EDA And Feature Engineering for the video game

Import Libraries

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
train = pd.read_csv('../input/train_V2.csv')
```

In [6]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4446966 entries, 0 to 4446965
Data columns (total 29 columns):
Id                object
groupId           object
matchId           object
assists           int64
boosts            int64
damageDealt       float64
DBNOs             int64
headshotKills     int64
heals             int64
killPlace         int64
killPoints        int64
kills             int64
killStreaks       int64
longestKill       float64
matchDuration     int64
matchType         object
maxPlace          int64
numGroups         int64
rankPoints        int64
revives           int64
rideDistance      float64
roadKills         int64
swimDistance      float64
teamKills         int64
vehicleDestroys   int64
walkDistance      float64
weaponsAcquired   int64
winPoints         int64
winPlacePerc      float64
dtypes: float64(6), int64(19), object(4)
memory usage: 983.9+ MB
```

Read data

In [7]:
```
train.head()
```

Out[7]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	killPoints	kills
0	7f96b2f878858a	4d4b580de459be	a10357fd1a4a91	0	0	0.00	0	0	0	60	1241	0
1	eef90569b9d03c	684d5656442f9e	aeb375fc57110c	0	0	91.47	0	0	0	57	0	0
2	1eaf90ac73de72	6a4a42c3245a74	110163d8bb94ae	1	0	68.00	0	0	0	47	0	0
3	4616d365dd2853	a930a9c79cd721	f1f1f4ef412d7e	0	0	32.90	0	0	0	75	0	0
4	315c96c26c9aac	de04010b3458dd	6dc8ff871e21e6	0	0	100.00	0	0	0	45	0	1

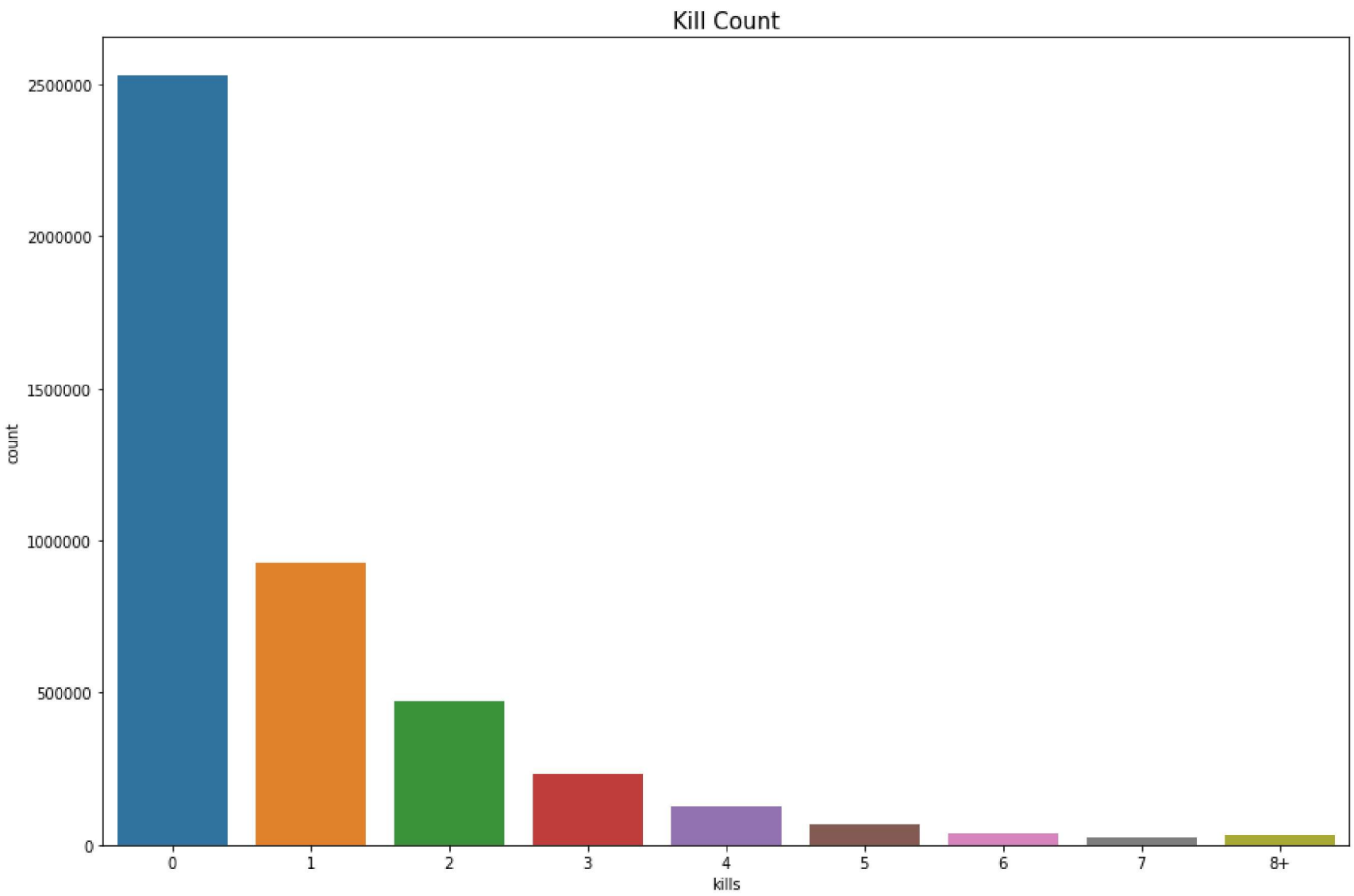
The Killers

In [8]:
```
print("The average person kills {:.4f} players, 99% of people have {} kills or less, while the most kills ever recorded
```

```
The average person kills 0.9248 players, 99% of people have 7.0 kills or less, while the most kills ever recorded is 7
2.
```

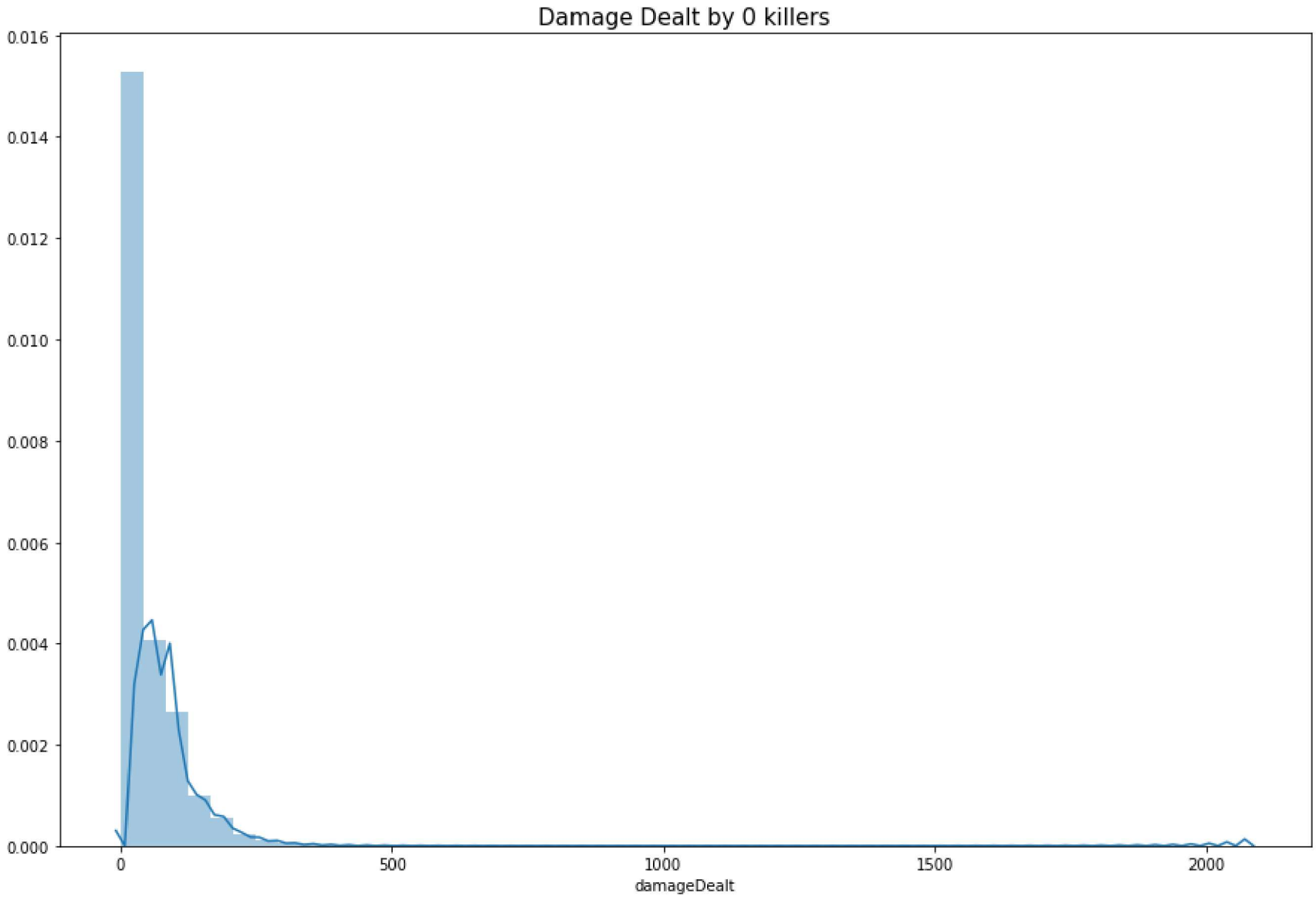
Let's plot the kill counts.

In [9]:
```
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.

In [10]:
```
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()
```



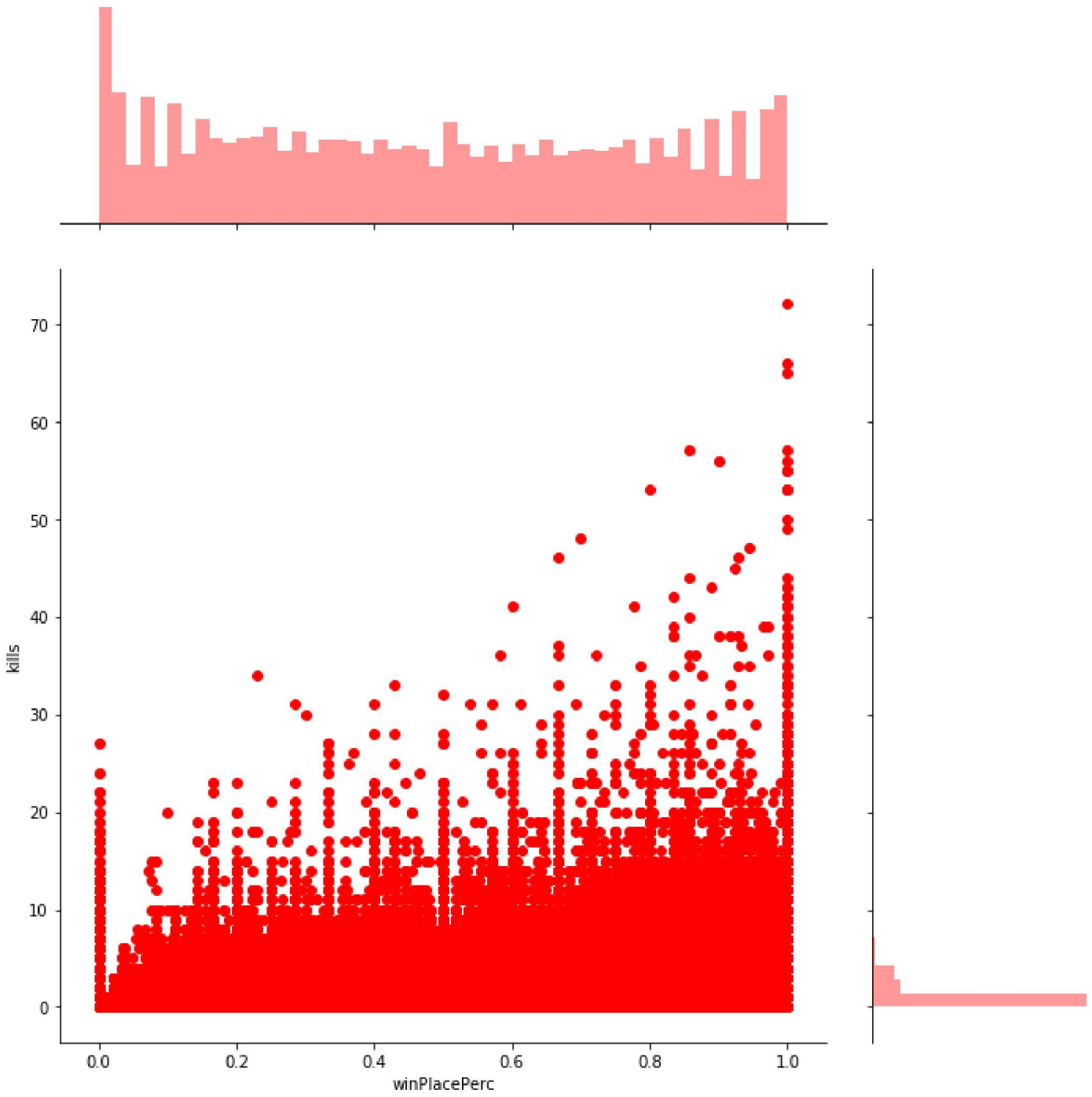
Let's investigate the exceptions.

In [11]:
```
print("{} players ({:.4f}%) have won without a single kill!".format(len(data[data['winPlacePerc']==1]), 100*len(data[da

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

```
16666 players (0.3748%) have won without a single kill!
4770 players (0.1073%) have won without dealing damage!
```

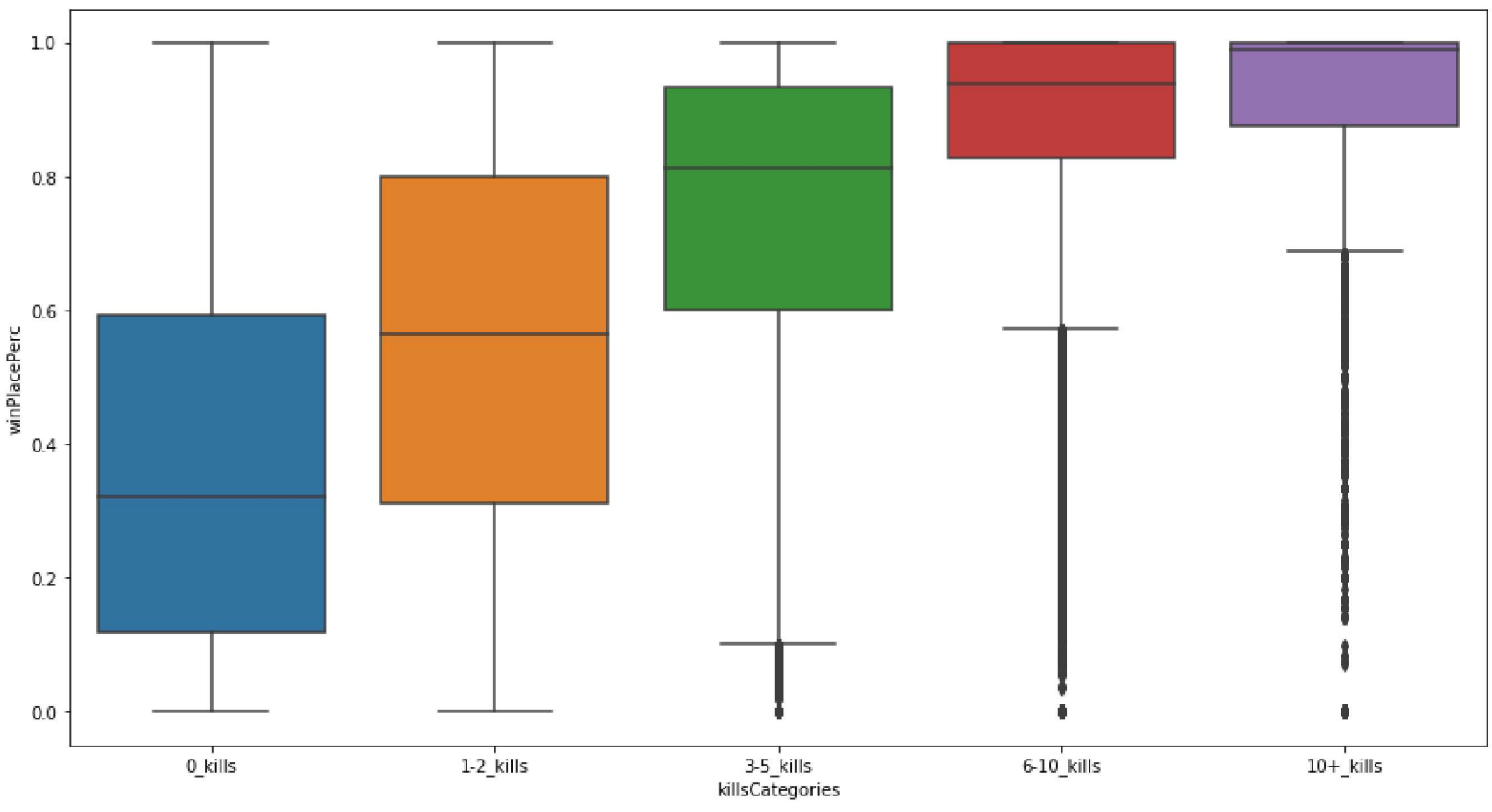
Plot win placement percentage vs kills.

In [12]:
```
sns.jointplot(x="winPlacePerc", y="kills", data=train, height=10, ratio=3, color="r")
plt.show()
```



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

In [13]:
```
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

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

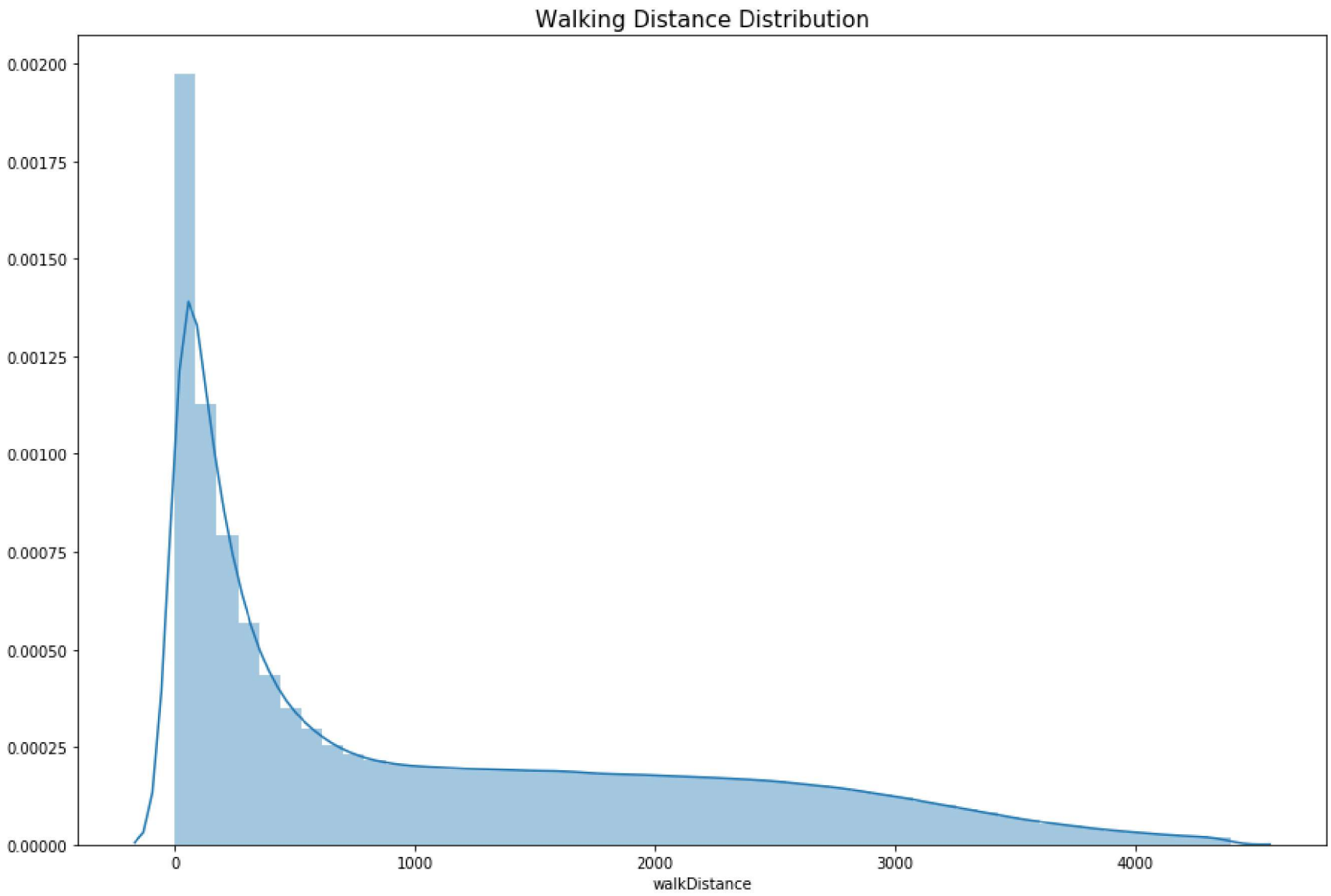


The Runners

In [14]:
```
print("The average person walks for {:.1f}m, 99% of people have walked {}m or less, while the marathoner champion walke
```

```
The average person walks for 1154.2m, 99% of people have walked 4396.0m or less, while the marathoner champion walked f
or 25780.0m.
```

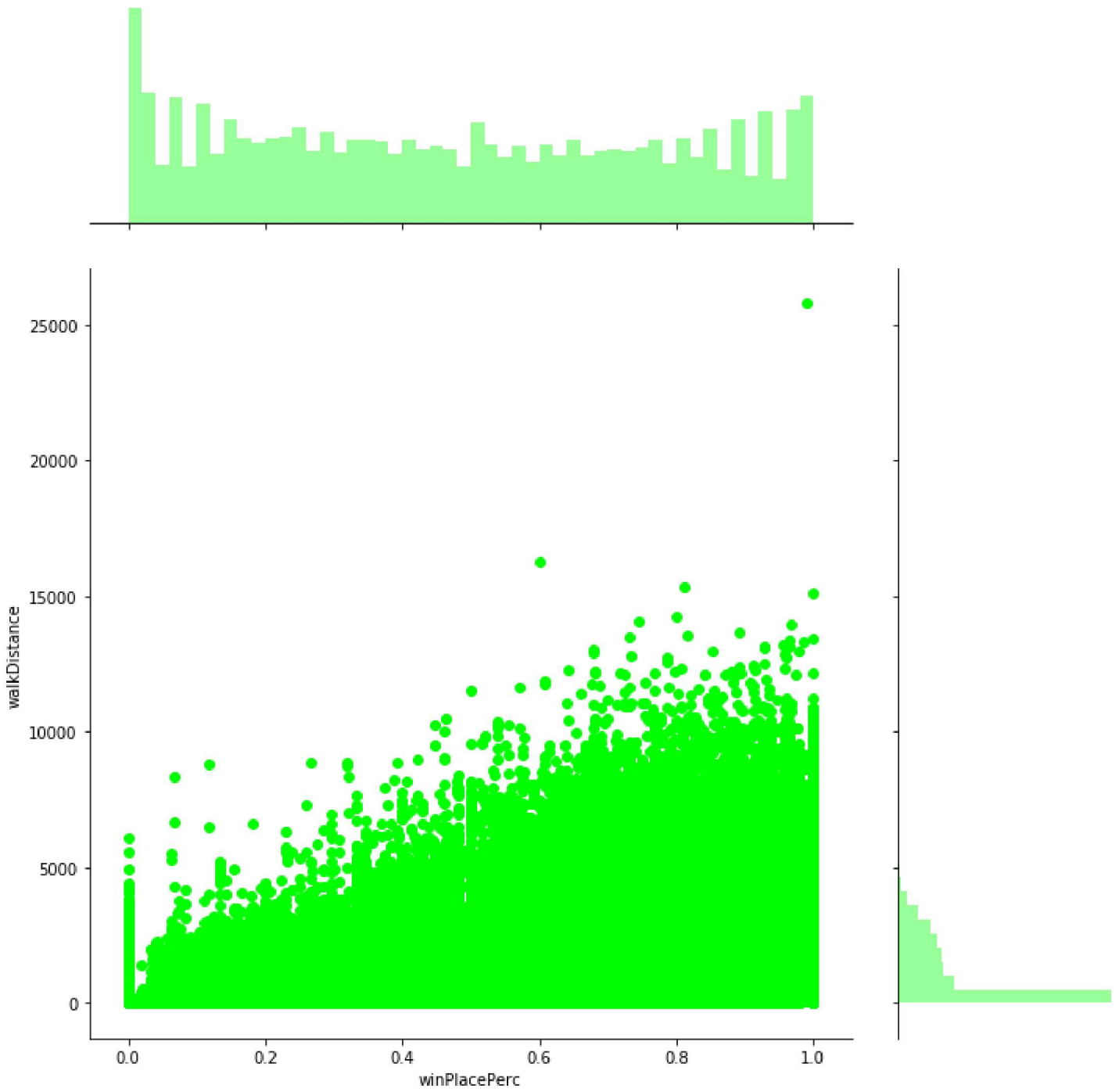
In [15]:
```
data = train.copy()
data = data[data['walkDistance'] < train['walkDistance'].quantile(0.99)]
plt.figure(figsize=(15,10))
plt.title("Walking Distance Distribution",fontsize=15)
sns.distplot(data['walkDistance'])
plt.show()
```



In [16]:
```
print("{} players ({:.4f}%) walked 0 meters. This means that they die before even taking a step or they are afk (more po
```

```
99603 players (2.0329%) walked 0 meters. This means that they die before even taking a step or they are afk (more possi
ble).
```

In [17]:
```
sns.jointplot(x="winPlacePerc", y="walkDistance",  data=train, height=10, ratio=3, color="lime")
plt.show()
```



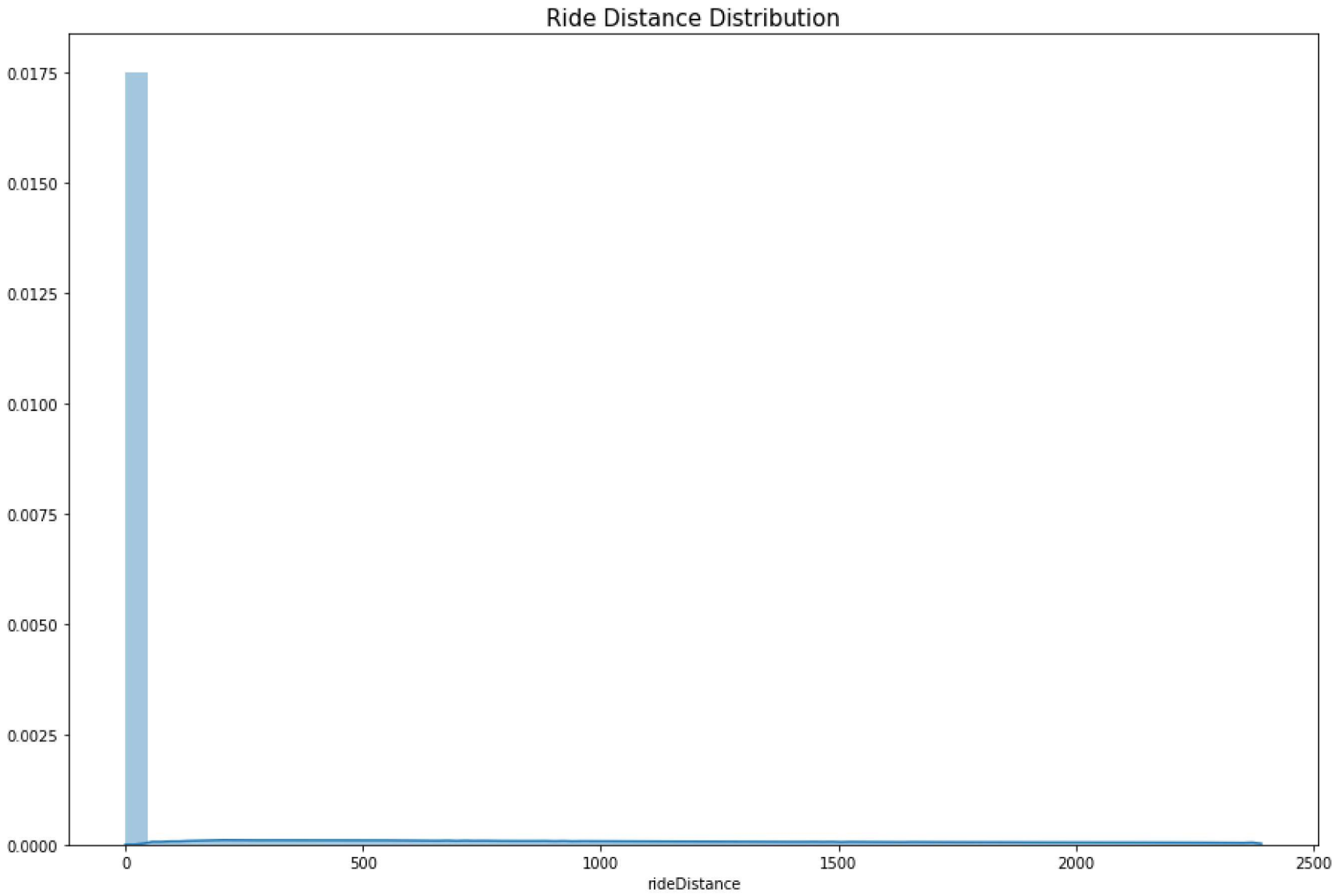
Apparently walking has a high correlation with winPlacePerc.

The Drivers

In [18]:
```
print("The average person drives for {:.1f}m, 99% of people have drived {}m or less, while the formula 1 champion drive
```

```
The average person drives for 606.1m, 99% of people have drived 6966.0m or less, while the formula 1 champion drived fo
r 40710.0m.
```

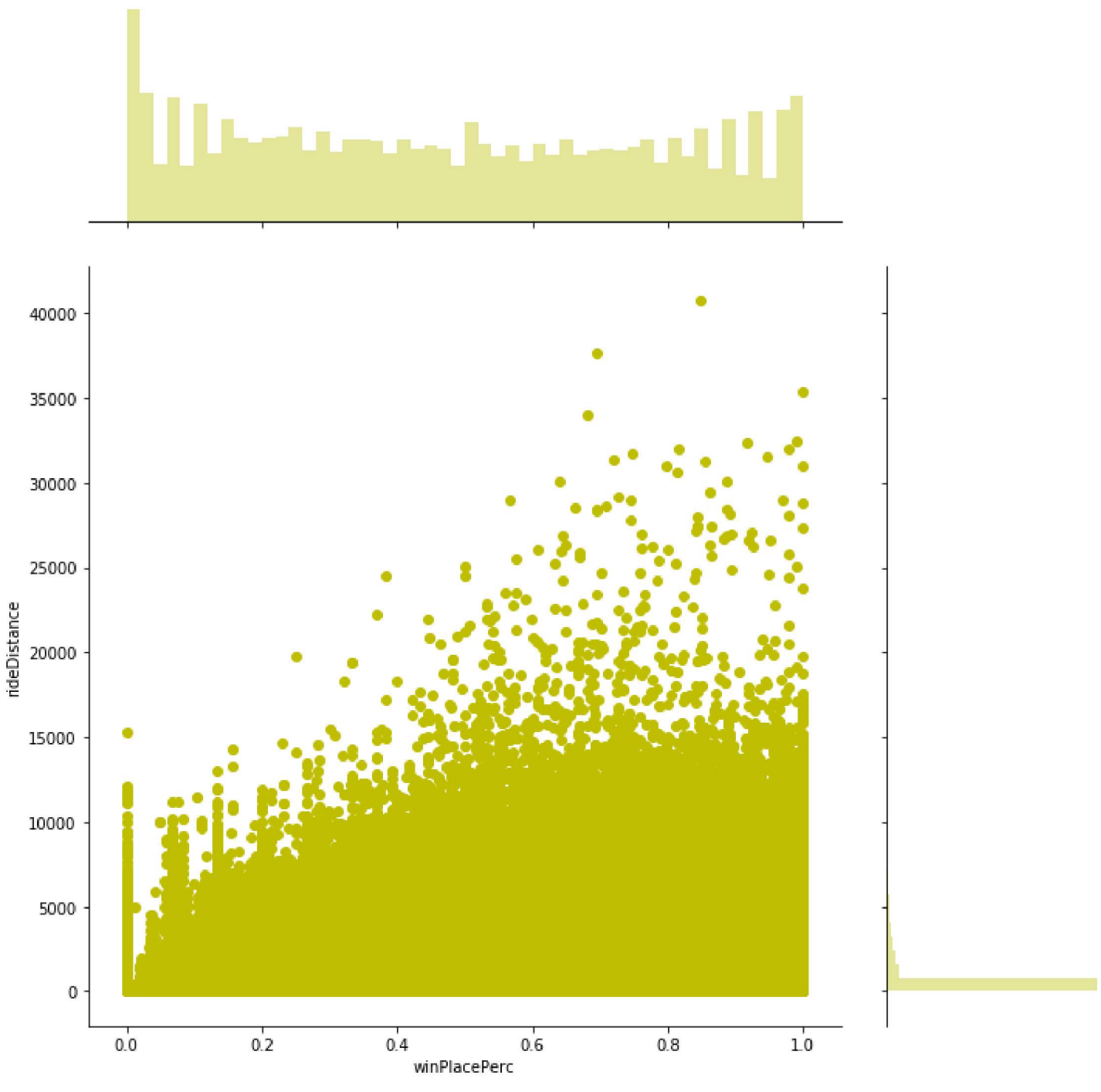
In [19]:
```
data = train.copy()
data = data[data['rideDistance'] < train['rideDistance'].quantile(0.9)]
plt.figure(figsize=(15,10))
plt.title("Ride Distance Distribution",fontsize=15)
sns.distplot(data['rideDistance'])
plt.show()
```



In [20]:
```
print("{} players ({:.4f}%) drived for 0 meters. This means that they don't have a driving licence yet.".format(len(dat
```

```
3309429 players (23.1022%) drived for 0 meters. This means that they don't have a driving licence yet.
```

In [21]:
```
sns.jointplot(x="winPlacePerc", y="rideDistance", data=train, height=10, ratio=3, color="y")
plt.show()
```

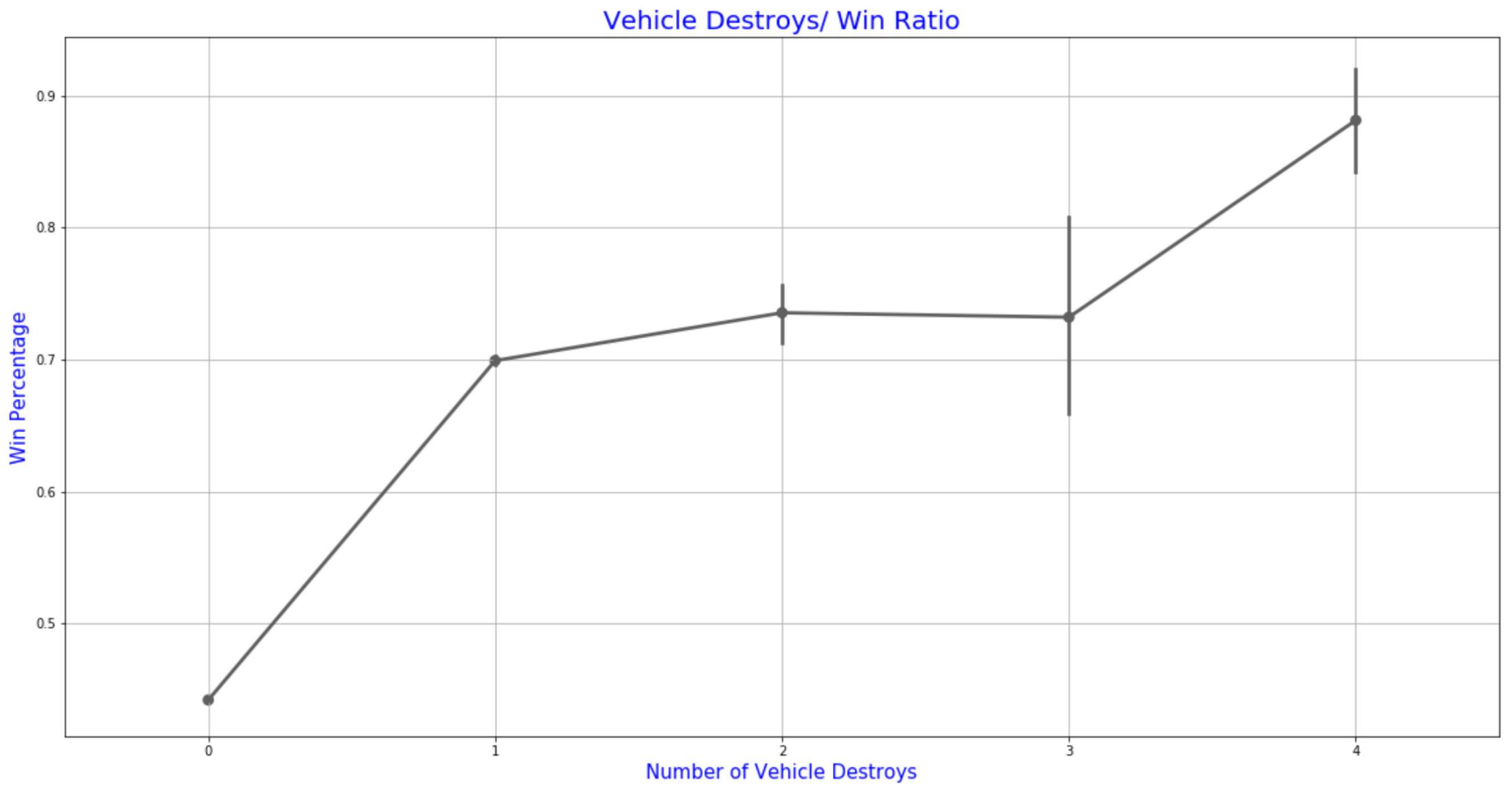


There is a small correlation between rideDistance and winPlacePerc.

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

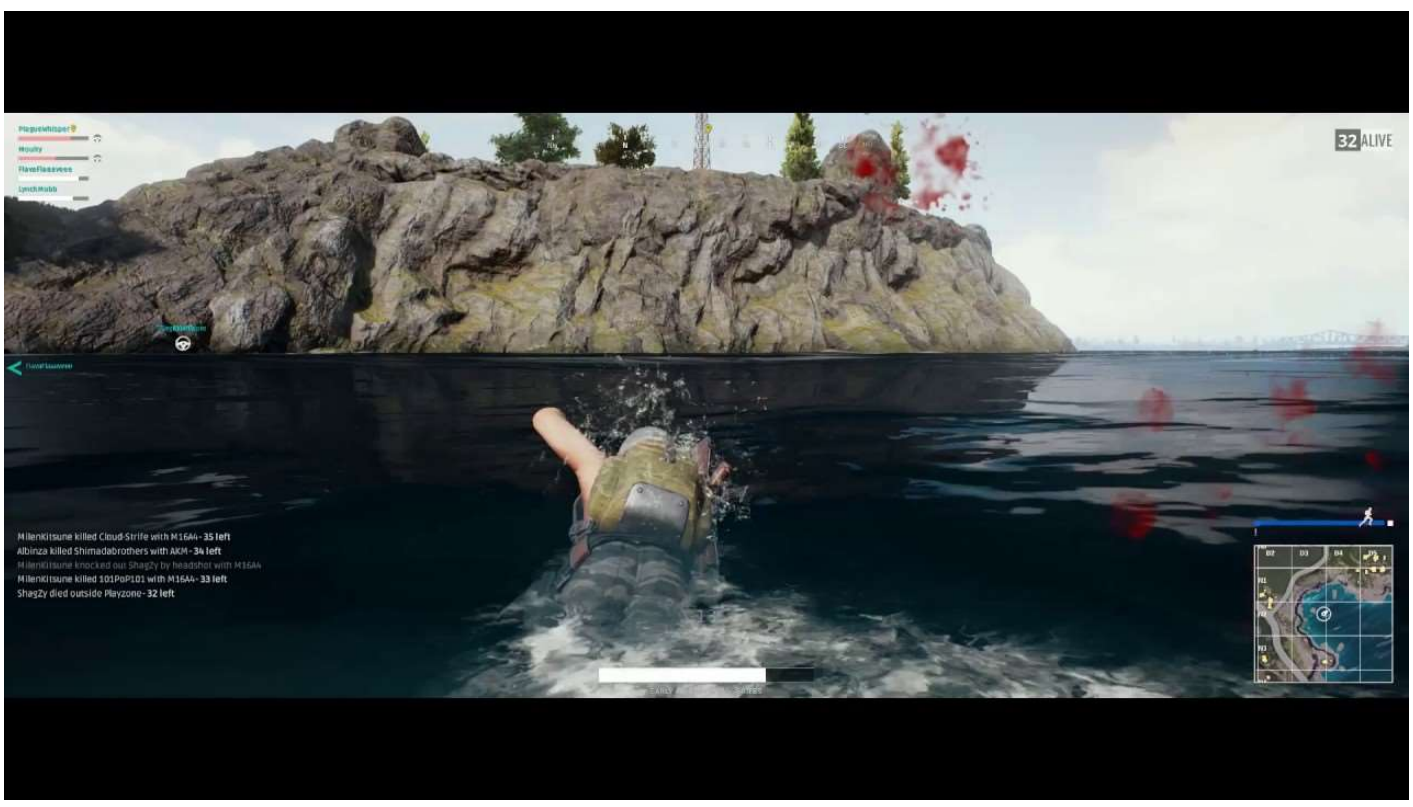
In [22]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='vehicleDestroys',y='winPlacePerc',data=data,color='#606060',alpha=0.8)
```

```
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.

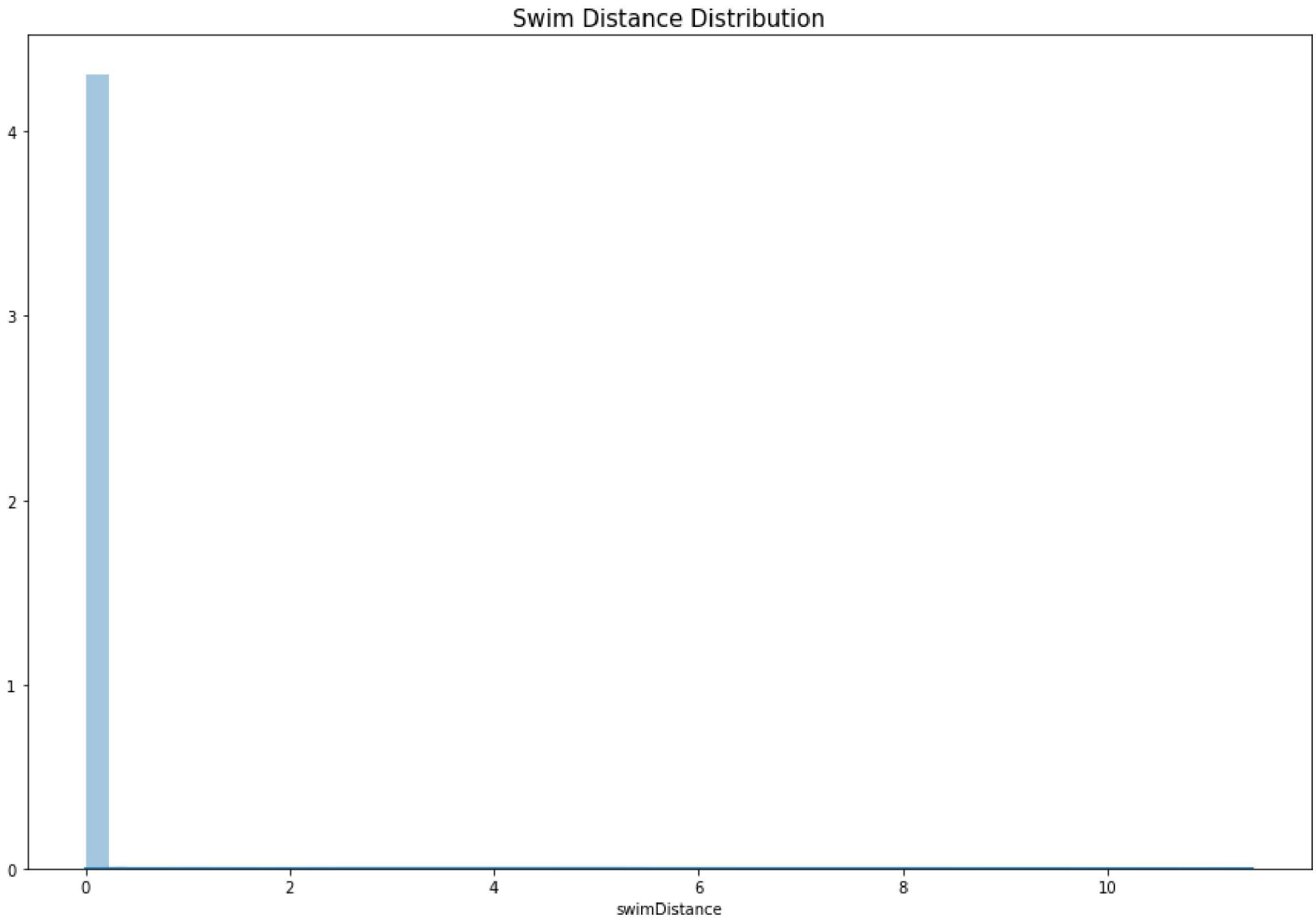
The Swimmers



In [23]:
```
print("The average person swims for {:.1f}m, 99% of people have swimemd {}m or less, while the olympic champion swimmed
```

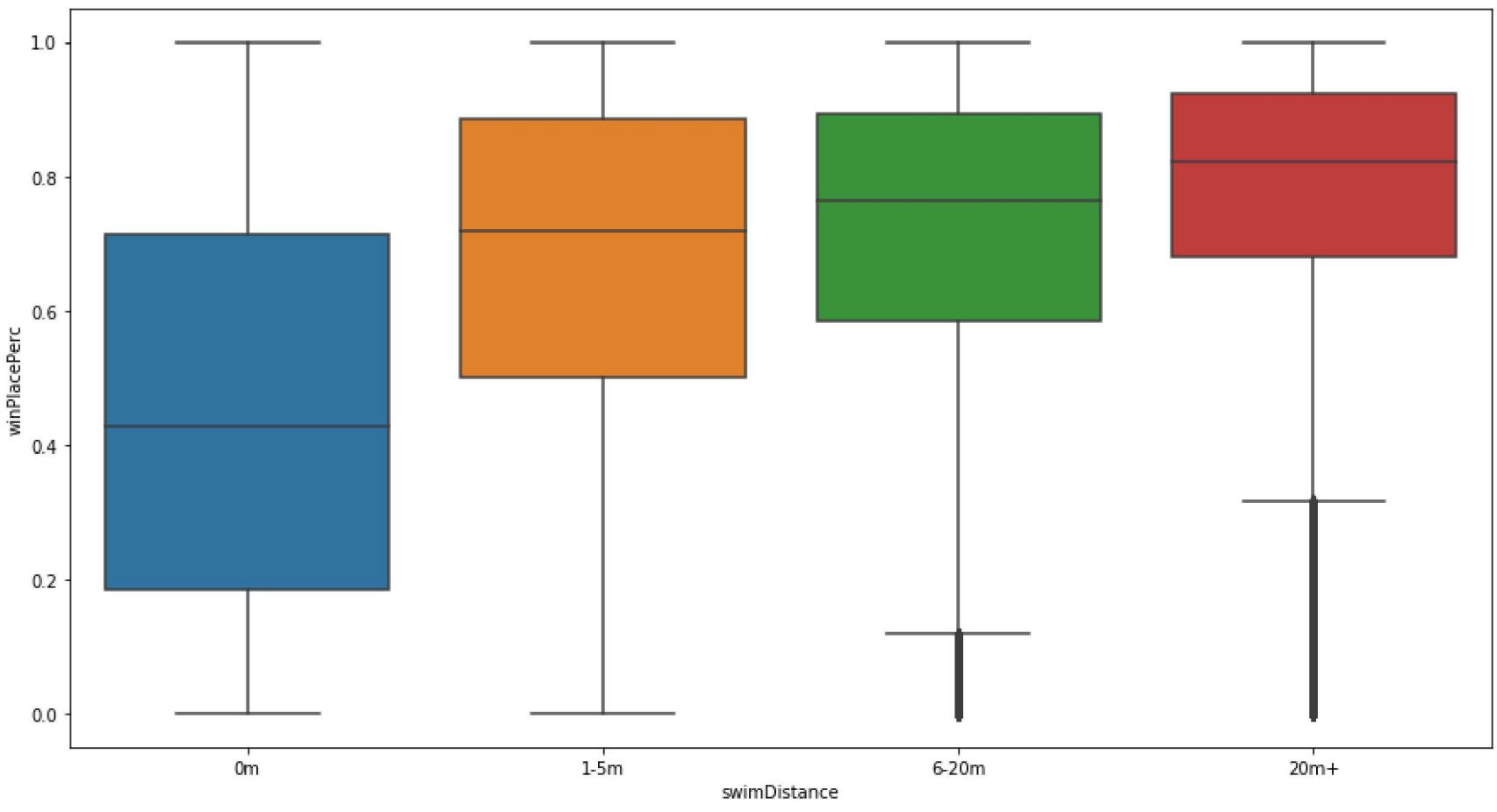
```
The average person swims for 4.5m, 99% of people have swimemd 123.0m or less, while the olympic champion swimmed for 38
23.0m.
```

In [24]:
```
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 [25]:
```
swim = train.copy()

swim['swimDistance'] = pd.cut(swim['swimDistance'], [-1, 0, 5, 20, 5286], labels=['0m','1-5m', '6-20m', '20m+'])

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



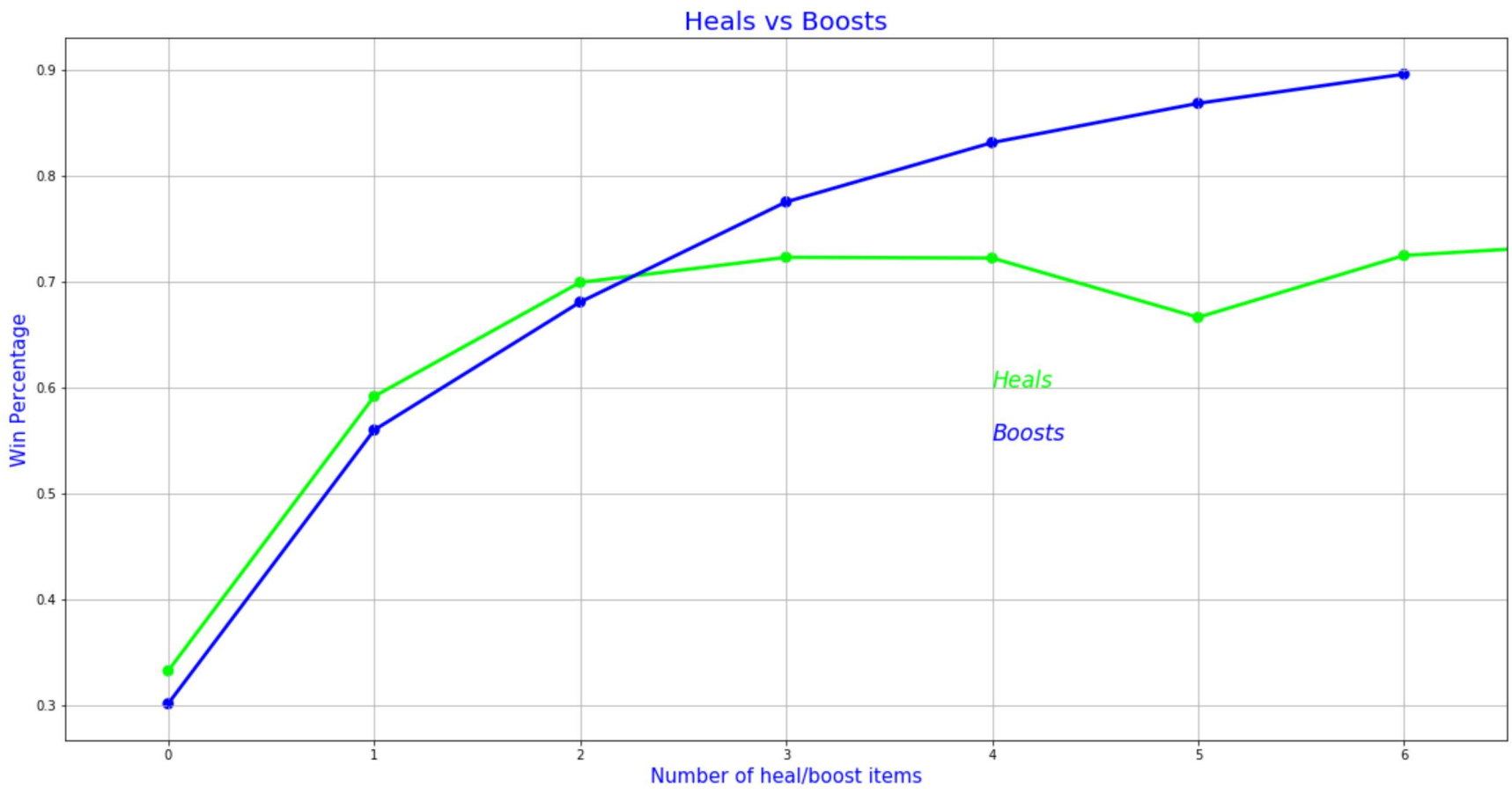
It seems that if you swim, you rise to the top.

The Healers

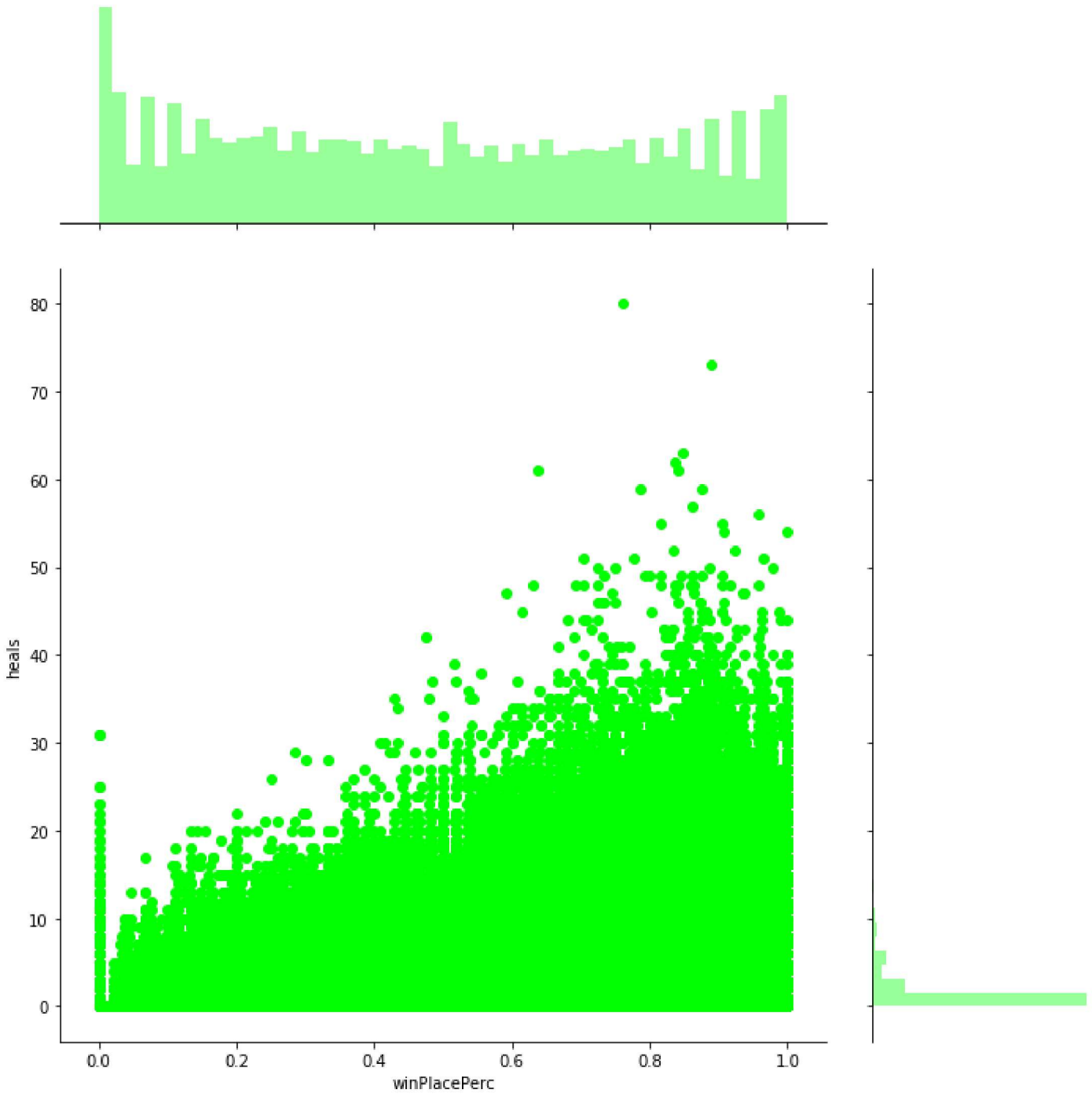
In [26]:
```
print("The average person uses {:.1f} heal items, 99% of people use {} or less, while the doctor used {}.".format(train
print("The average person uses {:.1f} boost items, 99% of people use {} or less, while the doctor used {}.".format(trai
```

```
The average person uses 1.4 heal items, 99% of people use 12.0 or less, while the doctor used 80.
The average person uses 1.1 boost items, 99% of people use 7.0 or less, while the doctor used 33.
```

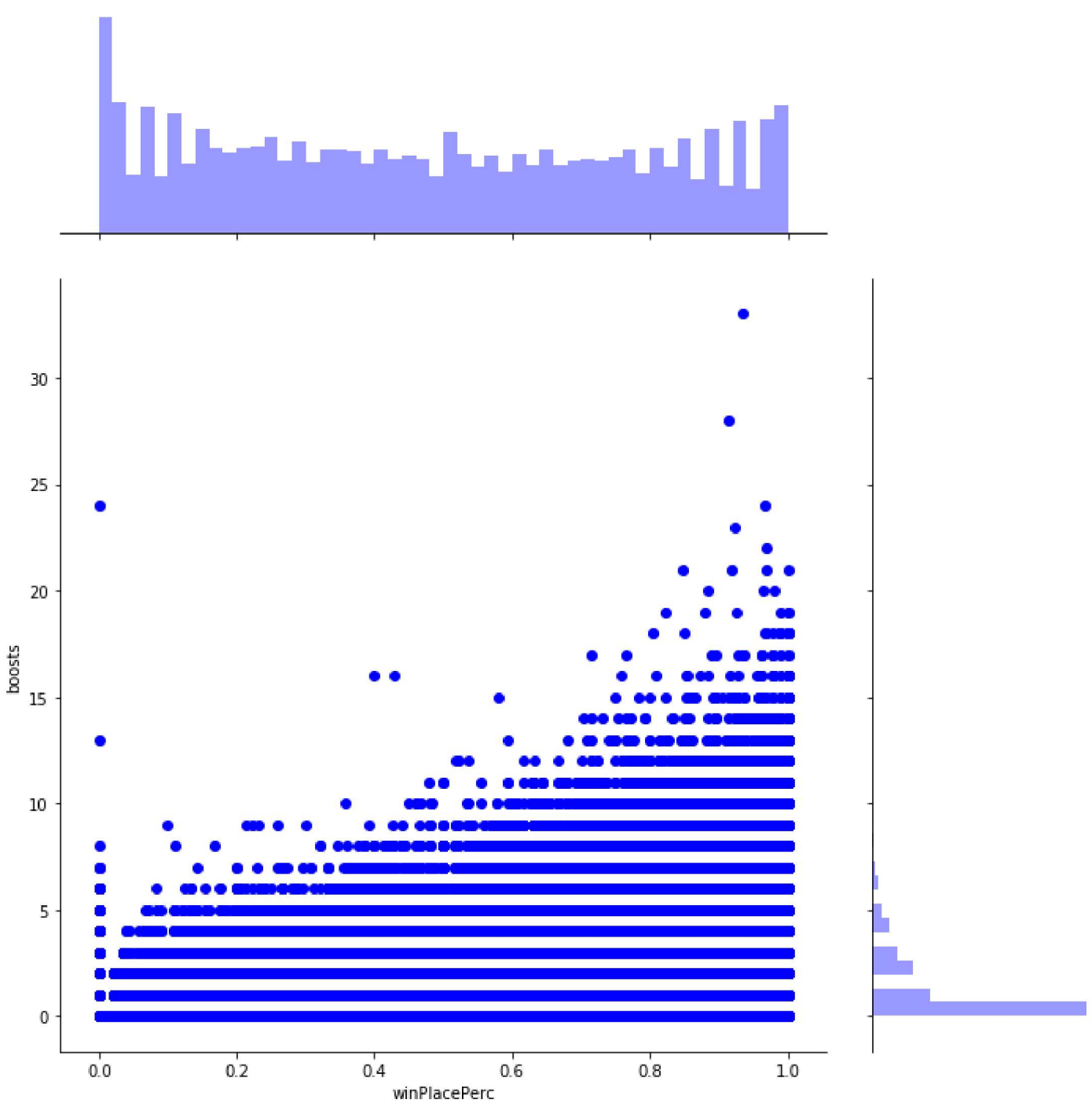
In [27]:
```
data = train.copy()
data = data[data['heals'] < data['heals'].quantile(0.99)]
data = data[data['boosts'] < data['boosts'].quantile(0.99)]

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 [28]:
```
sns.jointplot(x="winPlacePerc", y="heals", data=train, height=10, ratio=3, color="lime")
plt.show()
```



In [29]:
```
sns.jointplot(x="winPlacePerc", y="boosts", data=train, height=10, ratio=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.

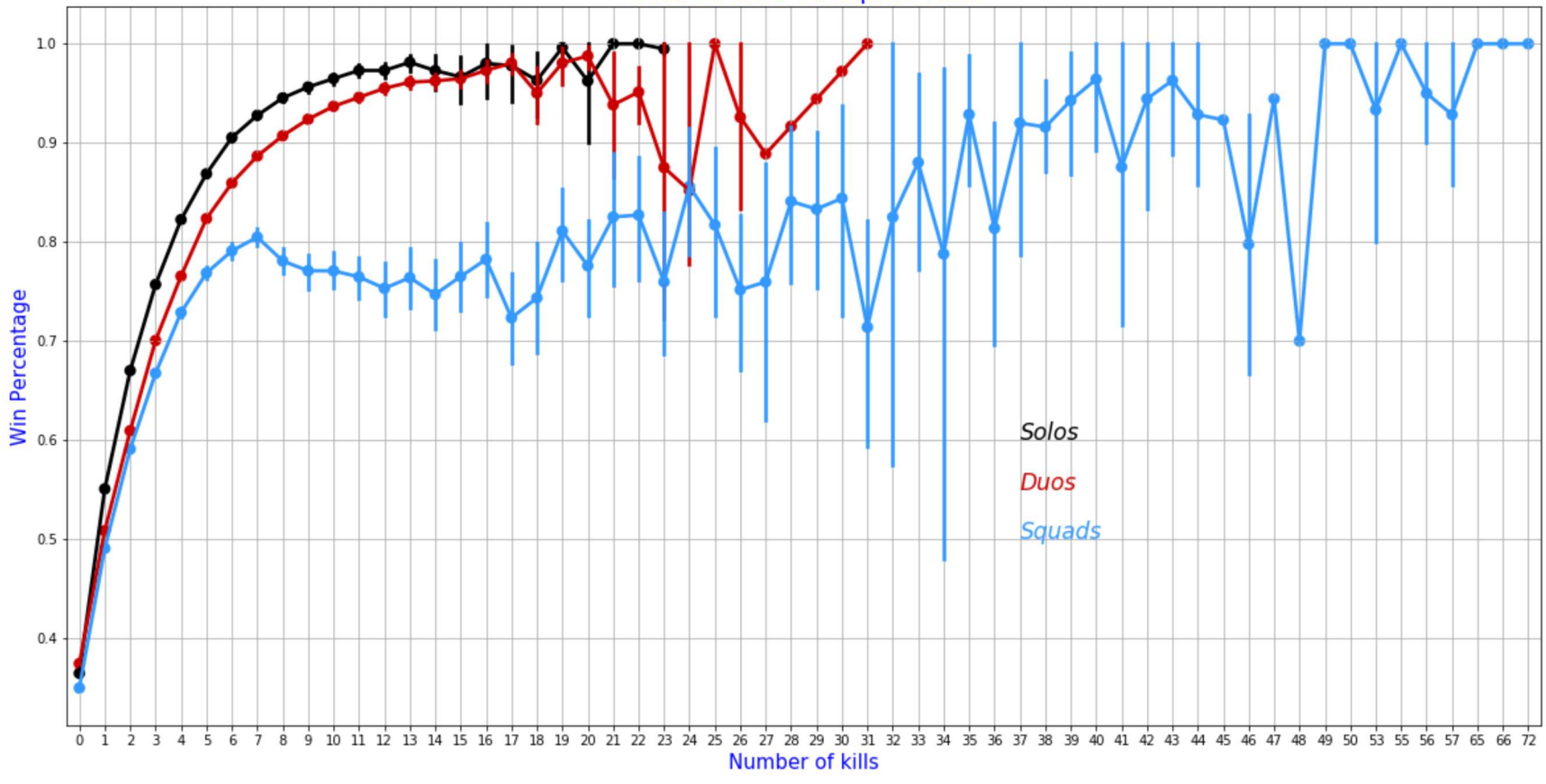
Solos, Duos and Squads

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 [30]:
```
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*
```

```
There are 709111 (15.95%) solo games, 3295326 (74.10%) duo games and 442529 (9.95%) squad games.
```

In [31]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='kills',y='winPlacePerc',data=solos,color='black',alpha=0.8)
sns.pointplot(x='kills',y='winPlacePerc',data=duos,color='#CC0000',alpha=0.8)
sns.pointplot(x='kills',y='winPlacePerc',data=squads,color='#3399FF',alpha=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 = 'italic')
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()
```



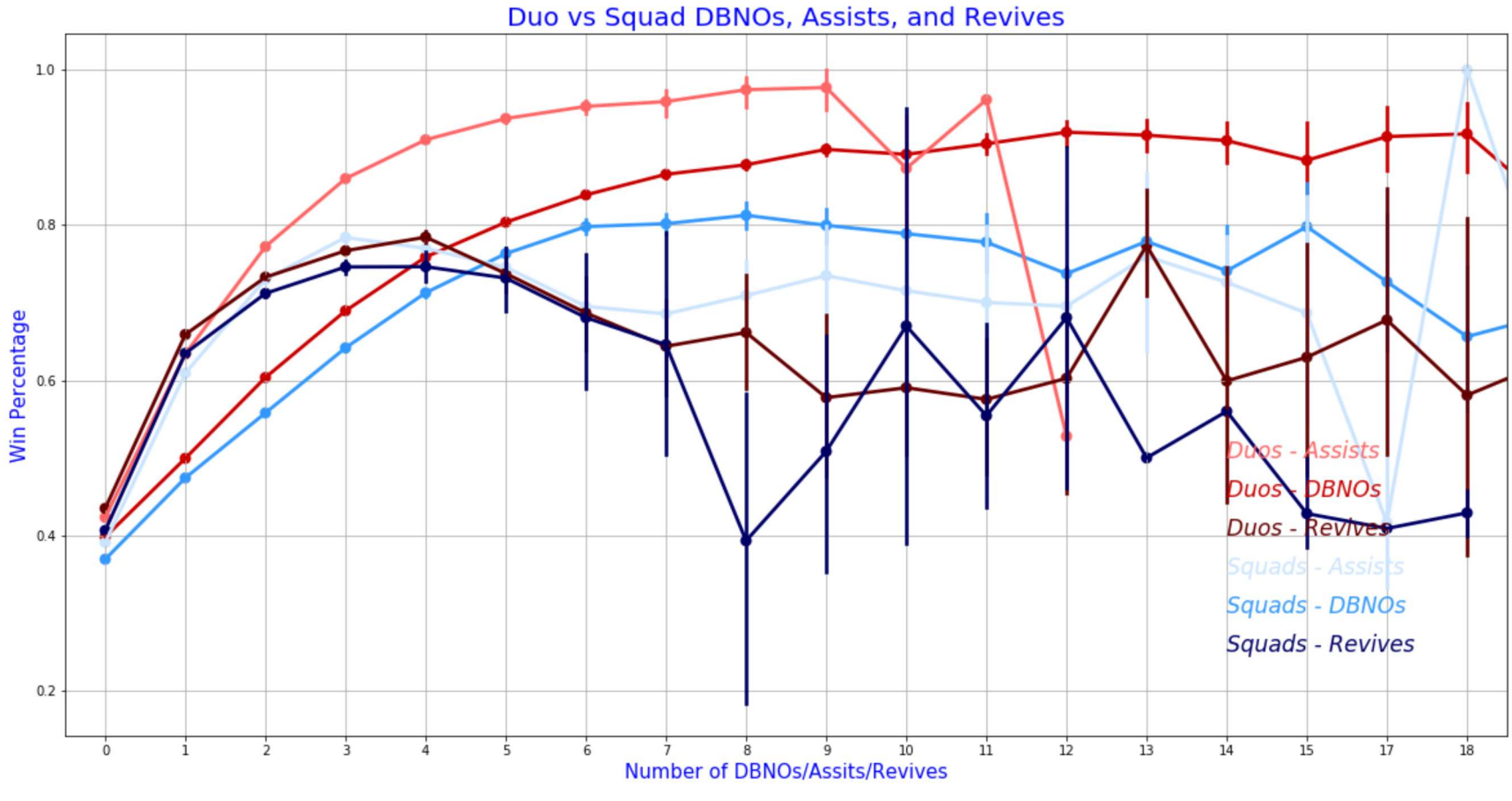
Solos and duos behave the same, but when playing squads kills don't matter that much.

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.

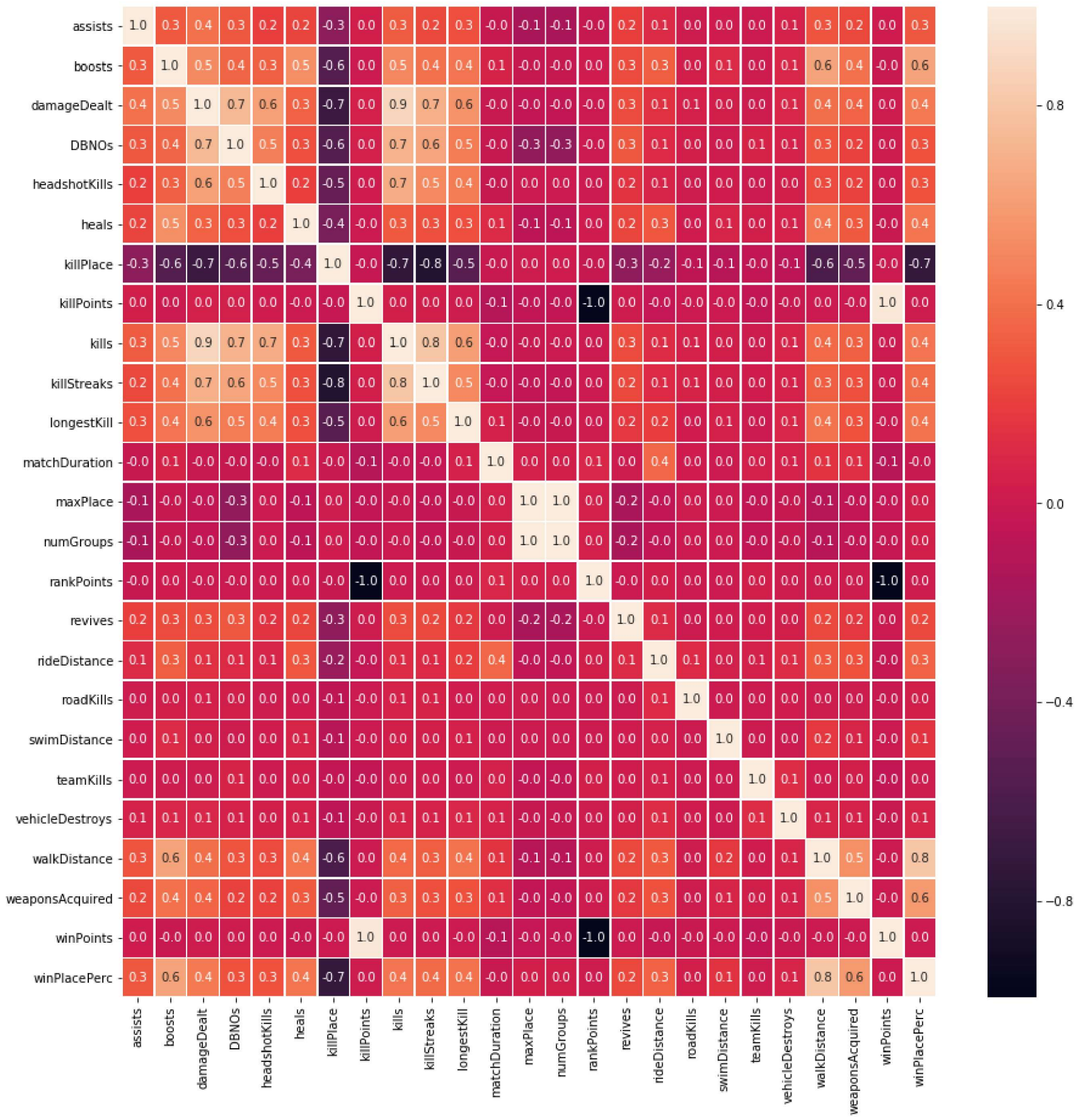
The attribute revive also happens in duos or squads.

In [32]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='DBNOs',y='winPlacePerc',data=duos,color='#CC0000',alpha=0.8)
sns.pointplot(x='DBNOs',y='winPlacePerc',data=squads,color='#3399FF',alpha=0.8)
sns.pointplot(x='assists',y='winPlacePerc',data=duos,color='#FF6666',alpha=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',alpha=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,style = '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,style = '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,color='blue')
plt.grid()
plt.show()
```



Pearson correlation between variables

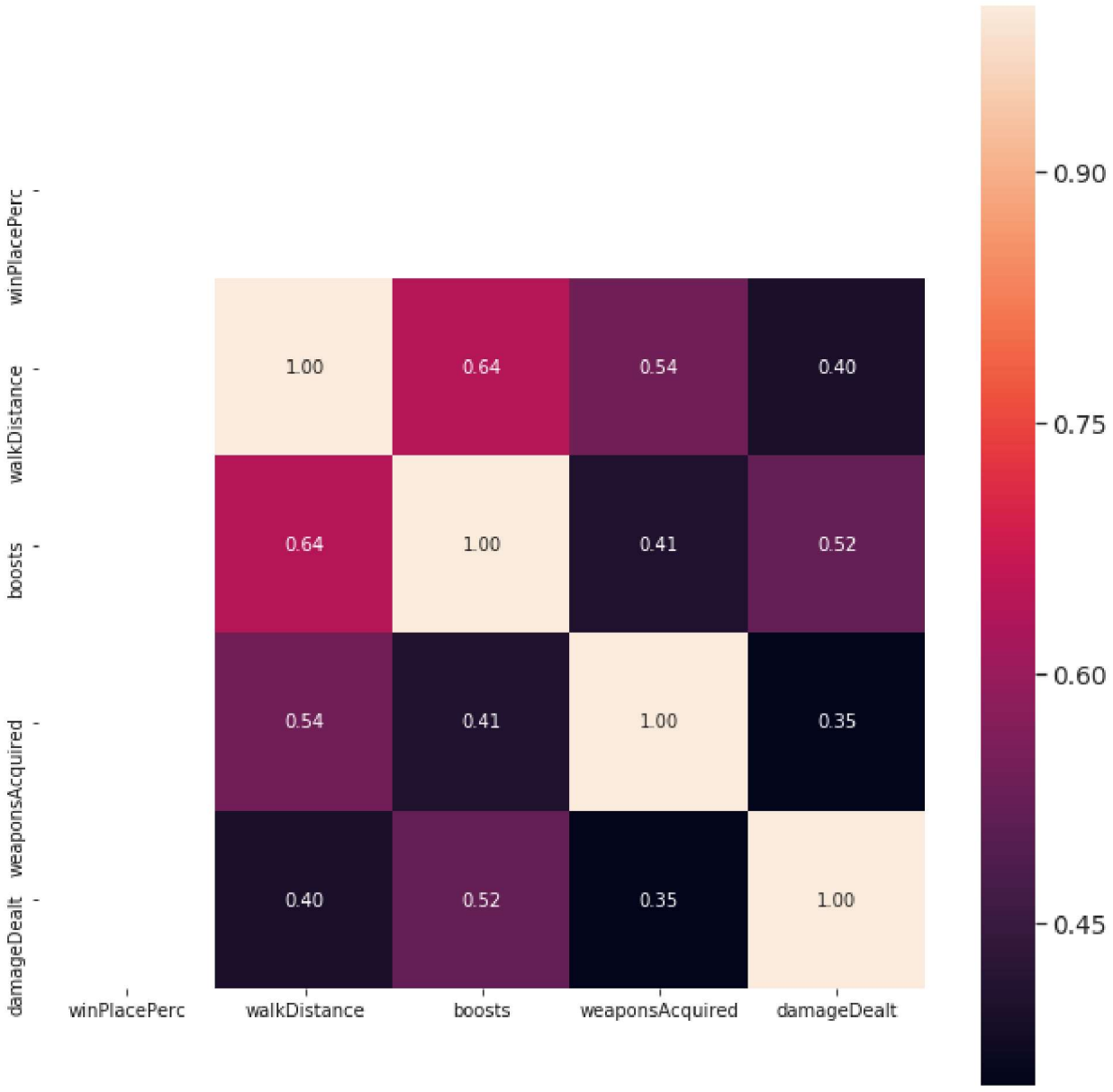
In [33]:
```
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(train.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.show()
```



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 [34]:
```
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', annot_kws={'size': 10}, yticklabels=cols.values, xt
plt.show()
```



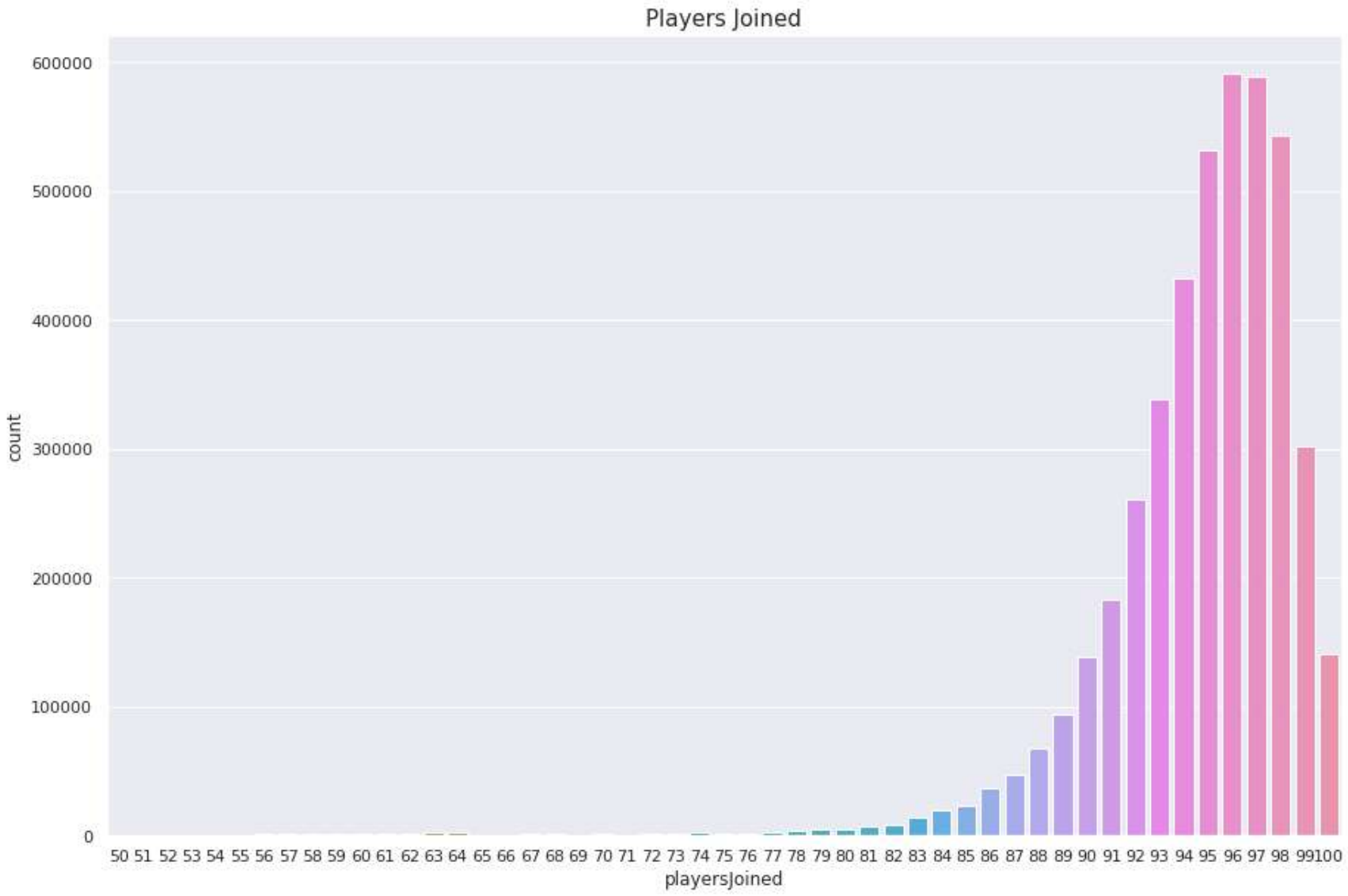
Let's plot the above variables and the killPlace variable as pairs.

Feature Engineering

A game 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 [36]:
```
train['playersJoined'] = train.groupby('matchId')['matchId'].transform('count')
```

In [37]:
```
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 [38]:
```
train['killsNorm'] = train['kills']*((100-train['playersJoined'])/100 + 1)
train['damageDealtNorm'] = train['damageDealt']*((100-train['playersJoined'])/100 + 1)
train[['playersJoined', 'kills', 'killsNorm', 'damageDealt', 'damageDealtNorm']][5:8]
```

Out[38]:

	playersJoined	kills	killsNorm	damageDealt	damageDealtNorm
5	95	1	1.05	100.000	105.00000
6	97	0	0.00	0.000	0.00000
7	96	0	0.00	8.538	8.87952

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

In [39]:
```
train['healsAndBoosts'] = train['heals']+train['boosts']
train['totalDistance'] = train['walkDistance']+train['rideDistance']+train['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 [40]:
```
train['boostsPerWalkDistance'] = train['boosts']/(train['walkDistance']+1) #The +1 is to avoid infinity, because there
train['boostsPerWalkDistance'].fillna(0, inplace=True)
train['healsPerWalkDistance'] = train['heals']/(train['walkDistance']+1) #The +1 is to avoid infinity, because there ar
train['healsPerWalkDistance'].fillna(0, inplace=True)
train['healsAndBoostsPerWalkDistance'] = train['healsAndBoosts']/(train['walkDistance']+1) #The +1 is to avoid infinity
train['healsAndBoostsPerWalkDistance'].fillna(0, inplace=True)
train[['walkDistance', 'boosts', 'boostsPerWalkDistance' ,'heals',  'healsPerWalkDistance', 'healsAndBoosts', 'healsAnd
```

Out[40]:

	walkDistance	boosts	boostsPerWalkDistance	heals	healsPerWalkDistance	healsAndBoosts	healsAndBoostsPerWalkDistance
40	327.30	1	0.003046	1	0.003046	2	0.006092
41	128.80	0	0.000000	0	0.000000	0	0.000000
42	52.52	0	0.000000	0	0.000000	0	0.000000
43	534.10	1	0.001869	0	0.000000	1	0.001869
44	2576.00	4	0.001552	6	0.002328	10	0.003880

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

In [41]:
```
train['killsPerWalkDistance'] = train['kills']/(train['walkDistance']+1) #The +1 is to avoid infinity, because there ar
train['killsPerWalkDistance'].fillna(0, inplace=True)
train[['kills', 'walkDistance', 'rideDistance', 'killsPerWalkDistance', 'winPlacePerc']].sort_values(by='killsPerWalkDi
```

Out[41]:

	kills	walkDistance	rideDistance	killsPerWalkDistance	winPlacePerc
4115816	29	0.0	0.0	29.0	0.7500
3083358	30	0.0	0.0	30.0	0.7500
422093	30	0.0	0.0	30.0	1.0000
2394021	31	0.0	0.0	31.0	0.5385
3057746	31	0.0	0.0	31.0	0.7500
2998470	35	0.0	0.0	35.0	1.0000
1158891	36	0.0	0.0	36.0	0.5833
3062788	36	0.0	0.0	36.0	0.8667
1068513	38	0.0	0.0	38.0	0.8333
1702541	43	0.0	0.0	43.0	1.0000