**PUBG Project Technical Document**

**Contents:**

Project Description

Initial Partitioning of the Data

Exploratory Data Analysis

Data Preparation

Model Building

Analysis of Impact of Certain Features on Outcomes

Predicting Fraudulent Play

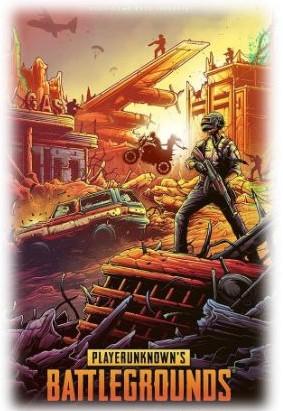
Deployment using Django

***Predicting Player Finishing Win Placement***

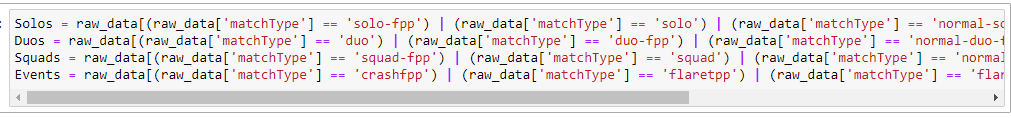
**Project Description**

*PlayerUnknown's Battlegrounds* (PUBG) is an online multiplayer battle royalegame developed and published by PUBG Corp. *Battlegrounds* is a player versus player shooter game in which up to one hundred players fight it out in a battle royale, a type of last man standing deathmatch.

Using a large set of data from anonymized PUBG game statistics comprised of player post-match statistics, the project objective is to build a model that predicts what a player's chances of winning are for a given set of post-match stats. Additionally we'll determine whether certain stats, specifically kill count and distance traveled in a game, have a measurable impact on the win chances. And finally we also seek to find ways to detect fraudulent players, i.e. cheaters, by analyzing the stats.

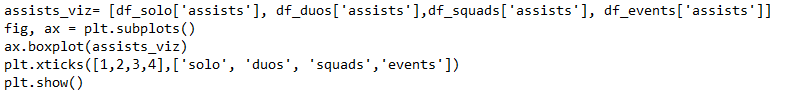


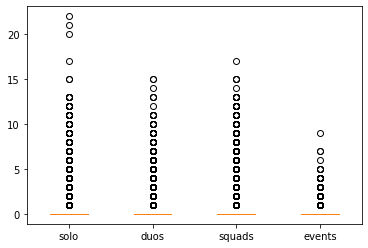
**Initial Partitioning of the Data**

The first step we took was to partition the data between the different game modes. We felt the model would be more accurate if made specific to the different modes, since the effects of game stats can be different when different game dynamics are at play in the different modes. This was done using a simple query on the pandas dataframe shown below then by exporting those new dataframes to new CSV files: 

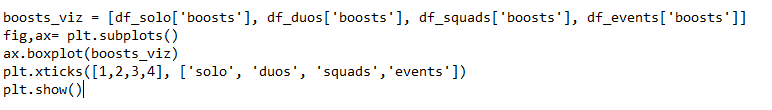
**Working on NAN Values and Outliers**

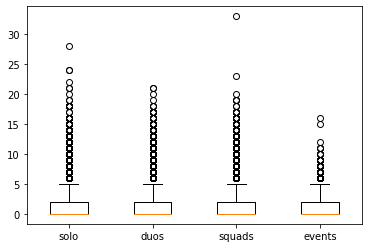
Visualize assists column



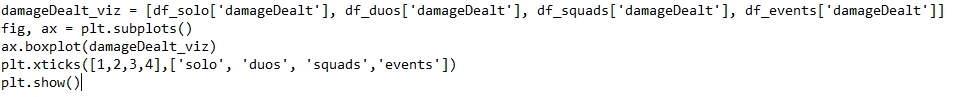


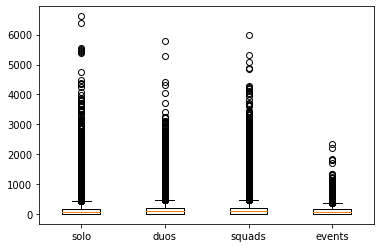
Visualize boosts column



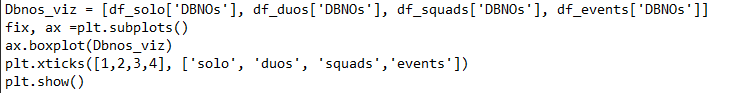


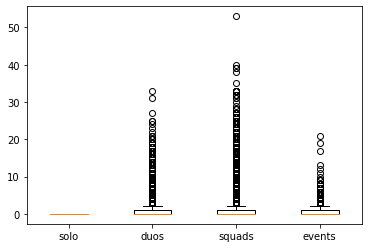
Visualize DamageDealt

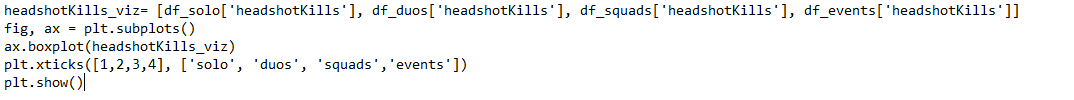


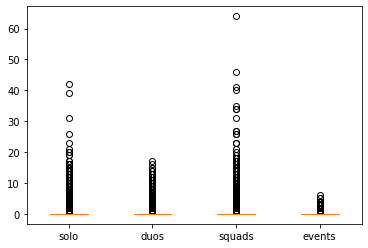


Visualize DBNOS



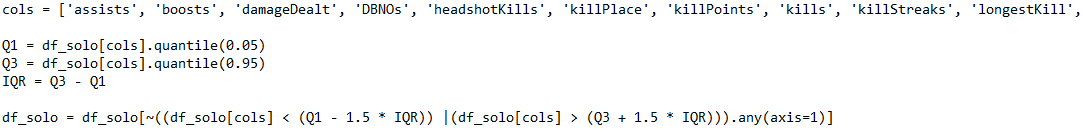




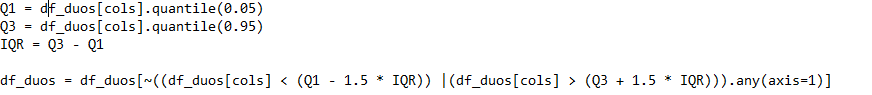


REMOVE OUTLIERS

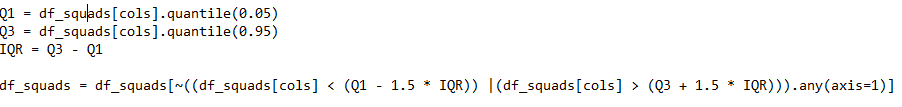
Removing outliers from solo



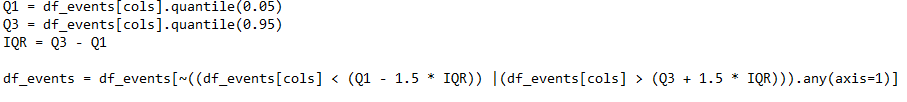
Removing outliers from duos



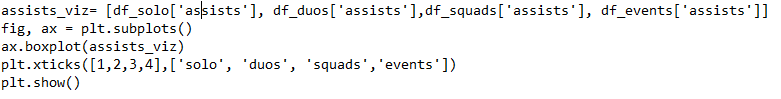
Removing outliers from squads

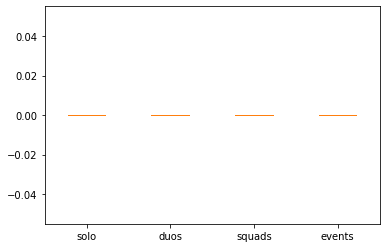


Removing outliers from events



Plot to check if there is outliers



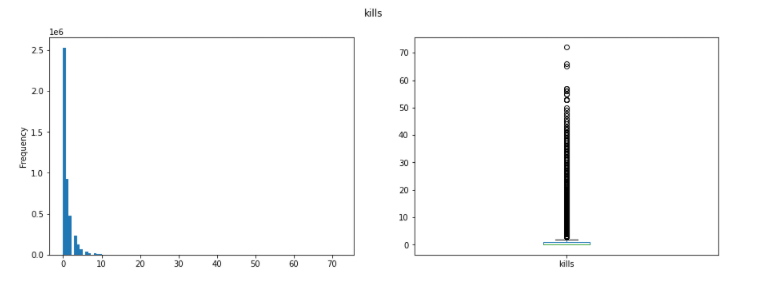


**Exploratory Data Analysis**

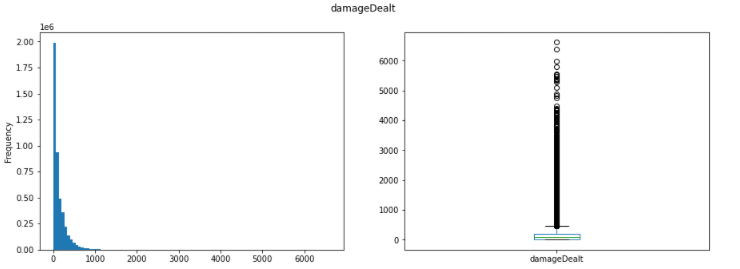
EDA done without removing cheaters/AFK

Histogram and box plots of all features

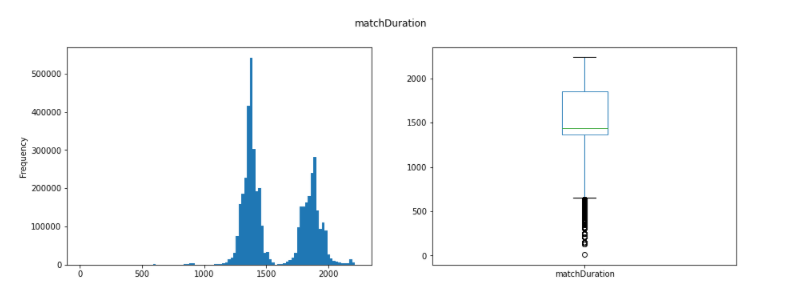
EDA shows that for almost all the features, data are skewed towards 0 and for some there are some minor data points that are on the extreme side such as having 60 kills when the maximum number of players are 100.



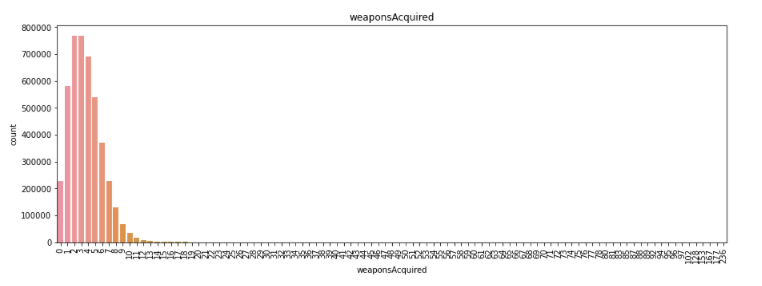
And this is also so for damage dealt with the data being skewed to 0



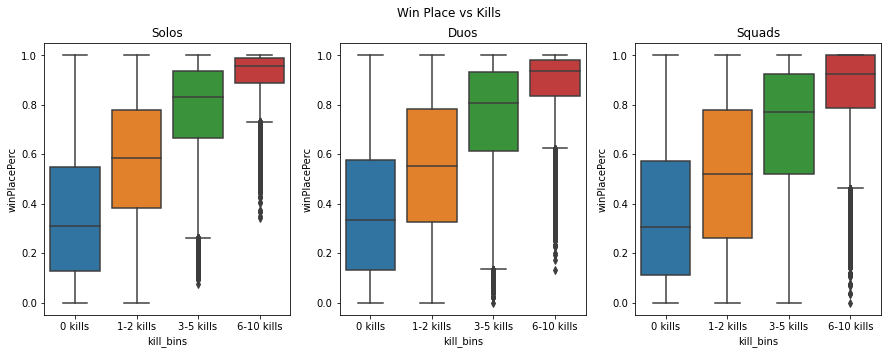
For match duration we found it interesting that it is bimodal and tested to see if match type might be the cause of it but it was bimodal for all game modes and we couldn’t find a cause for it.



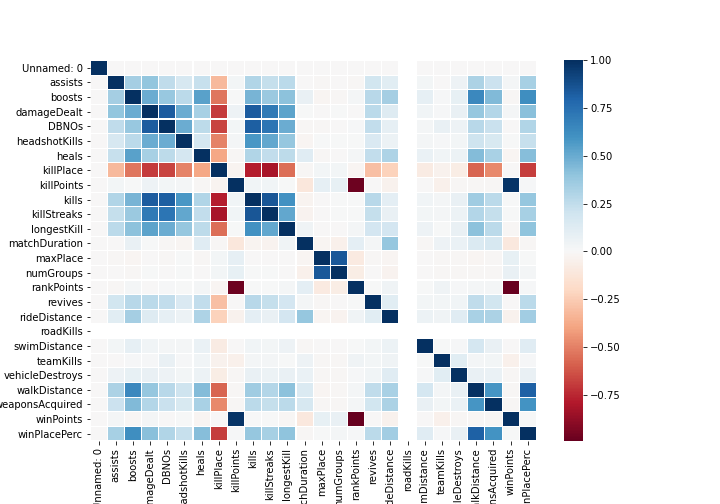
Another interesting we found that could be useful for cheater detection is that for weapons acquired it is almost normally distributed until around 13 weapons or so then there are some data entries where weapon acquired goes up to 236



We then tried to see how features such as kills affect our target variable, winPlacePerc. As seen from the plot below, the higher the kills the higher the mean of winPlacePerc which makes sense because the more you kill it will be more likely for you to be more skilled as well as have better gear and loot for more survivability



We then did a correlation heat map to visualize multi collinearity and we will be removing values with high collinearity by comparing their VIF later on



**Normalization**

Some normalization that will have to be done are for kills and damage as they will definitely affect the model heavily and some other features were also created as well.

**The main Normalized data**

| New Feature | Description |
| --- | --- |
| percent\_kill | Percentage from total kill for a player in the match (kills/kills\_in\_match) |
| percent\_team\_kill | Percentage team kills from total kills in the match (Team\_Total\_Kills/kills\_in\_match) |
| percent\_damage | Percentage of damage dealt from total damage dealt in the match (damageDealt/damage\_in\_match) |
| percent\_team\_damage | Percentage of damage dealt by team from total damage dealt in the match (Team\_Total\_Damage/damage\_in\_match) |
| headshot\_rate | Percentage of headshot kills from kills (headshotKills/kills) |

**Additional features created**

| New Feature | Description |
| --- | --- |
| players\_in\_match | The number of players in a match |
| players\_in\_team | The number of players in a team (out of 4) |
| kills\_in\_match | Total number of kills by everyone in the match |
| Team\_Total\_Kills | Total number of kills by team |
| damage\_in\_match | Total damage dealt by everyone in a match |
| Team\_Total\_Damage | Total damage dealt by team |
| heals\_and\_boost | Number of heals and boosts used (heals+boosts) |
| items | Total of items picked up (heals+boosts+weaponAcquired) |
| total\_distance | Total distance travelled (walkDistance + rideDistance + swimDistance) |

After feature engineering is done Id, groupId, matchId, damageDealt, kills and headshotKills are removed.

**Data Preparation**

*Detecting and Filtering for High Collinearity*

We used VIF (Variance Inflation Factor) calculations to find input variables with excessive collinearity with the other inputs. To do that first we wrote a function to produce a table of VIF values for each feature:

import statsmodels.formula.api as smf

### Generate smf.ols() argument string from list of input vars & dependent var

### Inputs: dependent variable, list of input variables

### Output: argument string for smf.ols() model building function.

def genArgStr(depVar, varLst):

argStr = depVar + " ~ "

for var in varLst:

argStr = argStr + var + " + "

return argStr[:-3]

### Function to generate a list of VIF values for a DataFrame's input variables.

### Inputs: DataFrame, list of input variables

### Returns: DataFrame of R-square & VIF values for each variable.

def genVIFs(df, varLst):

def vif(rsq):

return 1/(1 - rsq)

# Build list of string argument permutations to pass to smf.ols() method.

argStrings = []

for vr in varLst:

vrs = varLst[:]

vrs.remove(vr)

argStrings.append(genArgStr(vr, vrs))

# Get R-squares

rSqrs = []

for string in argStrings:

rSq = smf.ols(string, data = df).fit().rsquared

rSqrs.append(rSq)

# Build DataFrame with VIF values and return sorted.

VIF\_df = pd.DataFrame({"Variable": varLst, "R^2": rSqrs, "VIF": map(vif, rSqrs)})

return VIF\_df.sort\_values("VIF", ascending=False).reset\_index(drop = True)

We then applied it to the squad data, first extracting the dependent variable before entering the feature list.

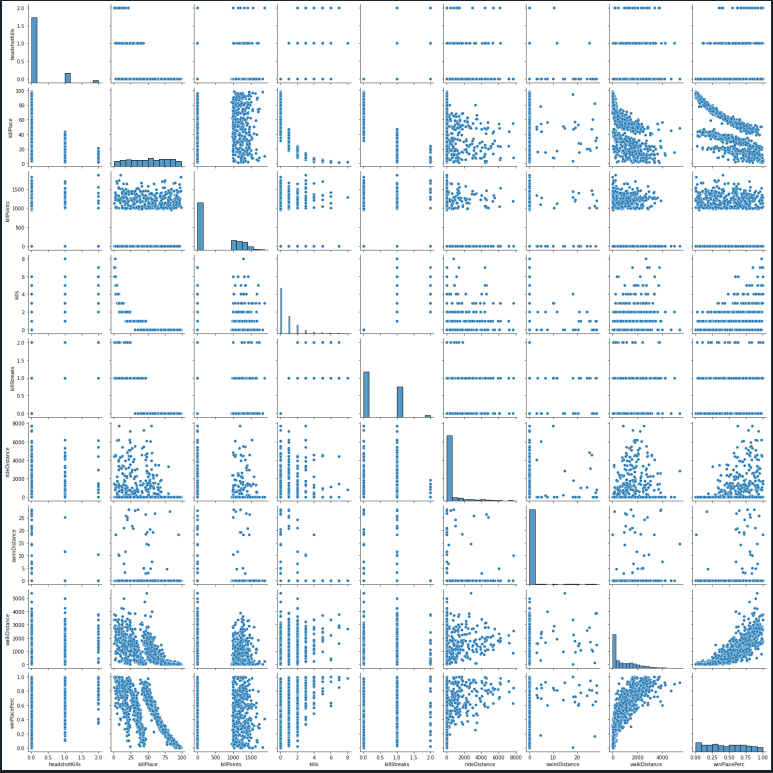
dpVar = "winPlacePerc"

varList = list(squadTr.columns)

varList.remove(dpVar)

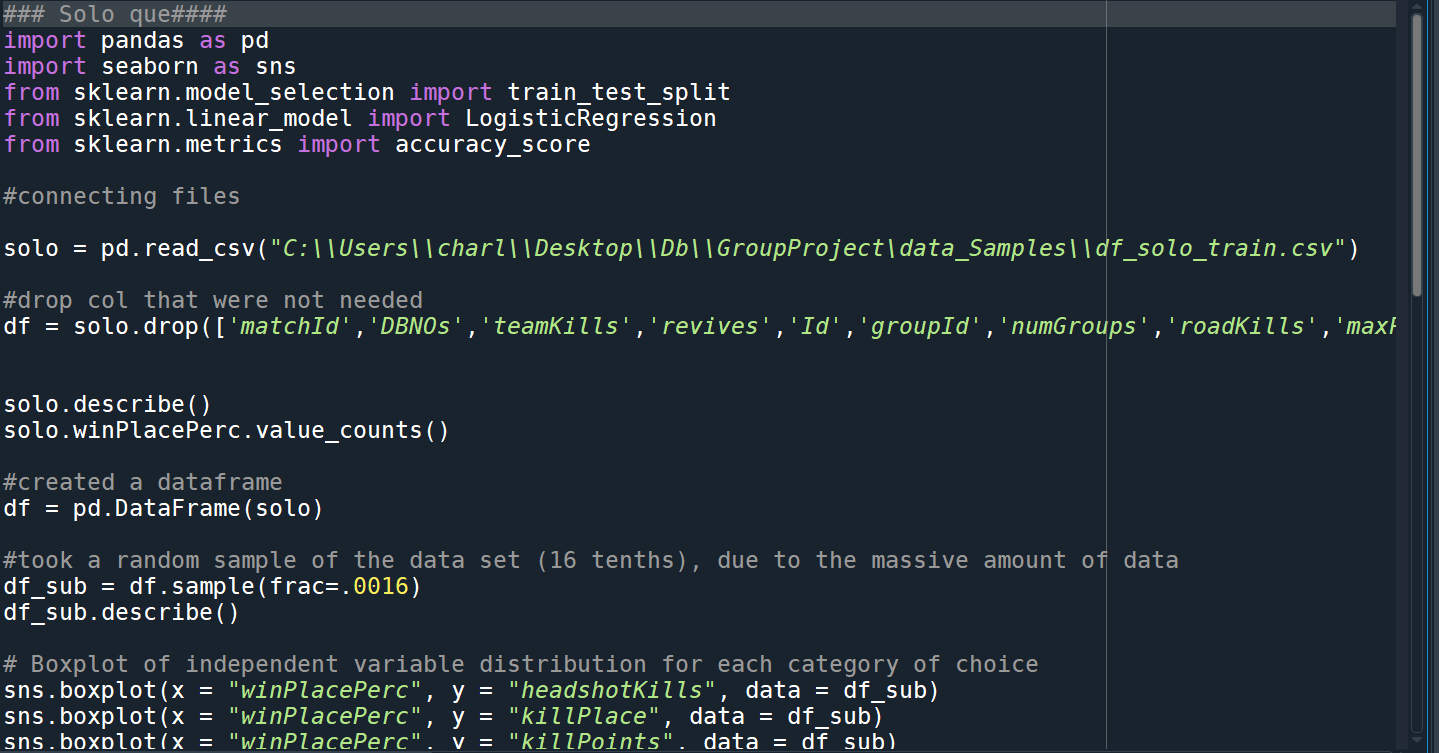
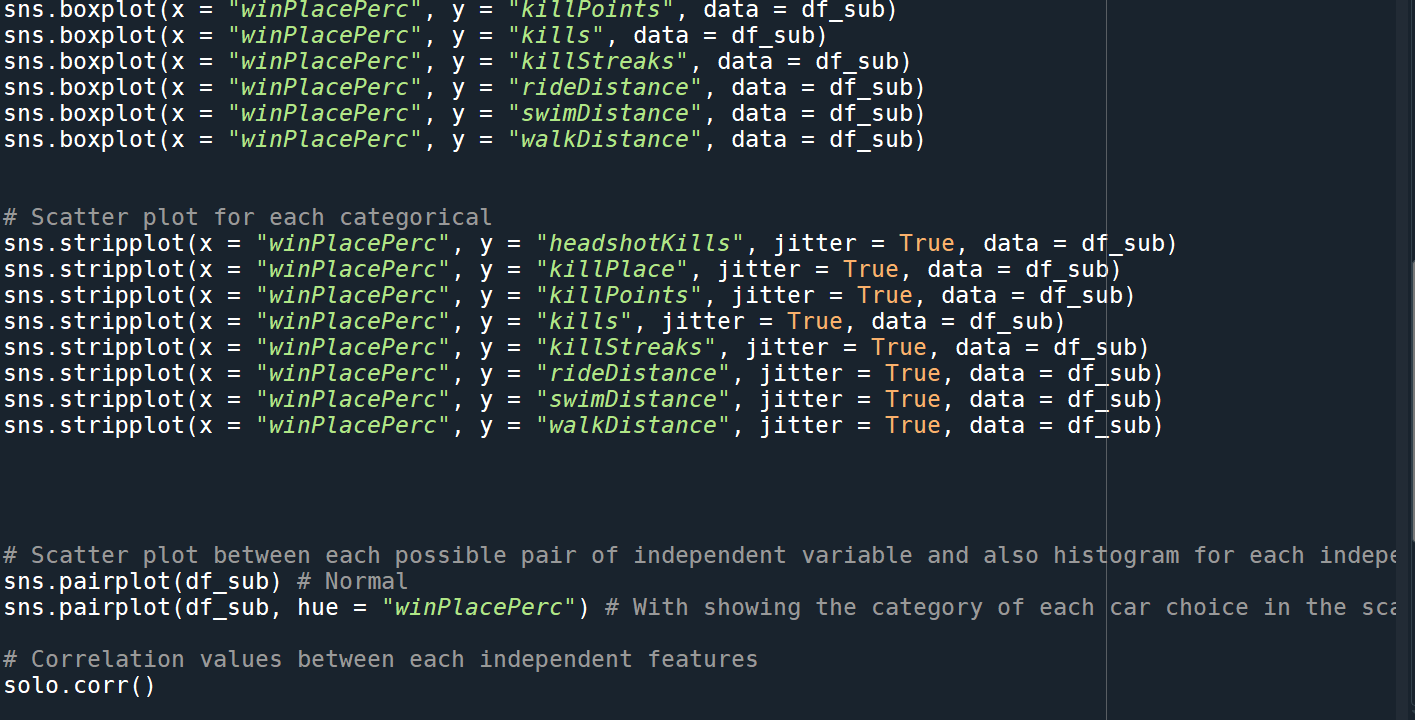
squadVIFs = genVIFs(squadTr, varList)

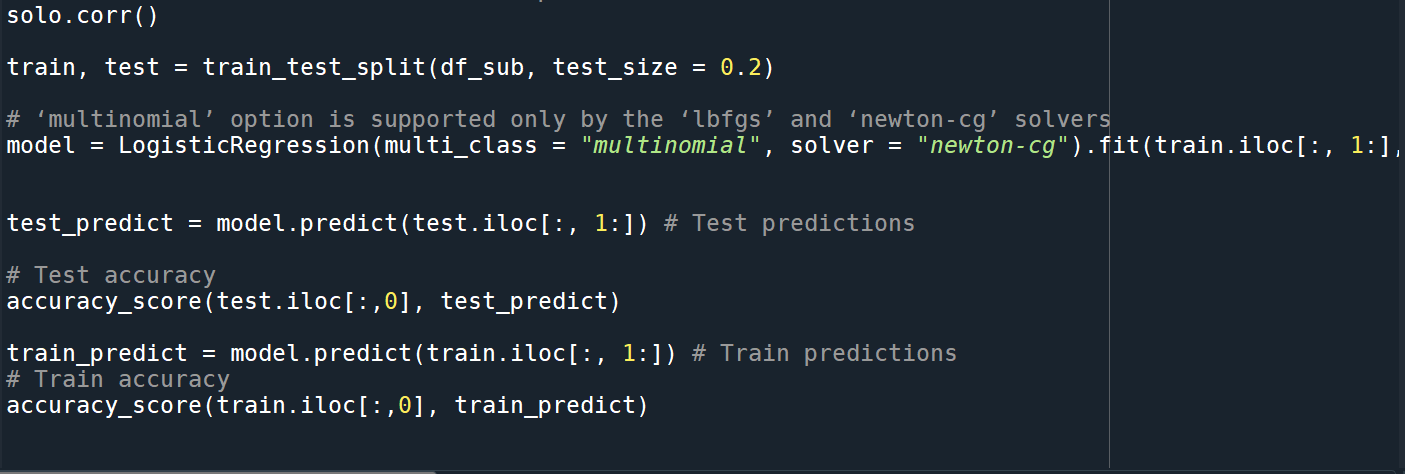
**Model Building**

**Multi-Regression**

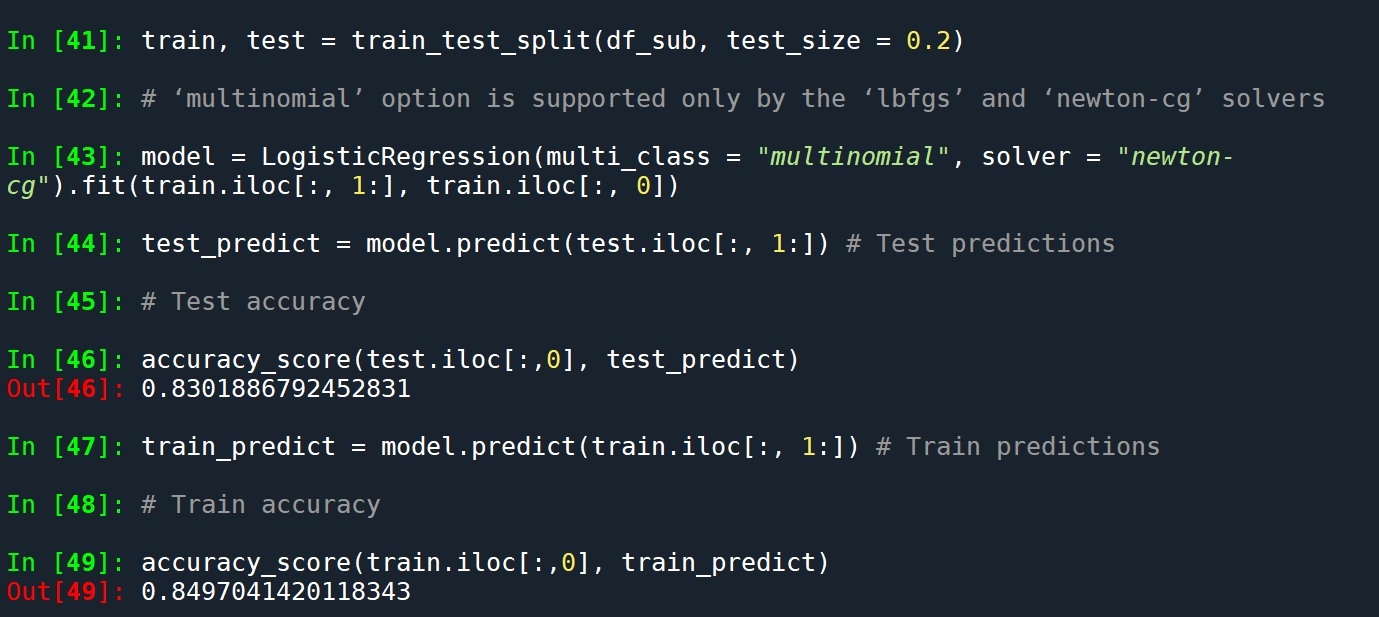
***Selected model: Multi-Regression***

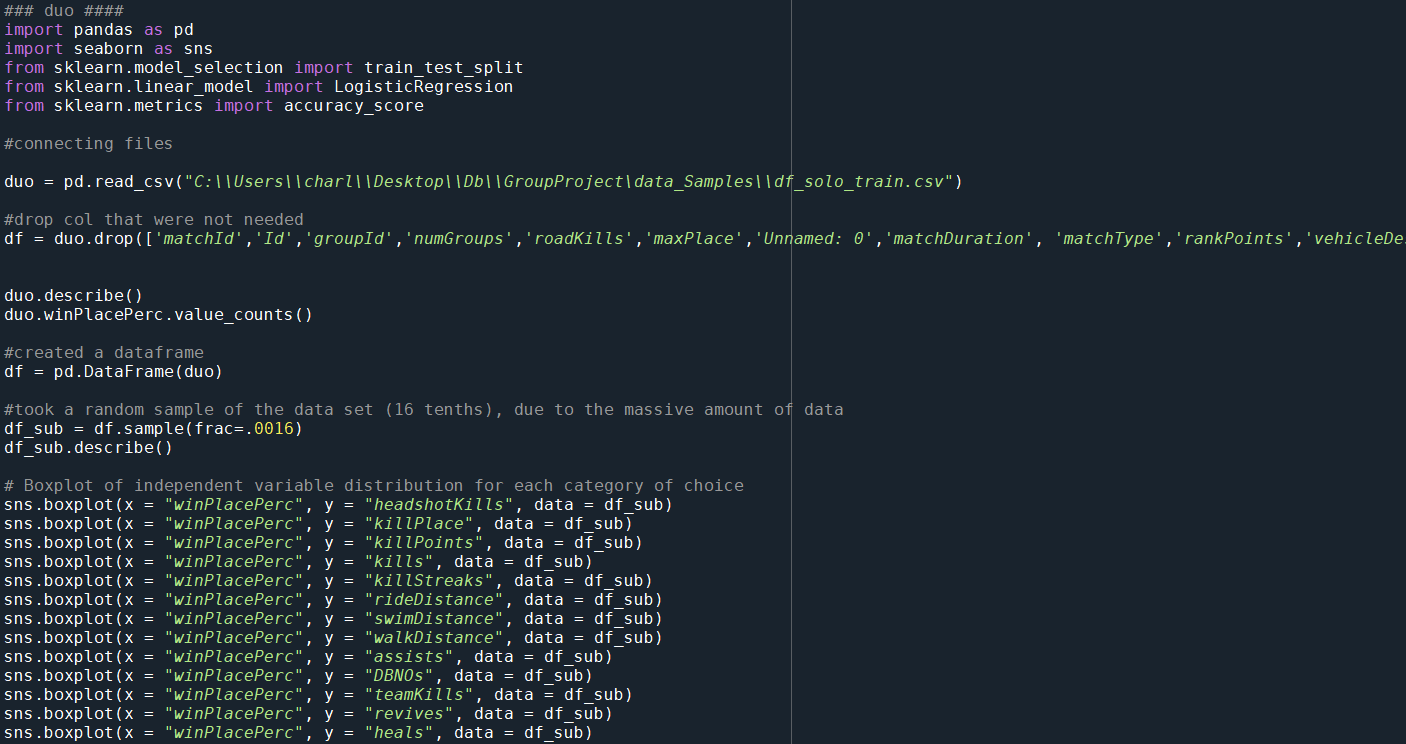
Using Multi-Regression techniques we tested many variables that could be used to factor the winning chance of a game of PUBG. Our main goal was to test to see if getting more kills or/ and traveling more could increase your chances of winning. We tested these theory with multiple game modes such as solo,which is just yourself playing again other single players .Duo, where you and one other player(friend or random) play against other two man teams.And Squad, where you and 3 other players (friend or random) play against other squads.

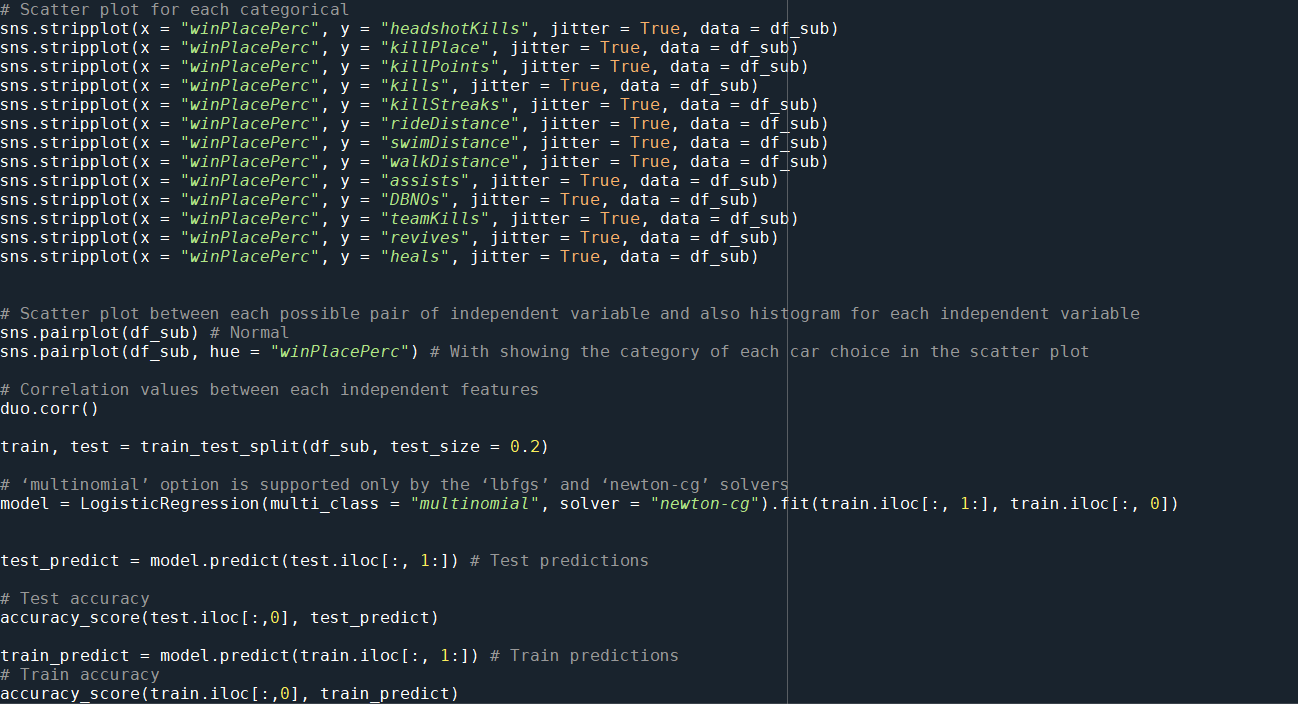
**Solo**

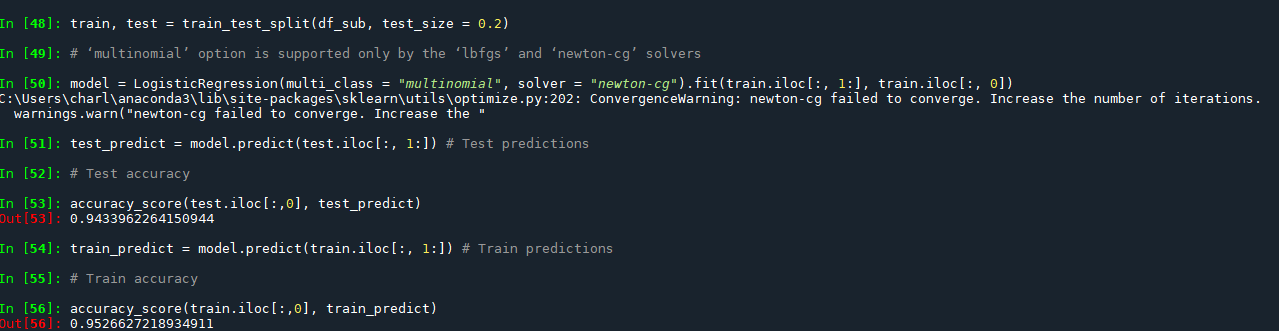


**OutPut:**

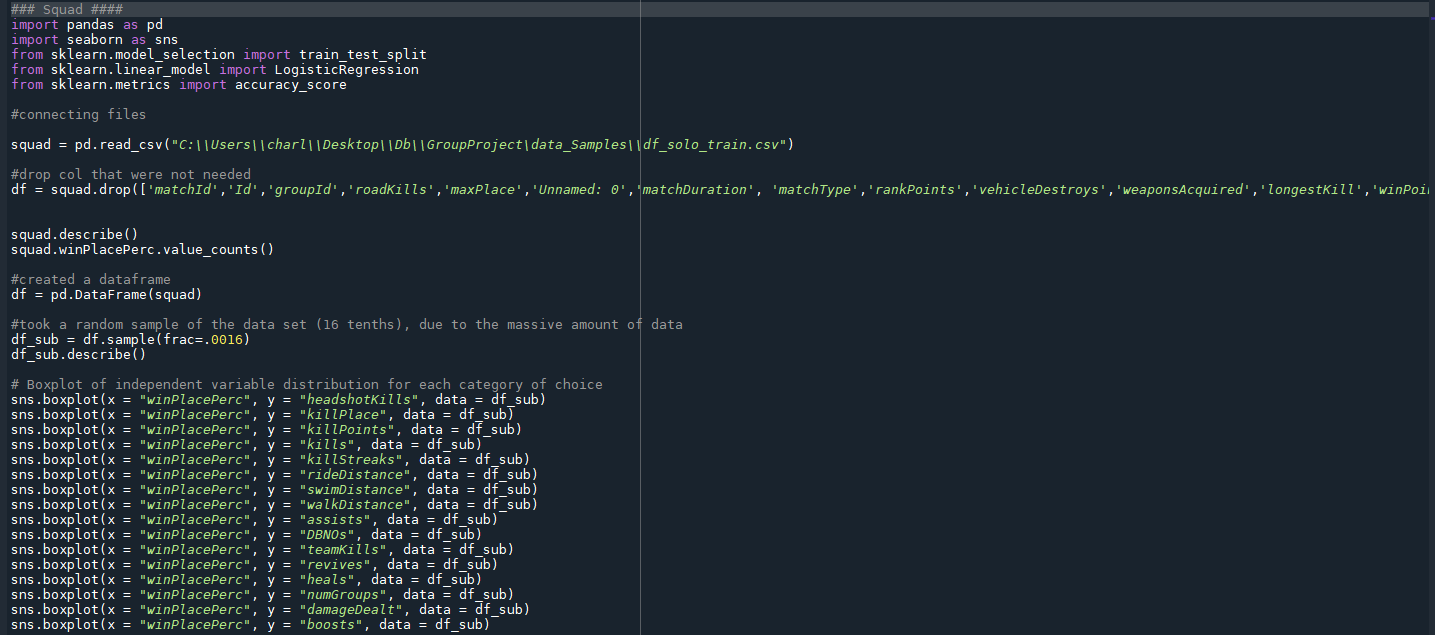


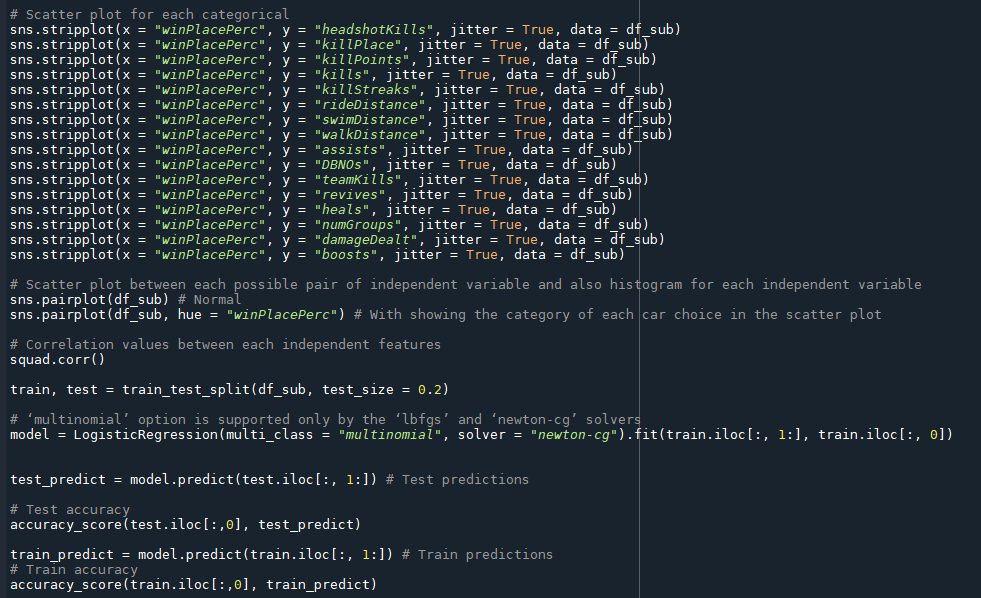
**DUO**

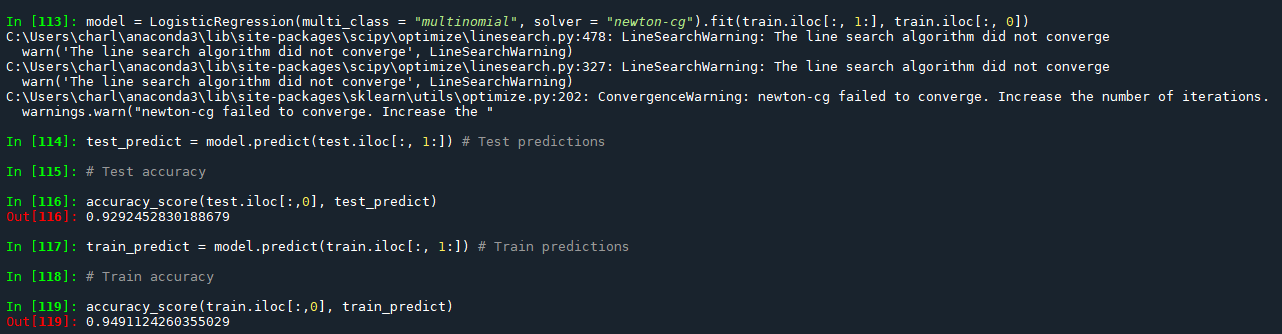


**OutPut:**

**Squad:**

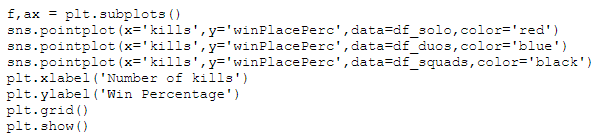
****

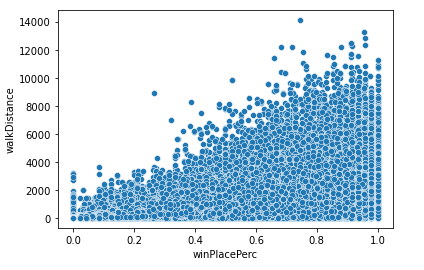
****

**OutPut:**

**Analysis of Impact of Certain Features on Outcomes**

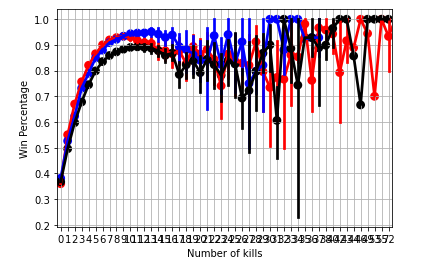
plotting winning chances of winning by walking distance:





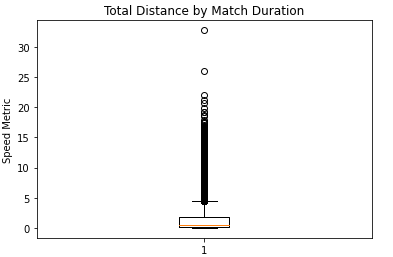
Plotting winning percentage by more kills:

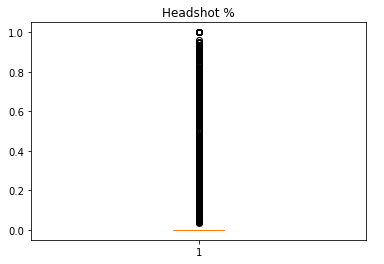




**Predicting Fraudulent Play**

Analysis were done on the basis of the charts below:



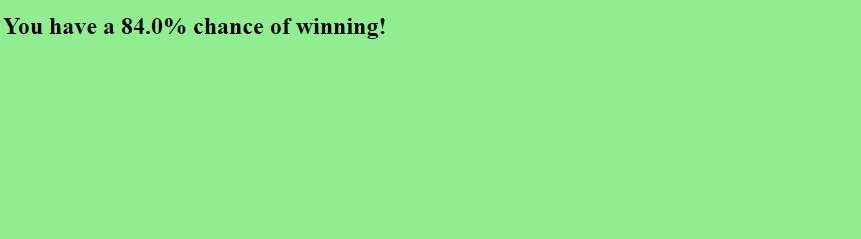
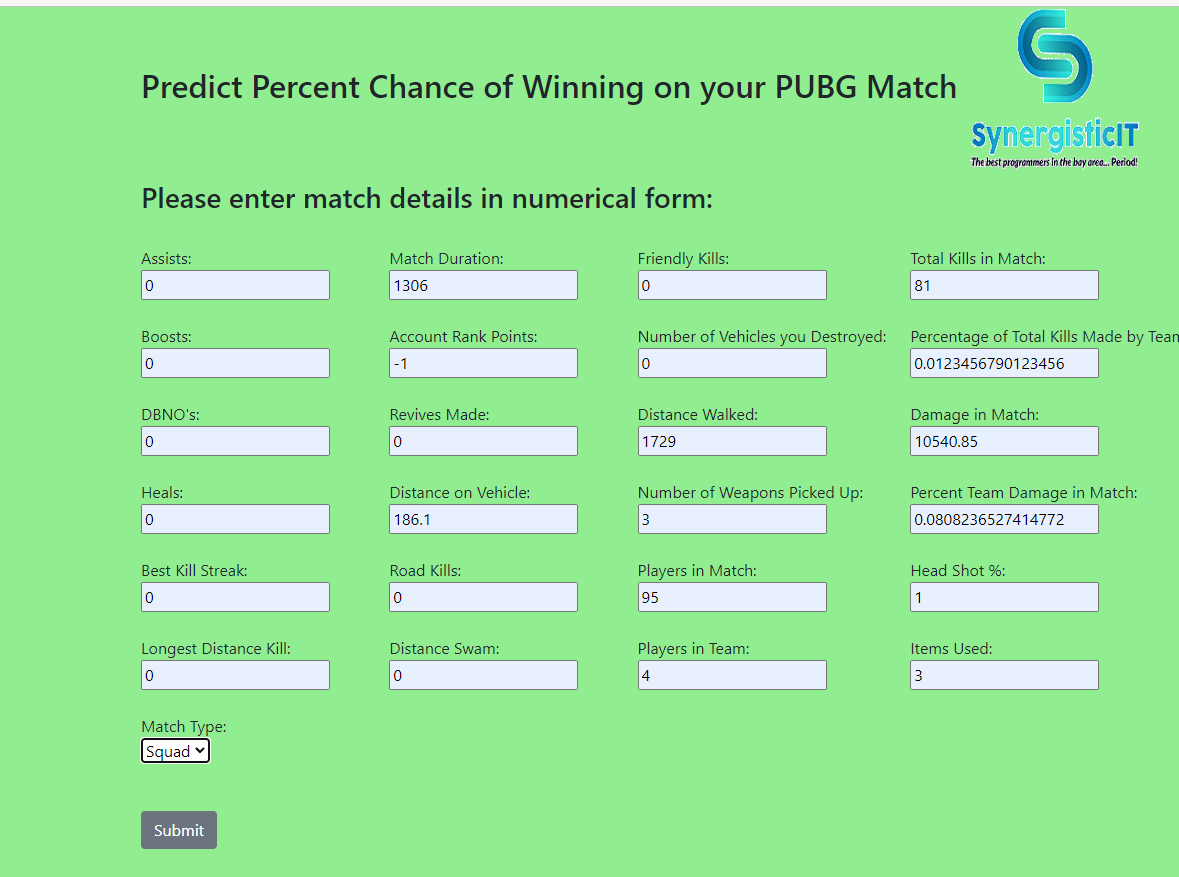
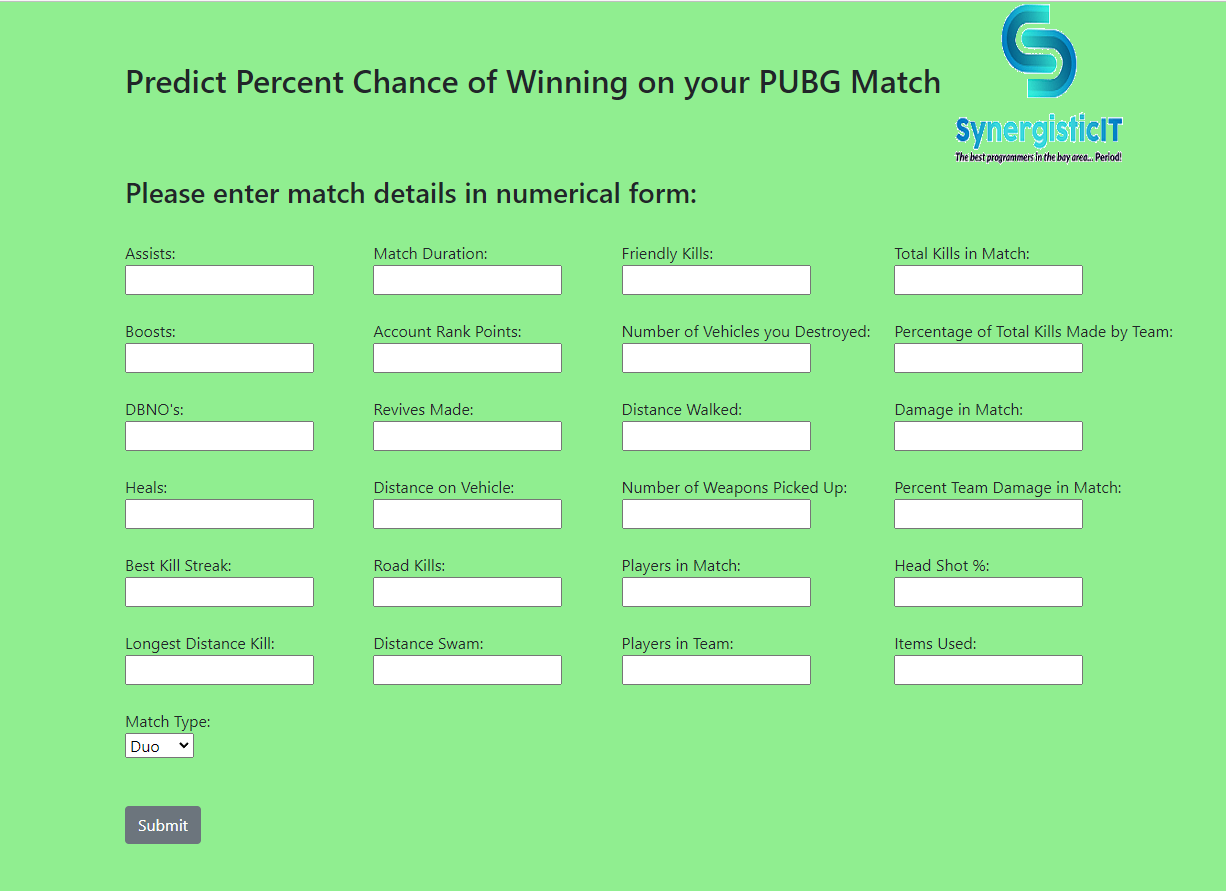


**Inferences from the above charts:**

* Simply using a distance outlier metric wasn’t going to be enough as it didn’t account for match duration or time alive.
* Total distance by match duration fixed this and allowed for a more clear metric that accounted for short and long matches alike.
* Speed Metric scores above approximately 6.5 should be further investigated for cheating.
* Headshot % is a little less straightforward on catching cheaters as it needs to be used in tandem with something else to be effective. I would suggest either number of kills with image recognition software.
* Any person who gets four or more kills in a game with a headshot % of over .8 should be flagged for image recognition software usage.
* This software would check for quick flicks in a player's view suggesting a machine controlling the aim on a player's controls.

**Deployment using Django**

The below images show the working deployment of the aforementioned models that were built on the three data sets:



**Notes on the above images:**

* The above prediction was taken from the test dataset which was unseen by our models.
* The prediction came out to 84% as seen above and the actual record estimate was 82.3% meaning our model came within 1.7% on a random data point.