:50:44	545	85	15.60%	0.71	34,097	2,996	8.79%
SIDEARM	89,303	10:42:08	431	101	23.43%	0.67	12,008	2,191	18.25%
SNIPER RIFLE	45,422	10:06:38	303	90	29.70%	0.50	2,703	601	22.23%
CARBINE	47,768	08:58:52	299	48	16.05%	0.55	15,671	1,973	12.59%
PDW	30,325	06:44:28	239	38	15.90%	0.59	13,359	1,572	11.77%
GADGET	-	05:40:22	129	0	0.00%	0.38	12,063	2,988	24.77%
SHOTGUN	13,191	01:39:53	65	5	7.69%	0.65	631	486	77.02%
KNIFE	-	00:00:00	56	0	0.00%	0.00	0	0	0.00%
GRENADE	8,459	00:19:37	55	0	0.00%	2.80	1,108	261	23.56%
DMR	1,774	00:30:48	17	5	29.41%	0.55	424	63	14.86%
BATTLE PICKUP	-	00:31:36	4	0	0.00%	0.13	145	18	12.41%

The problem with data expressed in a table like this is that it only represents one player, and in order to analyze multiple players, we need to have the data flattened out where all the stats are expressed in one row. For instance, a players "LMG" and "PDW" stats would be expressed in one row as "LMG_Kills", "LMG_headshots", "PDW_Kills", "PDW_headshots".

As we convert the individual player pkl files into dataframe, we wrote a function to conduct this transformation, the function would turn a 12 row * 9 column table to a 1 row * 108 column table. Since we are interested in players' vehicle and weapon usage, we only applied this on this function on the vehicle and weapon categories and ignored the other sub categorical data.

At the same time, in order to save space, we converted all NA values into the numerical value "999.99" so that all our data could be expressed using numbers. Eventually being able to convert our data into a pandas dataframe with the properties below:

Int64Index: 37759 entries Columns: 395 entries dtypes: int64(394), object(1)

To continue on, we can reindex this dataframe and drop any unneeded columns. Then we describe this data and see if there are any unusual information. There are certain columns that contains data where the min and max are the same. Usually this kind of data doesn't really contribute to analysis so we use loop to find all of them and have them dropped from the dataframe. Then we look for duplicate datasets. We found one entry that was collected twice using the value_count function, which auto-sorts counts by highest to lowest. We can remove the duplicate data set. We also notice that the 75% and 25% of the data for bestStreak is somehow the same, we plot bestStreak out to see what's going on and find that only a small percentage of players are able to make high bestStreak numbers whereas the majority of players only gain 0 to 5 bestStreak numbers, thus we delete this column and other columns similar to it where the 75% and 25% data are the same. Since in the previous processing step, we used 999.99 to replace all NA values, we can try to find which columns contain this value to check for missing values.

Int64Index: 37839 entries   Columns: 270 entries   dtypes: int64(269), object(1)

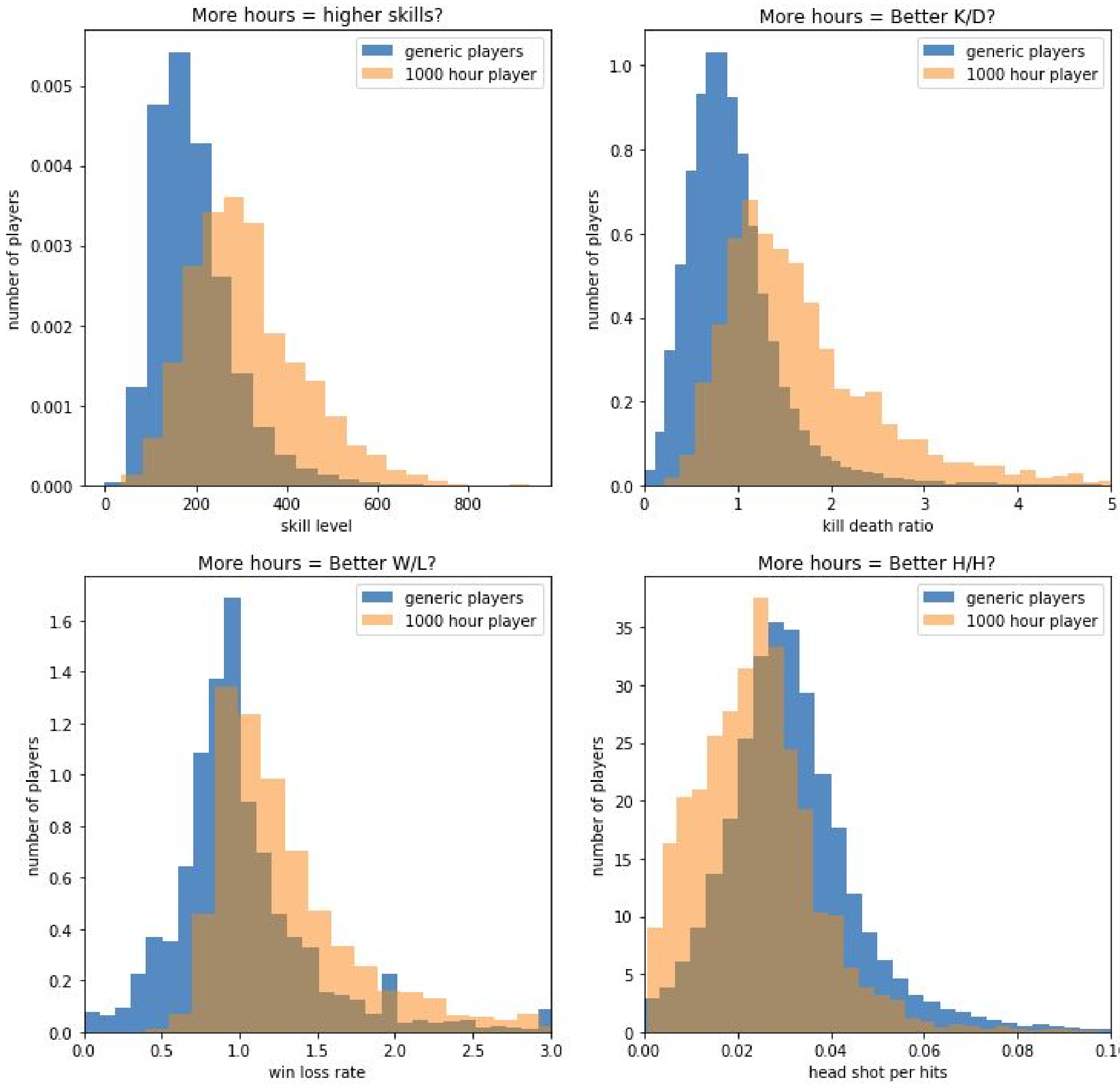
Since no value is returned, we can conclude that our dataframe doesn't contain any missing values. Video game data usually have a high level of completeness and missing data doesn't occur if the game's data collection process is well designed. Lastly, we merge this dataframe with our web scraped data to obtain our final data frame. We can now use this data frame for statistical and machine learning analysis.

Exploratory Data Analysis

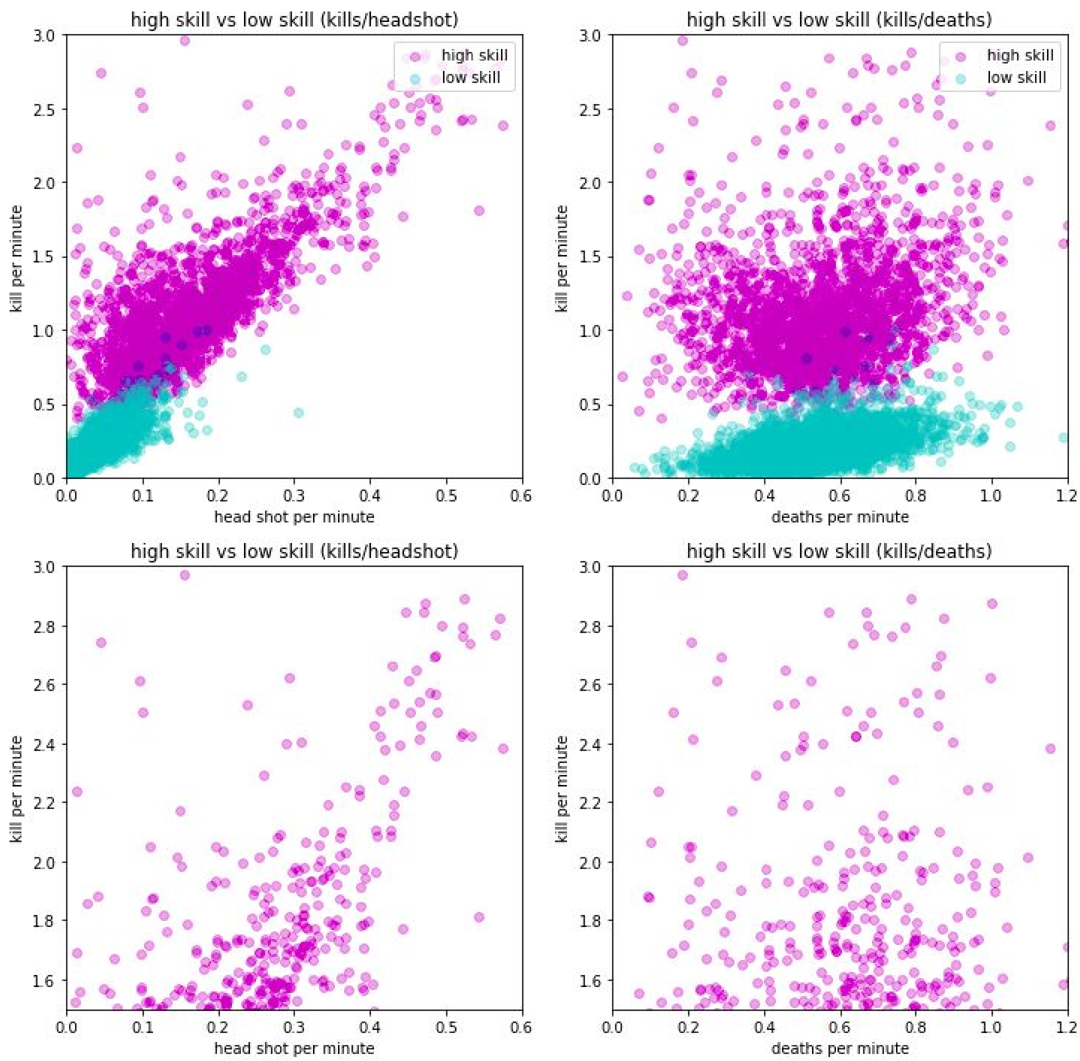
<u>Generic player performance stats (Analysis #1)</u>

In the beginning of the analysis, we mostly focused on overall player performance. The player_ids were removed, leaving only int64 formatted numbers. By looking at the skill level and player frequency, we found that the majority of our players had a skill level below 237 and most players had less than 1000 hours of play time. This meant that our data is heavily skewed left when it comes to hours.

Top players are in general very dedicated to their game. So in order to better understand this data, we split the players into two groups based on the 1000 hour play time, and used histograms to see if 1000 hour players did better than generic player who has less than 1000 hours of play time.



The results showed that contributing to more hours within the game did lead to better wins, higher skills and better Kill/Death ratios. However, one thing that stood out was headshots. In typical first person shooters, headshots are considered the most effective way of eliminating the enemy as it correlates with high damage. Yet in this scenario, players who have long playtime seems to be actively avoiding making headshots and going for body-shots instead.

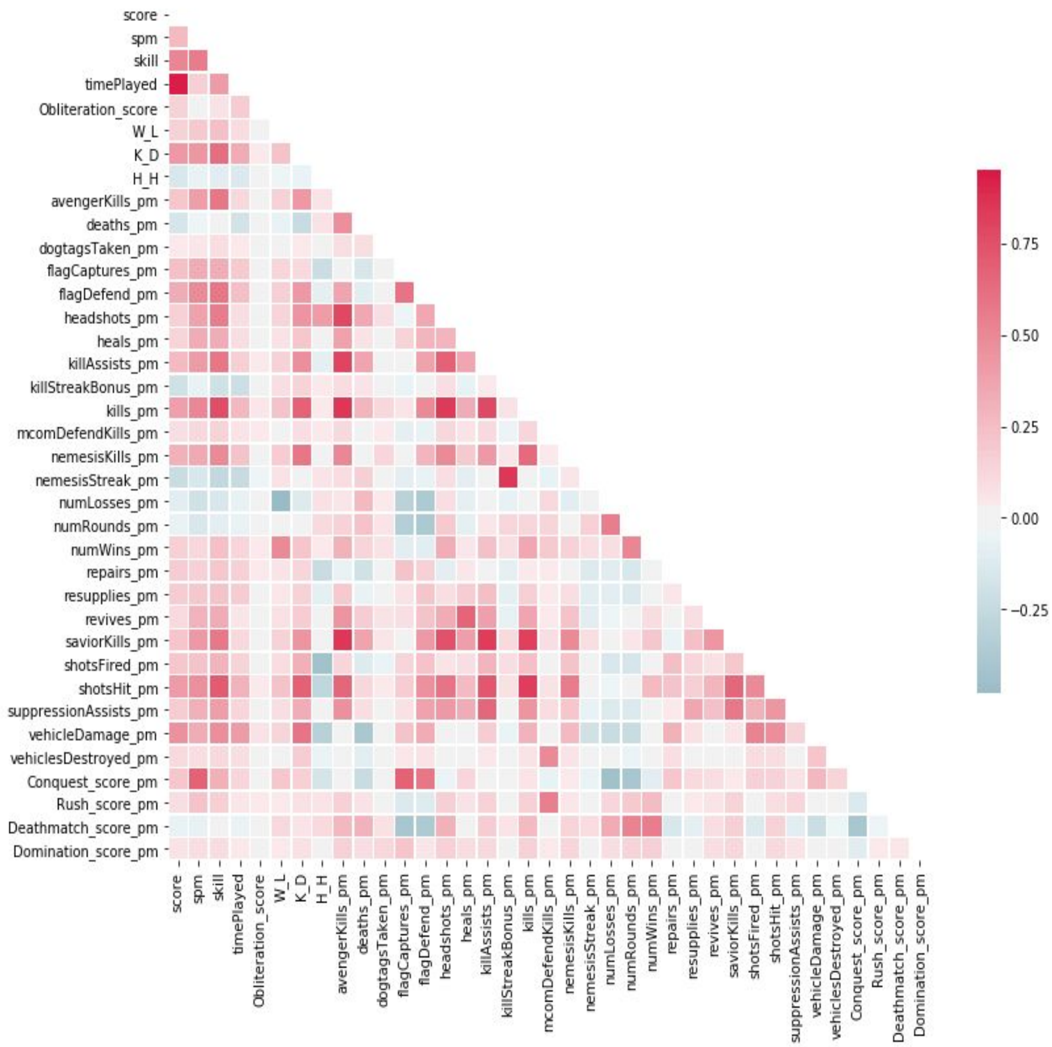


Even highly skilled players seem to only make a little more effort in getting more headshots. This may help us infer that headshots are not an effective way to make more gains within the BF4 game. A few other conclusions drawn from comparing groups and also introducing a correlation matrix:

1. Deaths per minute are nearly the same regardless of skill level;

2. The win-rate for every group is nearly the same since the server does a relatively good job at auto-match making

3. High level players are more likely to target and destroy vehicles which is more likely to contribute to higher win-rates



<u>Gun usage stats (Analysis #2)</u>

Within the gun usage data, we focused on the mean accuracy and standard deviation for all weapon categories. We found that Shotguns have the highest average accuracy, but a high std, which means, this weapon takes a lot of effort to master; LMG is the worst weapon in terms of accuracy; The best and most stable weapon seems to be SNIPER RIFLE, with low standard deviation and a relatively high accuracy.

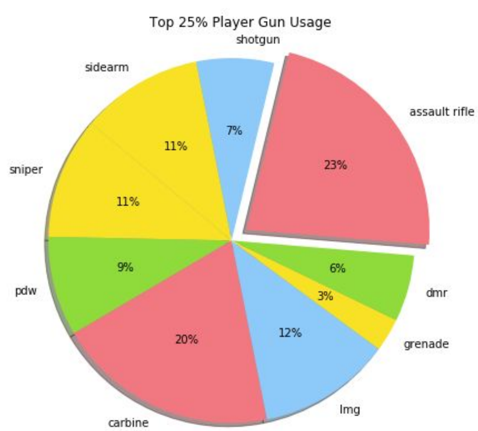
	mean	std	min	25%	50%	75%	max
GADGET_extra.accuracy	23.64	9.88	7.00	16.00	23.00	31.00	46.00
SNIPER RIFLE_extra.accuracy	24.79	5.44	14.00	21.00	25.00	29.00	35.00
PDW_extra.accuracy	11.91	2.97	6.00	10.00	12.00	14.00	17.00
CARBINE_extra.accuracy	12.83	2.88	7.00	11.00	13.00	15.00	18.00
BATTLE PICKUP_extra.accuracy	28.14	8.46	13.00	22.00	28.00	34.00	48.00
LMG_extra.accuracy	9.06	2.50	5.00	7.00	9.00	11.00	14.00
GRENADE_extra.accuracy	24.78	10.54	9.00	16.00	23.00	32.00	51.00
DMR_extra.accuracy	17.55	3.88	10.00	15.00	18.00	20.00	25.00
ASSAULT RIFLE_extra.accuracy	12.32	2.69	7.00	10.00	12.00	14.00	17.00
SHOTGUN_extra.accuracy	63.91	13.89	38.00	53.00	63.00	75.00	93.00
SIDEARM_extra.accuracy	17.90	4.42	9.00	15.00	18.00	21.00	26.00

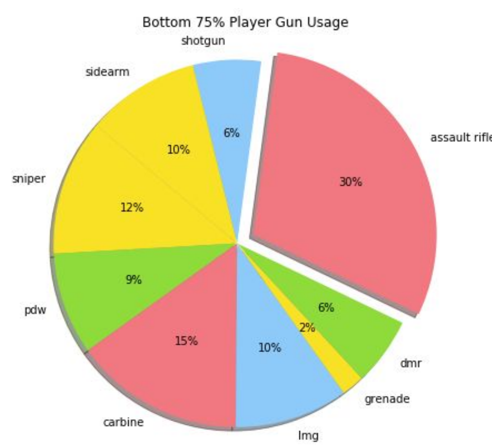
Since the mean and std of carbines were only slightly better than assault rifles, we compared the two groups using a student t-test. The null hypothesis for our t-test was H0: There is no statistically significant difference between the carbine and assault rifle accuracy.

The t-test resulted in the results below which meant the null hypothesis was rejected and that carbines were indeed better than assault rifles when it came to accuracy.

Ttest_indResult(statistic=-21.71692396390152, pvalue=3.8026573725064386e-104)

Another data group we looked at was weapon usage amongst the top 25% and bottom 75% of players. We used score as a way to look at usage frequency since score highly correlates with weapon usage time.





The null hypothesis was that there isn't a significant difference between top 25% of players and bottom 75% in terms of gun preference. We used the chi-square to compare the two groups and the results showed that the null hypothesis couldn't be rejected and there was no evidence that the two groups had significant differences.

Power_divergenceResult(statistic=4.55, pvalue=0.804410169231813)

Additional Conclusions & Next Steps

1. **Player patterns are governed by the features of the game.** Most players can't or don't make consistent headshots. To increase win-rate, players have to destroy vehicles. Assault rifles and carbines are popular guns due to its range and damage. Shotguns are pretty powerful, but difficult to use. Grenades are pretty effective doing spread damage, which helps ramp up points. The best and most stable weapon seems to be the sniper rifle, which has accuracy amongst players of all skills.

2. **The data may need added weight, centering and scaling.** Since the time spent on these weapons are all different, we may need to consider adding weight to certain types of weapons before proceeding. There are also issues surrounding scaling where some features have very high values and others have very low values. However, once the above steps are complete, we can introduce machine learning models to reduce dimensions and find more patterns.