EDA for the popular battle royale game PUBG

This kernel features:

- The Killers (https://www.kaggle.com/deffro/eda-is-fun#The-Killers)
- The Runners (https://www.kaggle.com/deffro/eda-is-fun#The-Runners)
- The Drivers (https://www.kaggle.com/deffro/eda-is-fun#The-Drivers)
- The Swimmers (https://www.kaggle.com/deffro/eda-is-fun#The-Swimmers)
- The Healers (https://www.kaggle.com/deffro/eda-is-fun#The-Healers)
- Solos, Duos and Squads (https://www.kaggle.com/deffro/eda-is-fun#Solos,-Duos-and-Squads)
- Correlation (https://www.kaggle.com/deffro/eda-is-fun#Pearson-correlation-between-variables)
- Feature Engineering (https://www.kaggle.com/deffro/eda-is-fun#Feature-Engineering)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
train = pd.read_csv('../input/train.csv')
```

In [3]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 4357336 entries, 0 to 4357335
Data columns (total 26 columns):
Id                int64
groupId           int64
matchId           int64
assists           int64
boosts            int64
damageDealt       float64
DBNOs             int64
headshotKills     int64
heals             int64
killPlace         int64
killPoints        int64
kills             int64
killStreaks       int64
longestKill       float64
maxPlace          int64
numGroups         int64
revives           int64
rideDistance      float64
roadKills         int64
swimDistance      float64
teamKills         int64
```

EDA is Fun!

Python notebook using data from PUBG Finish Placement Prediction (Kernels Only) · 47,583 views · 1y ago · data visualization, eda, feature engineering

986 Copy and Edit 874

```
winPlacePerc      float64
dtypes: float64(6), int64(20)
memory usage: 864.3 MB
```

- **groupId** - Integer ID to identify a group within a match. If the same group of players plays in different matches, they will have a different groupId each time.
- **matchId** - Integer ID to identify match. There are no matches that are in both the training and testing set.
- **assists** - Number of enemy players this player damaged that were killed by teammates.
- **boosts** - Number of boost items used.
- **damageDealt** - Total damage dealt. Note: Self inflicted damage is subtracted.
- **DBNOs** - Number of enemy players knocked.
- **headshotKills** - Number of enemy players killed with headshots.
- **heals** - Number of healing items used.
- **killPlace** - Ranking in match of number of enemy players killed.
- **killPoints** - Kills-based external ranking of player. (Think of this as an Elo ranking where only kills matter.)
- **kills** - Number of enemy players killed.
- **killStreaks** - Max number of enemy players killed in a short amount of time.
- **longestKill** - Longest distance between player and player killed at time of death. This may be misleading, as downing a - player and driving away may lead to a large longestKill stat.
- **maxPlace** - Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
- **numGroups** - Number of groups we have data for in the match.
- **revives** - Number of times this player revived teammates.
- **rideDistance** - Total distance traveled in vehicles measured in meters.
- **roadKills** - Number of kills while in a vehicle.
- **swimDistance** - Total distance traveled by swimming measured in meters.
- **teamKills** - Number of times this player killed a teammate.

- **teamKills** - Number of times this player killed a teammate.
- **vehicleDestroys** - Number of vehicles destroyed.
- **walkDistance** - Total distance traveled on foot measured in meters.
- **weaponsAcquired** - Number of weapons picked up.
- **winPoints** - Win-based external ranking of player. (Think of this as an Elo ranking where only winning matters.)
- **winPlacePerc** - The target of prediction. This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match. It is calculated off of maxPlace, not numGroups, so it is possible to have missing chunks in a match.

In [4]:
```
train.head()
```

Out[4]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace
0	0	24	0	0	5	247.30	2	0	4	17
1	1	440875	1	1	0	37.65	1	1	0	45
2	2	878242	2	0	1	93.73	1	0	2	54
3	3	1319841	3	0	0	95.88	0	0	0	86
4	4	1757883	4	0	1	0.00	0	0	1	58

The Killers

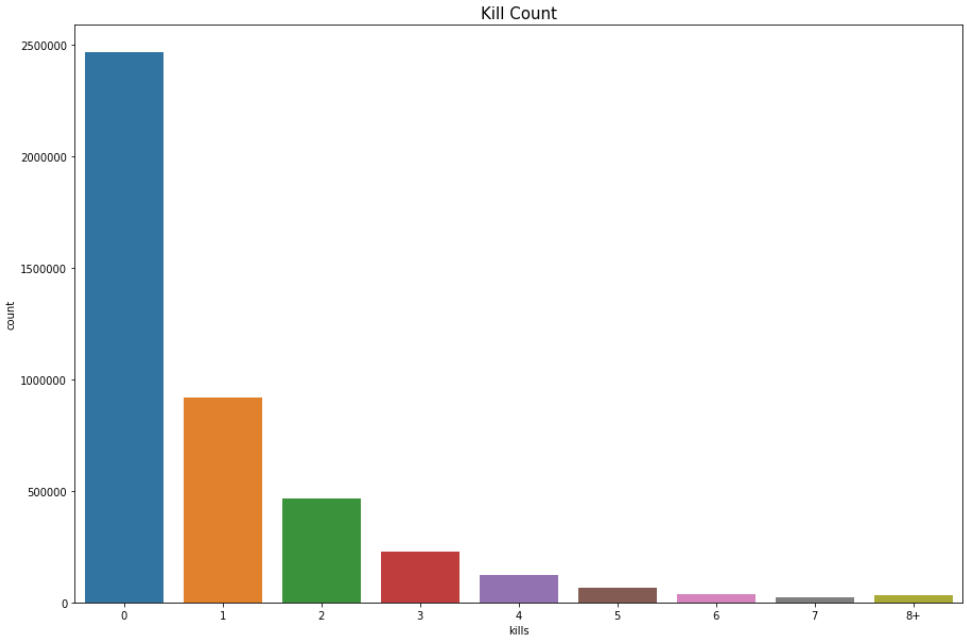
In [5]:
```
print("The average person kills {:.4f} players, 99% of people have {}
 kills or less, while the most kills ever recorded is {}.".format(trai
n['kills'].mean(),train['kills'].quantile(0.99), train['kills'].max
()))
```

```
The average person kills 0.9345 players, 99% of people have 7.0 kills
or less, while the most kills ever recorded is 60.
```

Let's plot the kill counts.

In [6]:

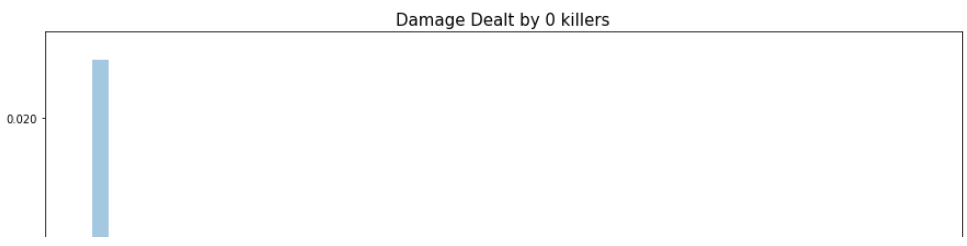
```
data = train.copy()
data.loc[data['kills'] > data['kills'].quantile(0.99)] = '8+'
plt.figure(figsize=(15,10))
sns.countplot(data['kills'].astype('str').sort_values())
plt.title("Kill Count",fontsize=15)
plt.show()
```

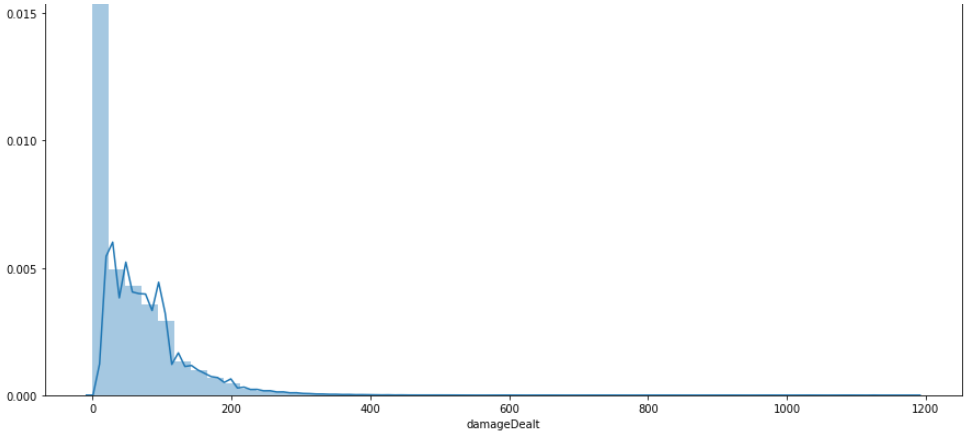


Most people can't make a single kill. At least do they do damage?

In [7]:

```
data = train.copy()
data = data[data['kills']==0]
plt.figure(figsize=(15,10))
plt.title("Damage Dealt by 0 killers",fontsize=15)
sns.distplot(data['damageDealt'])
plt.show()
```





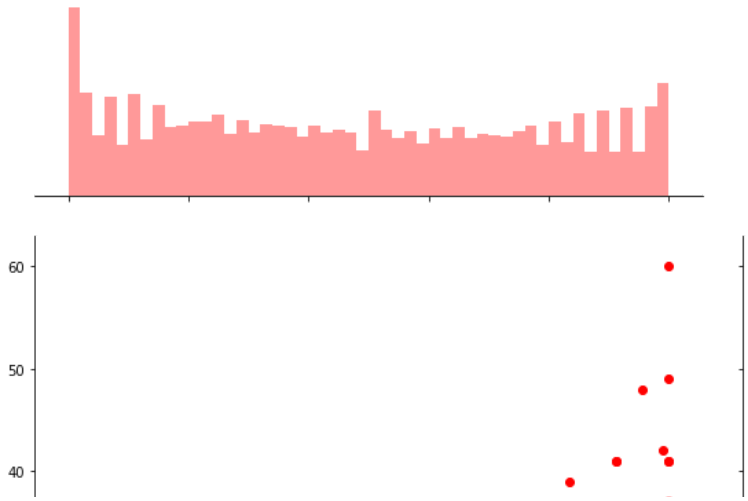
Well, most of them don't. Let's investigate the exceptions.

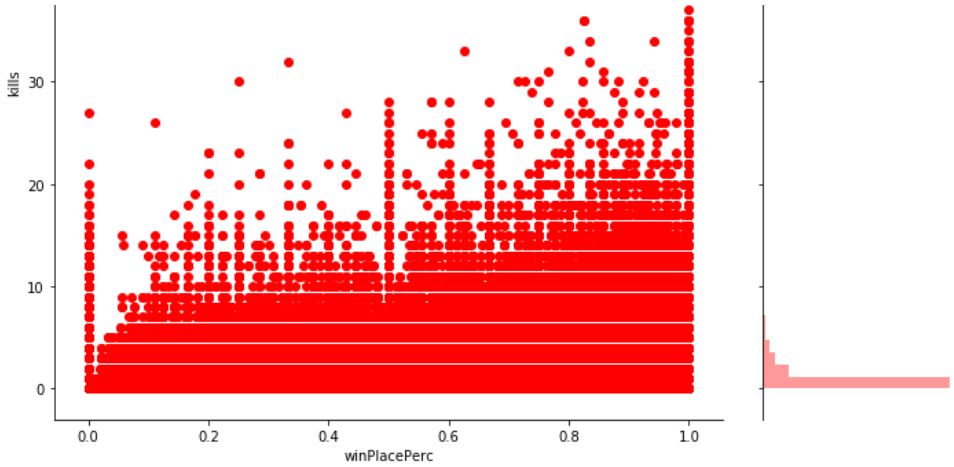
In [8]:
```
print("{} players ({:.4f}%) have won without a single kill!".format(le
n(data[data['winPlacePerc']==1]), 100*len(data[data['winPlacePerc']==1
])/len(train)))

data1 = train[train['damageDealt'] == 0].copy()
print("{} players ({:.4f}%) have won without dealing damage!".format(l
en(data1[data1['winPlacePerc']==1]), 100*len(data1[data1['winPlacePer
c']==1])/len(train)))
```

```
16722 players (0.3838%) have won without a single kill!
4262 players (0.0978%) have won without dealing damage!
```

Plot win placement percentage vs kills.

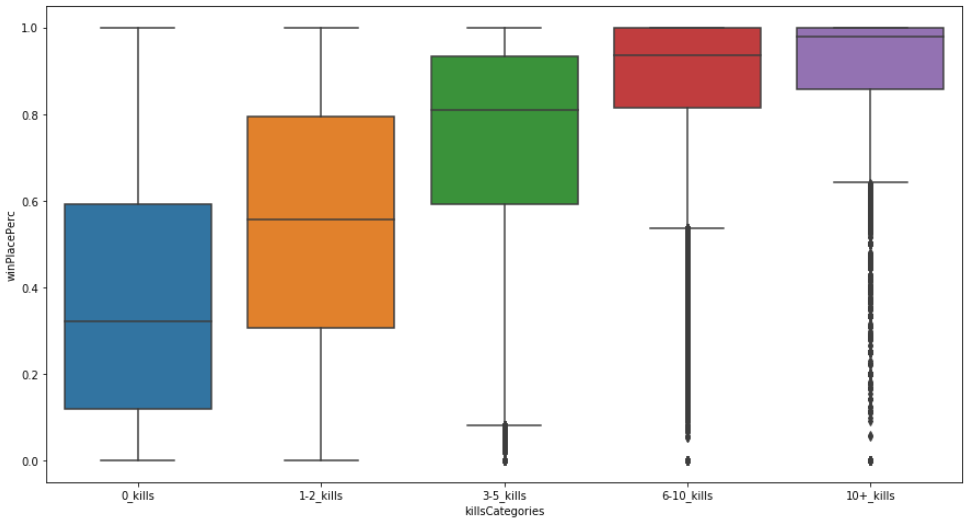
In [9]:
```
sns.jointplot(x="winPlacePerc", y="kills", data=train, height=10, rati
o=3, color="r")
plt.show()
```



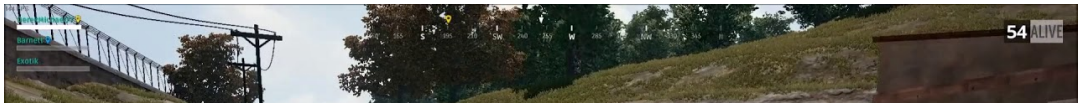


Apparentrly killing has a correlation with winning. Finally let's group players based on kills (0 kills, 1-2 kills, 3-5 kills, 6-10 kills and 10+ kills).

In [10]:
```
kills = train.copy()

kills['killsCategories'] = pd.cut(kills['kills'], [-1, 0, 2, 5, 10, 60
], labels=['0_kills','1-2_kills', '3-5_kills', '6-10_kills', '10+_kill
s'])

plt.figure(figsize=(15,8))
sns.boxplot(x="killsCategories", y="winPlacePerc", data=kills)
plt.show()
```



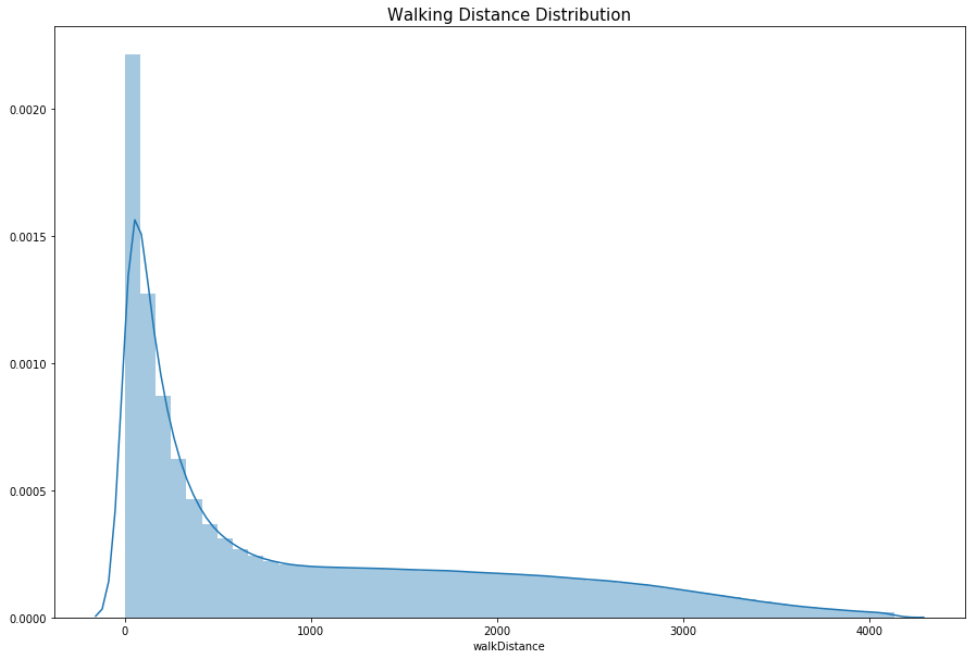
The Runners



In [11]:
```
print("The average person walks for {:.1f}m, 99% of people have walked {}m or less, while the marathoner champion walked for {}m.".format(train['walkDistance'].mean(), train['walkDistance'].quantile(0.99), train['walkDistance'].max()))
```

```
The average person walks for 1055.1m, 99% of people have walked 4138.0m or less, while the marathoner champion walked for 17300.0m.
```

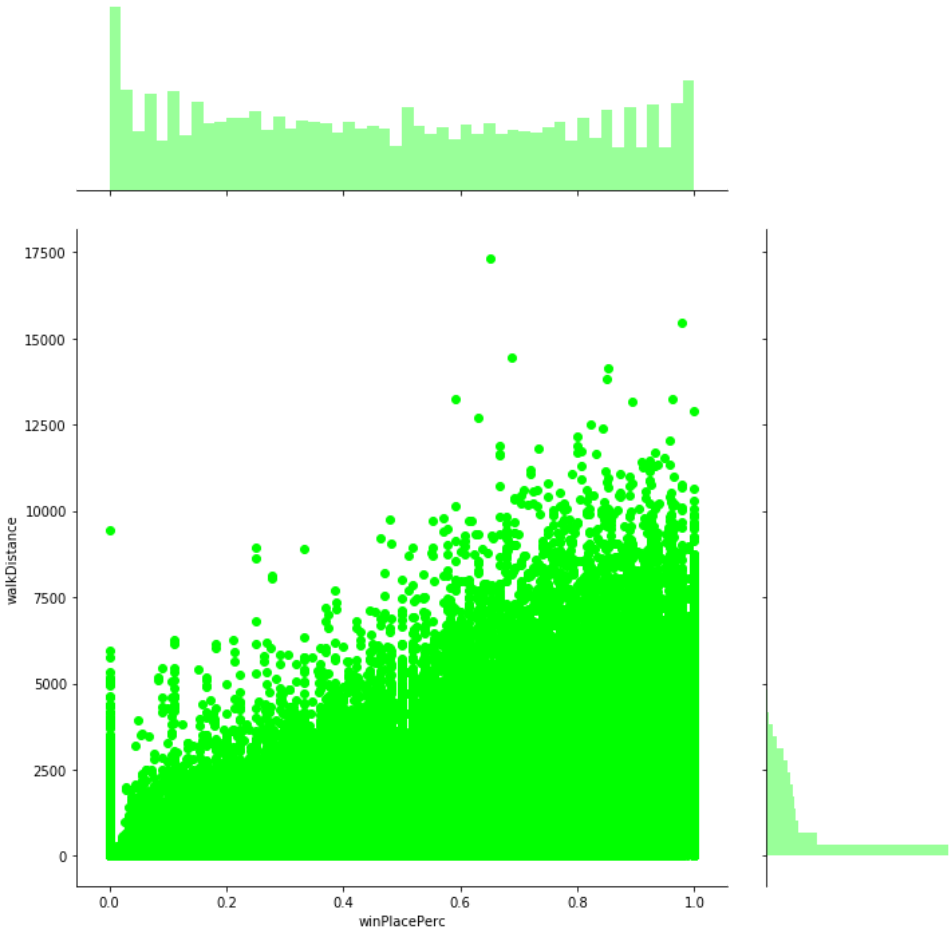
In [12]:
```
data = train.copy()
data = data[data['walkDistance'] < train['walkDistance'].quantile(0.99)]
plt.figure(figsize=(15,10))
plt.title("Walking Distance Distribution",fontsize=15)
sns.distplot(data['walkDistance'])
plt.show()
```



In [13]:
```
print("{} players ({:.4f}%) walked 0 meters. This means that they die
 before even taking a step or they are afk (more possible).".format(le
n(data[data['walkDistance'] == 0]), 100*len(data1[data1['walkDistance'
]==0])/len(train)))
```

```
94306 players (2.0581%) walked 0 meters. This means that they die befo
re even taking a step or they are afk (more possible).
```

In [14]:
```
sns.jointplot(x="winPlacePerc", y="walkDistance",  data=train, height=
10, ratio=3, color="lime")
plt.show()
```



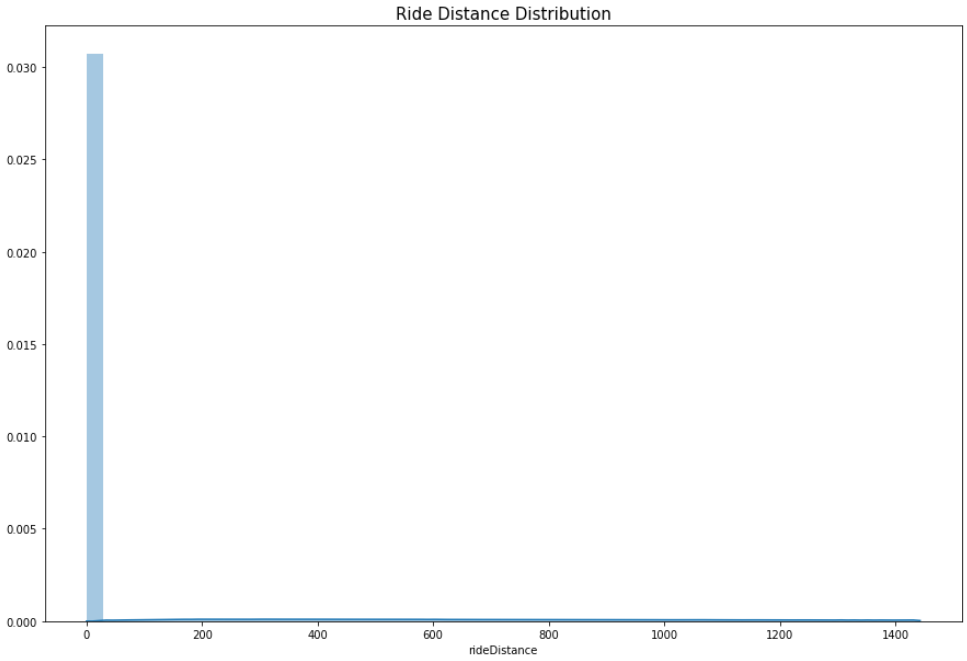
Apparently walking has a high correlation with winPlacePerc.

The Drivers

In [15]:
```
print("The average person drives for {:.1f}m, 99% of people have drive
d {}m or less, while the formula 1 champion drived for {}m.".format(tr
ain['rideDistance'].mean(), train['rideDistance'].quantile(0.99), trai
n['rideDistance'].max()))
```

```
The average person drives for 423.9m, 99% of people have drived 6133.0
m or less, while the formula 1 champion drived for 48390.0m.
```

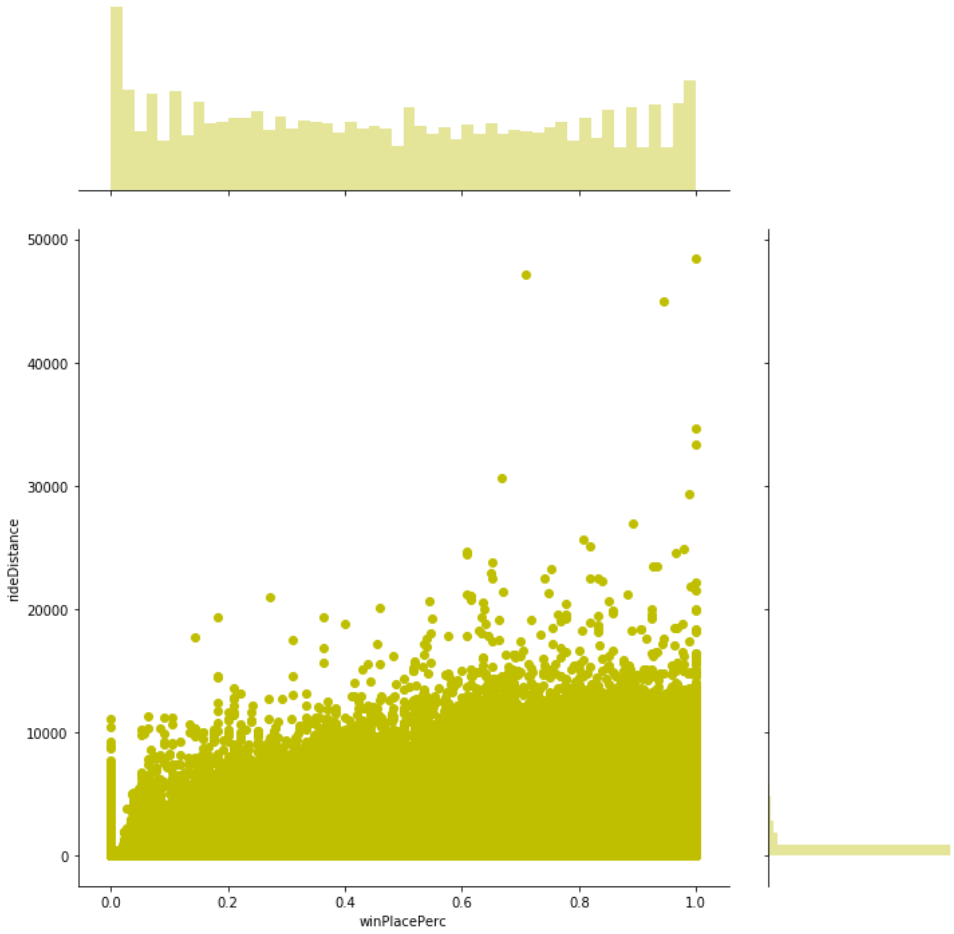
In [16]:
```
data = train.copy()
data = data[data['rideDistance'] < train['rideDistance'].quantile(0.9
)]
plt.figure(figsize=(15,10))
plt.title("Ride Distance Distribution",fontsize=15)
sns.distplot(data['rideDistance'])
plt.show()
```



In [17]:
```
print("{} players ({:.4f}%) drived for 0 meters. This means that they
 don't have a driving licence yet.".format(len(data[data['rideDistanc
e'] == 0]), 100*len(data1[data1['rideDistance']==0])/len(train)))
```

```
3439985 players (22.7940%) drived for 0 meters. This means that they d
on't have a driving licence yet.
```

In [18]:
```
sns.jointplot(x="winPlacePerc", y="rideDistance", data=train, height=1
0, ratio=3, color="y")
plt.show()
```

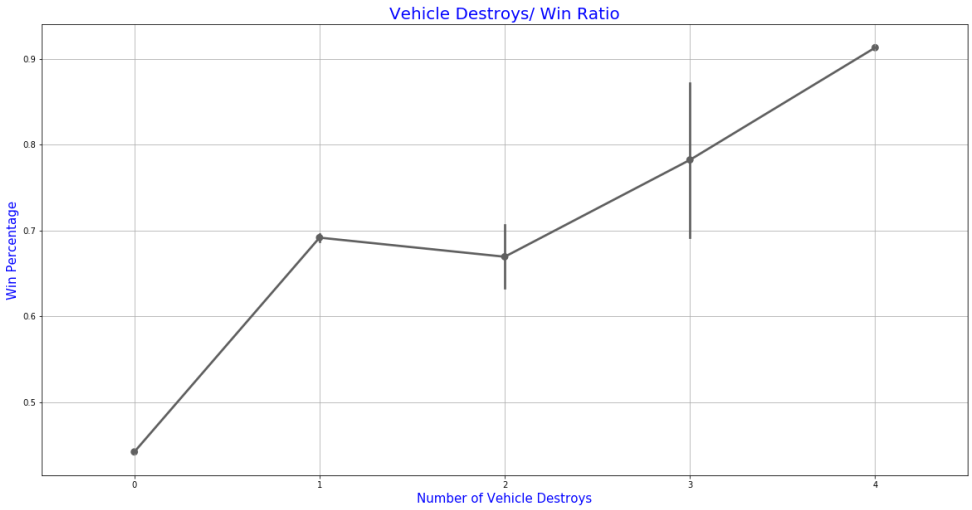


There is a small correlation between rideDistance and winPlacePerc.

Destroying a vehicle in my experience shows that a player has skills. Let's check it.

In [19]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='vehicleDestroys',y='winPlacePerc',data=data,color='#6
```

```
06060',alpha=0.8)
plt.xlabel('Number of Vehicle Destroys',fontsize = 15,color='blue')
plt.ylabel('Win Percentage',fontsize = 15,color='blue')
plt.title('Vehicle Destroys/ Win Ratio',fontsize = 20,color='blue')
plt.grid()
plt.show()
```



My experience was correct. Destroying a single vehicle increases your chances of winning!

The Swimmers

In [20]:

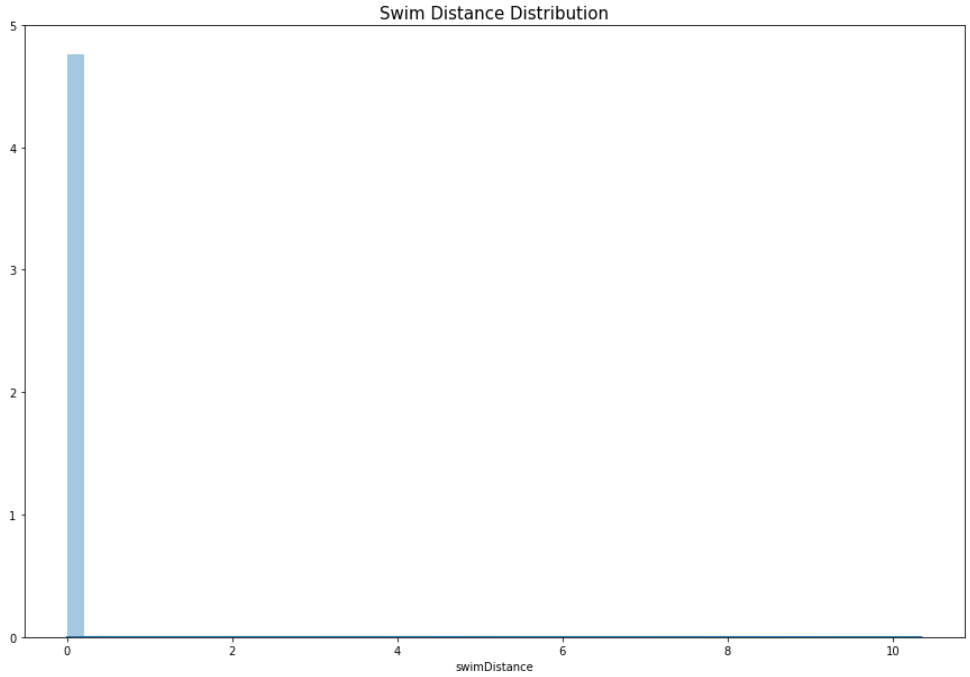
```
print("The average person swims for {:.1f}m, 99% of people have swimem
d {}m or less, while the olympic champion swimmed for {}m.".format(tra
in['swimDistance'].mean(), train['swimDistance'].quantile(0.99), train
```

```
['swimDistance'].max()))
```

```
The average person swims for 4.1m, 99% of people have swimemd 116.1999
9999999999m or less, while the olympic champion swimmed for 5286.0m.
```

In [21]:

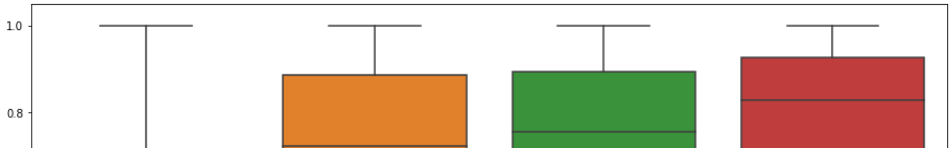
```
data = train.copy()
data = data[data['swimDistance'] < train['swimDistance'].quantile(0.95
)]
plt.figure(figsize=(15,10))
plt.title("Swim Distance Distribution",fontsize=15)
sns.distplot(data['swimDistance'])
plt.show()
```

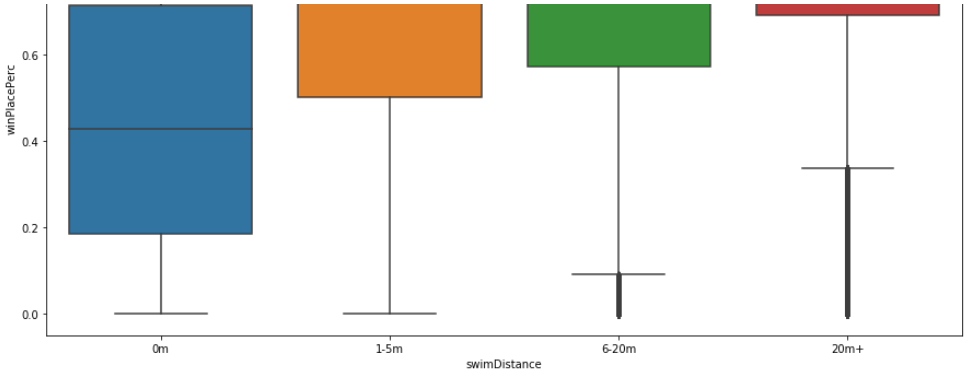


Almost no one swims. Let's group the swimming distances in 4 categories and plot vs winPlacePerc.

In [22]:

```
swim = train.copy()

swim['swimDistance'] = pd.cut(swim['swimDistance'], [-1, 0, 5, 20, 528
6], labels=['0m','1-5m', '6-20m', '20m+'])

plt.figure(figsize=(15,8))
sns.boxplot(x="swimDistance", y="winPlacePerc", data=swim)
plt.show()
```





It seems that if you swim, you rise to the top. In PUBG there are currently 3 maps. One of them has almost no water. Keep that in mind. I might plan on doing analysis to find out in which map a match is played.

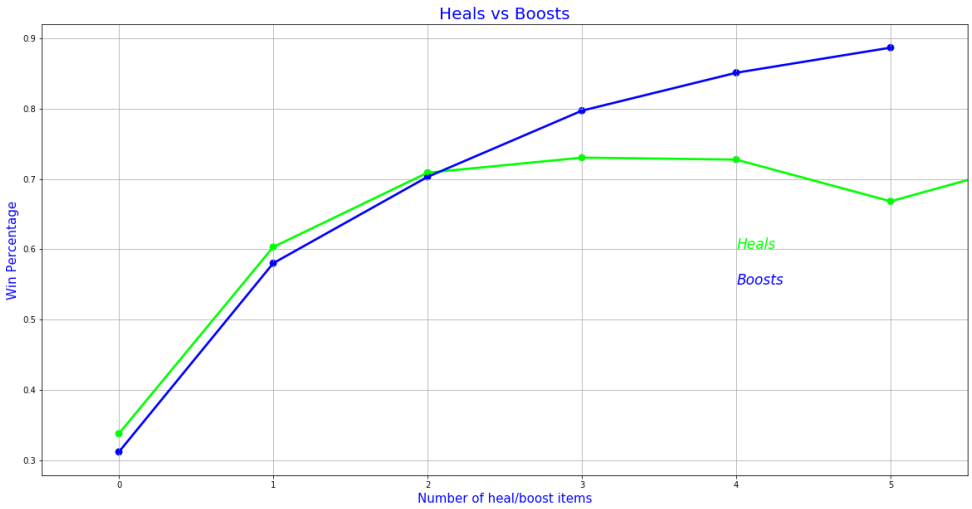
The Healers

In [23]:
```
print("The average person uses {:.1f} heal items, 99% of people use {}
or less, while the doctor used {}.".format(train['heals'].mean(), trai
n['heals'].quantile(0.99), train['heals'].max()))
print("The average person uses {:.1f} boost items, 99% of people use
{} or less, while the doctor used {}.".format(train['boosts'].mean(),
train['boosts'].quantile(0.99), train['boosts'].max()))
```

```
The average person uses 1.2 heal items, 99% of people use 11.0 or les
s, while the doctor used 59.
The average person uses 1.0 boost items, 99% of people use 7.0 or les
s, while the doctor used 18.
```

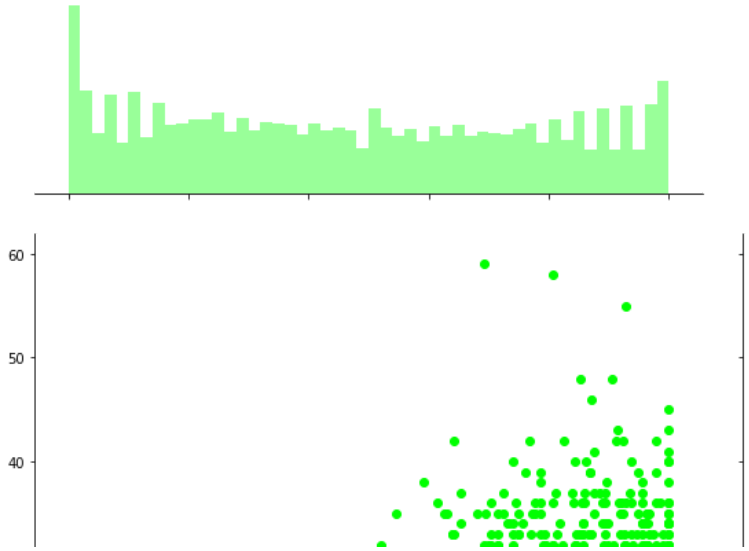
In [24]:

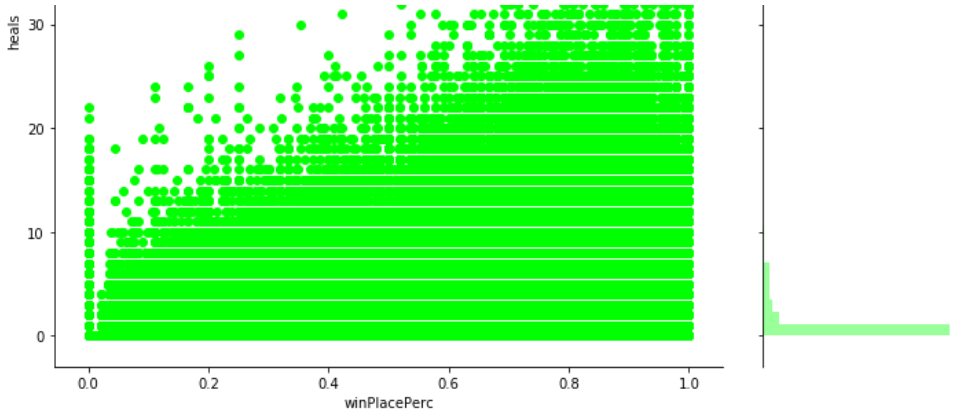
```
data = train.copy()
data = data[data['heals'] < data['heals'].quantile(0.99)]
data = data[data['boosts'] < data['boosts'].quantile(0.99)]

f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='heals',y='winPlacePerc',data=data,color='lime',alpha=
0.8)
sns.pointplot(x='boosts',y='winPlacePerc',data=data,color='blue',alpha
=0.8)
plt.text(4,0.6,'Heals',color='lime',fontsize = 17,style = 'italic')
plt.text(4,0.55,'Boosts',color='blue',fontsize = 17,style = 'italic')
plt.xlabel('Number of heal/boost items',fontsize = 15,color='blue')
plt.ylabel('Win Percentage',fontsize = 15,color='blue')
plt.title('Heals vs Boosts',fontsize = 20,color='blue')
plt.grid()
plt.show()
```



In [25]:

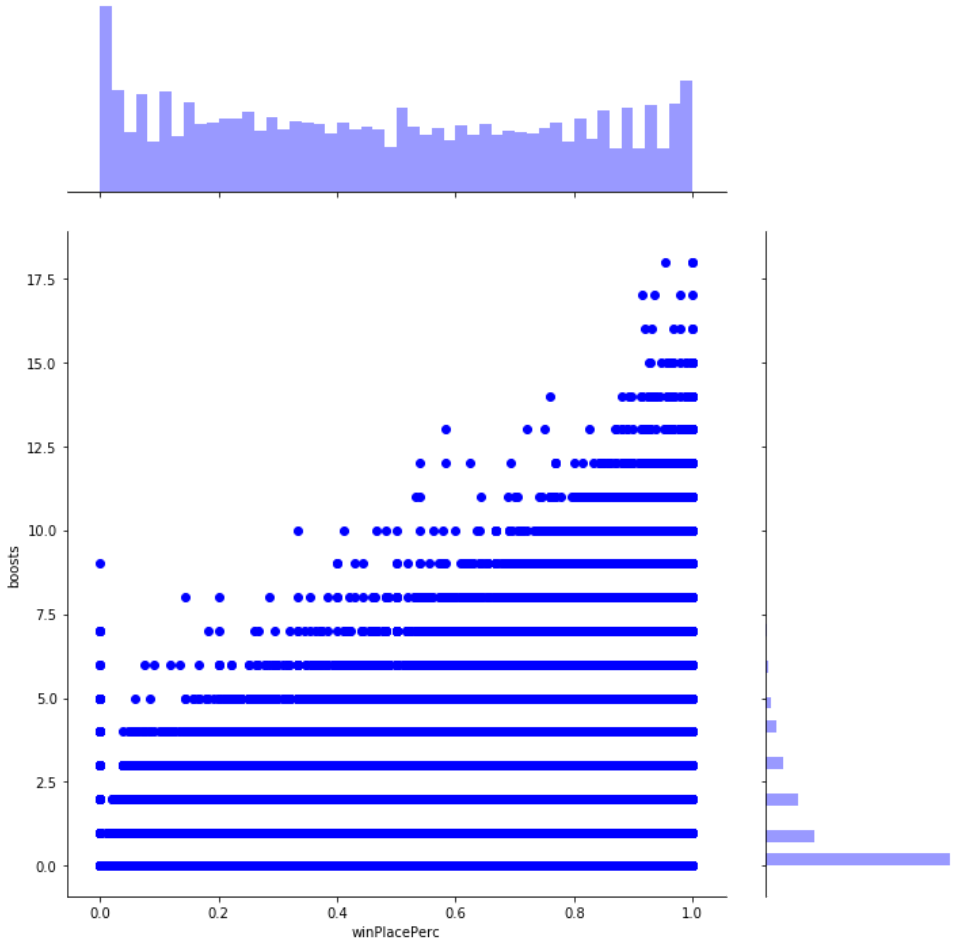
```
sns.jointplot(x="winPlacePerc", y="heals", data=train, height=10, rati
o=3, color="lime")
plt.show()
```





In [26]:

```
sns.jointplot(x="winPlacePerc", y="boosts", data=train, height=10, rat
io=3, color="blue")
plt.show()
```



So healing and boosting, definitely are correlated with winPlacePerc. Boosting is more.

In every plot, there is an abnormal behavior when values are 0.

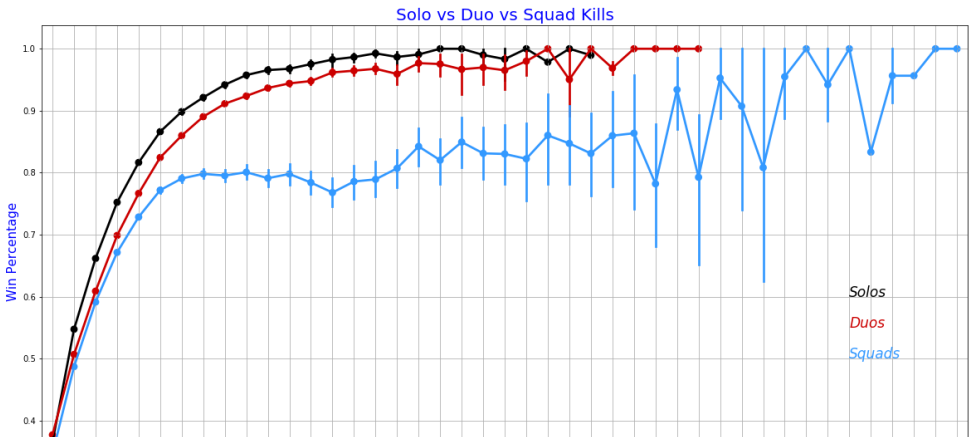
Solos, Duos and Squads

There are 3 game modes in the game. One can play solo, or with a friend (duo), or with 3 other friends (squad). 100 players join the same server, so in the case of duos the max teams are 50 and in the case of squads the max teams are 25.

In [27]:
```
solos = train[train['numGroups']>50]
duos = train[(train['numGroups']>25) & (train['numGroups']<=50)]
squads = train[train['numGroups']<=25]
print("There are {} ({:.2f}%) solo games, {} ({:.2f}%) duo games and
{} ({:.2f}%) squad games.".format(len(solos), 100*len(solos)/len(train
), len(duos), 100*len(duos)/len(train), len(squads), 100*len(squads)/l
en(train),))
```

```
There are 563279 (12.93%) solo games, 3070150 (70.46%) duo games and 7
23907 (16.61%) squad games.
```

In [28]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='kills',y='winPlacePerc',data=solos,color='black',alph
a=0.8)
sns.pointplot(x='kills',y='winPlacePerc',data=duos,color='#CC0000',alp
ha=0.8)
sns.pointplot(x='kills',y='winPlacePerc',data=squads,color='#3399FF',a
lpha=0.8)
plt.text(37,0.6,'Solos',color='black',fontsize = 17,style = 'italic')
plt.text(37,0.55,'Duos',color='#CC0000',fontsize = 17,style = 'italic'
)
plt.text(37,0.5,'Squads',color='#3399FF',fontsize = 17,style = 'itali
c')
plt.xlabel('Number of kills',fontsize = 15,color='blue')
plt.ylabel('Win Percentage',fontsize = 15,color='blue')
plt.title('Solo vs Duo vs Squad Kills',fontsize = 20,color='blue')
plt.grid()
plt.show()
```



Hmm, very interesting. Solos and duos behave the same, but when playing squads kills don't matter that much.

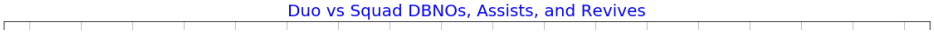
The attribute DBNOs means enemy players knocked. A "knock" can happen only in duos or squads, because the teammates have the chance to "revive" the knocked player in a given time. So a knocked player can be revived or die. If he is revived, the next time he will be knocked, his teammates will have less time to revive him.

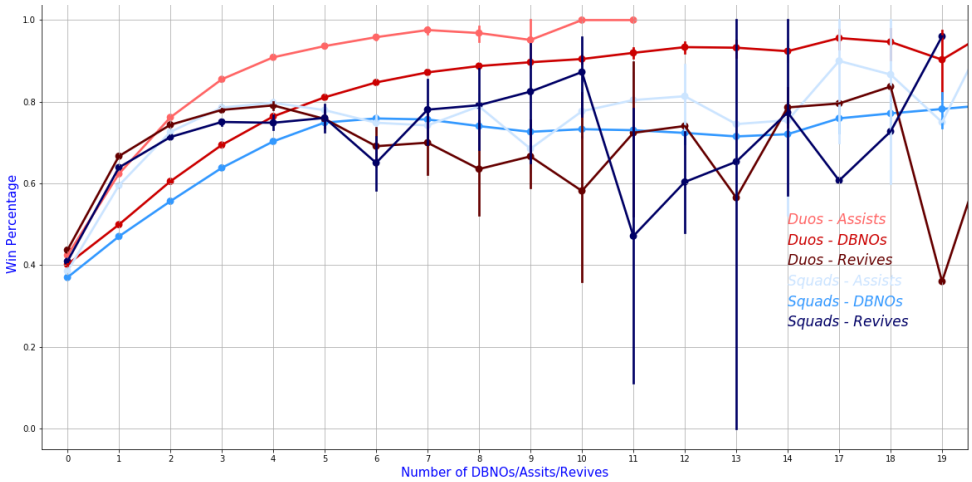
The attribute assist can also happen only in duos or squads. It generally means that the player had an involvement in a kill.

The attribute revive also happens in duos or squads.

In [29]:

```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='DBNOs',y='winPlacePerc',data=duos,color='#CC0000',alp
ha=0.8)
sns.pointplot(x='DBNOs',y='winPlacePerc',data=squads,color='#3399FF',a
lpha=0.8)
sns.pointplot(x='assists',y='winPlacePerc',data=duos,color='#FF6666',a
lpha=0.8)
sns.pointplot(x='assists',y='winPlacePerc',data=squads,color='#CCE5FF'
,alpha=0.8)
sns.pointplot(x='revives',y='winPlacePerc',data=duos,color='#660000',a
lpha=0.8)
sns.pointplot(x='revives',y='winPlacePerc',data=squads,color='#000066'
,alpha=0.8)
plt.text(14,0.5,'Duos - Assists',color='#FF6666',fontsize = 17,style =
'italic')
plt.text(14,0.45,'Duos - DBNOs',color='#CC0000',fontsize = 17,style =
'italic')
plt.text(14,0.4,'Duos - Revives',color='#660000',fontsize = 17,style =
'italic')
plt.text(14,0.35,'Squads - Assists',color='#CCE5FF',fontsize = 17,styl
e = 'italic')
plt.text(14,0.3,'Squads - DBNOs',color='#3399FF',fontsize = 17,style =
'italic')
plt.text(14,0.25,'Squads - Revives',color='#000066',fontsize = 17,styl
e = 'italic')
plt.xlabel('Number of DBNOs/Assits/Revives',fontsize = 15,color='blue'
)
plt.ylabel('Win Percentage',fontsize = 15,color='blue')
plt.title('Duo vs Squad DBNOs, Assists, and Revives',fontsize = 20,col
or='blue')
plt.grid()
plt.show()
```

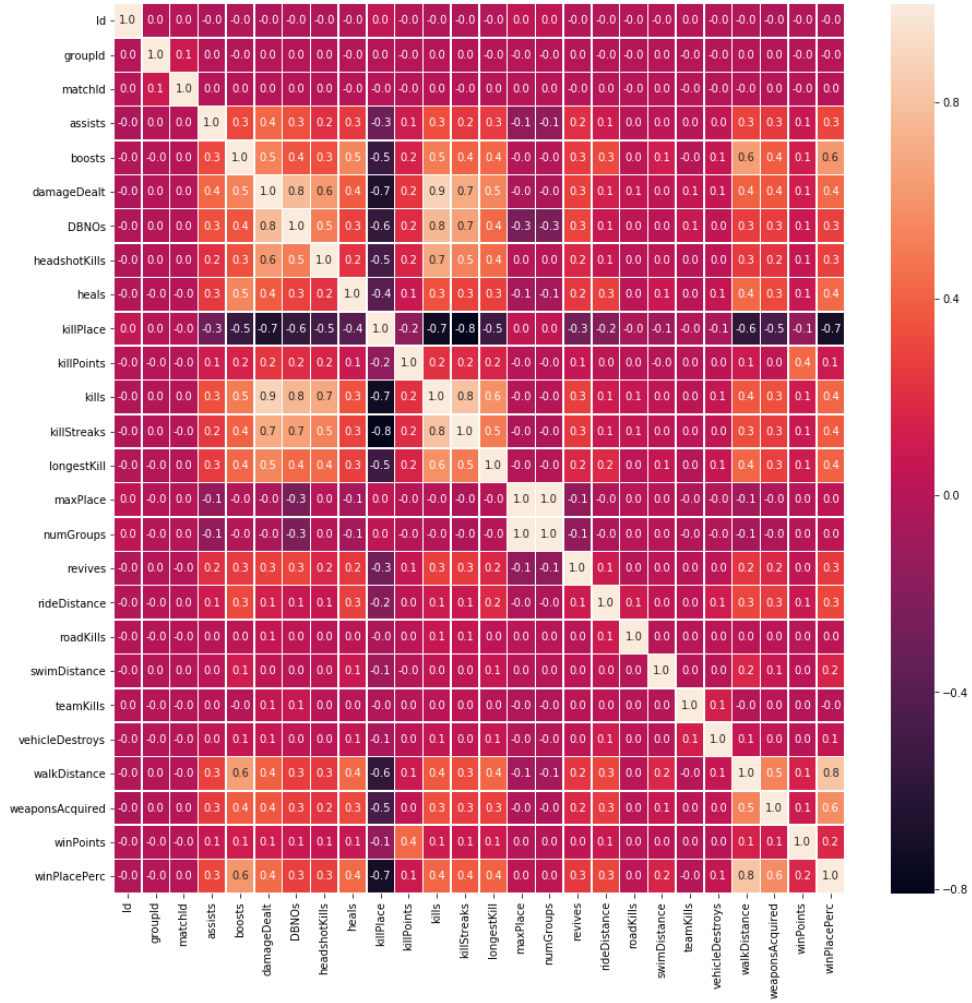




Pearson correlation between variables

In [30]:

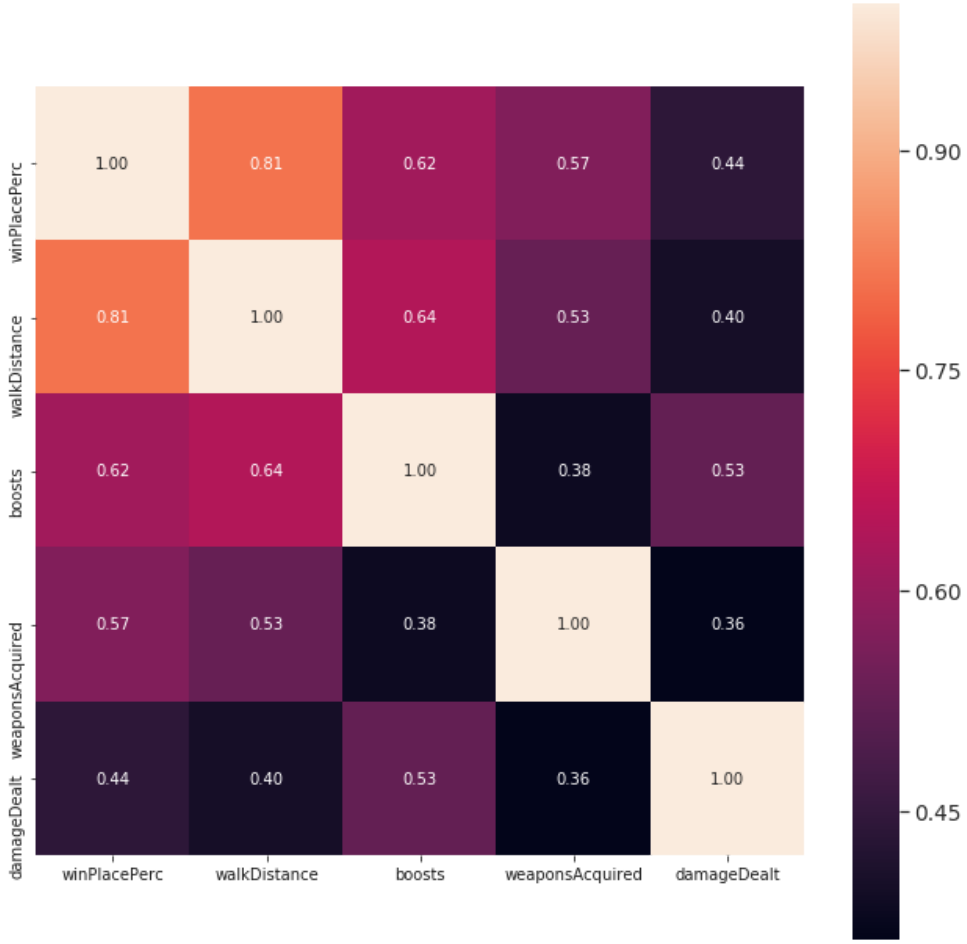
```
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(train.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.show()
```



In terms of the target variable (winPlacePerc), there are a few variables high medium to high correlation. The highest positive correlation is walkDistance and the highest negative the killPlace.

Let's zoom to the top-5 most positive correlated variables with the target.

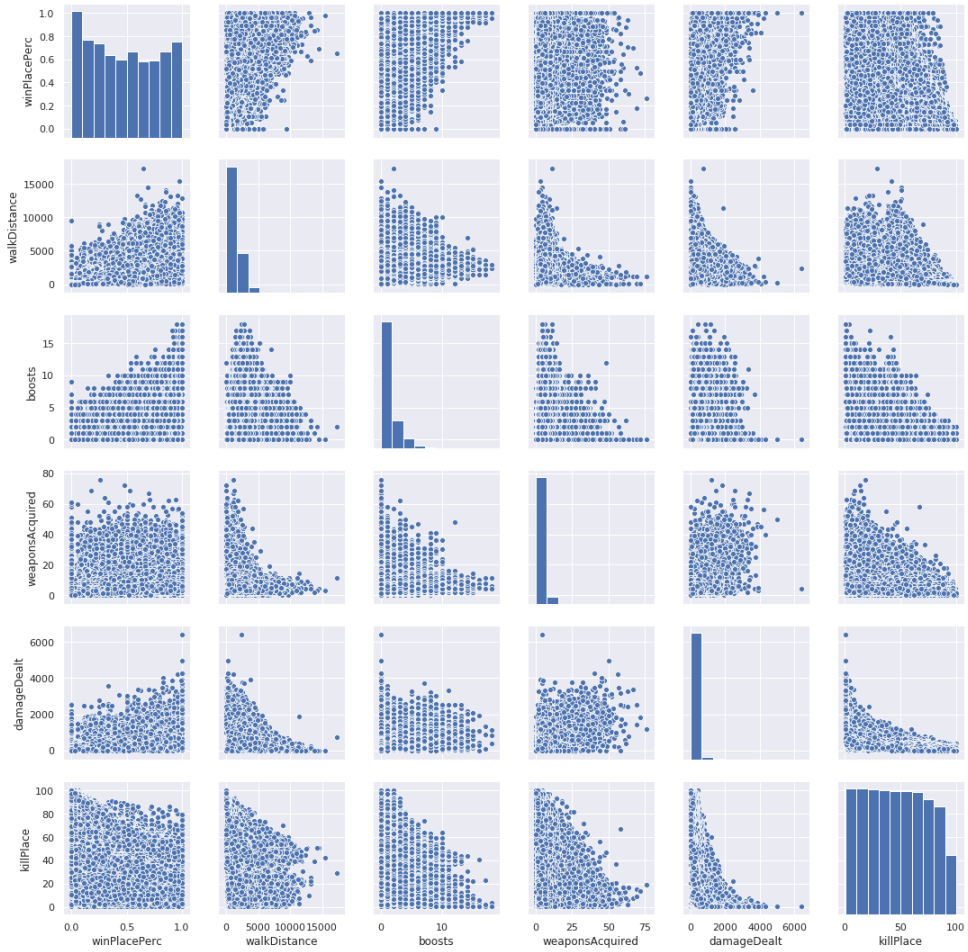
In [31]:
```
k = 5 #number of variables for heatmap
f,ax = plt.subplots(figsize=(11, 11))
cols = train.corr().nlargest(k, 'winPlacePerc')['winPlacePerc'].index
cm = np.corrcoef(train[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', an
not_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values
)
plt.show()
```



Let's plot the above variables and the killPlace variable as pairs.

In [32]:
```
sns.set()
cols = ['winPlacePerc', 'walkDistance', 'boosts', 'weaponsAcquired'
```

```
cols = ['winPlacePerc', 'walkDistance', 'boosts', 'weaponsAcquired',
'damageDealt', 'killPlace']
sns.pairplot(train[cols], size = 2.5)
plt.show()
```

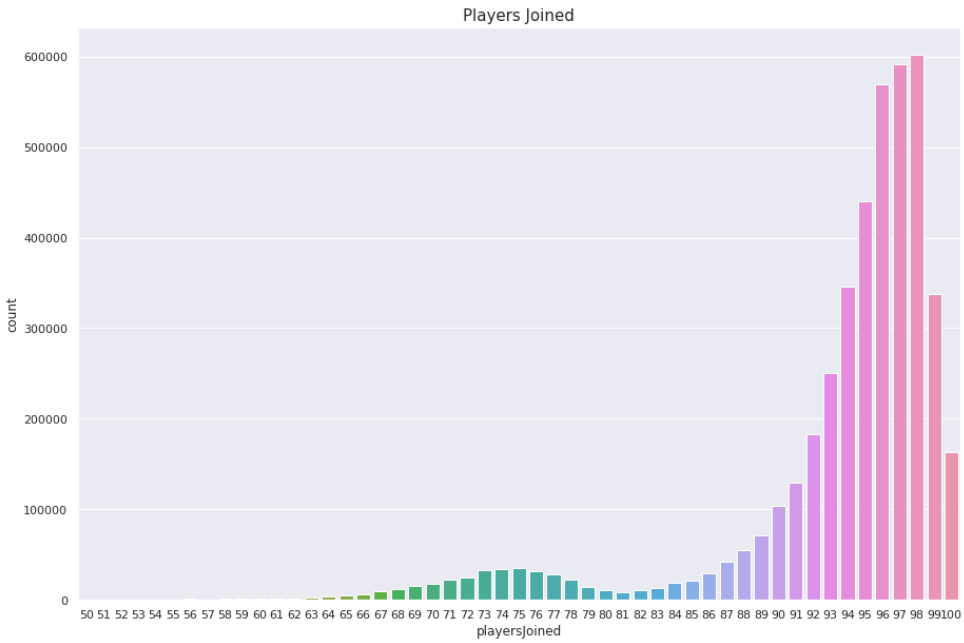


Feature Engineering

A game in PUBG can have up to 100 players fighting each other. But most of the times a game isn't "full". There is no variable that gives us the number of players joined. So lets create one.

In [33]:
```
train['playersJoined'] = train.groupby('matchId')['matchId'].transform
('count')
```

In [34]:
```
data = train.copy()
data = data[data['playersJoined']>49]
plt.figure(figsize=(15,10))
sns.countplot(data['playersJoined'])
plt.title("Players Joined",fontsize=15)
plt.show()
```



Based on the "playersJoined" feature we can create (or change) a lot of others to normalize their values. For example i will create the "killsNorm" and "damageDealtNorm" features. When there are 100 players in the game it might be easier to find and kill someone, than when there are 90 players. So i will normalize the kills in a way that a kill in 100 players will score 1 (as it is) and in 90 players it will score (100-90)/100 + 1 = 1.1. This is just an assumption. You can use different scales.

In [35]:
```
train['killsNorm'] = train['kills']*((100-train['playersJoined'])/100
+ 1)
train['damageDealtNorm'] = train['damageDealt']*((100-train['playersJo
ined'])/100 + 1)
train[['playersJoined', 'kills', 'killsNorm', 'damageDealt', 'damageDe
altNorm']][5:8]
```

Out[35]:

	playersJoined	kills	killsNorm	damageDealt	damageDealtNorm
5	96	1	1.04	128.1	133.224
6	86	1	1.14	130.3	148.542
7	92	5	5.40	661.8	714.744

Another simple feature is the sum of heals and boosts. Also the sum of total distance travelled.

In [36]:
```
train['healsAndBoosts'] = train['heals']+train['boosts']
train['totalDistance'] = train['walkDistance']+train['rideDistance']+t
rain['swimDistance']
```

When using boosting items you run faster. They also help staying out of the zone (PUBG term) and loot more (meaning walking more). So lets create a feature boosts per walking distance. Heals don't make you run faster, but they also help staying out of the zone and loot more. So lets create the same feature for heals also.

In [37]:
```
train['boostsPerWalkDistance'] = train['boosts']/(train['walkDistance'
]+1) #The +1 is to avoid infinity, because there are entries where boost
s>0 and walkDistance=0. Strange.
train['boostsPerWalkDistance'].fillna(0, inplace=True)
train['healsPerWalkDistance'] = train['heals']/(train['walkDistance']+
1) #The +1 is to avoid infinity, because there are entries where heals>0
and walkDistance=0. Strange.
train['healsPerWalkDistance'].fillna(0, inplace=True)
train['healsAndBoostsPerWalkDistance'] = train['healsAndBoosts']/(trai
n['walkDistance']+1) #The +1 is to avoid infinity.
train['healsAndBoostsPerWalkDistance'].fillna(0, inplace=True)
train[['walkDistance', 'boosts', 'boostsPerWalkDistance' ,'heals',  'h
ealsPerWalkDistance', 'healsAndBoosts', 'healsAndBoostsPerWalkDistanc
e']][40:45]
```

Out[37]:

	walkDistance	boosts	boostsPerWalkDistance	heals	healsPerWalkDistance	healsAndBoosts	h
40	2998.00	10	0.003334	4	0.001334	14	0.
41	3392.00	4	0.001179	1	0.000295	5	0.
42	4788.00	6	0.001253	6	0.001253	12	0.
43	581.20	2	0.003435	0	0.000000	2	0.
44	12.43	0	0.000000	0	0.000000	0	0.

Same, let's create the feature "killsPerWalkDistance".

In [38]:
```
train['killsPerWalkDistance'] = train['kills']/(train['walkDistance']+
1) #The +1 is to avoid infinity, because there are entries where kills>0
and walkDistance=0. Strange.
train['killsPerWalkDistance'].fillna(0, inplace=True)
train[['kills', 'walkDistance', 'rideDistance', 'killsPerWalkDistance'
, 'winPlacePerc']].sort_values(by='killsPerWalkDistance').tail(10)
```

Out[38]:

	kills	walkDistance	rideDistance	killsPerWalkDistance	winPlacePerc
1963256	25	0.0	0.0	25.0	1.0000
593613	25	0.0	0.0	25.0	1.0000
485441	25	0.0	0.0	25.0	0.9583
642937	25	0.0	0.0	25.0	0.8182
43076	26	0.0	0.0	26.0	1.0000
1869760	26	0.0	0.0	26.0	1.0000
381369	27	0.0	0.0	27.0	0.0000
32826	27	0.0	0.0	27.0	1.0000
67318	31	0.0	0.0	31.0	0.8571
841163	31	0.0	0.0	31.0	1.0000

0 walking distance and many kills? Also most have winPlacePerc=1. Definitelly cheaters.

Earlier in the kernel we did EDA for Solos, Duos and Squads. Lets create a column for them.

In [39]:
```
train['team'] = [1 if i>50 else 2 if (i>25 & i<=50) else 4 for i in tr
ain['numGroups']]
```

In [40]:
```
train.head()
```

Out[40]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace
0	0	24	0	0	5	247.30	2	0	4	17
1	1	440875	1	1	0	37.65	1	1	0	45
2	2	878242	2	0	1	93.73	1	0	2	54
3	3	1319841	3	0	0	95.88	0	0	0	86
4	4	1757883	4	0	1	0.00	0	0	1	58

So we created 10 features. I hope some of them to be useful.

Thank you if you reached this point! This is my first kernel ever. An upvote would be really appreciated and help me keep going!

In progress..

Did you find this Notebook useful?
Show your appreciation with an upvote

986

Data

Data Sources

- PUBG Finish Placement Prediction (Kernels Only)
  - sample_submission_V2.csv 1.93m x 2
  - test_V2.csv 1.93m x 28
  - train_V2.csv 4.45m x 29

PUBG Finish Placement Prediction (Kernels Only)

Can you predict the battle royale finish of PUBG Players?

Last Updated: a year ago

About this Competition

In a PUBG game, up to 100 players start in each match (matchId). Players can be on teams (groupId) which get ranked at the end of the game (winPlacePerc) based on how many other teams are still alive when they are eliminated. In game, players can pick up different munitions, revive downed-but-not-out (knocked) teammates, drive vehicles, swim, run, shoot, and experience all of the consequences -- such as falling too far or running themselves over and eliminating themselves.

You are provided with a large number of anonymized PUBG game stats, formatted so that each row contains one player's post-game stats. The data comes from matches of all types: solos, duos, squads, and custom; there is no guarantee of there being 100 players per match, nor at most 4 player per group.

You must create a model which predicts players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place).

File descriptions

- **train_V2.csv** - the training set
- **test_V2.csv** - the test set
- **sample_submission_V2.csv** - a sample submission file in the correct format

Data fields

Comments (162)

Sort by

All Comments | Hotness

Click here to comment...

Carlos Gutierrez • Posted on Version 3 of 5 • a year ago • Options • Reply 12

Good stuff man! I learned a lot!

One thing though, it's not that walking, swimming, or healing "help you win", I think is that the longer you survive, the more you are able to do all those things so it adds up.

Thank you for your contribution. :)

Dimitrios Effrosy... Kernel Author • Posted on Version 3 of 5 • a year ago • Options • Reply 0

You are correct my friend! Thanks for noticing and I am glad that you liked the kernel.

hybrideagle • Posted on Version 3 of 5 • a year ago • Options • Reply 8

I think this is a case of "correlation does not equal causation". Might want to update that.

Rajesh kumar jha • Posted on Latest Version • a year ago • Options • Reply 3

I am a beginner and your kernel helped a lot!
It's a really Impressive kernel and quite useful for learning purpose. Thanks for this.

Stian Fagerli Arntse... • Posted on Latest Version • a year ago • Options • Reply

5

Very nice overview! Increadibly useful for someone with limited experience in python (such as myself). But I have a question regarding the solo, duo and squad write up. You came to the conclusion that there were 3 million duo games - but that is 3 million players attending duo games - not 3 million duo matches, right? There are about 47 000 matches in the training data, if I am correct.

And as for defining match types, I have done a different approach. Even in a game with 92 numGroups, there are squads with 5 players - which point to this beein a custom game, not a solos game (since squads have a limit of 4 players in the squads mode). So instead, I use the maximum number of players in a matchId's squad to determine the game mode. If my approach is correct, it implies that the distribution is the following:

Custom 31293 (games where at least one squad has more than four players)
Duo 3940 (no squads with more than two players)
Solo 524 (no squads of more than one player)
Squads 11977 (no squads with more than four players)

Those numbers seem very strange to me, but I havent played the game, so I don't know what to expect.

But I may very well have done some errors both in coding and in logic, will post my kernel once I have had the time to clean it up.

EDIT: I did a search and found several other posts and discussions on this.

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply

1

Hello Stian, you are right about the 3 million duo games. I mean 3 million players attending duo games (not distinct). Your approach in detecting the custom game is very interesting. It can be used in the "teams" feature I created to make another category to define the custom games. Also, custom games may have other approaches to victory and thus your work is very usefull.

Hoang Nguyen • Posted on Latest Version • a year ago • Options • Reply

1

Yes I also wondered about this! I followed the maximum-number-of-players-in-group logic like yours, and also got the same number of each type of game!

An issue is that, by defining this way, there cases in which the game has 90-ish teams mostly consisting of only 1 players, but is still classified as "Duo" due to one or two teams of 2. Possibly in this situation the game is likely to be a custom, I think? Either this may not be important nor affect the data result that much.

Black Shen • Posted on Latest Version • a year ago • Options • Reply

1

it is a beautiful kernel. thanx

WEIQIPENG • Posted on Latest Version • a year ago • Options • Reply

1

Thank your notebook , I learn many knowleges

Dawg Welder • Posted on Latest Version • a year ago • Options • Reply

1

Useful EDA. Pretty good job so far, thanks!

Niranjan Christoph... • Posted on Latest Version • a year ago • Options • Reply

1

Great Content and Well Structured.
Thank you.

Arunsankar Kumar... • Posted on Latest Version • a year ago • Options • Reply

1

This is a fantastic kernel. I learned quite a bit from your kernel. Thanks Dimitrios!

sajal • Posted on Latest Version • a year ago • Options • Reply

1

Thanks for this amazing kernel. Helped a lot in understanding about the dataset.

Zack Ishio • Posted on Latest Version • a year ago • Options • Reply

1

Great kernel. It is so helpful.

Hossein Bahrami • Posted on Latest Version • a year ago • Options • Reply

1

Thank you. really nice kernel...

Carlo Lepelaars • Posted on Latest Version • a year ago • Options • Reply

3

Wow, this is an amazing kernel! It is no coincidence this kernel got so many upvotes!

Yokhesh Krishnasa... • Posted on Latest Version • a year ago • Options • Reply

1

Great Kernel

AR

Aditya Rathod • Posted on Latest Version • a year ago • Options • Reply

1

This is amazing. Thanks for the detailed analysis.

Nitin Datta • Posted on Latest Version • a year ago • Options • Reply

2

Amazing work learnt a lot from it

Michal Bogacz • Posted on Latest Version • a year ago • Options • Reply

1

Good job, I have one question. Is it sensible to check correlations between variables and "ID"? This is just a number that does not logically related to other variables. So in my opinion there is no point in checking the strength of correlation if it does not exist. Greetings! :)

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply 1

Yes of course there is no point to check for their correlation. ID will not be used as a feature for classification. Thanks!

n3dry • Posted on Latest Version • a year ago • Options • Reply 1

Love the data analysis and graphs!

Prudvi RajKumar • Posted on Latest Version • a year ago • Options • Reply 1

Nice kernel. Very useful!!!

Tirth Patel • Posted on Latest Version • a year ago • Options • Reply 2

This is the most entertaining and knowledgeable kernel. Thank you so much for providing such a awesome kernel!

Raafat • Posted on Latest Version • a year ago • Options • Reply 1

It's a beautiful kernal and very helpful. Thanks for this.

Li Ling • Posted on Latest Version • a year ago • Options • Reply 1

Really love your kernel!

Apoorv Patne • Posted on Latest Version • a year ago • Options • Reply 1

Thanks for such a great kernel. May I know how does this line of code give us the number of winners WITHOUT a single kill?

```
print("{} players ({:.4f}%) have won without a single
kill!".format(len(data[data['winPlacePerc']==1]),100*len(data[data['winPlacePerc']==1])/len(trai
n)))
```

Since according to data description, it indicates "The target of prediction. This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match." Also, fundamentally speaking, in a solo game, a player can not win a game without a kill. A player must have at least one kill in the final 1v1 if he wants the chiken dinner.

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply 1

Thank you!
Previously i wrote: data = data[data['kills']==0] filtering to the players with 0 kills. In this game it is possible to win with no kills. There is a continusly shrinking circle that deals damage to the players outside it. So the cirlce can kill you.

JoanQY • Posted on Latest Version • a year ago • Options • Reply

2

Great work!
I got a question. Do I have to split the training data into training set and test set before doing exploration and just use the training set to do exploration? Or do EDA first and then split data?

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply

1

I do the EDA on the train set, because it is the one that the target variable is known.

Sahit Sharma • Posted on Latest Version • a year ago • Options • Reply

1

Great kernel!

Shankar Pandala • Posted on Latest Version • a year ago • Options • Reply

1

I couldn't get that Solo, Duo and Squad thing.
You are saying, If I choose solo, all other players are also playing Solo not Duo or Squad??
Is this your assumption or the game works this way ?

Awesome kernel and and Awesome approach!!!

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply

0

Thanks! It is how the game works

Karan Jakhar • Posted on Latest Version • a year ago • Options • Reply

2

Very helpful kernel . Awesome . Thanks for sharing .

Akash Ravichandran • Posted on Latest Version • a year ago • Options • Reply

1

Well explained. Thanks for the great kernel.

nadayan • Posted on Latest Version • a year ago • Options • Reply

1

nice work!

NKN • Posted on Latest Version • a year ago • Options • Reply

1

Great work! Really helpful for me.

Akshay_Siras • Posted on Latest Version • a year ago • Options • Reply

1

i loved this kernal. Data set Explained in detailed EDA is really fun!

yasser latreche • Posted on Latest Version • a year ago • Options • Reply

1

Nice kernel
i have a question
i tried to write my own solution , and i tried by visualizing how much players per a team in a match , and the weird thig that i found is there are a lot of teams with plus of 4 players !!! and that's not logic cuz the max players per a team is 4 in the game

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply

0

Games with more than 4 players in a team are custom games and can be considered as outliers.

Jason • Posted on Latest Version • a year ago • Options • Reply

1

Hey Dimitrios, awesome work. You may want to have a look at the column 'matchType' and compare that to your team function. It's not listed in your column description but it is in the data. Thanks for sharing.

Overfitting • Posted on Latest Version • a year ago • Options • Reply

1

Hey bro, It's a really beautiful kernal and very helpful. Thanks for this.
I didn't understood the part of extracting the total number of players in a match . Can you explain you groupBy for totalplayersjoined.
Also, I think that number of headshot players are also pro. If we can extract the number of headshots per kill that would also help. what do you think?

Thanks again!

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply

1

Hey, thanks for your kind words. I think you need to check the transform function in order to understand the part you mentioned. Here is a great link for that.

If you are interested in creating a feature for headshops/kills use this code:

```
df['headshotPerKill'] = df['headshotKills'] / df['kills']
df['headshotPerKill'].fillna(0, inplace=True)
```

Mofii • Posted on Latest Version • a year ago • Options • Reply

1

Looks really cool!

Prem Stroke • Posted on Latest Version • a year ago • Options • Reply

1

Wow this is a great kernel. Thanks!

Mike • Posted on Latest Version • a year ago • Options • Reply

1

very nice and neat

GoldFish • Posted on Latest Version • a year ago • Options • Reply

1

Really nice! I learned a lot by this kernel.

KaHei Ng • Posted on Latest Version • a year ago • Options • Reply

1

Thanks for your work! Some findings are really non-trivial.

LIU,H.W • Posted on Latest Version • a year ago • Options • Reply

1

Really good !

Chirag Ved • Posted on Latest Version • a year ago • Options • Reply

2

Great kernal, especially for beginners like me.
I am building my first public kernal, thanks to you :)

Roman Sozonov • Posted on Latest Version • a year ago • Options • Reply

1

Wow! Good job!

Arun Sriraman • Posted on Latest Version • a year ago • Options • Reply

1

Fantastic One. Good way to look at a data.

Alexander Isaev • Posted on Latest Version • a year ago • Options • Reply

1

Thanks! You are inspired me to post my first public kernel! I took some parts from your one.

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply 1

This is exactly why i love kaggle

Felipe Moreira • Posted on Latest Version • a year ago • Options • Reply 1

Beautifull EDA.

PraneshKrishnamu... • Posted on Latest Version • a year ago • Options • Reply 1

Loved the way you've thought about the entire thing! Great work!

Senneth • Posted on Latest Version • a year ago • Options • Reply 1

Have you considered three-man team matchmake? Just a thought

Dimitrios Effrosy... Kernel Author • Posted on Latest Version • a year ago • Options • Reply 0

I believe they are included in the squads.

Siddharth Parashar • Posted on Latest Version • a year ago • Options • Reply 1

This is Awesome. Lessons learnt.

kurumi • Posted on Latest Version • a year ago • Options • Reply 1

Great Kernel!

Pavan Sanagapati • Posted on Latest Version • a year ago • Options • Reply 1

Excellent Kernel.Upvoted

YuGuiii • Posted on Latest Version • a year ago • Options • Reply 1

Although my English is very poor, I still learned a lot of useful things from your kernel.thank you very much!

Showing 50 of 162 comments. Show More

Similar Kernels