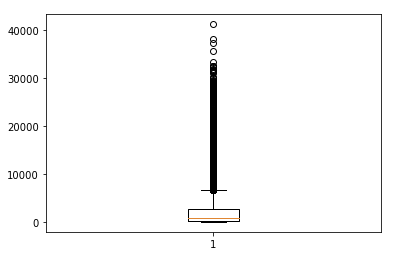
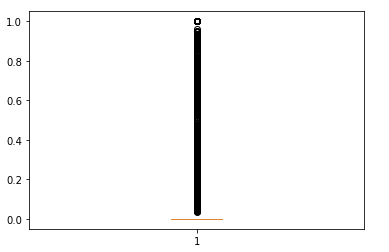
**Cheater Material**

Let's start by stating catching cheaters is a very tricky task. You must walk a fine line to find players who are truly skilled and talented and those who abuse cheats to achieve what would otherwise be unobtainable metrics. That being said, there are some dead giveaways in our data that allow us to find what must be cheaters, otherwise the stats they produce would be completely impossible. First let's state and talk about popular cheating methods and how we will use both our knowledge of the game as well as the data to leverage these methods into usable metrics to spot cheating in the game. Speed hacking is a popular form of cheating in which players increase their own speed in the game so they can avoid closing circles and be the first players to reach better areas. It also has the benefit of making you harder to hit while moving. Using average in-game speed and comparing that to a player's total distance traveled divided by the match duration both given in the data set we can create a box plot(shown below) to find abusers of this cheat.



As we can see the upper acceptable limit a player should move is around 8,000 meters. To be fair and not accidentally ban anyone who isn’t cheating we could move this metric to be 10 - 11 thousand meters. However as we can see some players have traveled above the 40,000 meter mark. This would be impossible without using a speed hack and a vehicle in tandem for nearly the entire game.

We are able to track one other cheat with the data received and that is aimbot. This cheat is known well across all shooters and involves running some form of external software that generally runs some form of image recognition. This software uses that output to control your mouse input and have you aim perfectly at images recognized as players. It takes a stronger PC to run this cheat but many gaming PC’s have the capability. To catch this cheat we can once again use a box plot(shown below) with our headshot percent data.



Now this metric is a little harder to form as we can see from the graph a majority of players never even score a headshot. This however does not indicate every player who hits the head is cheating. This means a little more research was needed to consider a metric. What was found was that aimbot uses what is called a hitscan box, this means as long as you hit the head on your screen the servers will always trigger it as a headshot meaning aimbot in practice never actually misses the head on accident. There are some aimbots that miss the head on purpose every i’th shot to confuse cheat catchers. That being said we see many games in our dataset where a perfect 100% headshot ratio was achieved, this could be caused by two things. One is a player got a single kill which happened to be a headshot although unlikely it isn’t impossible for this to happen. Second a low grade version of aimbot was used, one that simply always aims for the head. Using this graph and research it’s an easy jump to make that a image recognition software could be used to catch cheaters that meet a certain threshold to save the money on computational power. In other words it’s pointless to scan every game for cheating using image detection software when you only need to scan games that have a high kill and headshot percentage.

**Win Prediction**

Our win predictor was quite successful and will be written about in three parts which are steps taken in the model building and data preparation, the success rates and metrics of the model, and inferences we found from the model which could be useful. First is the steps taken to get the results. The first step taken was to handle both outliers and NAN value data, this was handled via removal over imputation as both outliers and NAN values together made up less than 2% of the total data. Next EDA was performed on the data and it was decided what sort of transformations needed to take place based on that EDA. It was decided to normalize the data to get all the data onto the same page across all records. Finally our last step before model building was to check for linearity, multicollinearity, and heavily influential data entries. These were handled for the most part via removal and feature engineering.

Now let's move onto model evaluation. We had very good model performance with every one of our models (solos, duos, squads) performing well into the 90% accuracy on both train and test sets. This is a very well performing model for the task given and helps give a lot of insight into winning. We will now go into our third part of outlining these insights which can be used to help players improve their game. First we can indicate which features had a heavy impact on our models. Each model had a different training so we’ll split it into three parts. The Solo model was heavily impacted by percent\_kill and assists. This is very interesting as it gives us insight that helping other players with kills and getting kills is a large part of winning games in solos. This may seem obvious but it is a well known strategy in solos to hide until the end of the game, this data points to that strategy worsening your chances of winning. Next the duos model was impacted by kill\_streak, percent\_team\_damage, percent\_damage, and percent\_team\_kill. This once again shows us a passive gameplay style that tends to lose games. It also points to staying alive being important as kill streaks require you to not be downed. Finally for squads important features include percent\_damage, percent\_team\_damage, percent\_kill, percent\_team\_kill, killStreaks. What’s interesting is that besides kill\_streaks all other features are negative. What this means is a passive playstyle where you take kills at the last moment seems to be the optimal playstyle for squads.

**Kills and Distance Impact**