In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

In [3]:
```
df = pd.read_csv('train_V2.csv')
```

In [4]:
```
test = pd.read_csv('test_V2.csv')
```

In [5]:
```
df.head()
```

Out[5]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	he
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 [6]:
```
df.shape
```

Out[6]:
```
(4446966, 29)
```

check the NA in dataframe

In [7]:
```
df.isna().sum()
```

Out[7]:
```
Id                  0
groupId             0
matchId             0
assists             0
boosts              0
damageDealt         0
DBNOs               0
headshotKills       0
heals               0
killPlace           0
killPoints          0
kills               0
killStreaks         0
longestKill         0
matchDuration       0
matchType           0
maxPlace            0
numGroups           0
rankPoints          0
revives             0
rideDistance        0
roadKills           0
swimDistance        0
teamKills           0
vehicleDestroys     0
walkDistance        0
weaponsAcquired     0
winPoints           0
winPlacePerc        1
dtype: int64
```

In [8]:
```
df.winPlacePerc[df['winPlacePerc'].isna()]
```

Out[8]:
```
2744604   NaN
Name: winPlacePerc, dtype: float64
```

In [9]:
```
df.dropna(inplace = True)
```

In [10]:
```
df.isna().sum()
```

Out[10]:
```
Id                  0
groupId             0
matchId             0
assists             0
boosts              0
damageDealt         0
DBNOs               0
headshotKills       0
heals               0
killPlace           0
killPoints          0
kills               0
killStreaks         0
longestKill         0
matchDuration       0
matchType           0
maxPlace            0
numGroups           0
rankPoints          0
revives             0
rideDistance        0
roadKills           0
swimDistance        0
teamKills           0
vehicleDestroys     0
walkDistance        0
weaponsAcquired     0
winPoints           0
winPlacePerc        0
dtype: int64
```

encoding of categorical features

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4446965 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: 1017.8+ MB
```

In [12]:
```
m_types = df.loc[:,"matchType"].value_counts().to_frame().reset_index()
m_types.columns = ["Type","Count"]
m_types
```

Out[12]:

	Type	Count
0	squad-fpp	1756186
1	duo-fpp	996691
2	squad	626526
3	solo-fpp	536761
4	duo	313591
5	solo	181943
6	normal-squad-fpp	17174
7	crashfpp	6287
8	normal-duo-fpp	5489
9	flaretpp	2505
10	normal-solo-fpp	1682
11	flarefpp	718
12	normal-squad	516
13	crashtpp	371
14	normal-solo	326
15	normal-duo	199

In [13]:
```
range(df['matchType'].count())
```

Out[13]:
```
range(0, 4446965)
```

since there are only three types of game: solo,dual(2 players each team) and squad(4 players each team). 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 [14]:
```
df['matchType'] = df['matchType'].apply(lambda x: 'solo' if x == 'solo-fpp'or x=='solo' or x=='normal-solo-fpp' or x=='normal-solo'
                                                    else ('duo' if x=='duo'or x=='duo-fpp'or x=='normal-duo-fpp'or x=='normal-duo' else ('squad'if x=='squad'or x=='squad-fpp' or x=='normal-squad-fpp'or x=='normal-squad'else 'custome')))
```

In [15]:
```
df['matchType'].value_counts()
```

Out[15]:
```
squad      2400402
duo        1315970
solo        720712
custome       9881
Name: matchType, dtype: int64
```

In [16]:
```
df1 = pd.get_dummies(df,columns = ['matchType'])
```

In [17]:
```
df1[:5]
```

Out[17]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	he
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 × 32 columns

In [18]:
```
test['matchType'].value_counts()
```

Out[18]:
```
squad-fpp           752137
duo-fpp             441667
squad               275830
solo-fpp            235778
duo                 140935
solo                 77989
normal-squad-fpp      4161
crashfpp              2701
normal-duo-fpp        1676
flaretpp               634
normal-squad           186
crashtpp               178
flarefpp               137
normal-solo-fpp         99
normal-solo             58
normal-duo               8
Name: matchType, dtype: int64
```

In [19]:
```
test['matchType'] = test['matchType'].apply(lambda x: 'solo' if x == 'so
lo-fpp'or x=='solo' or x=='normal-solo-fpp' or x=='normal-solo'
                                            else ('duo' if x=='duo'or x
=='duo-fpp'or x=='normal-duo-fpp'or x=='normal-duo' else ('squad'if x==
'squad'or x=='squad-fpp' or x=='normal-squad-fpp'or x=='normal-squad'els
e 'custome')))
```

In [20]:
```
test['matchType'].value_counts()
```

Out[20]:
```
squad      1032314
duo         584286
solo        313924
custome       3650
Name: matchType, dtype: int64
```

In [21]:
```
df1['Id'].nunique()
```

Out[21]:
```
4446965
```

In [22]:
```
df1.shape
```

Out[22]:
```
(4446965, 32)
```

In [23]:
```
#make sure each row have unique players
```

feature scaling

In [24]:
```
df2 = df1.drop(['Id', 'groupId','matchId'], axis=1)
```

In [25]:
```
df2[:5]
```

Out[25]:

	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	killPoints	kills	killStre
0	0	0	0.00	0	0	0	60	1241	0	
1	0	0	91.47	0	0	0	57	0	0	
2	1	0	68.00	0	0	0	47	0	0	
3	0	0	32.90	0	0	0	75	0	0	
4	0	0	100.00	0	0	0	45	0	1	

5 rows × 29 columns

In [26]:
```
df2.keys()
```

Out[26]:
```
Index(['assists', 'boosts', 'damageDealt', 'DBNOs', 'headshotKills', 'h
eals',
       'killPlace', 'killPoints', 'kills', 'killStreaks', 'longestKil
l',
       'matchDuration', 'maxPlace', 'numGroups', 'rankPoints', 'revive
s',
       'rideDistance', 'roadKills', 'swimDistance', 'teamKills',
       'vehicleDestroys', 'walkDistance', 'weaponsAcquired', 'winPoint
s',
       'winPlacePerc', 'matchType_custome', 'matchType_duo', 'matchType
_solo',
       'matchType_squad'],
      dtype='object')
```

In [27]:
```
df2.describe().astype('int64')
```

Out[27]:

	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	killPoints	444
count	4446965	4446965	4446965	4446965	4446965	4446965	4446965	4446965	
mean	0	1	130	0	0	1	47	505	
std	0	1	170	1	0	2	27	627	
min	0	0	0	0	0	0	1	0	
25%	0	0	0	0	0	0	24	0	
50%	0	0	84	0	0	0	47	0	
75%	0	2	186	1	0	2	71	1172	
max	22	33	6616	53	64	80	101	2170	

8 rows × 29 columns

In [28]:
```
from sklearn.model_selection import train_test_split
```

In [29]:
```
df3 = df2.drop(['winPlacePerc'],axis = 1)
```

In [30]:
```
X_train,X_test,y_train,y_test = train_test_split(df3,df2['winPlacePerc'
],test_size = 0.25,random_state=1)
```

In [31]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [32]:
```
scaler = MinMaxScaler()
```

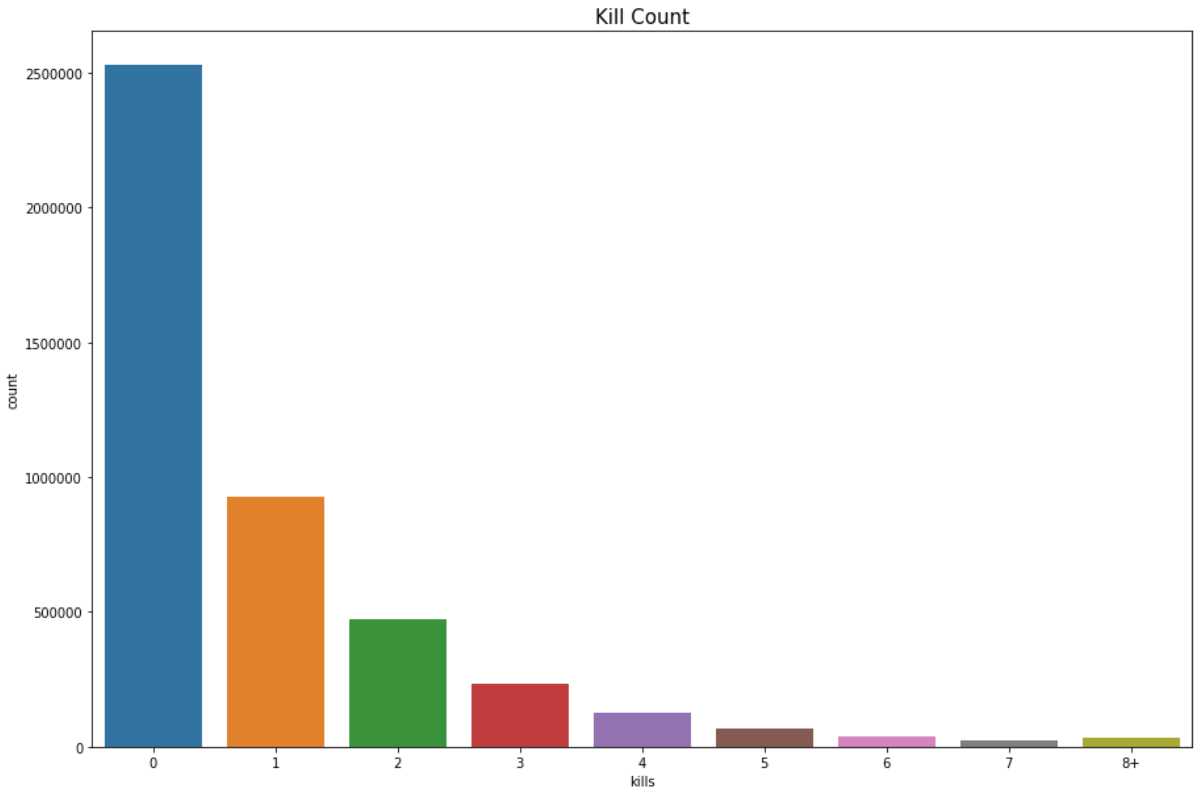
In [33]:
```
scaler.fit(X_train)
```

Out[33]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

data screening

In [35]:
```
#compare the kills and damage and win
```

In [36]:
```
df2.loc[df2['kills'] > df2['kills'].quantile(0.99)] = '8+'
plt.figure(figsize=(15,10))
sns.countplot(df2['kills'].astype('str').sort_values())
plt.title("Kill Count",fontsize=15)
plt.show()
```



In [41]:
```
df.info()
```

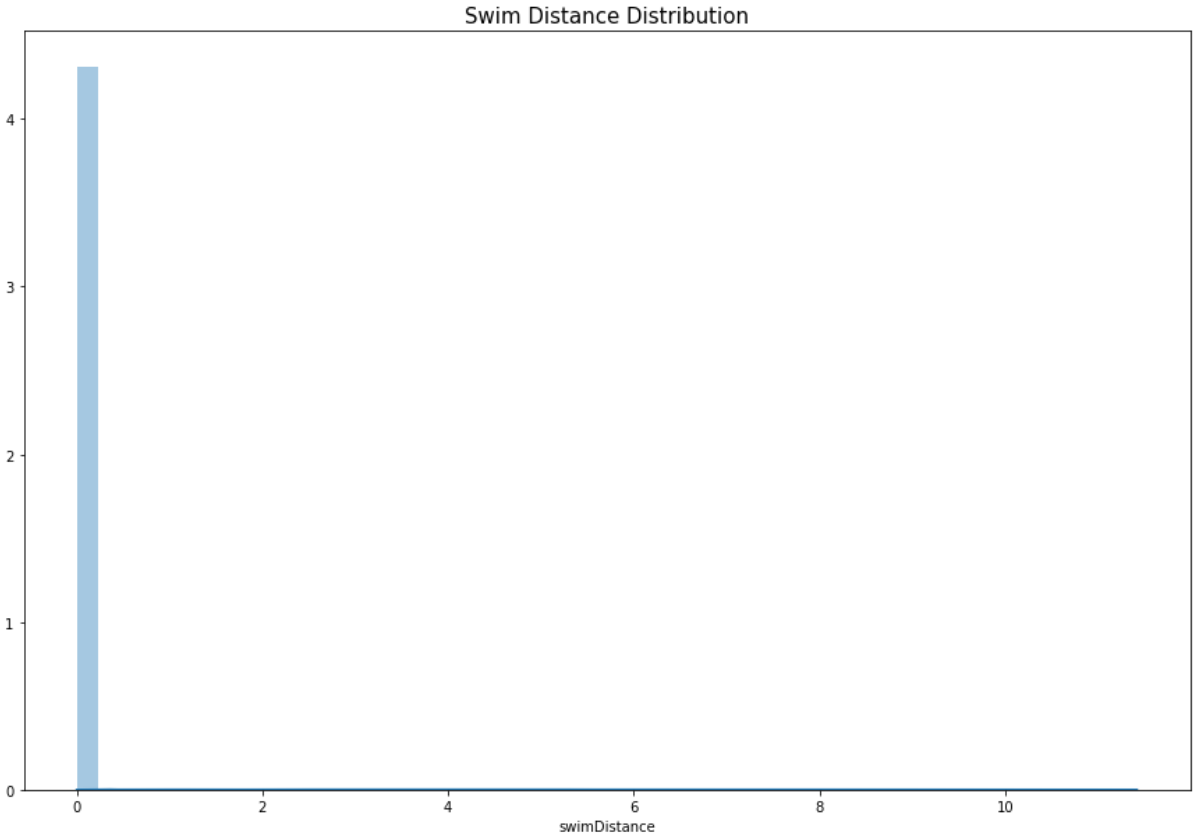
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4446965 entries, 0 to 4446965
Data columns (total 29 columns):
Id                 object
groupId            object
matchId            object
assists            int64
boosts             int64
damageDealt        float64
DBNOs              int64
headshotKills      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: 1017.8+ MB
```

In [45]:
```
# SWIM
```

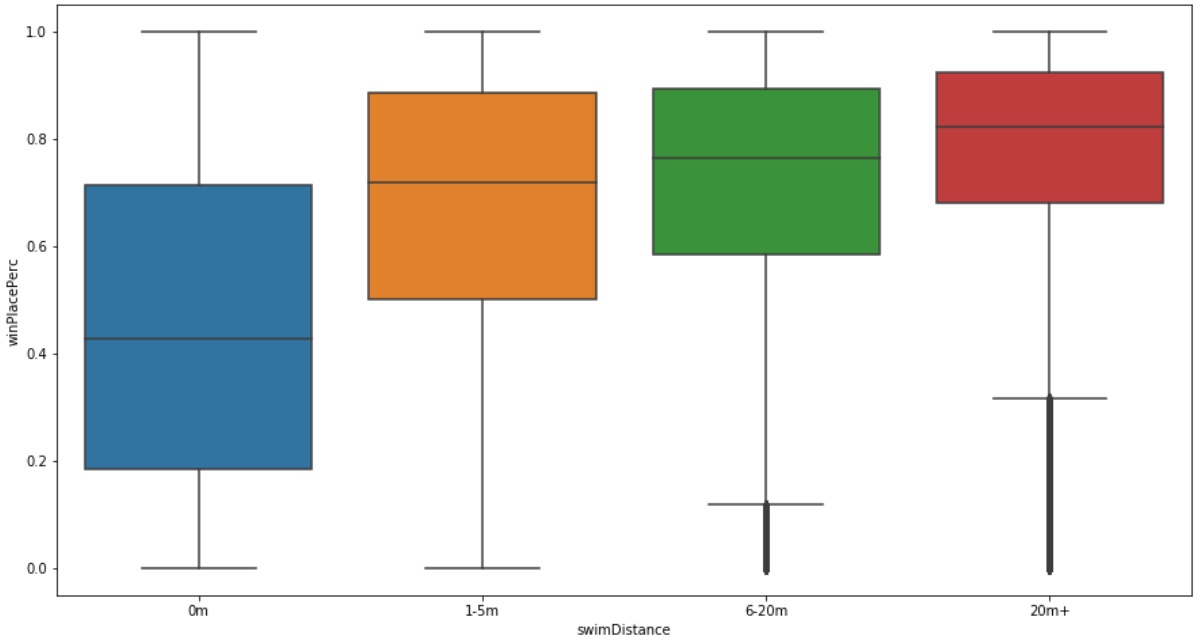
In [42]:
```
print("The average person swims for {:.1f}m, 99% of people have swimemd
{}m or less, \
while the olympic champion swimmed for {}m."
      .format(df['swimDistance'].mean(), df['swimDistance'].quantile(0.9
9), df['swimDistance'].max()))
```

```
The average person swims for 4.5m, 99% of people have swimemd 123.0m or
less, while the olympic champion swimmed for 3823.0m.
```

In [44]:
```
data = df.copy()
data = data[data['swimDistance'] < df['swimDistance'].quantile(0.95)]
plt.figure(figsize=(15,10))
plt.title("Swim Distance Distribution",fontsize=15)
sns.distplot(data['swimDistance'])
plt.show()
```



In [46]:
```
swim = df.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()
```

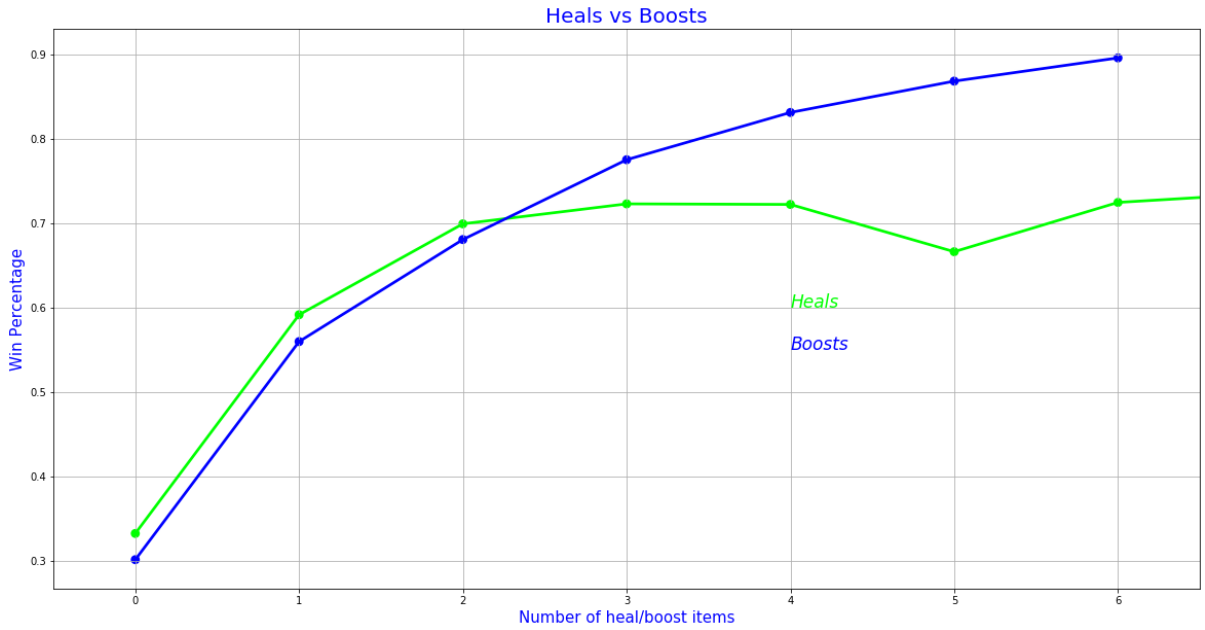


In [74]:
```
# Healers
```

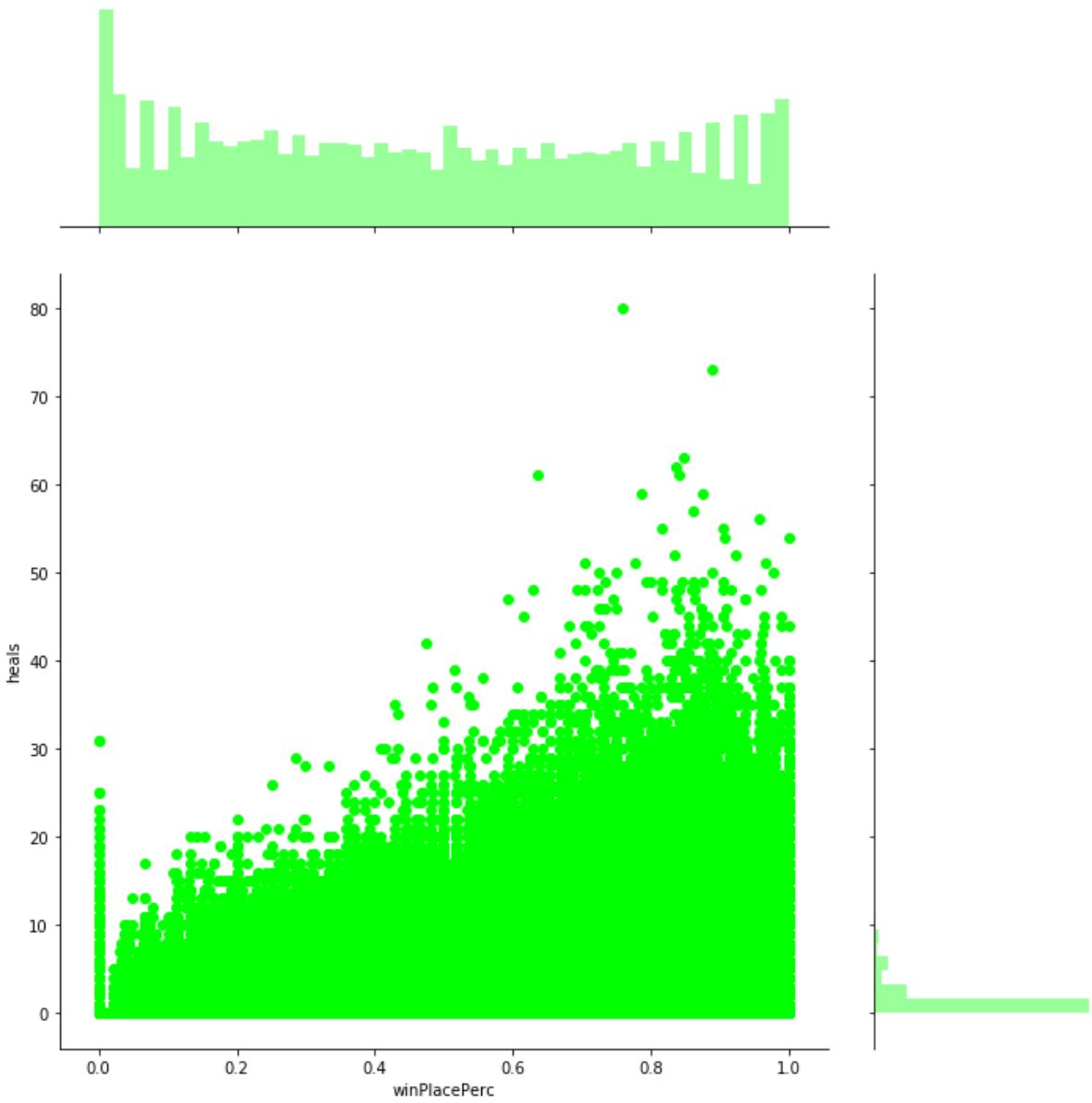
In [75]:
```
print("The average person uses {:.1f} heal items, 99% of people use {} o
r less,\
while the doctor used {}.".format(df['heals'].mean(), df['heals'].quanti
le(0.99), df['heals'].max()))
print("The average person uses {:.1f} boost items, 99% of people use {}
 or less, \
while the doctor used {}.".format(df['boosts'].mean(), df['boosts'].quan
tile(0.99), df['boosts'].max()))
```

```
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.
```

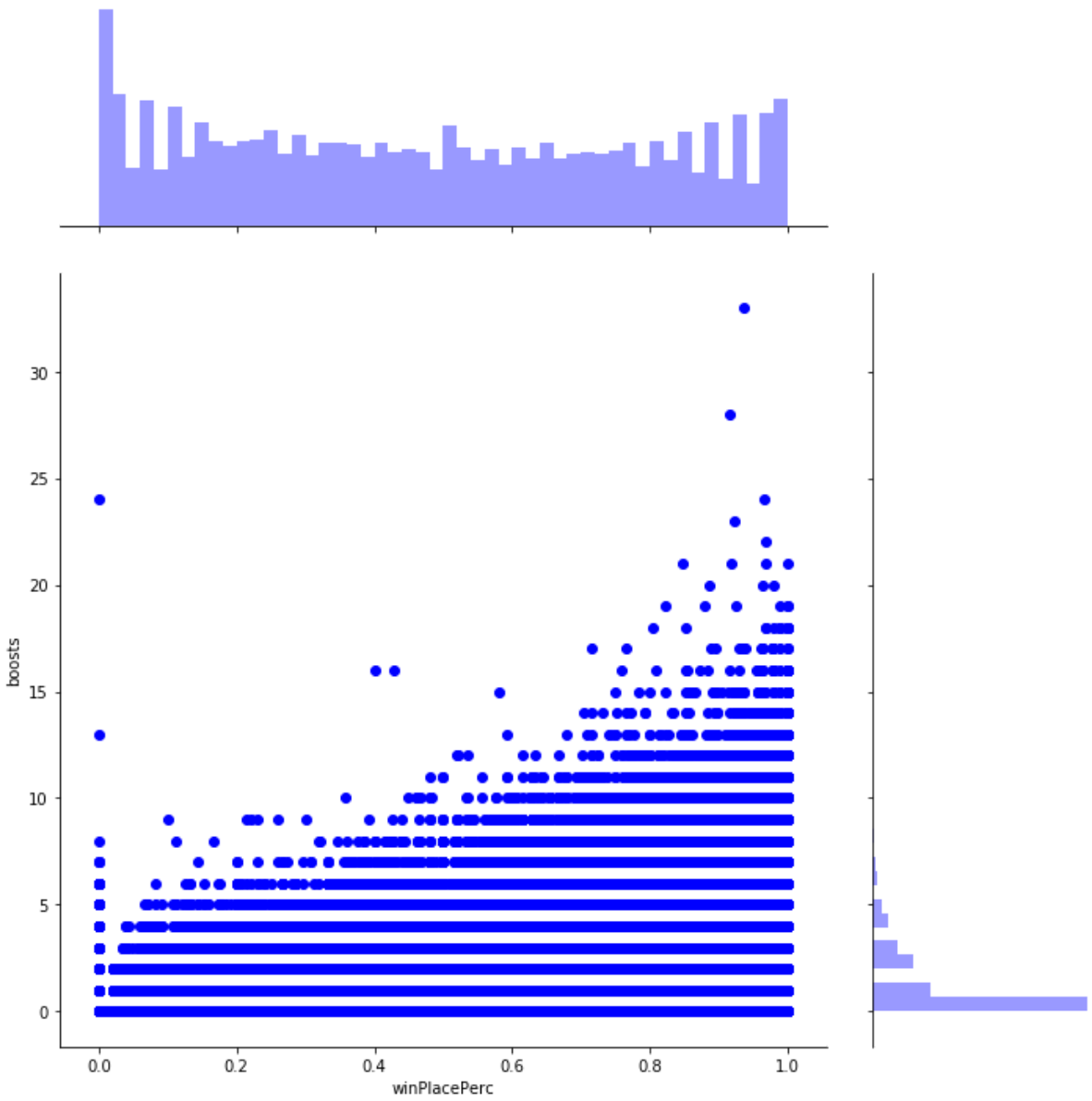
In [76]:
```
data = df.copy()
data = data[data['heals'] < data['heals'].quantile(0.99)]
data = data[data['boosts'] < data['boosts'].quantile(0.99)]

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 [77]:
```
sns.jointplot(x="winPlacePerc", y="heals", data=df, height=10, ratio=3,
color="lime")
plt.show()
```



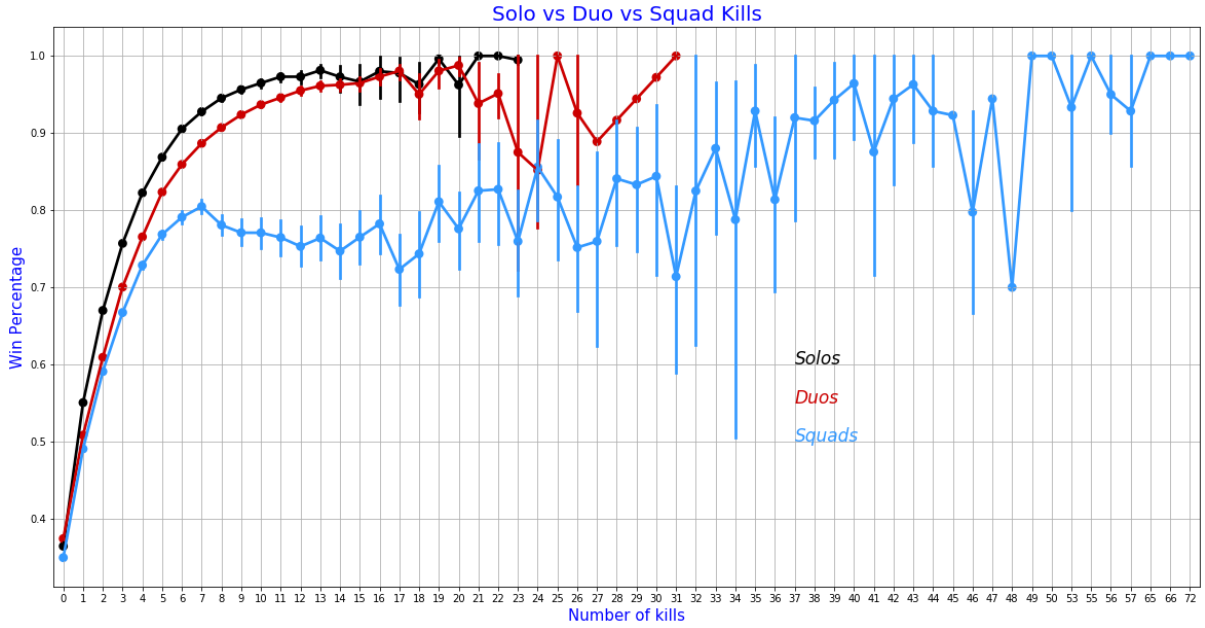
In [78]:
```
sns.jointplot(x="winPlacePerc", y="boosts", data=df, height=10, ratio=3,
color="blue")
plt.show()
```



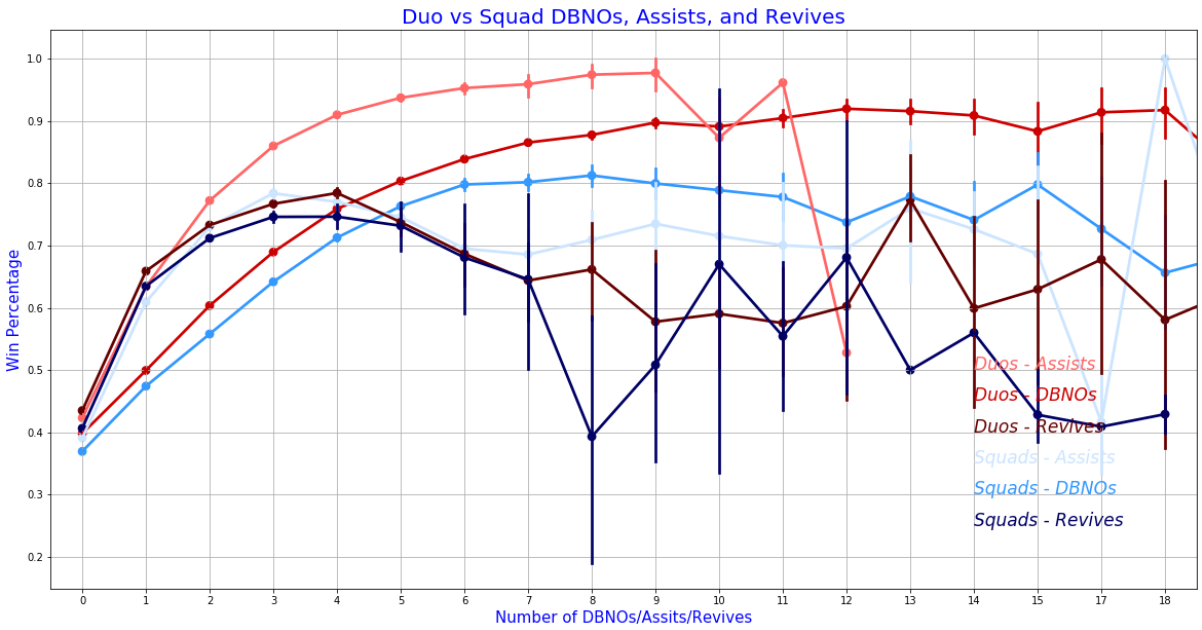
In [81]:
```
solos = df[df['numGroups']>50]
duos = df[(df['numGroups']>25) & (df['numGroups']<=50)]
squads = df[df['numGroups']<=25]
print("There are {} ({:.2f}%) solo games, {} ({:.2f}%) duo games and {}
 ({:.2f}%) squad games."
      .format(len(solos), 100*len(solos)/len(df), len(duos), 100*len(duo
s)/len(df), len(squads),
              100*len(squads)/len(df),))
```

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

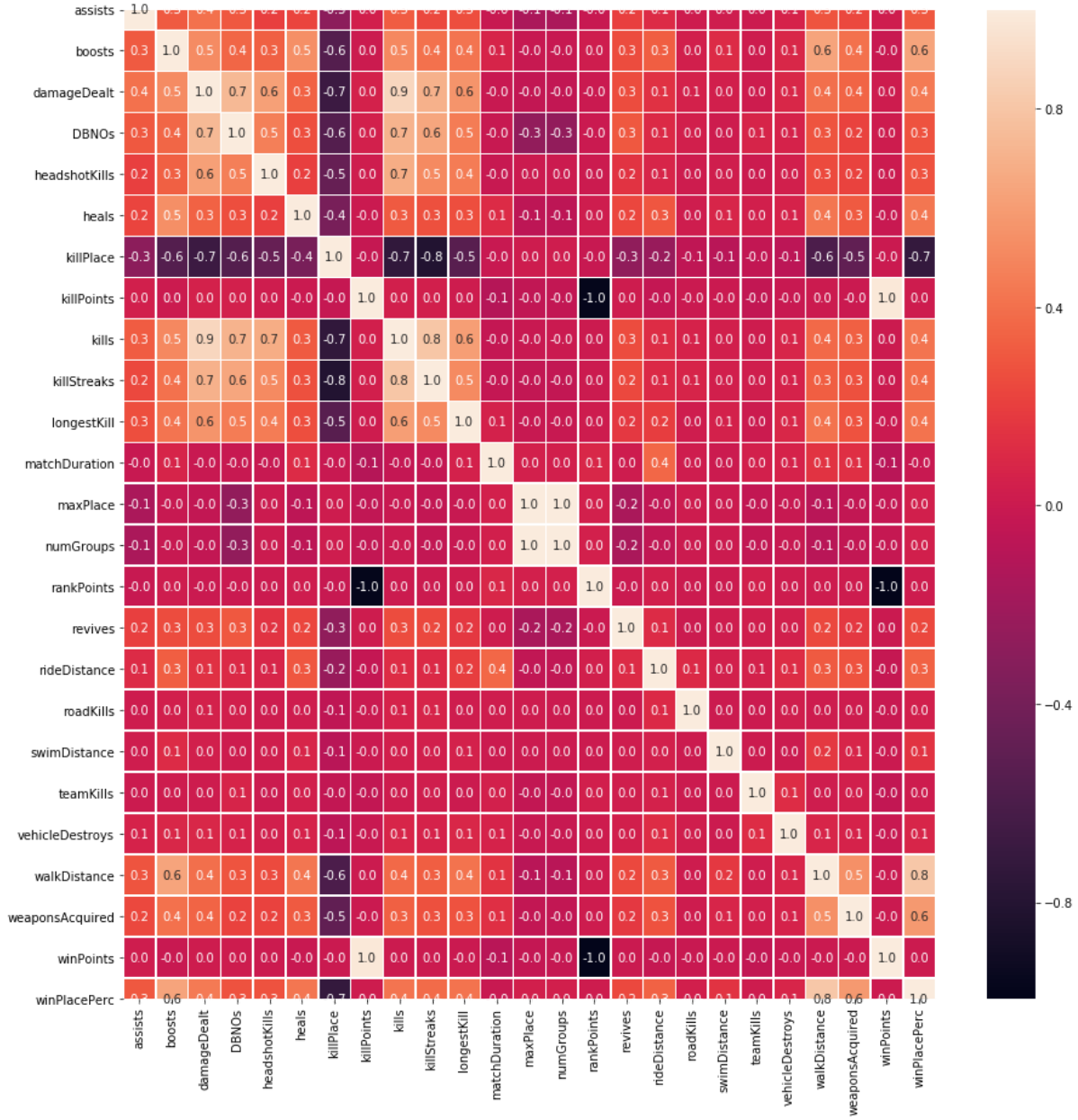
In [82]:
```
f,ax1 = plt.subplots(figsize =(20,10))
sns.pointplot(x='kills',y='winPlacePerc',data=solos,color='black',alpha=
0.8)
sns.pointplot(x='kills',y='winPlacePerc',data=duos,color='#CC0000',alpha
=0.8)
sns.pointplot(x='kills',y='winPlacePerc',data=squads,color='#3399FF',alp
ha=0.8)
plt.text(37,0.6,'Solos',color='black',fontsize = 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()
```



In [83]:
```
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',alp
ha=0.8)
sns.pointplot(x='assists',y='winPlacePerc',data=duos,color='#FF6666',alp
ha=0.8)
sns.pointplot(x='assists',y='winPlacePerc',data=squads,color='#CCE5FF',a
lpha=0.8)
sns.pointplot(x='revives',y='winPlacePerc',data=duos,color='#660000',alp
ha=0.8)
sns.pointplot(x='revives',y='winPlacePerc',data=squads,color='#000066',a
lpha=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 = 'i
talic')
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()
```



In [84]:
```
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(df.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.show()
```



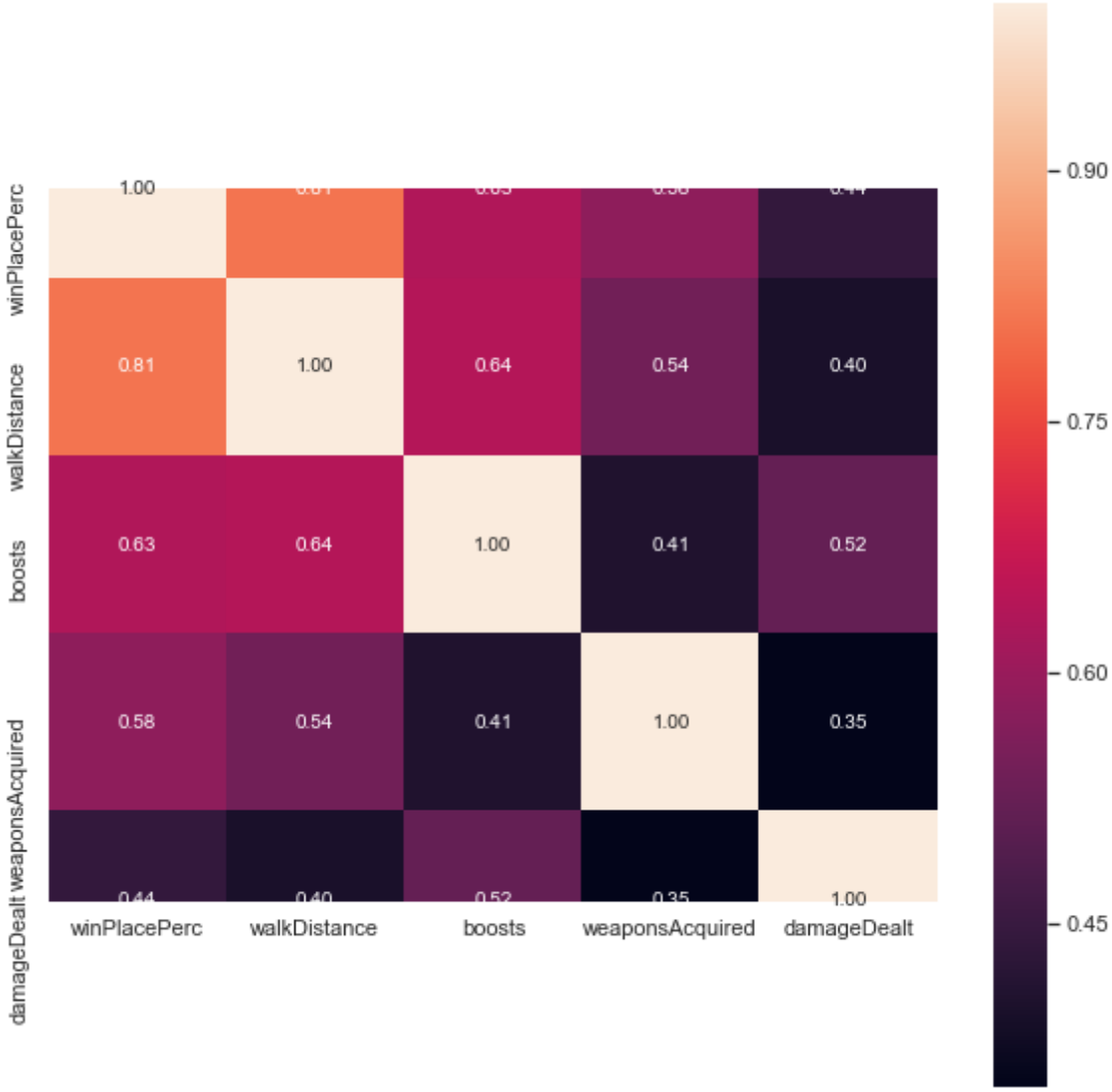
In [87]:
```
df[:5]
```

Out[87]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	he
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	

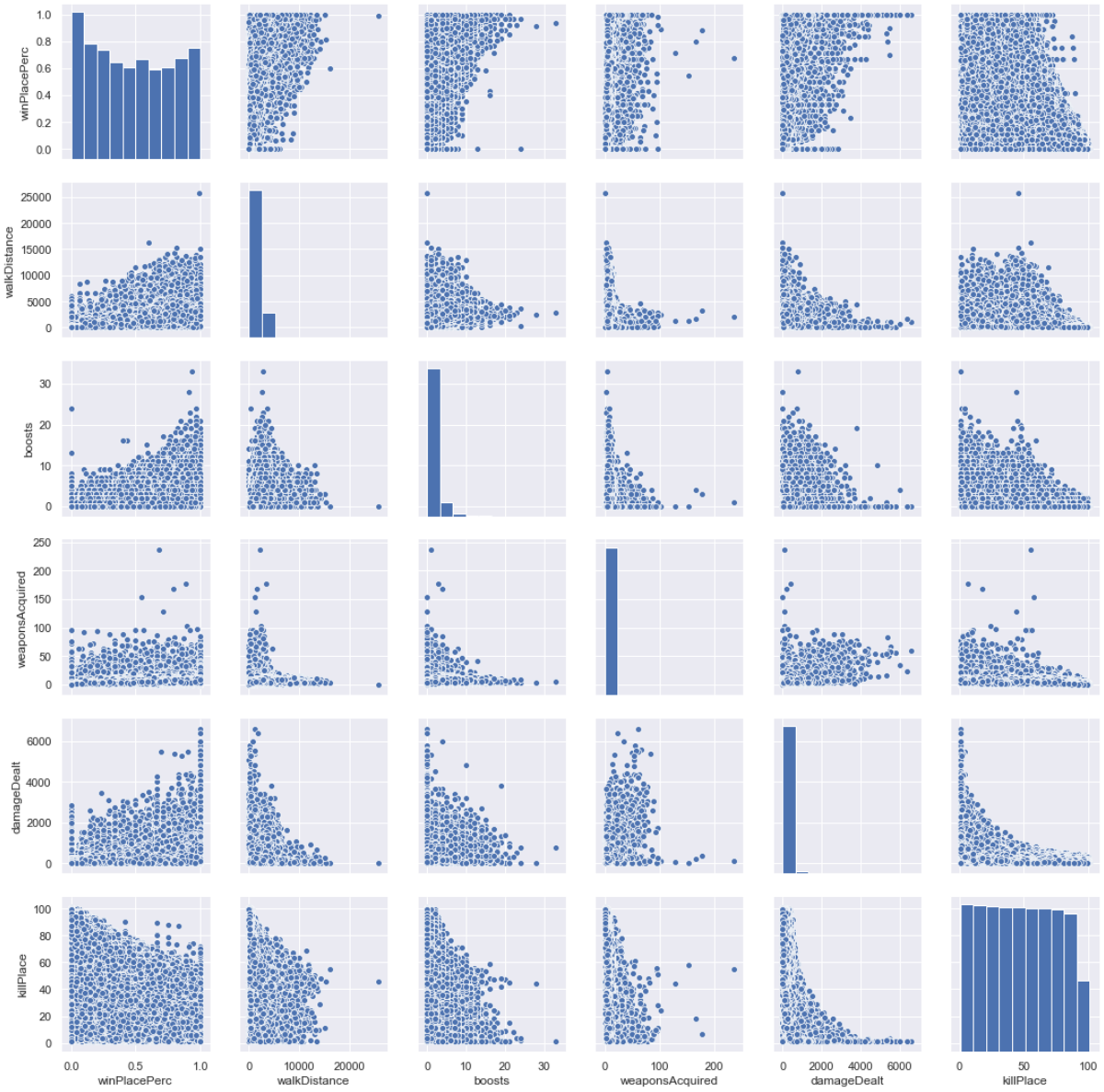
5 rows × 29 columns

In [94]:
```
k = 5 #number of variables for heatmap
f,ax = plt.subplots(figsize=(10, 10))
cols = df.corr().nlargest(k, 'winPlacePerc')['winPlacePerc'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.0)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', anno
t_kws={'size': 10},
                 yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



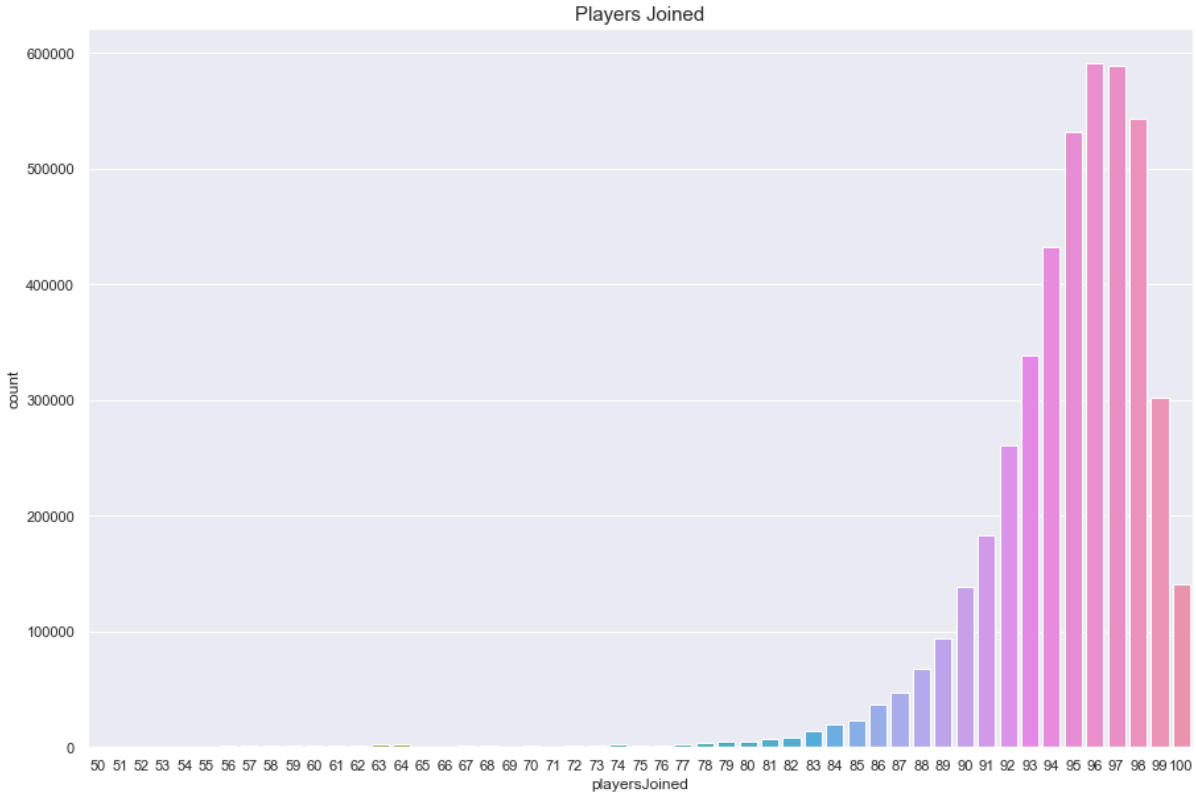
In [95]:
```
sns.set()
cols = ['winPlacePerc', 'walkDistance', 'boosts', 'weaponsAcquired', 'da
mageDealt', 'killPlace']
sns.pairplot(df[cols], size = 2.5)
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2065: Us
erWarning: The `size` parameter has been renamed to `height`; pleaes up
date your code.
  warnings.warn(msg, UserWarning)
```



In [96]:
```
df['playersJoined'] = df.groupby('matchId')['matchId'].transform('count'
)
```

In [97]:
```
data = df.copy()
data = data[data['playersJoined']>49]
plt.figure(figsize=(15,10))
sns.countplot(data['playersJoined'])
plt.title("Players Joined",fontsize=15)
plt.show()
```



In [99]:
```
df['killsNorm'] = df['kills']*((100-df['playersJoined'])/100 + 1)
df['damageDealtNorm'] = df['damageDealt']*((100-df['playersJoined'])/100
+ 1)
df[['playersJoined', 'kills', 'killsNorm', 'damageDealt', 'damageDealtNo
rm']][5:8]
```

Out[99]:

	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

In [100]:
```
df['healsAndBoosts'] = df['heals']+df['boosts']
df['totalDistance'] = df['walkDistance']+df['rideDistance']+df['swimDist
ance']
```

In [101]:
```
df['boostsPerWalkDistance'] = df['boosts']/(df['walkDistance']+1)
#The +1 is to avoid infinity, because there are entries where boosts>0 a
nd walkDistance=0. Strange.
df['boostsPerWalkDistance'].fillna(0, inplace=True)
df['healsPerWalkDistance'] = df['heals']/(df['walkDistance']+1)
#The +1 is to avoid infinity, because there are entries where heals>0 an
d walkDistance=0. Strange.
df['healsPerWalkDistance'].fillna(0, inplace=True)
df['healsAndBoostsPerWalkDistance'] = df['healsAndBoosts']/(df['walkDist
ance']+1)
#The +1 is to avoid infinity.
df['healsAndBoostsPerWalkDistance'].fillna(0, inplace=True)
df[['walkDistance', 'boosts', 'boostsPerWalkDistance' ,'heals',
       'healsPerWalkDistance', 'healsAndBoosts', 'healsAndBoostsPerWalkD
istance']][40:45]
```

Out[101]:

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

In [103]:
```
df['killsPerWalkDistance'] = df['kills']/(df['walkDistance']+1)
#The +1 is to avoid infinity, because there are entries where kills>0 an
d walkDistance=0. Strange.
df['killsPerWalkDistance'].fillna(0, inplace=True)
df[['kills', 'walkDistance', 'rideDistance', 'killsPerWalkDistance', 'wi
nPlacePerc']]\
.sort_values(by='killsPerWalkDistance').tail(10)
```

Out[103]:

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

In [106]:
```
df['team'] = [1 if i>50 else 2 if (i>25 & i<=50) else 4 for i in df['numGroups']]
```

In [107]:
```
df.head()
```

Out[107]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	he
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 × 39 columns

In [38]:
```
#jddjfjfjf
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

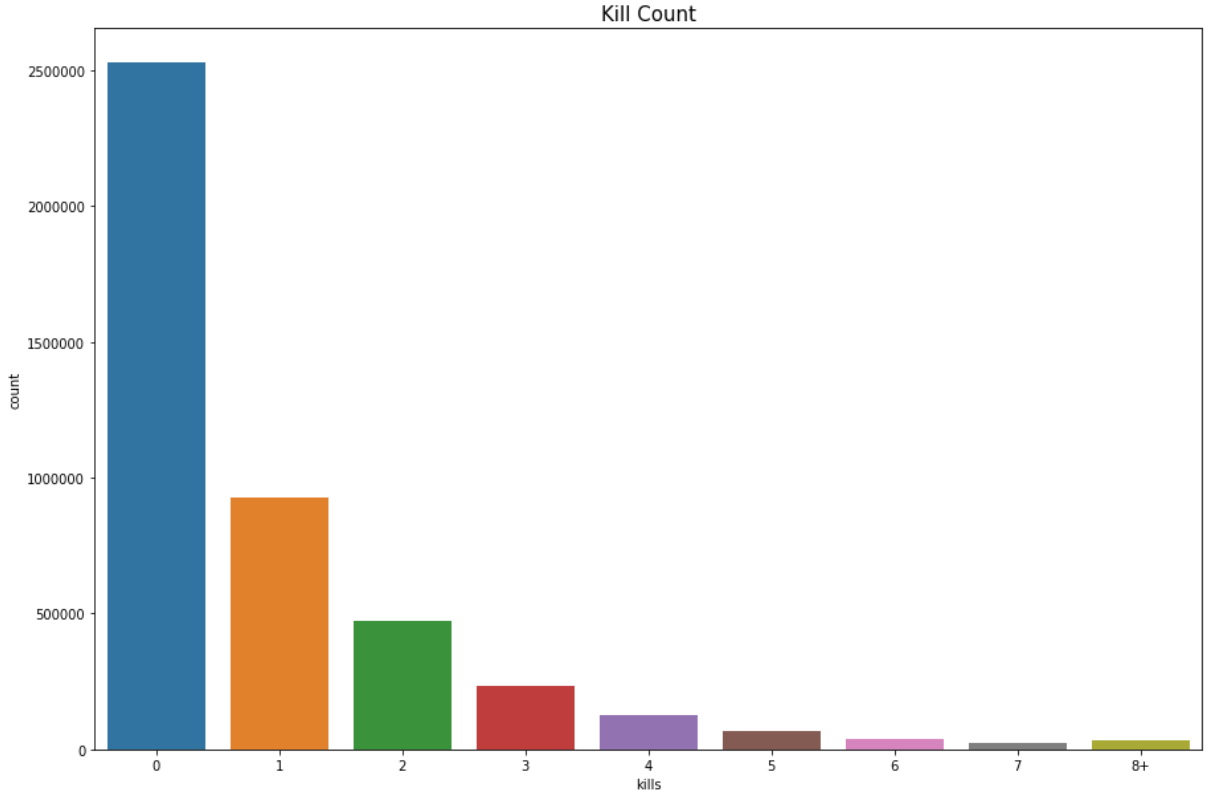
In [ ]:

In [ ]:

data screening

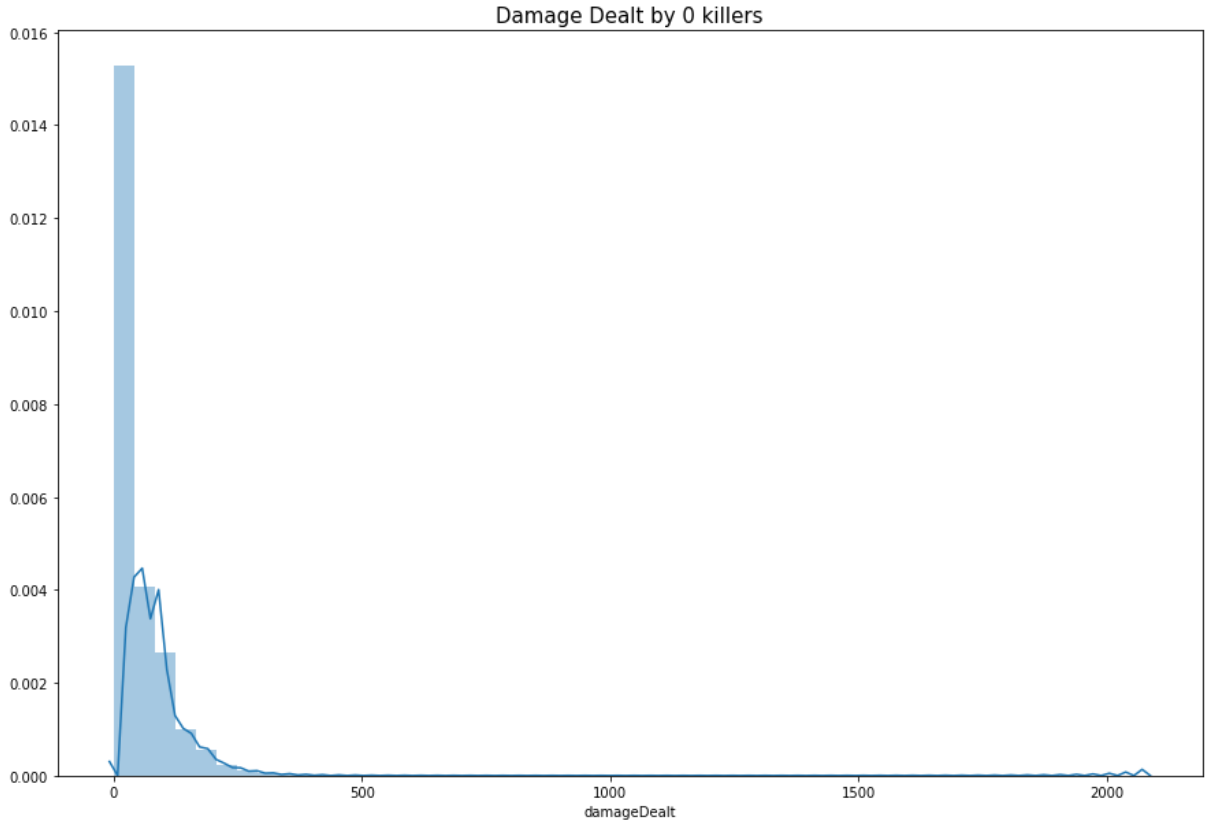
In [39]:
```
#compare the kills and damage and win
```

In [40]:
```
data = df2.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. Let's see tha damage they can make.

In [46]:
```
data = df2.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()
```



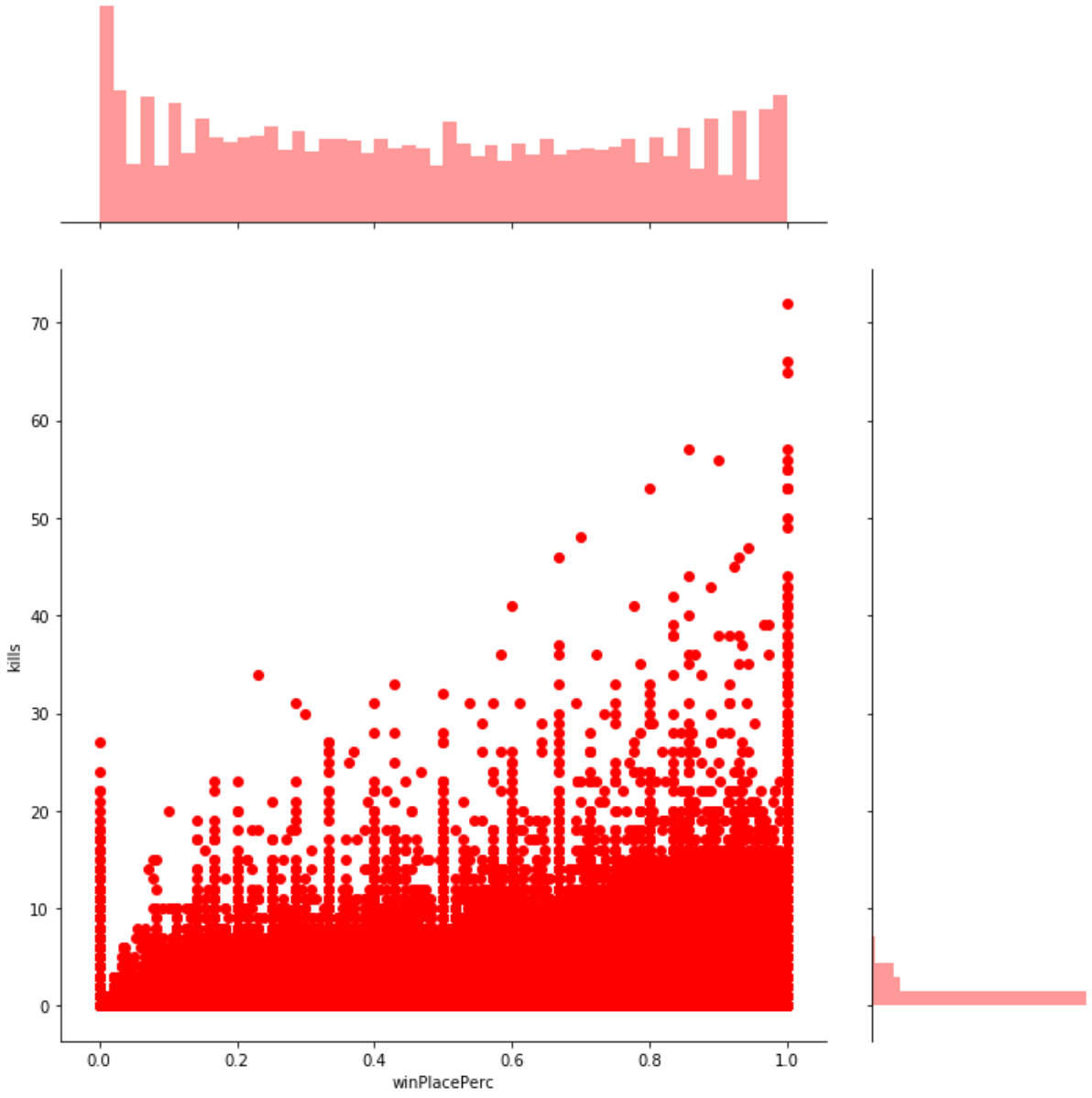
Well, most of them don't. Let's investigate the exceptions.

In [47]:
```
print("{} players ({:.4f}%) have won without a single kill!".format(len(data[data[
'winPlacePerc']==1]), 100*len(data[data['winPlacePerc']==1])/len(df2)))

data1 = df2[df2['damageDealt'] == 0].copy()
print("{} players ({:.4f}%) have won without dealing damage!".format(len(data1[data
1['winPlacePerc']==1]), 100*len(data1[data1['winPlacePerc']==1])/len(df2)))
```

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

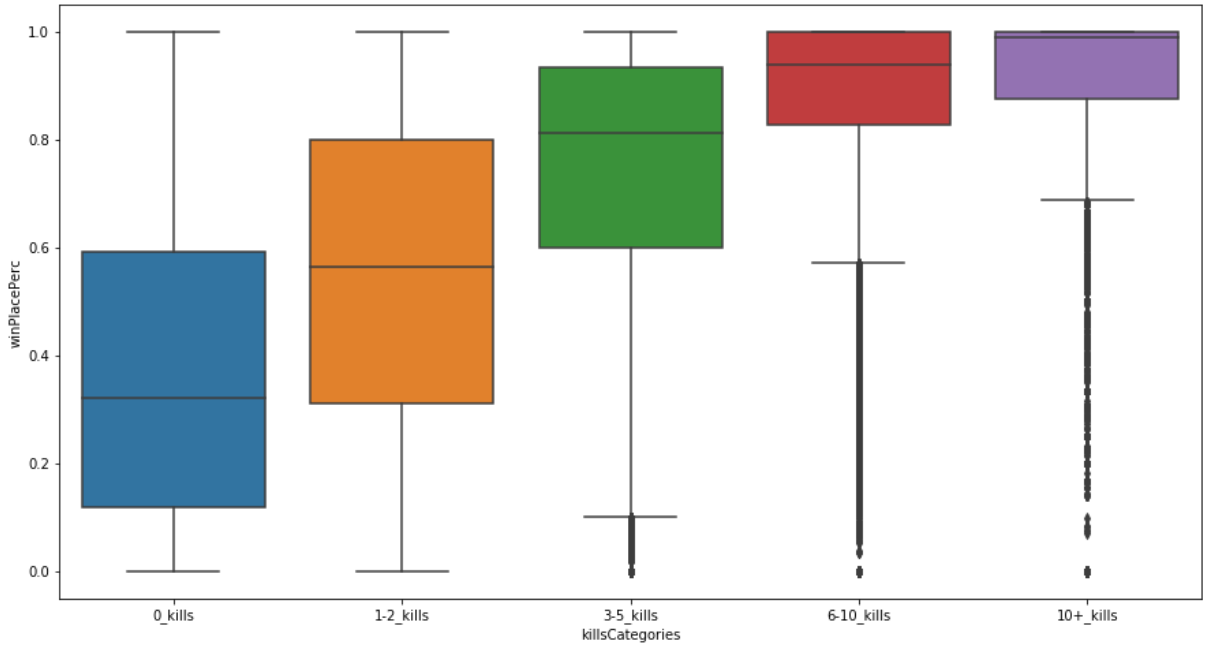
In [66]:
```
# Plot win placement percentage vs kills.
```

In [48]:
```
sns.jointplot(x="winPlacePerc", y="kills", data=df2, height=10, ratio=3, color="r")
plt.show()
```



In [67]:
```
# 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 [49]:
```
kills = df2.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+_kills'])

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

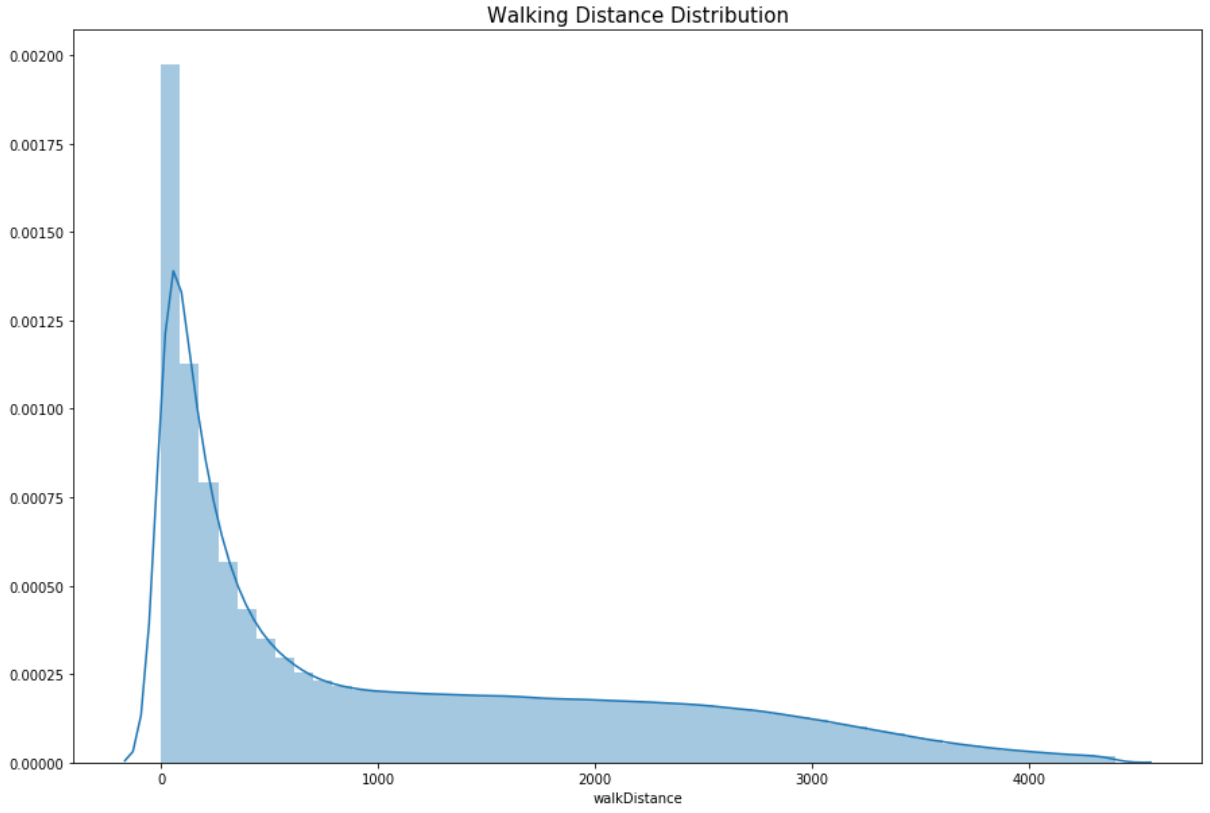


In [68]:
```
# The Runners
```

In [53]:
```
print("The average person walks for {:.1f}m, 99% of people have walked {}m or less, while the marathoner champion walked for {}m.".format(df2['walkDistance'].mean(), \

df2['walkDistance'].quantile(0.99), \

df2['walkDistance'].max()))
```

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

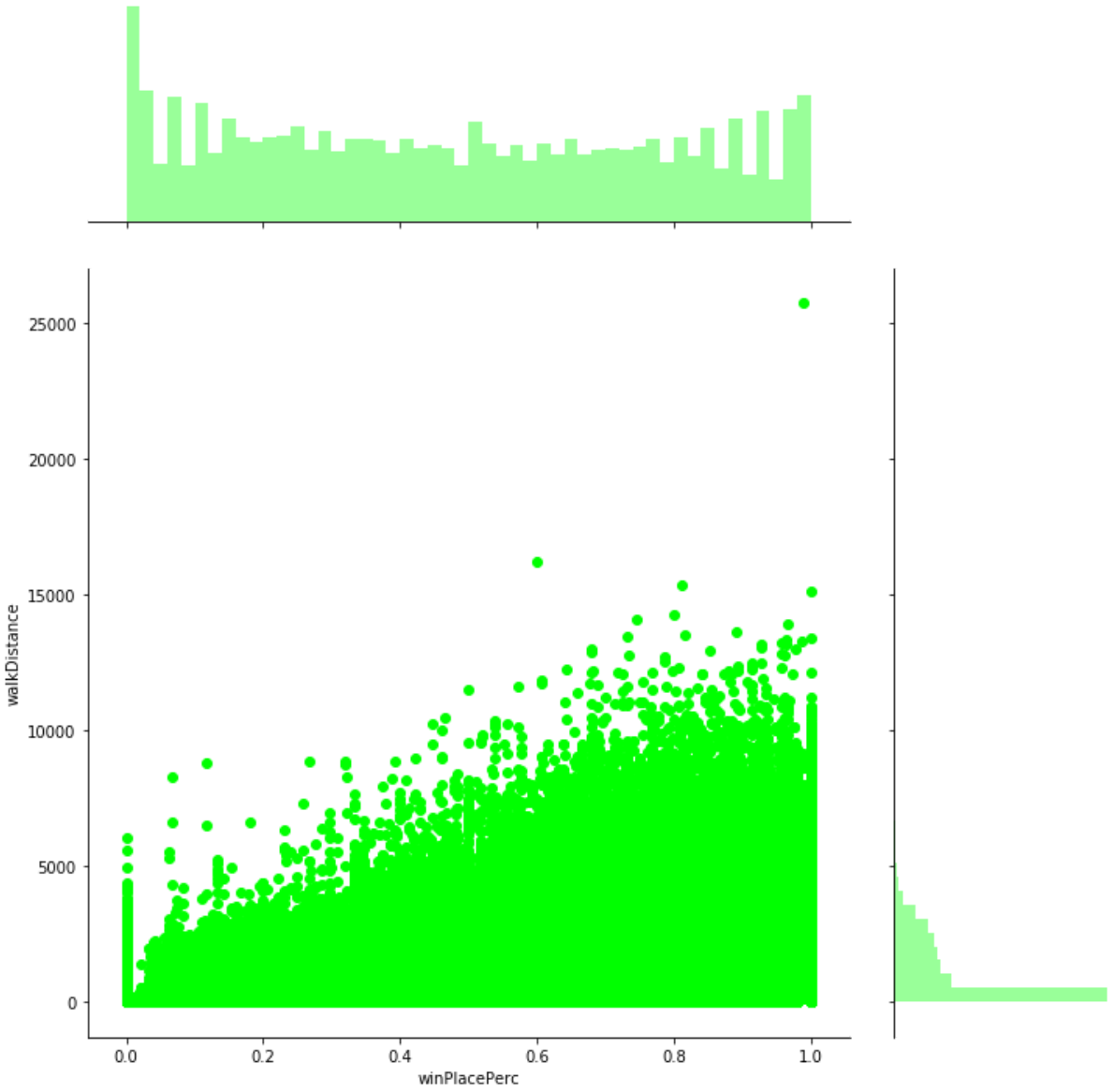
In [55]:
```
data = df2.copy()
data = data[data['walkDistance'] < df2['walkDistance'].quantile(0.99)]
plt.figure(figsize=(15,10))
plt.title("Walking Distance Distribution",fontsize=15)
sns.distplot(data['walkDistance'])
plt.show()
```



In [56]:
```
print("{} players ({:.4f}%) walked 0 meters. This means that they die before even t
aking a step or they are afk (more possible).".format(len(data[data['walkDistance']
== 0]), \

100*len(data1[data1['walkDistance']==0])/len(df2)))
```

```
99602 players (2.0328%) walked 0 meters. This means that they die before even taki
ng a step or they are afk (more possible).
```

In [57]:
```
sns.jointplot(x="winPlacePerc", y="walkDistance",  data=df2, height=10, ratio=3, co
lor="lime")
plt.show()
```



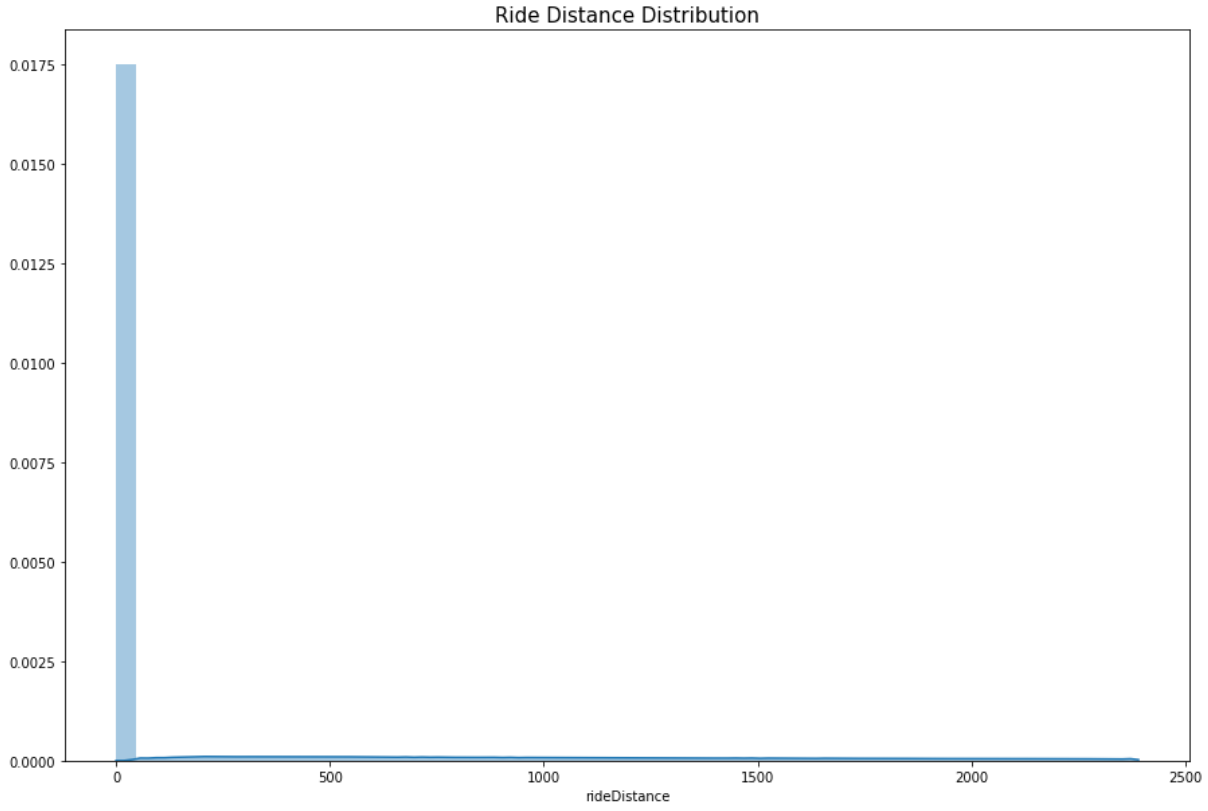
Apparently walking has a high correlation with winPlacePerc.

In [69]:
```
# The Drivers
```

In [58]:
```
print("The average person drives for {:.1f}m, 99% of people have drived {}m or les
s, while the formula 1 champion drived for {}m.".\
      format(df2['rideDistance'].mean(), df2['rideDistance']\
             .quantile(0.99), df2['rideDistance'].max()))
```

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

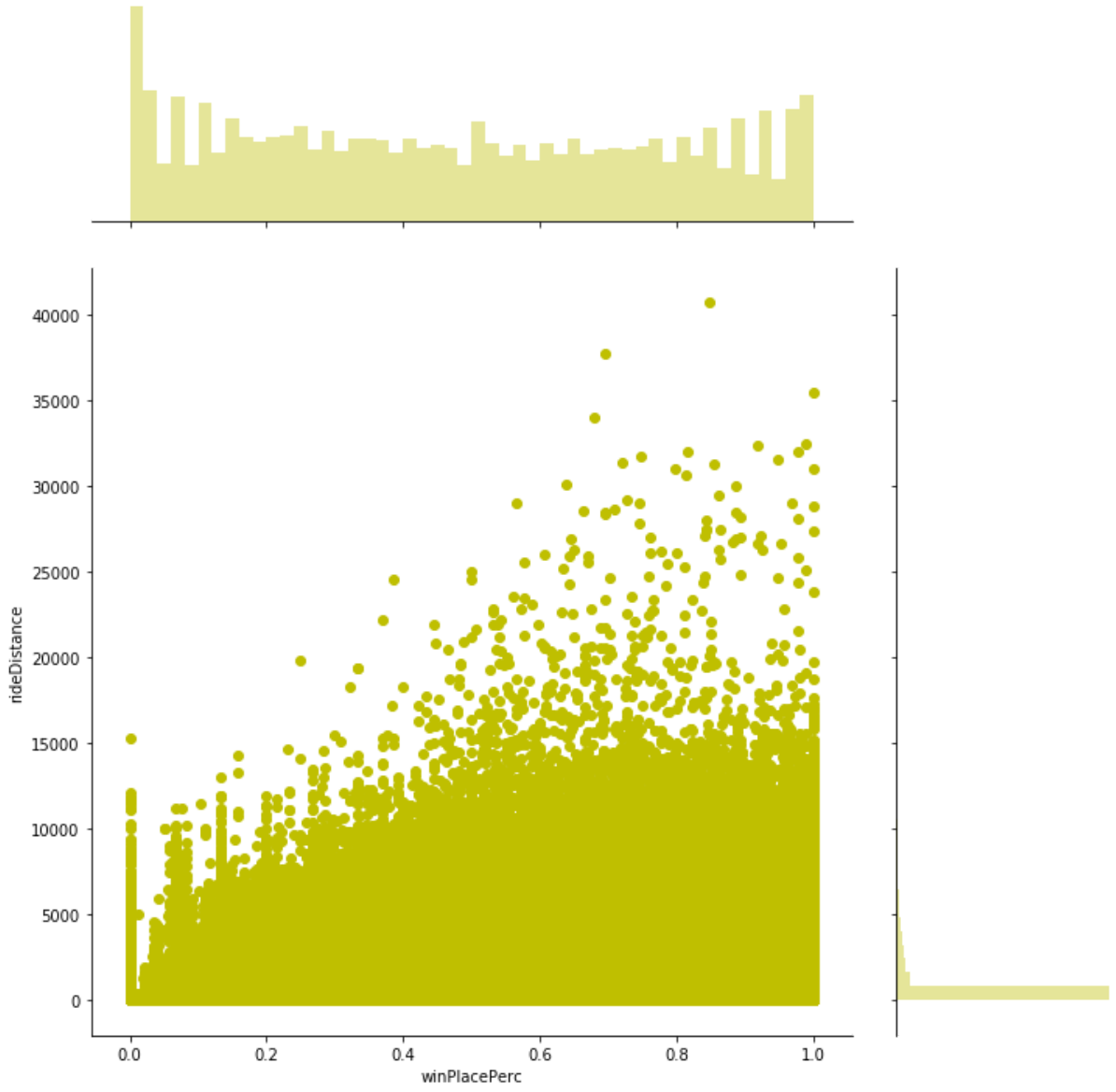
In [60]:
```
data = df2.copy()
data = data[data['rideDistance'] < df1['rideDistance'].quantile(0.9)]
plt.figure(figsize=(15,10))
plt.title("Ride Distance Distribution",fontsize=15)
sns.distplot(data['rideDistance'])
plt.show()
```



In [61]:
```
print("{} players ({:.4f}%) drived for 0 meters. This means that they don't have a
 driving licence yet."\
      .format(len(data[data['rideDistance'] == 0]), \
              100*len(data1[data1['rideDistance']==0])/len(df2)))
```

```
3309428 players (23.1022%) drived for 0 meters. This means that they don't have a
driving licence yet.
```

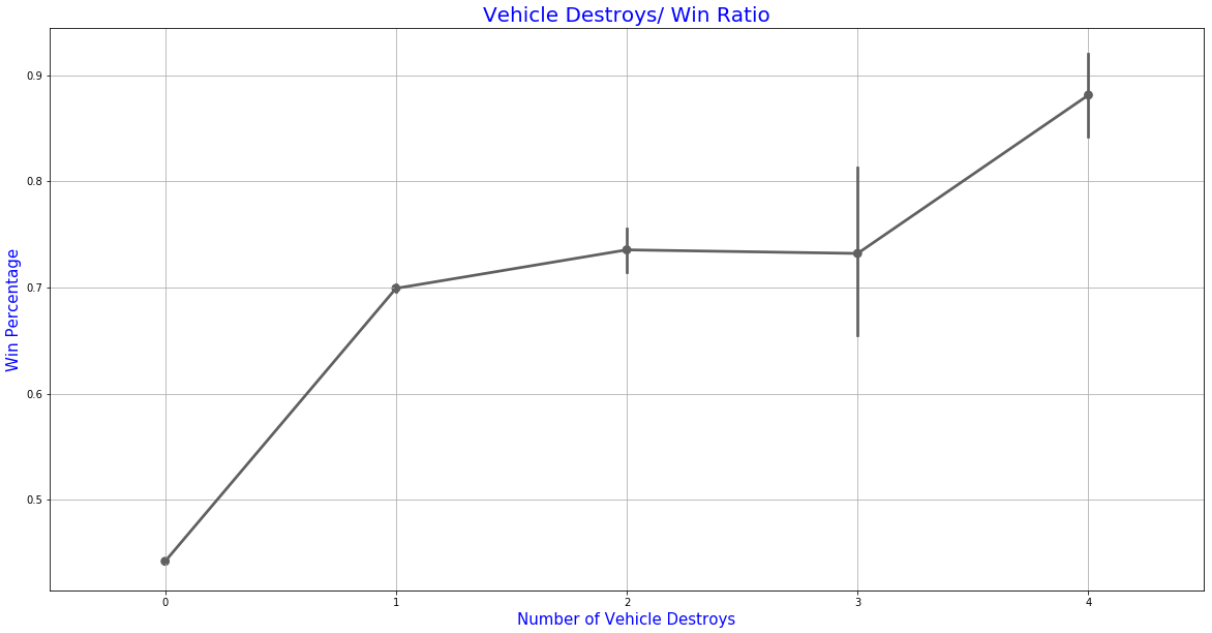
In [62]:
```
sns.jointplot(x="winPlacePerc", y="rideDistance", data=df2, height=10, ratio=3, color="y")
plt.show()
```



There is a small correlation between rideDistance and winPlacePerc.

In [71]:
```
# Destroying a vehicle in my experience shows that a player has skills. Let's check it.
```

In [63]:
```
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!

In [72]:
```
# The Swimmers
```

In [65]:
```
print("The average person swims for {:.1f}m, 99% of people have swimemd {}m or less, while the olympic champion swimmed for {}m."\
      .format(df2['swimDistance'].mean(), df2['swimDistance']\
              .quantile(0.99), df2['swimDistance'].max()))
```

```
The average person swims for 4.5m, 99% of people have swimemd 123.0m or less, while the olympic champion swimmed for 3823.0m.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [26]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

random forest

In [14]:
```
param_grid = [{'n_estimators':[10,50,100,200],'max_depth':[5,10,15,20],
               'max_features':[3,5,6,8]}]
```

In [19]:
```
temp_df2 = df2[:1000]
```

In [20]:
```
X1 = temp_df2.drop(['winPlacePerc'],axis=1)
```

In [21]:
```
y1 = temp_df2['winPlacePerc']
```

In [18]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [22]:
```
grid = GridSearchCV(RandomForestRegressor(), param_grid=param_grid, cv=5
)
```

In [23]:
```
grid.fit(X1, y1)
```

Out[23]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion
='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             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='warn', n_job
s=None,
                                             oob_score=False, random_st
ate=None,
                                             verbose=0, warm_start=Fals
e),
             iid='warn', n_jobs=None,
             param_grid=[{'max_depth': [5, 10, 15, 20],
                          'max_features': [3, 5, 6, 8],
                          'n_estimators': [10, 50, 100, 200]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fa
lse,
             scoring=None, verbose=0)
```

In [24]:
```
print(grid.best_params_)
```

```
{'max_depth': 15, 'max_features': 8, 'n_estimators': 200}
```

In [25]:
```
random_forest = RandomForestRegressor(n_estimators=200,max_features=8,ma
x_depth=15)
```

In [28]:
```
random_forest.fit(X_train, y_train)
```

Out[28]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=15,
                      max_features=8, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=Non
e,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=200,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [29]:
```
predrf = random_forest.predict(X_test)
```

In [30]:
```
from sklearn.metrics import mean_squared_error
```

In [32]:
```
mean_squared_error(y_test,predrf)
```

Out[32]:
```
0.007477802922092167
```

In [33]:
```
R_squared = random_forest.score(X_train,y_train)
R_squared
```

Out[33]:
```
0.9263411451146588
```

In [34]:
```
cvs_rf = np.mean(cross_val_score(random_forest, X1, y1, cv=3))
```

In [35]:
```
print(cvs_rf)
```

```
0.8616735500885596
```

In [ ]:
```
# For random forest mode, the MSE is 0.0074778, R_squared is 0.9263, cro
ss validation score is 0.8617
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [61]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
```

In [62]:
```
lab_enc = preprocessing.LabelEncoder()
y_train_new = lab_enc.fit_transform(y_train)
y_train_new
```

Out[62]:
```
array([1374, 2952,    0, ...,  616, 1751,   95], dtype=int64)
```

In [63]:
```
y1 = pd.DataFrame()
```

In [64]:
```
y1['trian'] = y_train_new
y1 = y1[:10000]
X = X_train[:10000]
```

In [65]:
```
param_grid = [{'min_samples_split':range(10,500,20),'max_depth': range(1,20,1)}]
grid = GridSearchCV(DecisionTreeRegressor(), param_grid=param_grid, cv=5)
grid.fit(X, y1)
print(grid.best_params_)
```

```
{'max_depth': 9, 'min_samples_split': 70}
```

In [126]:
```
# Prediction based on test file
decision_tree = DecisionTreeRegressor(max_depth=9, min_samples_split=70)
decision_tree.fit(X_train, y_train)
Y_pred = decision_tree.predict(X_test)
```

In [167]:
```
# The test file Y
```

In [165]:
```
# Prediction based on real file
Y_pred = decision_tree.predict(X_test)
Y_p = pd.DataFrame()
Y_p['winPlacePerc']= Y_pred_final
Y_p[:5]
```

Out[165]:

	winPlacePerc
0	0.245682
1	0.879634
2	0.712843
3	0.602237
4	0.925602

In [168]:
```
# The real prediction from decision tree model
```

In [166]:
```
Y_pred_final = decision_tree.predict(test2)
Y_p_final = pd.DataFrame()
Y_p_final['winPlacePerc']= Y_pred_final
Y_p_final[:5]
```

Out[166]:

	winPlacePerc
0	0.245682
1	0.879634
2	0.712843
3	0.602237
4	0.925602

In [194]:
```
# Cross Validation of test file
cvs_tree = np.mean(cross_val_score(decision_tree, X_test, y_test, cv=3))
print('The Cross Validation value is: ', cvs_tree)
```

```
The Cross Validation value is:  0.8898589363593024
```

In [190]:
```
SSE = mean_squared_error(y_test, Y_pred) * len(y_train)
R_squared = r2_score(y_test, Y_pred)
MSE = mean_squared_error(y_test,Y_pred)
print('The SSE value is: ', SSE)
print('The R_squared value is :', R_squared)
print('The MSE value is:', MSE)
```

```
The SSE value is:  34584.44557390435
The R_squared value is : 0.8902422540621205
The MSE value is: 0.01036945522800255
```

In [ ]:

In [ ]:

In [ ]:

In [26]:
```
df2.describe().astype('int64')
```

Out[26]:

	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	killPoints	444
count	4446965	4446965	4446965	4446965	4446965	4446965	4446965	4446965	
mean	0	1	130	0	0	1	47	505	
std	0	1	170	1	0	2	27	627	
min	0	0	0	0	0	0	1	0	
25%	0	0	0	0	0	0	24	0	
50%	0	0	84	0	0	0	47	0	
75%	0	2	186	1	0	2	71	1172	
max	22	33	6616	53	64	80	101	2170	

8 rows × 29 columns

In [27]:
```
from sklearn.model_selection import train_test_split
```

In [28]:
```
df3 = df2.drop(['winPlacePerc'],axis = 1)
```

In [29]:
```
X_train,X_test,y_train,y_test = train_test_split(df3,df2['winPlacePerc'
],test_size = 0.25,random_state=1)
```

In [ ]:
```
# apply prediction models, please see the below
```

Gradient Boost Regression

In [73]:
```
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBClassifier
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
# since our train dataset size has 4 millons, grid search spend too much
time for the whole train model
# we just select teh first 10thousand to find the best paramters
```

In [41]:
```
temp_df2 = df2[:10000]
```

In [42]:
```
X1 = temp_df2.drop(['winPlacePerc'],axis=1)
```

In [43]:
```
y1 = temp_df2['winPlacePerc']
```

In [44]:
```
X1_train,X1_test,y1_train,y1_test = train_test_split(X1,y1,test_size =
0.33,
                                                     random_state = 1)
```

In [62]:
```
param_grid = [{'learning_rate':[0.01,0.025,0.1,0.25,0.5,0.8],'max_depth'
:[3,5,10,15],
               'n_estimators':[10,50,100,200]}]
```

In [63]:
```
grid = GridSearchCV(GradientBoostingRegressor(), param_grid=param_grid,
cv=5)
```

In [64]:
```
grid.fit(X1, y1)
```

Out[64]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_ms
e',
                                                 init=None, learning_ra
te=0.1,
                                                 loss='ls', max_depth=
3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=
0.0,
                                                 min_impurity_split=Non
e,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_le
af=0.0,
                                                 n_estimators=100,
                                                 n_iter_no_change=None,
                                                 presort='auto',
                                                 random_state=None,
                                                 subsample=1.0, tol=0.0
001,
                                                 validation_fraction=0.
1,
                                                 verbose=0, warm_start=
False),
             iid='warn', n_jobs=None,
             param_grid=[{'learning_rate': [0.01, 0.025, 0.1, 0.25, 0.
5, 0.8],
                          'max_depth': [3, 5, 10, 15],
                          'n_estimators': [10, 50, 100, 200]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fa
lse,
             scoring=None, verbose=0)
```

In [65]:
```
print(grid.best_params_)
```

```
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
```

In [68]:
```
xgboost = GradientBoostingRegressor(n_estimators = 200,learning_rate =
0.1,max_depth=5)
```

In [69]:
```
xgboost.fit(X_train,y_train)
```

Out[69]:
```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=Non
e,
                          learning_rate=0.1, loss='ls', max_depth=5,
                          max_features=None, 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=20
0,
                          n_iter_no_change=None, presort='auto',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_star
t=False)
```

In [70]:
```
predxg = xgboost.predict(X_test)
```

In [74]:
```
mean_squared_error(y_test,predxg)
```

Out[74]:
```
0.006864648741405337
```

In [75]:
```
R_squared = xgboost.score(X_train,y_train)
R_squared
```

Out[75]:
```
0.927846634172564
```

In [71]:
```
cvs_xgboost = np.mean(cross_val_score(xgboost, X1, y1, cv=3))
```

In [76]:
```
print(cvs_xgboost)
```

```
0.9147104623229937
```

In [55]:
```
y1
```

Out[55]:
```
0       0.4444
1       0.6400
2       0.7755
3       0.1667
4       0.1875
         ...
9995    0.0000
9996    0.9773
9997    0.5000
9998    0.6522
9999    0.1071
Name: winPlacePerc, Length: 10000, dtype: float64
```

In [ ]:
```
# From the gradient boosting, we find that MSE is 0.006, R_squared is 0.
9278, and cross validation score is 0.9147
# The MSE here is quite small
```

Multiple linear regression

In [77]:
```
from scipy.special import comb
from sklearn.linear_model import LinearRegression
```

In [78]:
```
model1 = LinearRegression(n_jobs=-1,normalize = True).fit(X_train,y_trai
n)
```

In [79]:
```
pred = model1.predict(X_test)
mean_squared_error(y_test,pred)
```

Out[79]: 0.015254254100941219

In [80]:
```
kfold = KFold(n_splits=5, random_state = 1)
```

In [81]:
```
measure = 'neg_mean_squared_error'
```

In [83]:
```
cvs_log = np.mean(cross_val_score(model1, X1, y1, cv=kfold))
print(cvs_log)
```

```
0.8404535181996102
```

In [ ]:
```
# For Multiple linear regression, MSE is 0.01525, and cross validation s
core is 0.84045
```

logistic regression

since the logistic regression is only used for binary prediction, while the y-train and y_test is the float number, so we cannot used the logistic regression

as we compare all the model's MSE together. the gradient boosting regressor has the smallest MSE and largest r_square. so we choose the gradient boost as the best and fittest model

In [92]:
```
test = pd.read_csv('test_V2.csv')
```

In [93]:
```
test.shape
```

Out[93]: (1934174, 28)

In [94]:
```
test.isna().sum()
```

Out[94]:
```
Id                0
groupId           0
matchId           0
assists           0
boosts            0
damageDealt       0
DBNOs             0
headshotKills     0
heals             0
killPlace         0
killPoints        0
kills             0
killStreaks       0
longestKill       0
matchDuration     0
matchType         0
maxPlace          0
numGroups         0
rankPoints        0
revives           0
rideDistance      0
roadKills         0
swimDistance      0
teamKills         0
vehicleDestroys   0
walkDistance      0
weaponsAcquired   0
winPoints         0
dtype: int64
```

In [95]:
```
test['matchType'] = test['matchType'].apply(lambda x: 'solo' if x == 'so
lo-fpp'or x=='solo' or x=='normal-solo-fpp' or x=='normal-solo'
                                            else ('duo' if x=='duo'or x
=='duo-fpp'or x=='normal-duo-fpp'or x=='normal-duo' else ('squad'if x==
'squad'or x=='squad-fpp' or x=='normal-squad-fpp'or x=='normal-squad'els
e 'custome')))
```

In [96]:
```
test1 = pd.get_dummies(test,columns = ['matchType'])
```

In [97]:
```
test1[:5]
```

Out[97]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	he
0	9329eb41e215eb	676b23c24e70d6	45b576ab7daa7f	0	0	51.46	0	
1	639bd0dcd7bda8	430933124148dd	42a9a0b906c928	0	4	179.10	0	
2	63d5c8ef8dfe91	0b45f5db20ba99	87e7e4477a048e	1	0	23.40	0	
3	cf5b81422591d1	b7497dbdc77f4a	1b9a94f1af67f1	0	0	65.52	0	
4	ee6a295187ba21	6604ce20a1d230	40754a93016066	0	4	330.20	1	

5 rows × 31 columns

In [98]:
```
test2 = test1.drop(['Id', 'groupId','matchId'], axis=1)
```

In [99]:
```
result = xgboost.predict(test2)
```

In [102]:
```
submit = pd.DataFrame()
submit['Id'] = test1.Id
submit['winPlacePerc'] = result
```

In [104]:
```
submit.to_csv('submit.csv',index=False,header=True)
```

In [ ]:
```
# we got the prediction from test model using xgbooting model
```