In [54]:
```
import pandas as pd

import numpy as np
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import mean_absolute_error

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,pr
ecision_score,recall_score
```

In [2]:
```
train=pd.read_csv('train_V2.csv')
```

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

Out[3]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	hea
0	7f96b2f878858a	4d4b580de459be	a10357fd1a4a91	0	0	0.00	0	
1	eef90569b9d03c	684d5656442f9e	aeb375fc57110c	0	0	91.47	0	
2	1eaf90ac73de72	6a4a42c3245a74	110163d8bb94ae	1	0	68.00	0	
3	4616d365dd2853	a930a9c79cd721	f1f1f4ef412d7e	0	0	32.90	0	
4	315c96c26c9aac	de04010b3458dd	6dc8ff871e21e6	0	0	100.00	0	

5 rows × 29 columns

In [4]:
```
# Checking row with winPlacePerc with NULL values
train[train['winPlacePerc'].isnull()]
```

Out[4]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
2744604	f70c74418bb064	12dfbede33f92b	224a123c53e008	0	0	0.0	0

1 rows × 29 columns

In [5]:
```
#Deleting the row
train.drop(2744604, inplace=True)
```

In [6]:
```
train[train['winPlacePerc'].isnull()]
```

Out[6]:

Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	...

0 rows × 29 columns

In [17]:
```
# Create feature totalDistance
train['totalDistance'] = train['rideDistance'] + train['walkDistance']
+ train['swimDistance']
# Create feature killsWithoutMoving
train['killsWithoutMoving'] = ((train['kills'] > 0) & (train['totalDist
ance'] == 0))
```

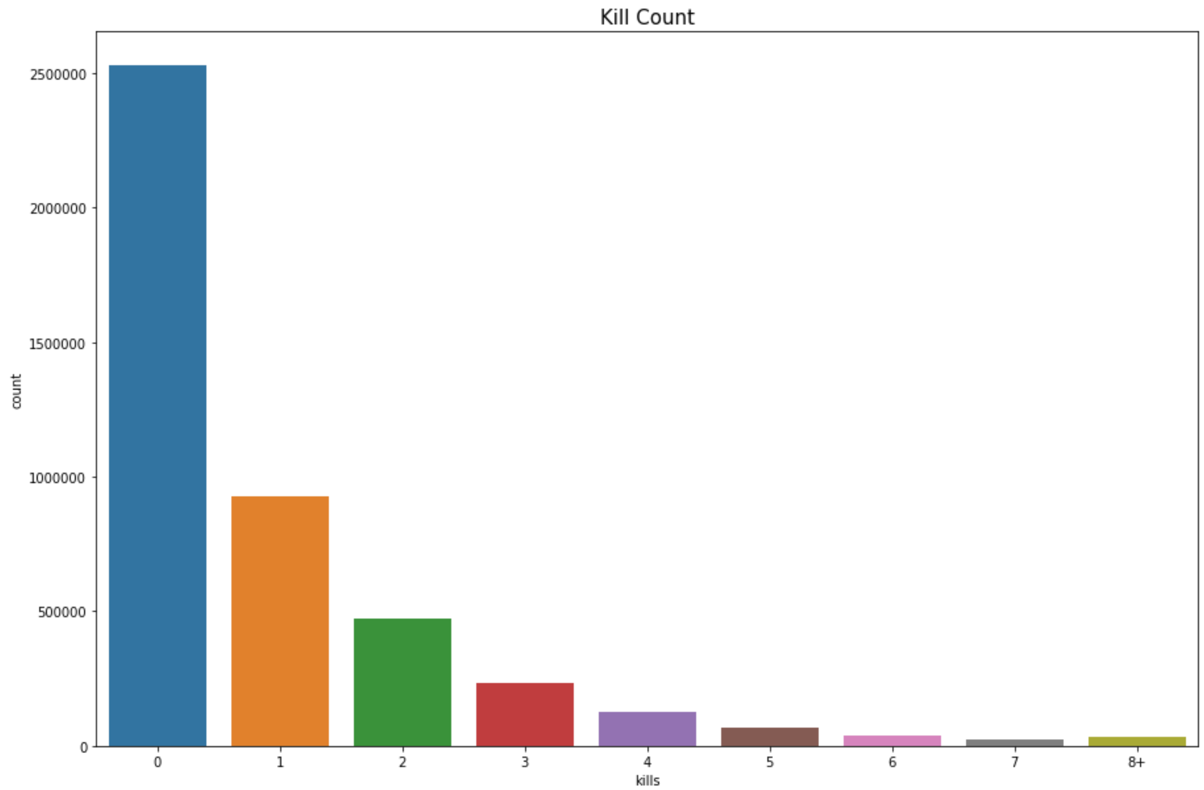
In [ ]:

The killer Players

In [52]:
```
print("The average person kills {:.4f} players, 99% of people have {} k
ills or less, while the most kills ever \
recorded is {}.".format(train['kills'].mean(),train['kills'].quantile(
0.99), train['kills'].max()))
```

```
The average person kills 0.9223 players, 99% of people have 7.0 kills o
r less, while the most kills ever recorded is 30.
```

In [53]:
```
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()
```

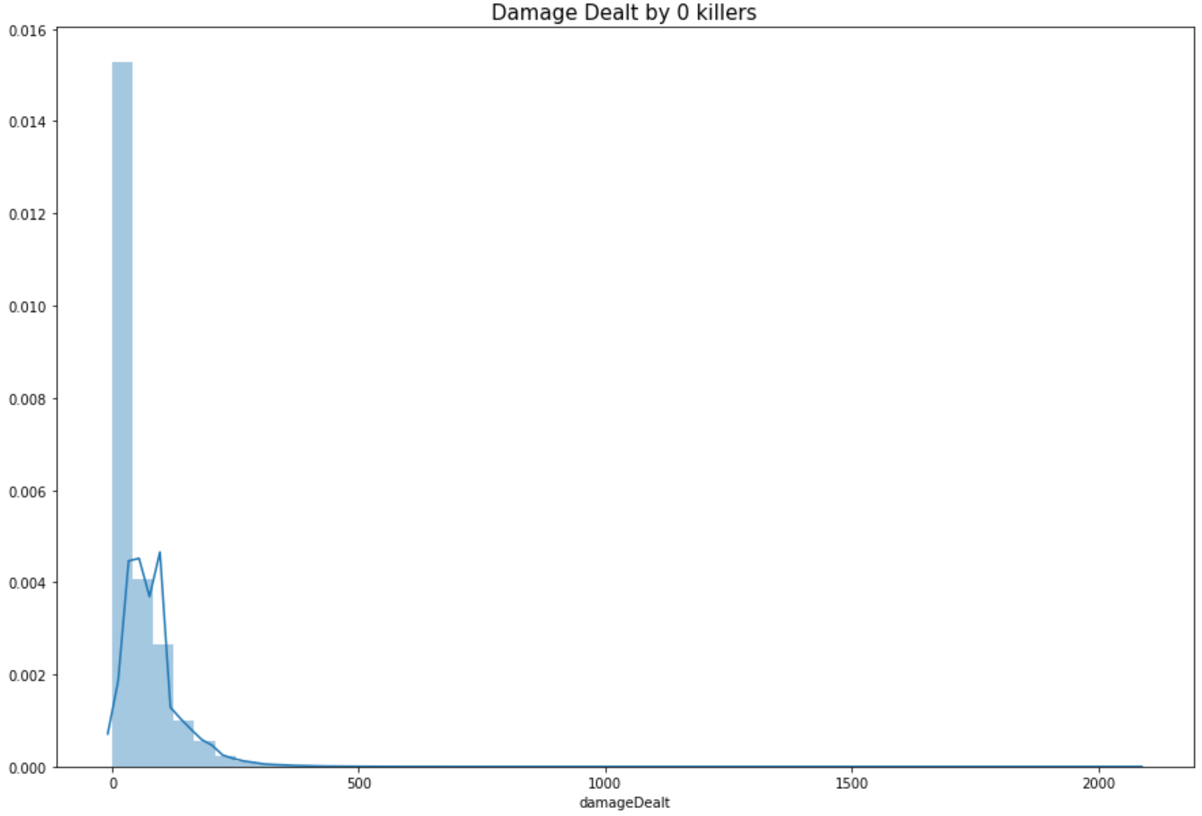


Most players are not able to kill a single person, so now we will plot the damage done by players with 0 kills.

In [54]:
```
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()
```

```
/usr/local/lib/python3.5/site-packages/scipy/stats/stats.py:1713: Futur
eWarning: Using a non-tuple sequence for multidimensional indexing is d
eprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future t
his will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Most of the player don't do any damage, but some have. So, we are

checking who have won without killing anyone and without doing any damage.

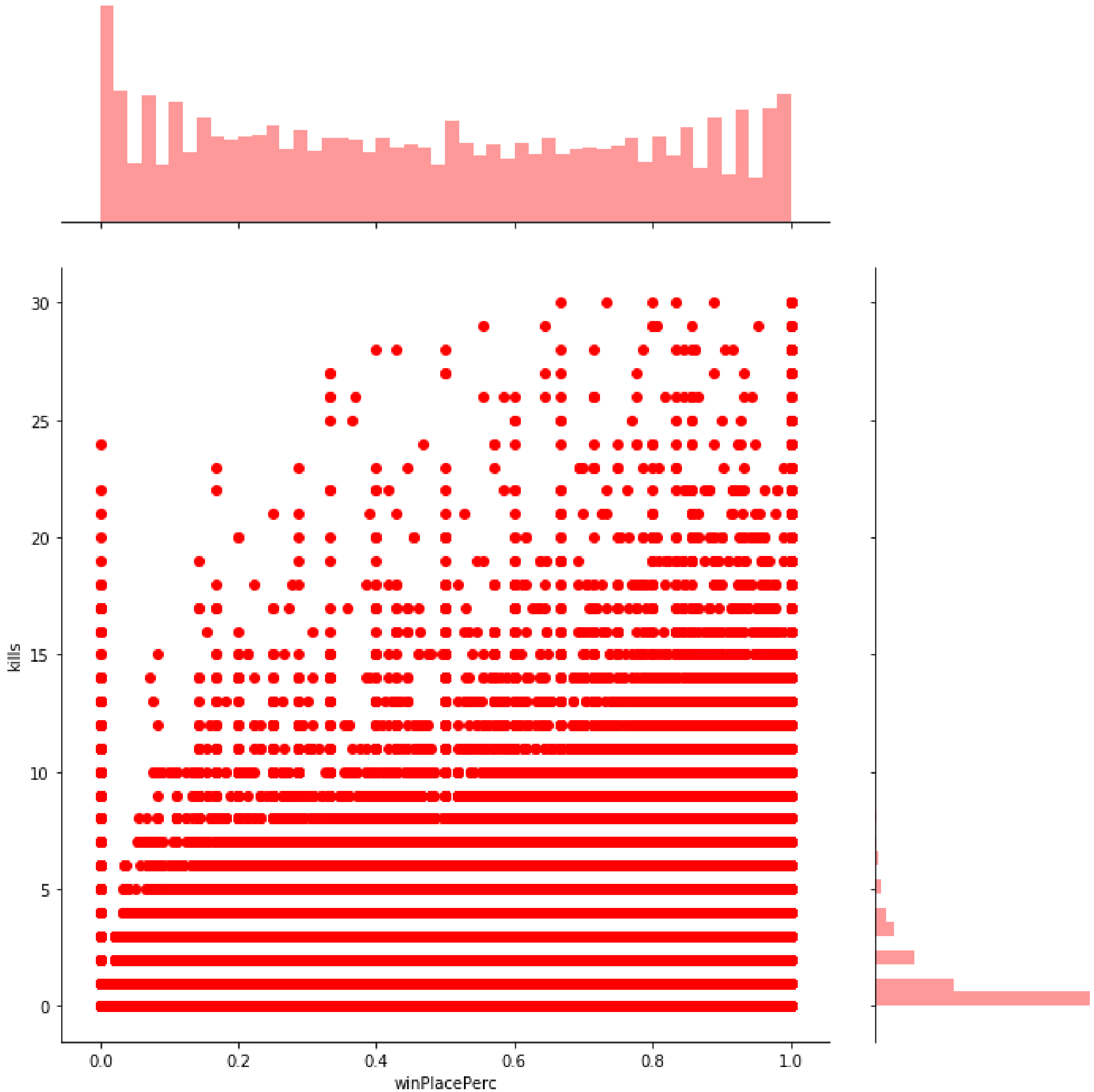
In [55]:
```
print("{} players ({:.4f}%) have won without a single kill!".format(len
(data[data['winPlacePerc']==1]), 100*len(data[data['winPlacePerc']==1])
/len(train)))

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

```
16661 players (0.3748%) have won without a single kill!
4769 players (0.1073%) have won without dealing damage!
```

Plotting win placement percentage vs kills

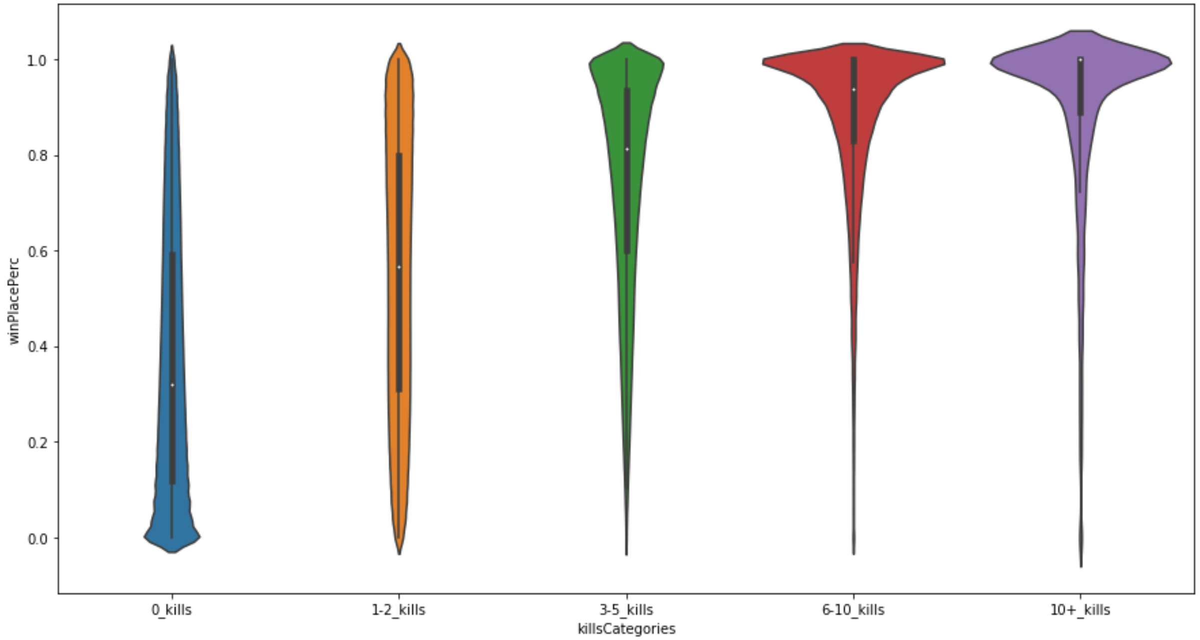
In [56]:
```
sns.jointplot(x="winPlacePerc", y="kills",height=10, data=train, ratio=
3, color="r")
plt.show()
```



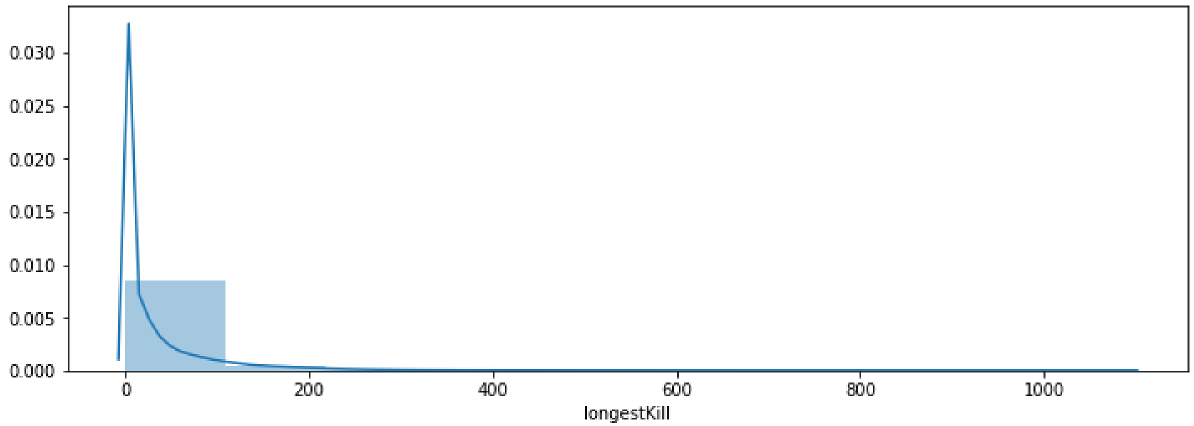
killing has a correlation with winning. So, we are grouping players

based on kills (0 kills, 1-2 kills, 3-5 kills, 6-10 kills and 10+ kills).

In [57]:
```
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.violinplot(x="killsCategories", y="winPlacePerc", data=kills)
plt.show()
```



In [26]:
```
# Plot the distribution of longestKill
plt.figure(figsize=(12,4))
sns.distplot(train['longestKill'], bins=10)
plt.show()
```



In [27]:
```
# Check out players who made kills with a distance of more than 1 km
display(train[train['longestKill'] >= 1000].shape)
train[train['longestKill'] >= 1000].head(10)
```

```
(20, 31)
```

Out[27]:

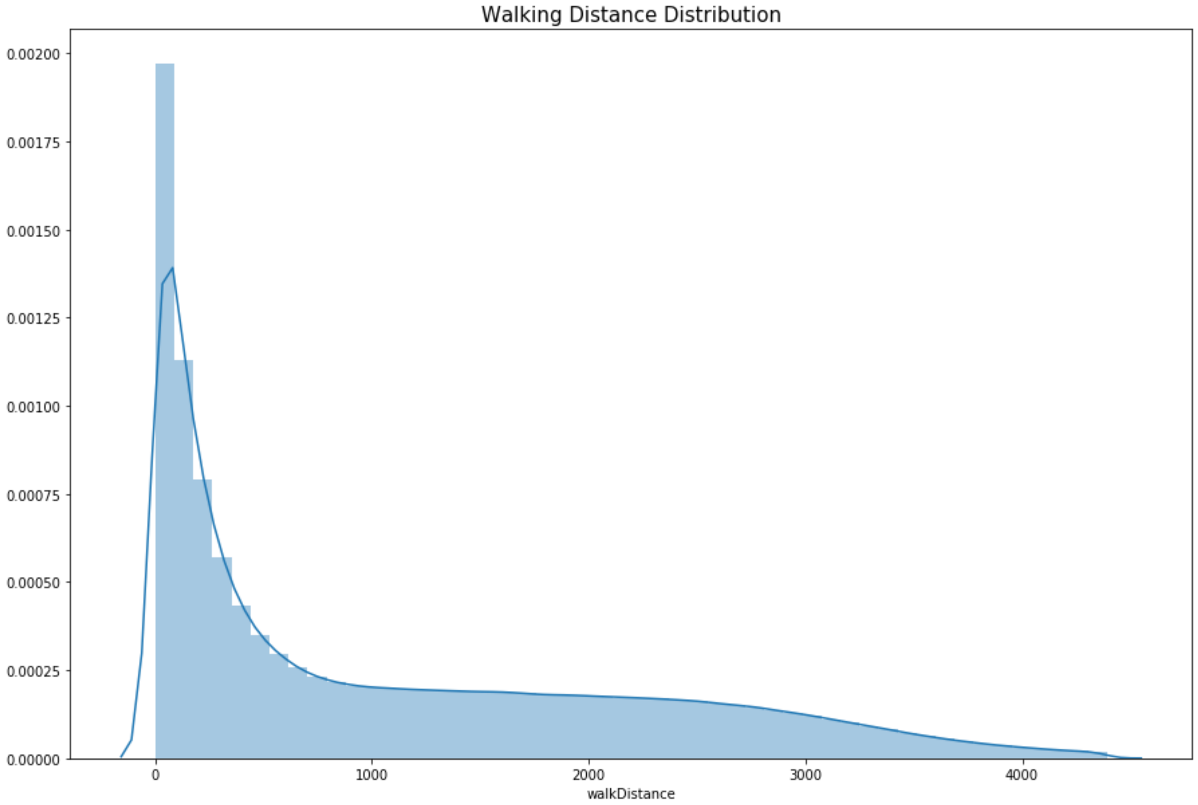
	Id	groupId	matchId	assists	boosts	damageDealt	DBNC
202281	88e2af7d78af5a	34ddeede52c042	4346bc63bc67fa	0	3	783.9	
240005	41c2f5c0699807	9faecf87ab4275	634edab75860b3	5	0	1284.0	
324313	ef390c152bcc3d	30fd444be3bbc1	4f7f8d6cf558b4	2	0	1028.0	
656553	9948b058562163	c8cb8491112bf6	0104eeb664494d	6	0	1410.0	1
803632	4e7e6c74e3c57d	94698690918933	da91b0c3d875f8	0	0	196.8	
895411	1f5ba6e0cfb968	512ea24b831be3	5fb0d8b1fc16cf	4	0	1012.0	1
1172437	303a93cfa1f46c	8795d39fd0df86	9c8962b58bb3e3	2	1	329.3	
1209416	528659ff1c1aec	7d1ba83423551d	ea9386587d5888	0	6	1640.0	
1642712	91966848e08e2f	0ee4fbd27657c9	17dea22cefe62a	3	2	2103.0	
2015559	5ff0c1a9fab2ba	2d8119b1544f87	904cecf36217df	3	3	1302.0	

10 rows × 31 columns

In [28]:
```
# Remove outliers
train.drop(train[train['longestKill'] >= 1000].index, inplace=True)
```

The Runner Players

In [58]:
```
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()
```

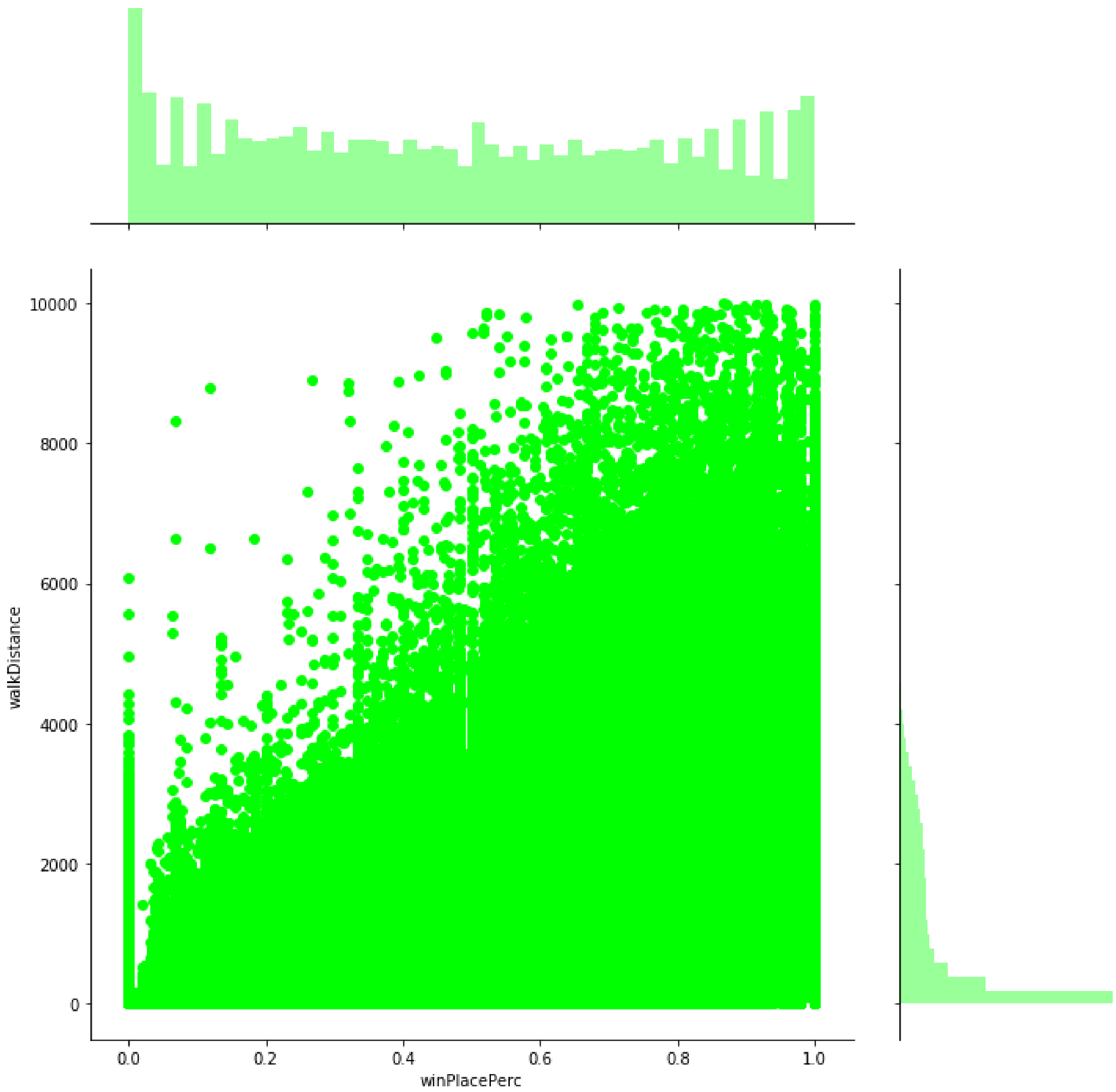


In [55]:
```
print("{} players ({:.4f}%) walked 0 meters. This means that they die b
efore even taking a step.".format(len(data[data['walkDistance'] == 0]),
 100*len(data1[data1['walkDistance']==0])/len(train)))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
<ipython-input-55-73636b236287> in <module>
----> 1 print("{} players ({:.4f}%) walked 0 meters. This means that th
ey die before even taking a step.".format(len(data[data['walkDistance']
 == 0]), 100*len(data1[data1['walkDistance']==0])/len(train)))

NameError: name 'data1' is not defined
```

In [60]:
```
sns.jointplot(x="winPlacePerc", y="walkDistance",  data=train, height=10, ratio=3, color="lime")
plt.show()
```



Walking has a high correlation with Win Placement Percentage.**

In [18]:
```
# Check players who kills without moving
display(train[train['killsWithoutMoving'] == True].shape)
train[train['killsWithoutMoving'] == True].head(10)
```

```
(1535, 31)
```

Out[18]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
1824	b538d514ef2476	0eb2ce2f43f9d6	35e7d750e442e2	0	0	593.00	0
6673	6d3a61da07b7cb	2d8119b1544f87	904cecf36217df	2	0	346.60	0
11892	550398a8f33db7	c3fd0e2abab0af	db6f6d1f0d4904	2	0	1750.00	0
14631	58d690ee461e9d	ea5b6630b33d67	dbf34301df5e53	0	0	157.80	0
15591	49b61fc963d632	0f5c5f19d9cc21	904cecf36217df	0	0	100.00	0
20881	40871bf43ddac7	2cea046b7d1dce	0600f86f11c6e4	0	0	506.10	4
23298	b950836d0427da	1f735b1e00d549	ad860f4e162bbc	1	0	1124.00	0
24640	aeced11d46de19	d4009ffa95bb4f	73f3ed869c9171	2	0	529.90	0
25659	6626c4d47cffa0	ee3fe5c0d917c3	341341834b7941	0	1	128.90	0
30079	869331b90bfa3f	869ea3ad036e53	fa373e28ff5062	0	0	85.56	0

10 rows × 31 columns

In [19]:
```
# Remove outliers
train.drop(train[train['killsWithoutMoving'] == True].index, inplace=Tr
ue)
```

In [20]:
```
# Players who got more than 10 roadKills
train[train['roadKills'] > 10]
```

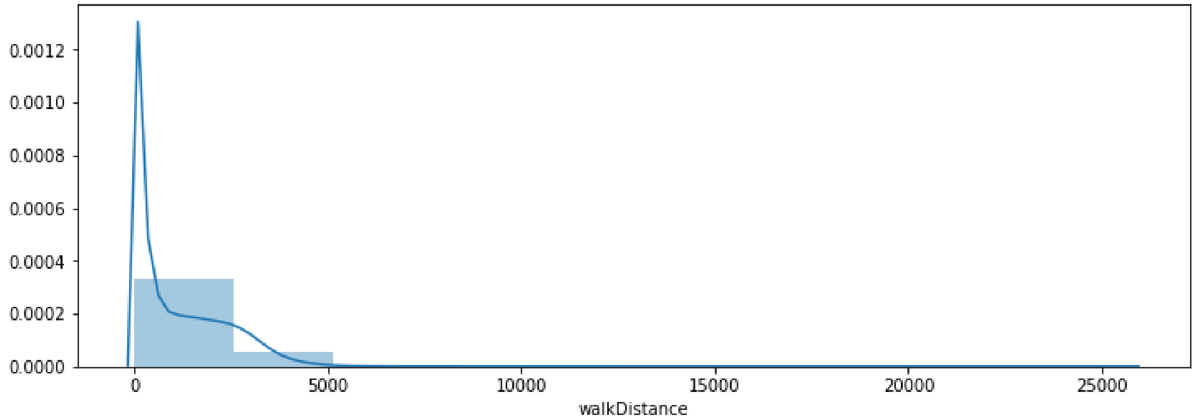
Out[20]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNC
2733926	c3e444f7d1289f	489dd6d1f2b3bb	4797482205aaa4	0	0	1246.0	
2767999	34193085975338	bd7d50fa305700	a22354d036b3d6	0	0	1102.0	
2890740	a3438934e3e535	1081c315a80d14	fe744430ac0070	0	8	2074.0	
3524413	9d9d044f81de72	8be97e1ba792e3	859e2c2db5b125	0	3	1866.0	

4 rows × 31 columns

In [29]:

```
# Plot the distribution of walkDistance
plt.figure(figsize=(12,4))
sns.distplot(train['walkDistance'], bins=10)
plt.show()
```



In [30]:

```
# walkDistance anomalies
display(train[train['walkDistance'] >= 10000].shape)
train[train['walkDistance'] >= 10000].head(10)
```

(219, 31)

Out[30]:

Id	groupId	matchId	assists	boosts	damageDealt	DBNOs

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
23026	8a6562381dd83f	23e638cd6eaf77	b0a804a610e9b0	0	1	0.00	0
34344	5a591ecc957393	6717370b51c247	a15d93e7165b05	0	3	23.22	0
49312	582685f487f0b4	338112cd12f1e7	d0afbf5c3a6dc9	0	4	117.20	1
68590	8c0d9dd0b4463c	c963553dc937e9	926681ea721a47	0	1	32.34	0
94400	d441bebd01db61	7e179b3366adb8	923b57b8b834cc	1	1	73.08	0
125103	db5a0cdc969dcb	50cc466757950e	c306a9745c4c1d	0	4	37.73	0
136421	955e60b09a96b1	30df08fe22a901	8669d01725f135	0	1	0.00	0
136476	0d75d05b5c988c	3da040ce77cd0b	65bc5211a569dd	0	3	0.00	0
154080	7e8a71d23381cd	e2c9f4f92840b2	a721de1aa05408	0	3	0.00	0
154128	32fdde4c716787	390ae9a51c11b8	82610ed1b4d033	0	4	52.16	0

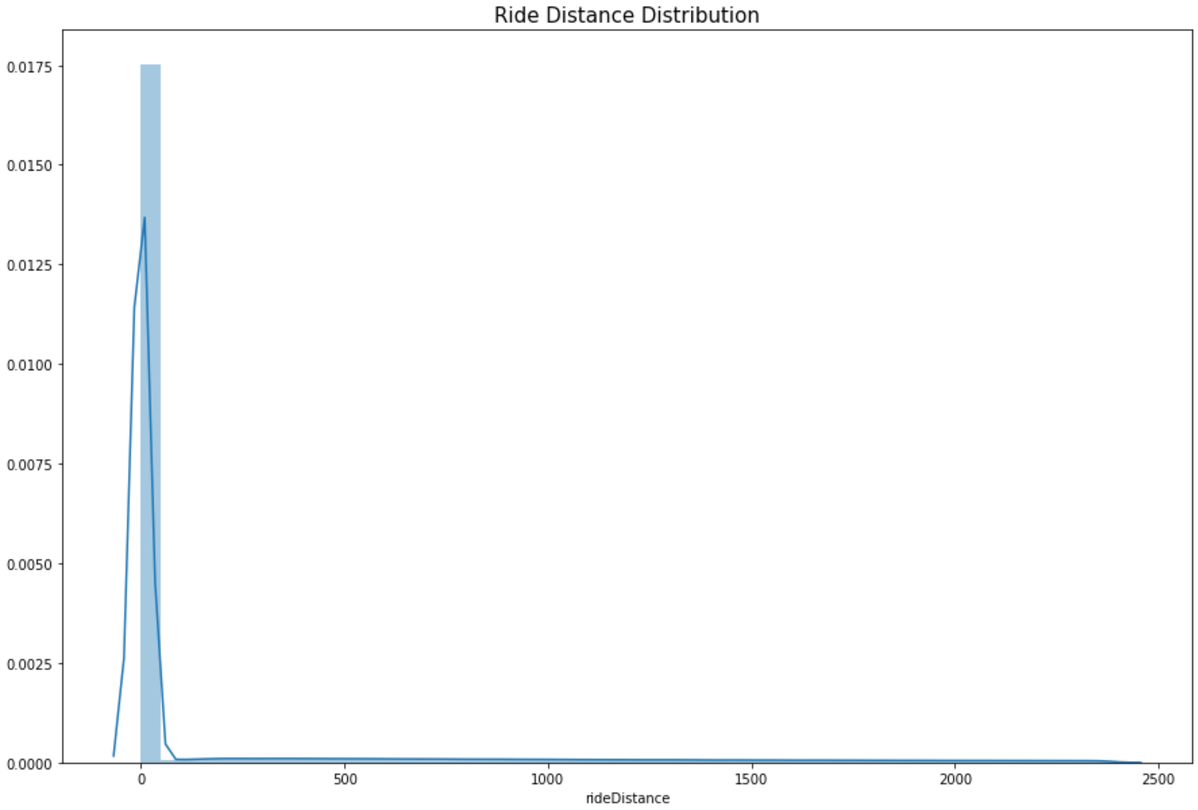
10 rows × 31 columns

In [31]:
```
# Remove outliers
train.drop(train[train['walkDistance'] >= 10000].index, inplace=True)
```

The Driver Players

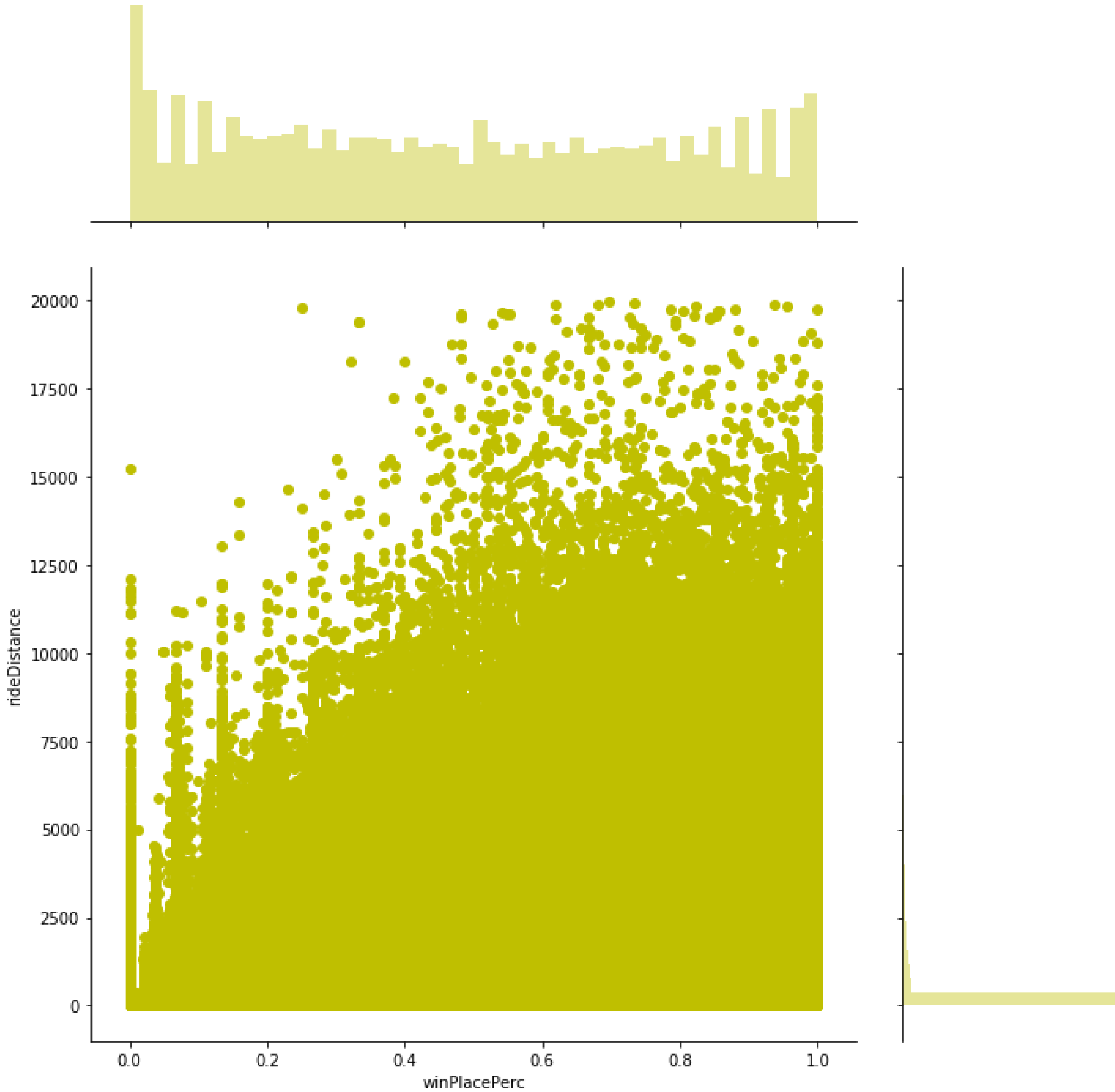
In [ ]:
```
print("The average person drives for {:.1f}m, 99% of people have drived {}m or less, while the maximum distance covered by a single person was {}m.".format(train['rideDistance'].mean(), train['rideDistance'].quantile(0.99), train['rideDistance'].max()))
```

In [63]:
```
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()
```



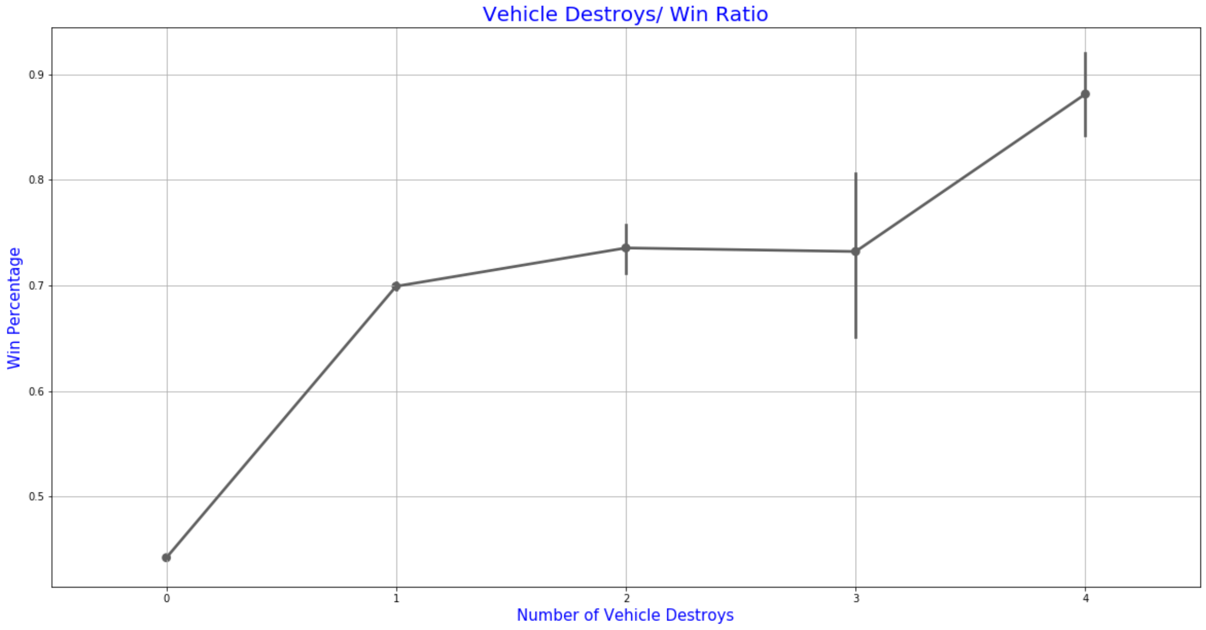
In [ ]:
```
print("{} players ({:.4f}%) drived for 0 meters.".format(len(data[data[
'rideDistance'] == 0]), 100*len(data1[data1['rideDistance']==0])/len(tr
ain)))
```

In [65]:
```
sns.jointplot(x="winPlacePerc", y="rideDistance", data=train, height=10
, ratio=3, color="y")
plt.show()
```



There is a small correlation between Ride Distance and Win Placement Percentage.

In [66]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='vehicleDestroys',y='winPlacePerc',data=data,color='#60
6060',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()
```

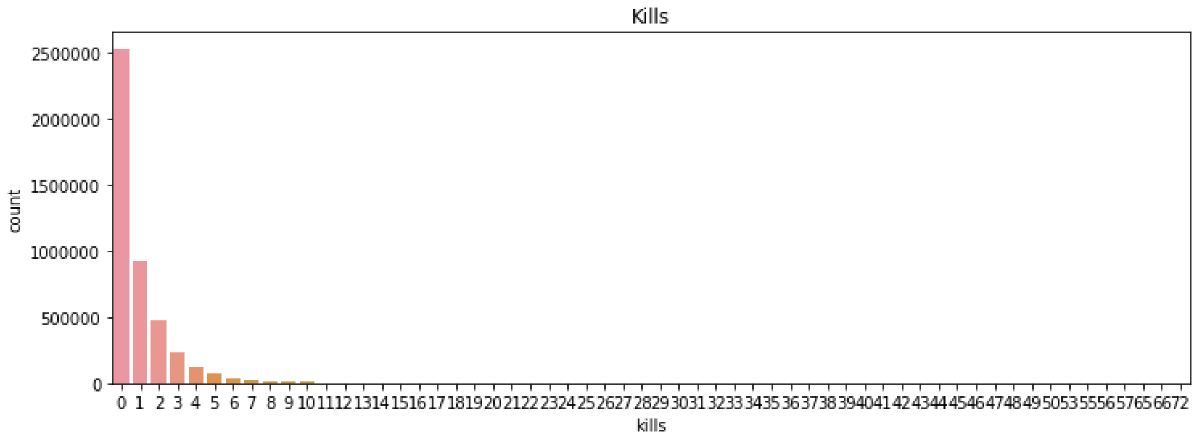


The winning percentage increases with increase in number of vehicles destroyed

In [21]:
```
# Drop roadKill outliers
train.drop(train[train['roadKills'] > 10].index, inplace=True)
```

In [22]:
```
# Plot the distribution of kills
plt.figure(figsize=(12,4))
```

```
sns.countplot(data=train, x=train['kills']).set_title('Kills')
plt.show()
```



In [23]:

```
# Players who got more than 30 kills
display(train[train['kills'] > 30].shape)
train[train['kills'] > 30].head(10)
```

(95, 31)

Out[23]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
57978	9d8253e21ccbbd	ef7135ed856cd8	37f05e2a01015f	9	0	3725.0	0
87793	45f76442384931	b3627758941d34	37f05e2a01015f	8	0	3087.0	0
156599	746aa7eabf7c86	5723e7d8250da3	f900de1ec39fa5	21	0	5479.0	0
160254	15622257cb44e2	1a513eeecfe724	db413c7c48292c	1	0	4033.0	0
180189	1355613d43e2d0	f863cd38c61dbf	39c442628f5df5	5	0	3171.0	0
334400	810f2379261545	7f3e493ee71534	f900de1ec39fa5	20	0	6616.0	0
353128	f3e9746e3ff151	4bc1f00f07b304	a9e84c456cc859	2	0	3834.0	0
457829	265e23756baa0b	9d94424171c2a1	664dee9ed8f646	3	0	2907.0	0
488335	31a0682922ef45	275a27a3ee4cc8	3037f74ef8a3a3	2	0	3055.0	0

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
662650	dd424a8b74bd49	ac9dea6d62f2e6	8a728def0644be	9	0	3454.0	38

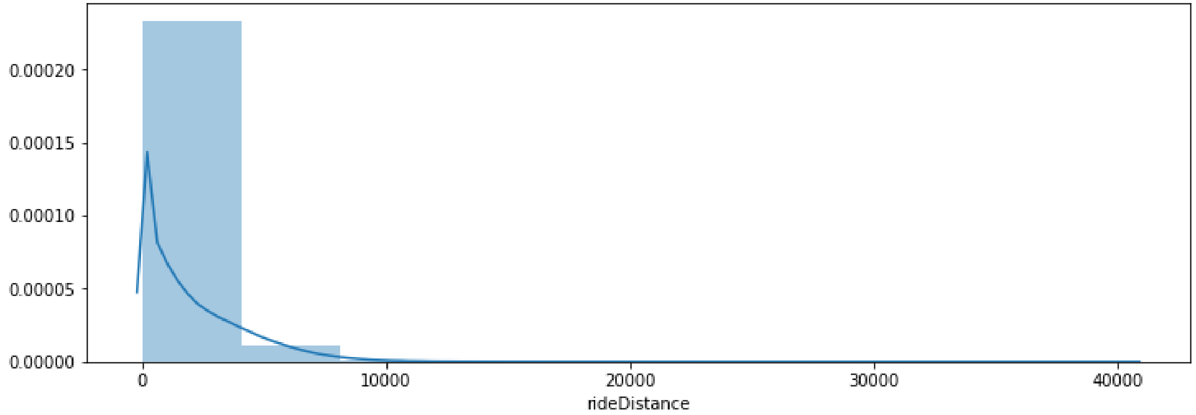
10 rows × 31 columns

In [24]:

```
# Remove outliers
train.drop(train[train['kills'] > 30].index, inplace=True)
```

In [32]:

```
# Plot the distribution of rideDistance
plt.figure(figsize=(12,4))
sns.distplot(train['rideDistance'], bins=10)
plt.show()
```



In [33]:

```
# rideDistance anomalies
display(train[train['rideDistance'] >= 20000].shape)
train[train['rideDistance'] >= 20000].head(10)
```

```
(150, 31)
```

Out[33]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
28588	6260f7c49dc16f	b24589f02eedd7	6ebea3b4f55b4a	0	0	99.20	0

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
63015	adb7dae4d0c10a	8ede98a241f30a	8b36eac66378e4	0	0	0.00	0
70507	ca6fa339064d67	f7bb2e30c3461f	3bfd8d66edbeff	0	0	100.00	0
72763	198e5894e68ff4	ccf47c82abb11f	d92bf8e696b61d	0	0	0.00	0
95276	c3fabfce7589ae	15529e25aa4a74	d055504340e5f4	0	7	778.20	0
140097	9944fbbea2b91e	18b4d5f4bb1906	d9d4a3e50cae75	1	0	12.55	0
297186	88904c200175b6	012a61a01e146e	7a270c25e9b70c	0	1	0.00	0
371098	f7071357f6b762	f3ee20821f4627	ac47c86bf385bf	0	0	72.92	1
403647	c65da7b3fceef5	814d1b3736e276	ff9f570b555d48	0	2	0.00	0
426708	149e224a2330ae	6d8cb80b3de8ff	f8b8e2643f60ee	0	2	0.00	0

10 rows × 31 columns

In [34]:
```
# Remove outliers
train.drop(train[train['rideDistance'] >= 20000].index, inplace=True)
```

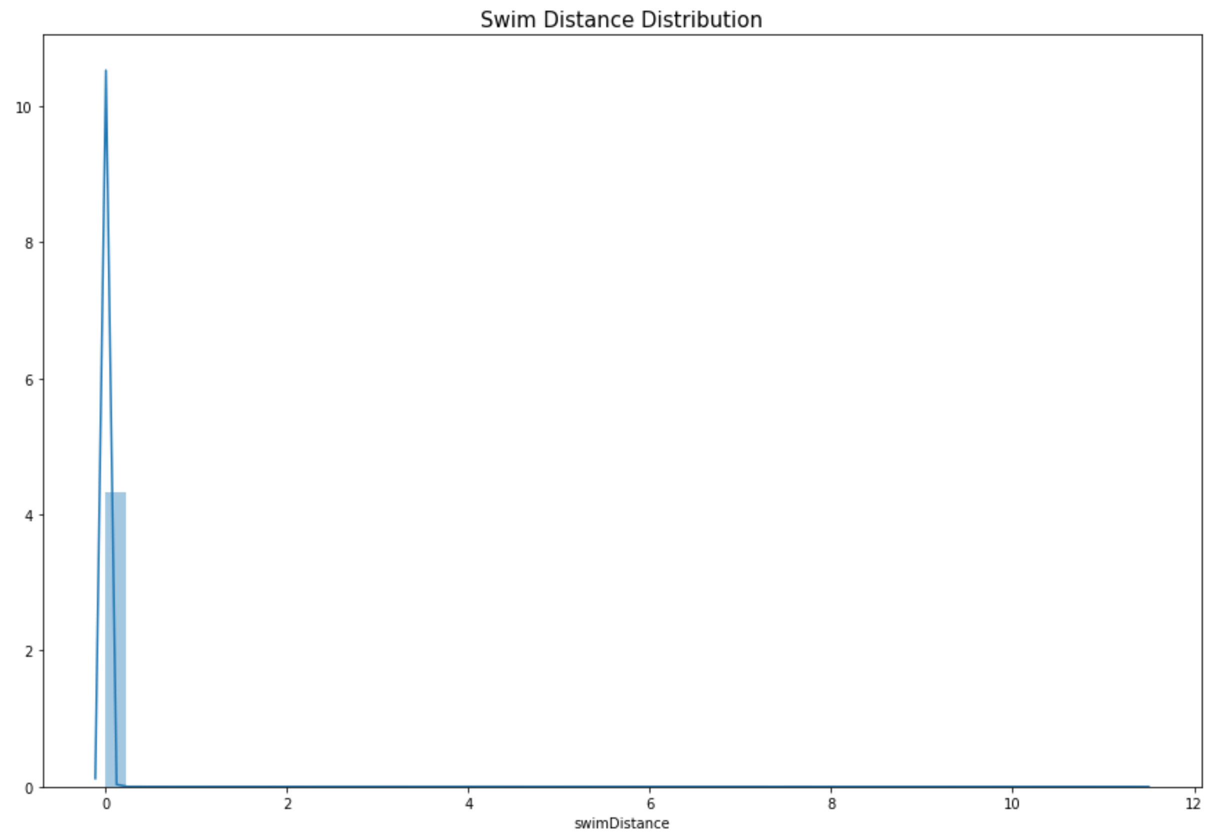
The swimmer Players

In [56]:
```
print("The average person swims for {:.1f}m, 99% of people have swimmed {}m or less, while the maximum distance covered was {}m.".format(train['swimDistance'].mean(), train['swimDistance'].quantile(0.99), train['swimDistance'].max()))
```

```
The average person swims for 4.5m, 99% of people have swimmed 122.9m or less, while the maximum distance covered was 1980.0m.
```

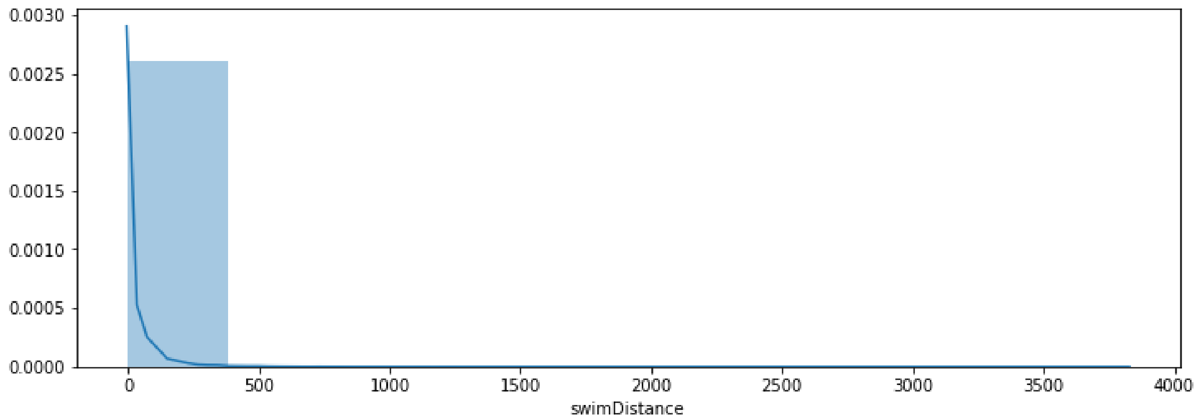
In [68]:
```
data = train.copy()
data = data[data['swimDistance'] < train['swimDistance'].quantile(0.95)]
plt.figure(figsize=(15,10))
```

```
plt.title("Swim Distance Distribution",fontsize=15)
sns.distplot(data['swimDistance'])
plt.show()
```



Almost no one swims. So, there is no use of proceeding with this.

In [35]:
```
# Plot the distribution of swimDistance
plt.figure(figsize=(12,4))
sns.distplot(train['swimDistance'], bins=10)
plt.show()
```



In [36]:

```
# Players who swam more than 2 km
train[train['swimDistance'] >= 2000]
```

Out[36]:

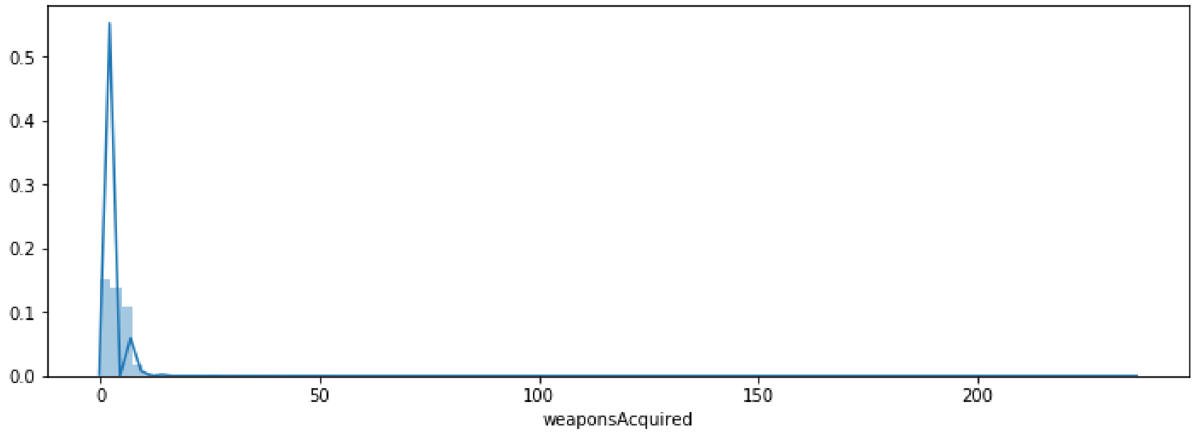
	Id	groupId	matchId	assists	boosts	damageDealt	DBNC
177973	c2e9e5631f4e54	23213058f83abe	f01eb1073ef377	0	5	78.12	
274258	ba5e3dfb5a0fa0	383db055216ec2	d6e13468e28ab4	0	4	53.32	
1005337	d50c9d0e65fe2a	4996575c11abcb	668402592429f8	0	1	503.00	
1195818	f811de9de80b70	d08ddf7beb6252	8a48703ab52ec8	0	7	352.30	
1227362	a33e917875c80e	5b72674b42712b	5fb0d8b1fc16cf	0	1	589.20	
1889163	bd8cc3083a9923	1d5d17140d6fa4	8e2e6022d6e5c8	0	0	0.00	
2065940	312ccbb27b99aa	47c7f4d69e2fb1	b4b11756321f3a	1	3	49.59	
2327586	8773d0687c6aae	b17f46f9f6666c	56ee5897512c86	3	1	474.40	
2784855	a8653b87e83892	383db055216ec2	d6e13468e28ab4	1	4	843.80	
3359439	3713b36e1ba9e1	1f7aed9240864a	584447ed875c85	0	0	0.00	
3513522	aff482b8c08486	383db055216ec2	d6e13468e28ab4	0	4	109.80	
4132225	2496e3223a8b5d	78980ab36f7642	23ec7dd5546022	0	0	0.00	

12 rows × 31 columns

```
In [37]: # Remove outliers
train.drop(train[train['swimDistance'] >= 2000].index, inplace=True)
```

Weapons acquired

```
In [38]: # Plot the distribution of weaponsAcquired
plt.figure(figsize=(12,4))
sns.distplot(train['weaponsAcquired'], bins=100)
plt.show()
```




```
In [39]: # Players who acquired more than 80 weapons
display(train[train['weaponsAcquired'] >= 80].shape)
train[train['weaponsAcquired'] >= 80].head()
```

(19, 31)

Out[39]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNO
233643	7c8c83f5f97d0f	b33b210a52a2f8	2e8a0917a71c43	0	0	67.11	
588387	c58e3e0c2ba678	3d3e6100c07ff0	d04dbb98249f76	0	1	175.30	

	Id	groupId	matchId	assists	boosts	damageDealt	DBNO
1437471	8f0c855d23e4cd	679c3316056de8	fbaf1b3ae1d884	1	0	100.00	
1449293	db54cf45b9ed1c	898fccaeeb041d	484b4ae51fe80f	0	0	0.00	
1592744	634a224c53444e	75fa7591d1538c	f900de1ec39fa5	9	0	1726.00	

5 rows × 31 columns

In [40]:
```
# Remove outliers
train.drop(train[train['weaponsAcquired'] >= 80].index, inplace=True)
```

The Healer and booster Players

In [69]:
```
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(), tra
in['boosts'].quantile(0.99), train['boosts'].max()))
```

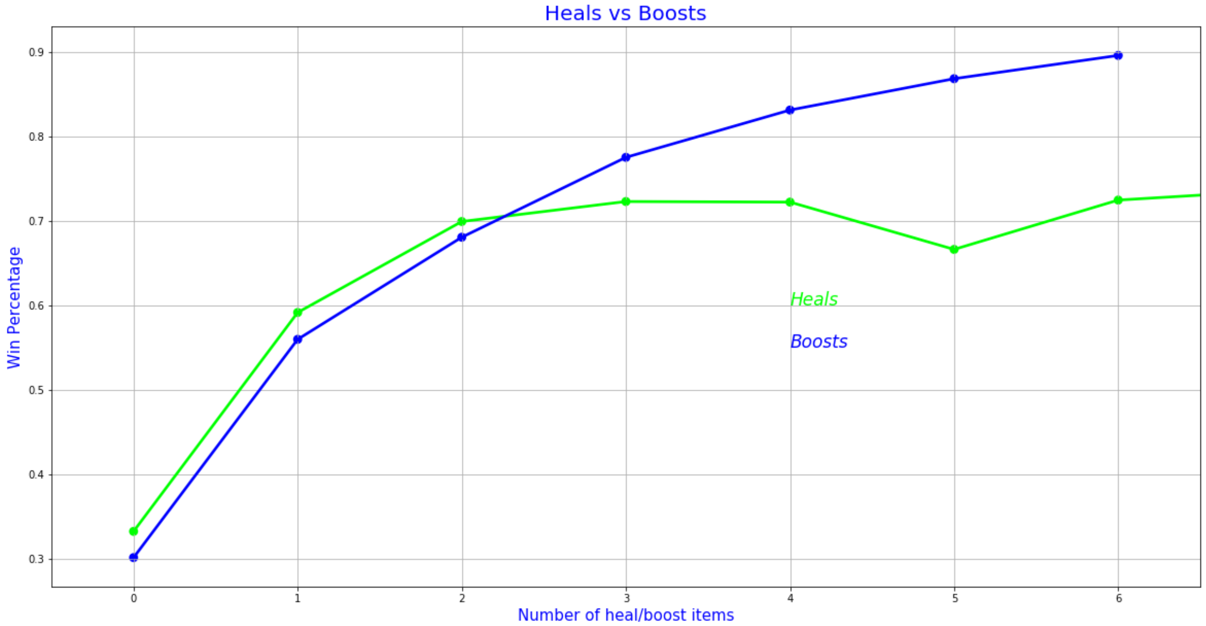
```
The average person uses 1.4 heal items, 99% of people use 12.0 or less,
while the doctor used 39.
The average person uses 1.1 boost items, 99% of people use 7.0 or less,
while the doctor used 33.
```

In [7]:
```
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')
```

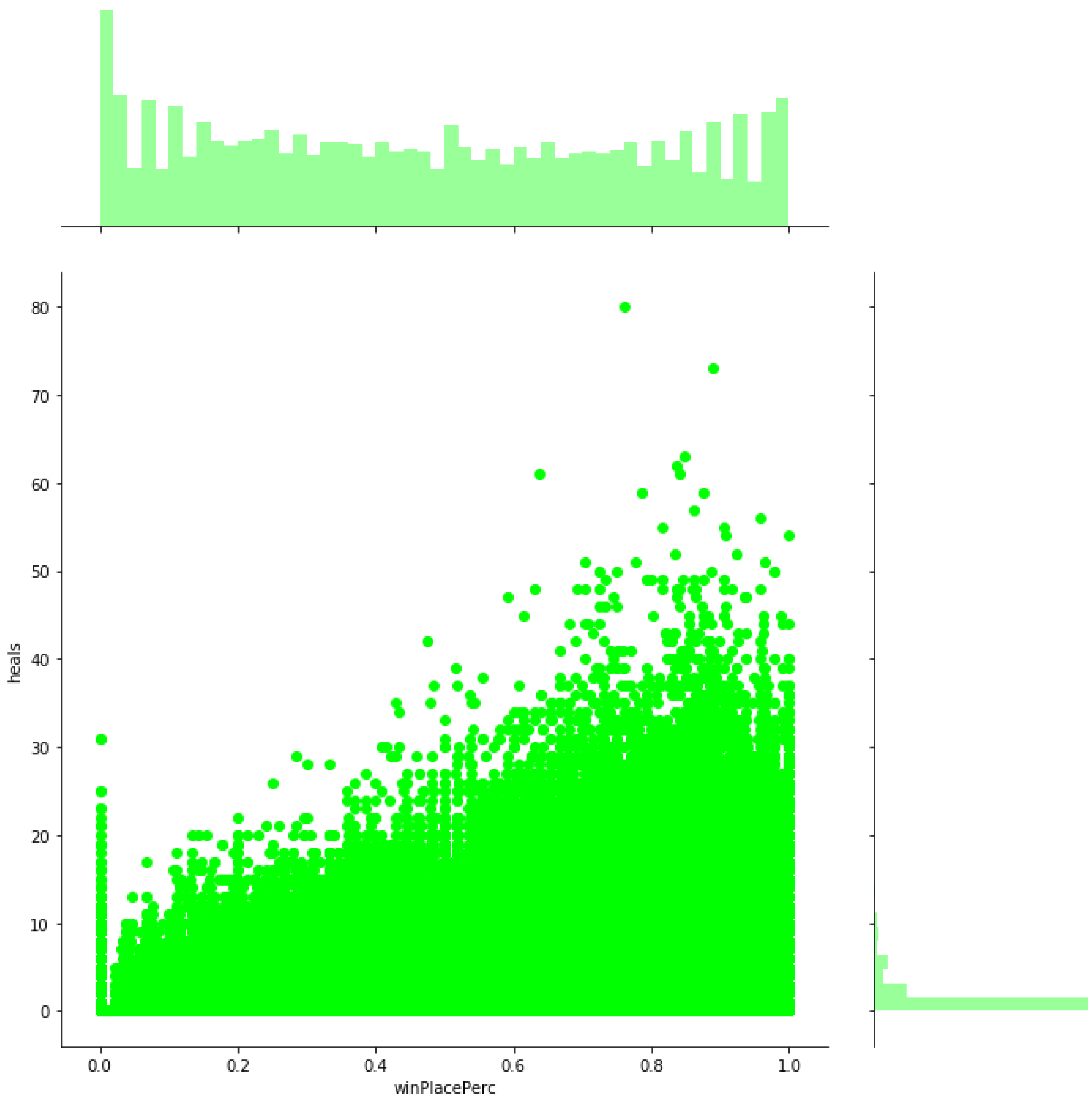
```
plt.text(4,0.55,'Boosts',color='blue',fontsize = 17,style = 'italic')
plt.xlabel('Number of heal/boost items',fontsize = 15,color='blue')
plt.ylabel('Win Percentage',fontsize = 15,color='blue')
plt.title('Heals vs Boosts',fontsize = 20,color='blue')
plt.grid()
plt.show()
```

```
/usr/local/lib/python3.5/site-packages/scipy/stats/stats.py:1713: Futur
eWarning: Using a non-tuple sequence for multidimensional indexing is d
eprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future t
his will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

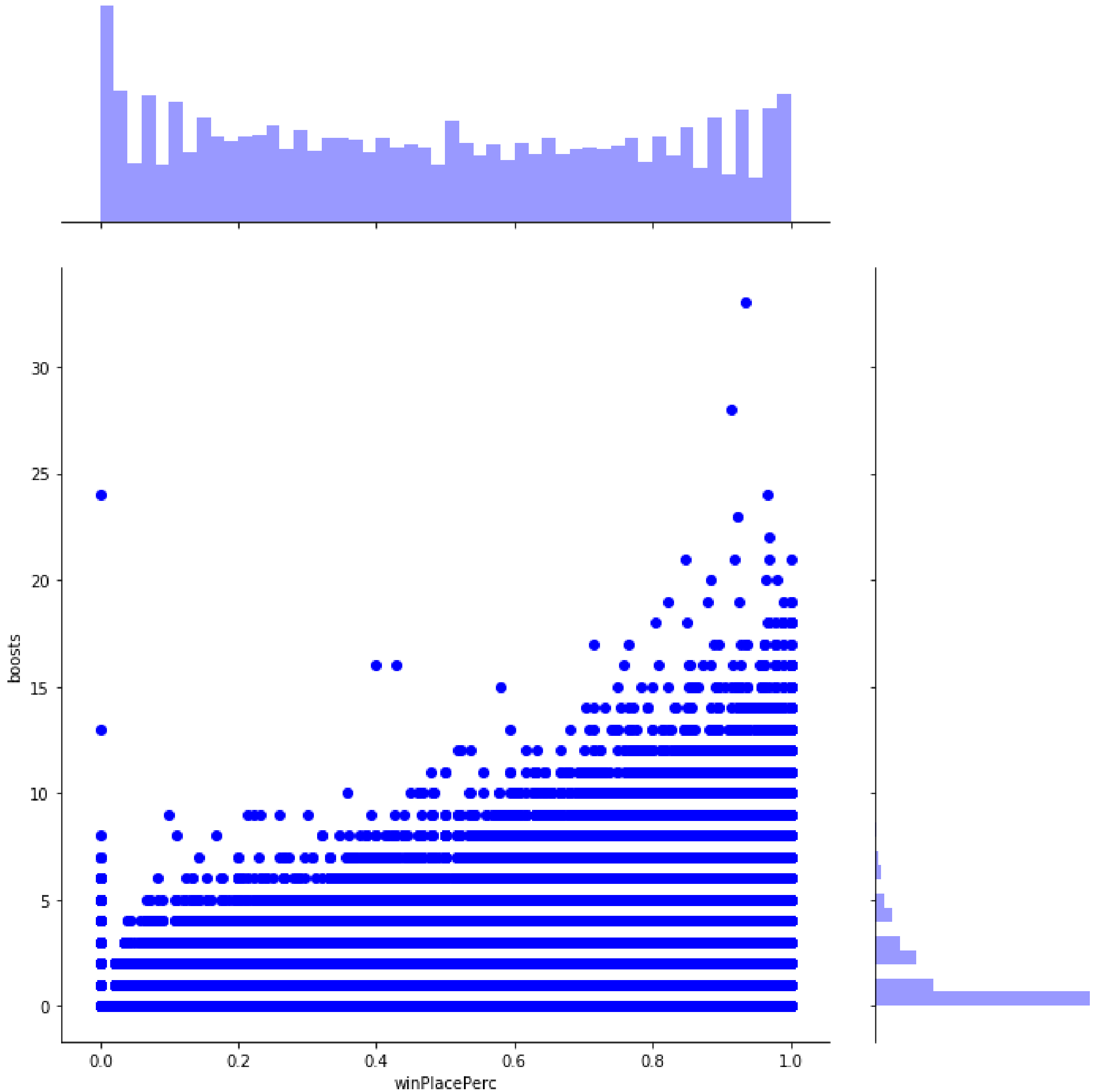


Here, we can see that as number of heal and boost items increases, there is increase in win percentage, but at a x=5, the heal item used decreases the winning percentage while boost does not.

In [8]:
```
sns.jointplot(x="winPlacePerc", y="heals", data=train, height=10, ratio
=3, color="lime")
plt.show()
```

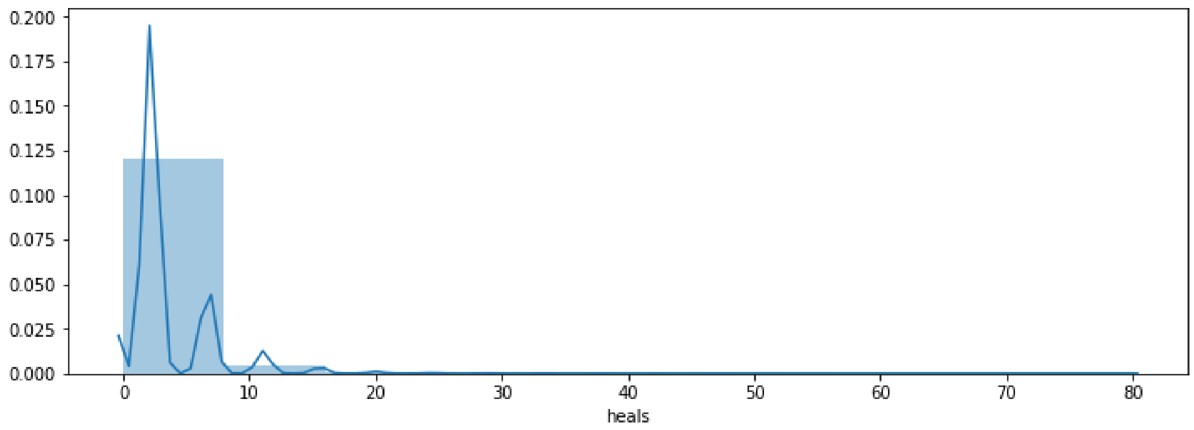


In [9]:
```
sns.jointplot(x="winPlacePerc", y="boosts", data=train, height=10, ratio=3, color="blue")
plt.show()
```



In [41]:
```
# Distribution of heals
plt.figure(figsize=(12,4))
```

```python
sns.distplot(train['heals'], bins=10)
plt.show()
```



In [42]:

```python
# 40 or more healing items used
display(train[train['heals'] >= 40].shape)
train[train['heals'] >= 40].head(10)
```

```
(135, 31)
```

Out[42]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
18405	63ab976895d860	927eeba5614c4f	69473402649f11	0	2	0.0	(
54463	069ddee7c9d26a	58ab5a1ce8e06f	942416b6caf21e	1	4	182.0	(
126439	c45bd6917146e2	81ab9f863957cb	4335664c6716fa	0	2	0.0	(
259351	86910c38335c2f	2738398928d28c	7d2911e944bfaa	0	10	0.0	(
268747	a007734fbc6ebf	5bf702dfa1e5d4	ad6b5669d33a2c	0	5	0.0	(
269098	a0891dbc2950ea	dde848d90491ba	b4fd3348551b73	0	2	0.0	(
284195	91a2fb00455eb3	f639b09774c5b1	65b73c71653822	0	3	123.0	(
300204	1f4f2efc86bfcb	3d668492d1fca9	d3638466a43d38	0	6	175.0	2
349908	7725ad71ad2ff7	4b2a7cf86d1546	cfa2775c9ef944	3	0	2348.0	(

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs
375156	d64866c78ebcb0	aa0f089ae6430c	4dbc4ebba33ec6	0	7	278.5	3

10 rows × 31 columns

In [43]:
```
# Remove outliers
train.drop(train[train['heals'] >= 40].index, inplace=True)
```

In [44]:
```
# Remaining players in the training set
train.shape
```

Out[44]:
```
(4444776, 31)
```

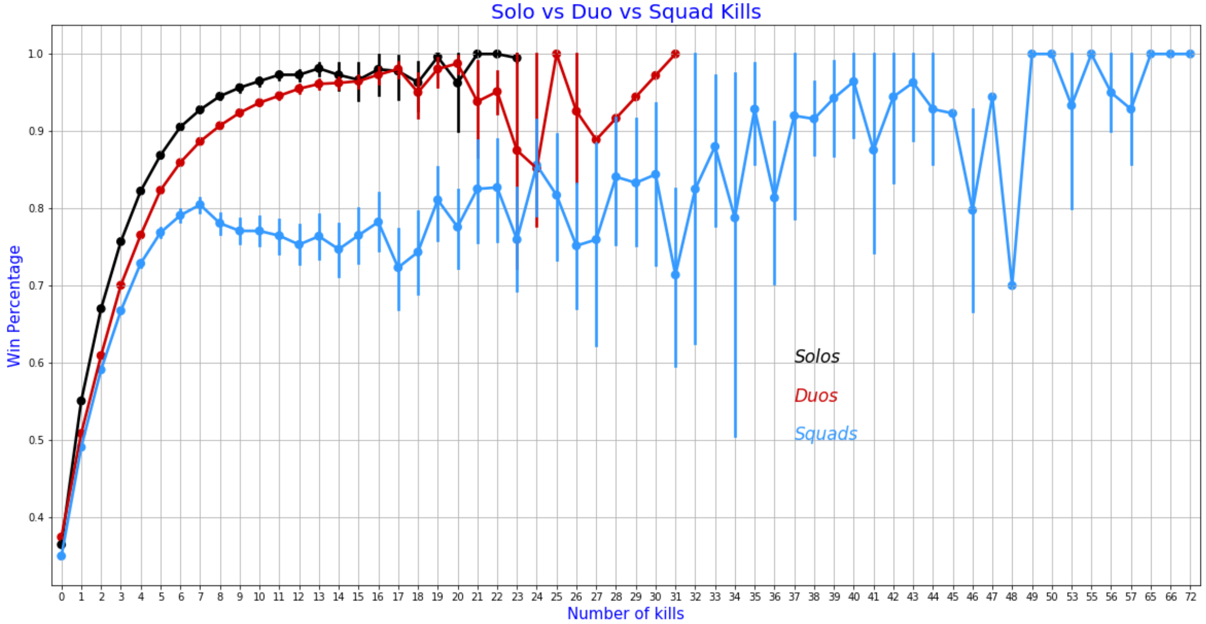
Solos, Duos, Squads

In [10]:
```
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)/len(t
rain),))
```

```
There are 709111 (15.95%) solo games, 3295326 (74.10%) duo games and 44
2528 (9.95%) squad games.
```

In [11]:
```
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',alph
a=0.8)
sns.pointplot(x='kills',y='winPlacePerc',data=squads,color='#3399FF',al
pha=0.8)
plt.text(37,0.6,'Solos',color='black',fontsize = 17,style = 'italic')
```

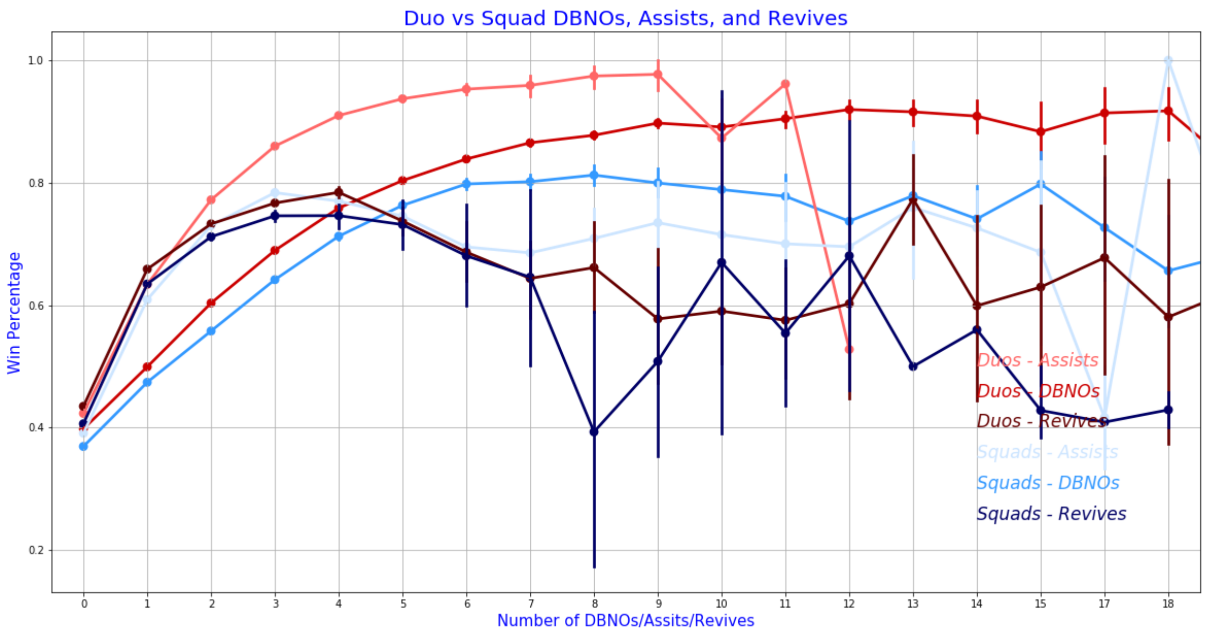
```
plt.text(37,0.55,'Duos',color='#CC0000',fontsize = 17,style = 'italic')
plt.text(37,0.5,'Squads',color='#3399FF',fontsize = 17,style = 'italic'
)
plt.xlabel('Number of kills',fontsize = 15,color='blue')
plt.ylabel('Win Percentage',fontsize = 15,color='blue')
plt.title('Solo vs Duo vs Squad Kills',fontsize = 20,color='blue')
plt.grid()
plt.show()
```



After number of kills=7, the win percentage of squads is not depending on no. of kills, whereas as no of kills increases, win percentage increases.

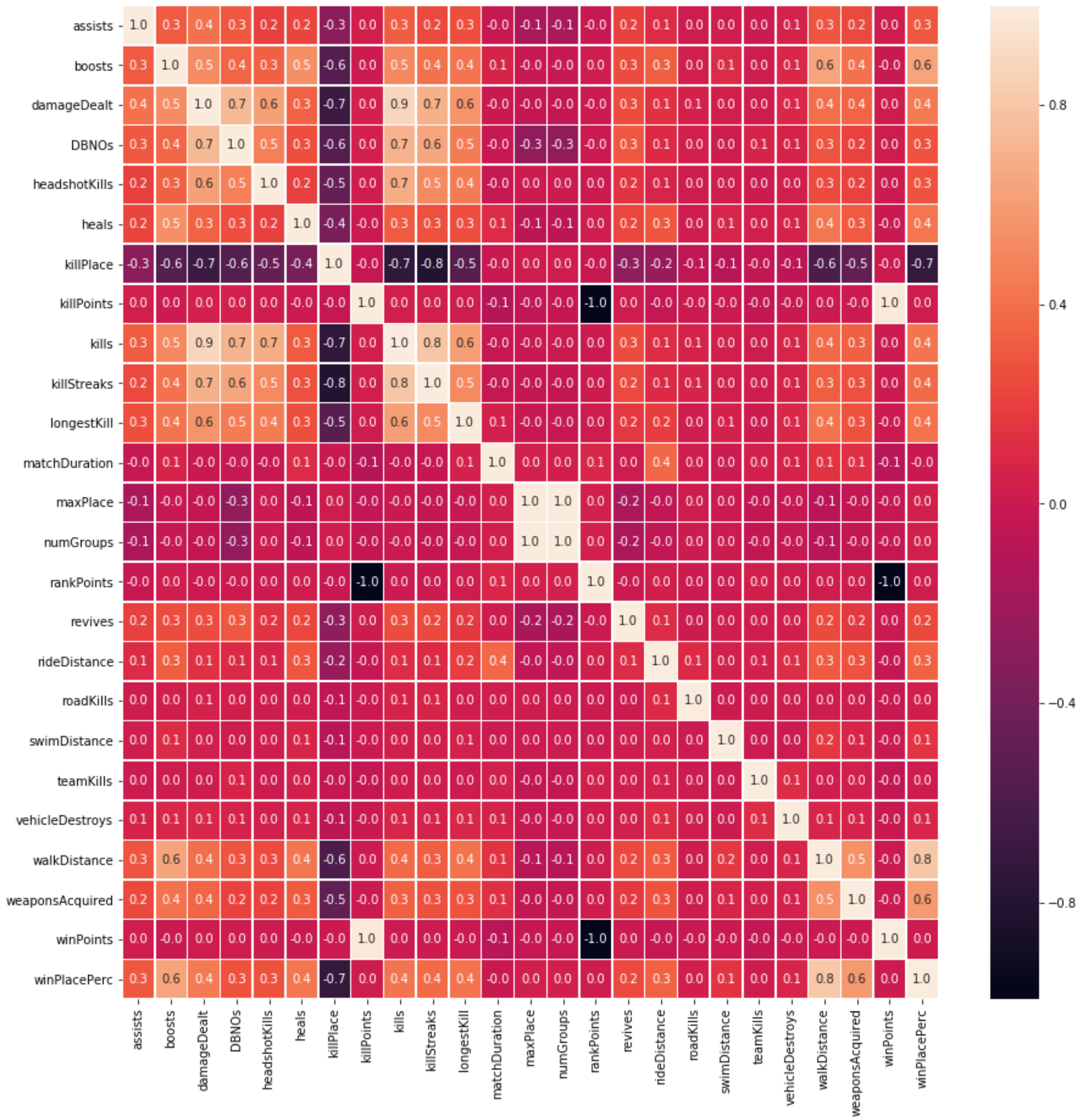
Now, we will be plotting features which are available only for duos and squads: DBNOs,revives and assists.

In [14]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='DBNOs',y='winPlacePerc',data=duos,color='#CC0000',alph
a=0.8)
sns.pointplot(x='DBNOs',y='winPlacePerc',data=squads,color='#3399FF',al
pha=0.8)
sns.pointplot(x='assists',y='winPlacePerc',data=duos,color='#FF6666',al
pha=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',al
pha=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,colo
r='blue')
plt.grid()
plt.show()
```



Pearson correlation between features.

In [16]:
```
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(train.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.show()
```


assists
boosts
damageDealt
DBNOs
headshotKills
heals
killPlace
killPoints
kills
killStreaks
longestKill
matchDuration
maxPlace
numGroups
rankPoints
revives
rideDistance
roadKills
swimDistance
teamKills
vehicleDestroys
walkDistance
weaponsAcquired
winPoints
winPlacePerc
0.8
0.4
0.0
−0.4
−0.8

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.

In [46]:

```
# One hot encode matchType
train = pd.get_dummies(train, columns=['matchType'])

# Take a look at the encoding
matchType_encoding = train.filter(regex='matchType')
matchType_encoding.head()
```

Out[46]:

	matchType_crashfpp	matchType_crashtpp	matchType_duo	matchType_duo-fpp	matchType_flarefp
0	0	0	0	0	
1	0	0	0	0	
2	0	0	1	0	
3	0	0	0	0	
4	0	0	0	0	

In [47]:

```
# Turn groupId and match Id into categorical types
train['groupId'] = train['groupId'].astype('category')
train['matchId'] = train['matchId'].astype('category')

# Get category coding for groupId and matchID
train['groupId_cat'] = train['groupId'].cat.codes
train['matchId_cat'] = train['matchId'].cat.codes

# Get rid of old columns
train.drop(columns=['groupId', 'matchId'], inplace=True)

# Lets take a look at our newly created features
train[['groupId_cat', 'matchId_cat']].head()
```

Out[47]:

	groupId_cat	matchId_cat
0	613591	30085
1	827582	32751
2	843273	3143
3	1340072	45260
4	1757338	20531

In [48]:

```
# Drop Id column, because it probably won't be useful for our Machine L
earning algorithm,
# because the test set contains different Id's
train.drop(columns = ['Id'], inplace=True)
```

In [50]:

```
sample = 500000
df_sample = train.sample(sample)
# Split sample into training data and target variable
df = df_sample.drop(columns = ['winPlacePerc']) #all columns except tar
get
y = df_sample['winPlacePerc'] # Only target variable

x_train, x_test, y_train, y_test=train_test_split(df,y,test_size=0.3, r
andom_state=0)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(350000, 44)
(150000, 44)
(350000,)
(150000,)
```

In [51]:

```
# Function for splitting training and validation data
def split_vals(a, n : int):
    return a[:n].copy(), a[n:].copy()
val_perc = 0.20 # % to use for validation set
```

```
n_valid = int(val_perc * 350000)
n_trn = len(x_train)-n_valid
# Split data
raw_train, raw_valid = split_vals(df_sample, n_trn)
X_train, X_valid = split_vals(x_train, n_trn)
y_train, y_valid = split_vals(y_train, n_trn)

# Check dimensions of samples
print('Sample train shape: ', X_train.shape,
      'Sample target shape: ', y_train.shape,
      'Sample validation shape: ', X_valid.shape)
```

```
Sample train shape:  (280000, 44) Sample target shape:  (280000,) Sampl
e validation shape:  (70000, 44)
```

In [52]:

```
base_learners = [300]
depth = [50]
param_grid = {'n_estimators': base_learners,'max_depth': depth}
RFC = RandomForestRegressor(max_features='sqrt')
model = GridSearchCV(RFC, param_grid, cv=3 , n_jobs = -1,pre_dispatch=2
)
model.fit(X_train, y_train)
print("Model with best parameters :\n",model.best_estimator_)
# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)

optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)
```

```
Model with best parameters :
 RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=50,
           max_features='sqrt', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=300, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=Fa
lse)
```

```
The optimal number of base learners is :  300
The optimal number of depth is :  50
```

In [57]:

```
y_pred=model.predict(x_test)
print("Misclassified samples: %d"%(y_test!=y_pred).sum())
mse = mean_squared_error(y_test,y_pred)
print("RMSE :", np.sqrt(mse))
mae=mean_absolute_error(y_test,y_pred)
print("MAE :", mae)
```

```
Misclassified samples: 149983
RMSE : 0.08935445877043544
MAE : 0.06389176893623487
```

In [ ]:

```
# Metric used for the PUBG competition (Mean Absolute Error (MAE))
# Function to print the MAE (Mean Absolute Error) score
# This is the metric used by Kaggle in this competition
def print_score(m : RandomForestRegressor):
    res = ['mae train: ', mean_absolute_error(m.predict(X_train), y_tra
in),
           'mae val: ', mean_absolute_error(m.predict(X_valid), y_valid
)]
    if hasattr(m, 'oob_score_'): res.append(m.oob_score_)
    print(res)
```