scikit-RF

July 13, 2019

In [1]:
```
## Something went wrong when importing fastai.structured.
## We fixed this by put the whole source code of fastai.structured in the notebook.
## This was copied from: https://github.com/anandsaha/fastai.part1.v2/blob/master/fastai

from sklearn_pandas import DataFrameMapper
from sklearn.preprocessing import LabelEncoder, Imputer, StandardScaler
from pandas.api.types import is_string_dtype, is_numeric_dtype
from sklearn.ensemble import forest
from sklearn.tree import export_graphviz

def get_sample(df,n):
    """ Gets a random sample of n rows from df, without replacement.
    Parameters:
    -----------
    df: A pandas data frame, that you wish to sample from.
    n: The number of rows you wish to sample.
    Returns:
    --------
    return value: A random sample of n rows of df.
    Examples:
    ---------
    >>> df = pd.DataFrame({'col1' : [1, 2, 3], 'col2' : ['a', 'b', 'a']})
    >>> df
       col1 col2
    0     1    a
    1     2    b
    2     3    a
    >>> get_sample(df, 2)
       col1 col2
    2     3    a
    1     2    b
    """
    idxs = sorted(np.random.permutation(len(df))[:n])
    return df.iloc[idxs].copy()

def proc_df(df, y_fld, skip_flds=None, do_scale=False, na_dict=None,
            preproc_fn=None, max_n_cat=None, subset=None, mapper=None):
```

```
""" proc_df takes a data frame df and splits off the response variable, and
changes the df into an entirely numeric dataframe.
Parameters:
-----------
df: The data frame you wish to process.
y_fld: The name of the response variable
skip_flds: A list of fields that dropped from df.
do_scale: Standardizes each column in df,Takes Boolean Values(True,False)
na_dict: a dictionary of na columns to add. Na columns are also added if there
    are any missing values.
preproc_fn: A function that gets applied to df.
max_n_cat: The maximum number of categories to break into dummy values, instead
    of integer codes.
subset: Takes a random subset of size subset from df.
mapper: If do_scale is set as True, the mapper variable
    calculates the values used for scaling of variables during training time(mean an
Returns:
--------
[x, y, nas, mapper(optional)]:
    x: x is the transformed version of df. x will not have the response variable
        and is entirely numeric.
    y: y is the response variable
    nas: returns a dictionary of which nas it created, and the associated median.
    mapper: A DataFrameMapper which stores the mean and standard deviation of the co
    variables which is then used for scaling of during test-time.
Examples:
---------
>>> df = pd.DataFrame({'col1' : [1, 2, 3], 'col2' : ['a', 'b', 'a']})
>>> df
   col1 col2
0     1    a
1     2    b
2     3    a
note the type of col2 is string
>>> train_cats(df)
>>> df
   col1 col2
0     1    a
1     2    b
2     3    a
now the type of col2 is category { a : 1, b : 2}
>>> x, y, nas = proc_df(df, 'col1')
>>> x
   col2
0     1
1     2
2     1
>>> data = DataFrame(pet=["cat", "dog", "dog", "fish", "cat", "dog", "cat", "fish"],
```

```
                    children=[4., 6, 3, 3, 2, 3, 5, 4],
                    salary=[90, 24, 44, 27, 32, 59, 36, 27])
        >>> mapper = DataFrameMapper([(:pet, LabelBinarizer()),
                                ([:children], StandardScaler())])
        >>>round(fit_transform!(mapper, copy(data)), 2)
        8x4 Array{Float64,2}:
        1.0  0.0  0.0   0.21
        0.0  1.0  0.0   1.88
        0.0  1.0  0.0  -0.63
        0.0  0.0  1.0  -0.63
        1.0  0.0  0.0  -1.46
        0.0  1.0  0.0  -0.63
        1.0  0.0  0.0   1.04
        0.0  0.0  1.0   0.21
        """
        if not skip_flds: skip_flds=[]
        if subset: df = get_sample(df,subset)
        df = df.copy()
        if preproc_fn: preproc_fn(df)
        y = df[y_fld].values
        df.drop(skip_flds+[y_fld], axis=1, inplace=True)

        if na_dict is None: na_dict = {}
        for n,c in df.items(): na_dict = fix_missing(df, c, n, na_dict)
        if do_scale: mapper = scale_vars(df, mapper)
        for n,c in df.items(): numericalize(df, c, n, max_n_cat)
        res = [pd.get_dummies(df, dummy_na=True), y, na_dict]
        if do_scale: res = res + [mapper]
        return res

    def rf_feat_importance(m, df):
        return pd.DataFrame({'cols':df.columns, 'imp':m.feature_importances_}
                            ).sort_values('imp', ascending=False)

    def set_rf_samples(n):
        """ Changes Scikit learn's random forests to give each tree a random sample of
        n random rows.
        """
        forest._generate_sample_indices = (lambda rs, n_samples:
            forest.check_random_state(rs).randint(0, n_samples, n))

    def reset_rf_samples():
        """ Undoes the changes produced by set_rf_samples.
        """
        forest._generate_sample_indices = (lambda rs, n_samples:
            forest.check_random_state(rs).randint(0, n_samples, n_samples))
```

In [2]:
```
# Standard libraries
```

```
import os
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from pdpbox import pdp
#from plotnine import *
from pandas_summary import DataFrameSummary
from sklearn.ensemble import RandomForestRegressor
from IPython.display import display

# Machine Learning
import sklearn
from sklearn import metrics
from scipy.cluster import hierarchy as hc
from fastai.imports import *

# Directories
KAGGLE_DIR = './'
```

In [3]:
```
train = pd.read_csv('./train_V2.csv')
test = pd.read_csv('./test_V2.csv')
```

1 Illegal Match

Fellow Kaggler 'averagemn' brought to our attention that there is one particular player with a 'winPlacePerc' of NaN. The case was that this match had only one player. We will delete this row from our dataset.

In [4]:
```
# Check row with NaN value
train[train['winPlacePerc'].isnull()]
```

Out[4]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	...	revives	rideDistance	roadKills	swimDistance	teamKills	vehicleDestroys	walkDistance	weaponsAcquired	winPoints	winPlacePerc
2744604	f70c74418bb064	12dfbede33f92b	224a123c53e008	0	0	0.0	0	0	0	1	...	0	0.0	0	0.0	0	0	0.0	0	0	NaN

[1 rows x 29 columns]

Let's delete this entry:

```
In [5]: # Drop row with NaN 'winPlacePerc' value
        train.drop(2744604, inplace=True)
```

And he's gone!

```
In [6]: # The row at index 2744604 will be gone
        train[train['winPlacePerc'].isnull()]
```

```
Out[6]: Empty DataFrame
        Columns: [Id, groupId, matchId, assists, boosts, damageDealt, DBNOs, headshotKills, heal
        Index: []

        [0 rows x 29 columns]
```

2 Feature Engineering

Earlier in this kernel we created the new features "totalDistance" and "headshot_rate". In this section we add more interesting features to improve the predictive quality of our machine learning models.

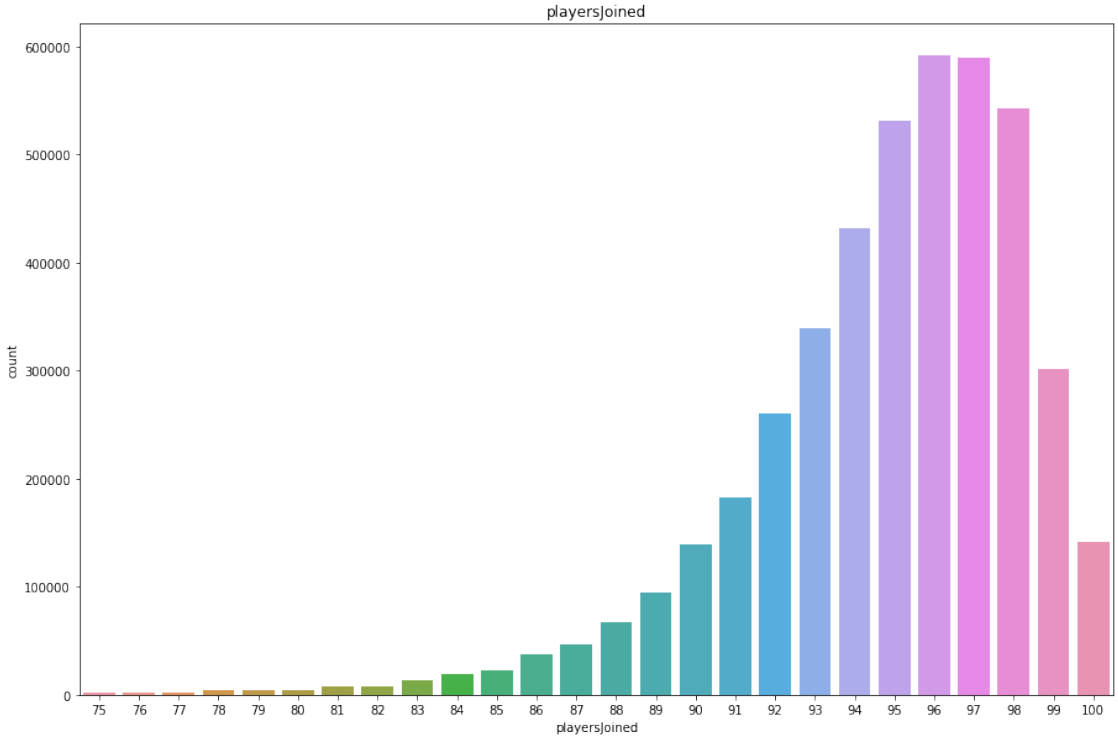
Initial ideas for this section come from this amazing kernel.

Note: It is important with feature engineering that you also add the engineered features to your test set!

2.0.1 Players Joined

This is likely a very valuable feature for our model. If we know how many people are in a match we can normalize other features and get stronger predictions on individual players.

```
In [7]: # playersJoined
        train['playersJoined'] = train.groupby('matchId')['matchId'].transform('count')
        plt.figure(figsize=(15,10))
        sns.countplot(train[train['playersJoined']>=75]['playersJoined'])
        plt.title('playersJoined')
        plt.show()
```



There are a few matches with fewer than 75 players that are not displayed here. As you can see most of the matches are nearly packed a have nearly 100 players. It is nevertheless interesting to take these features into our analysis.

2.0.2 Normalized features

```
In [8]: # Create normalized features
        train['killsNorm'] = train['kills']*((100-train['playersJoined'])/100 + 1)
        train['damageDealtNorm'] = train['damageDealt']*((100-train['playersJoined'])/100 + 1)
        train['maxPlaceNorm'] = train['maxPlace']*((100-train['playersJoined'])/100 + 1)
        train['matchDurationNorm'] = train['matchDuration']*((100-train['playersJoined'])/100 +
        # Compare standard features and normalized features
        to_show = ['Id', 'kills','killsNorm','damageDealt', 'damageDealtNorm', 'maxPlace', 'maxP
        train[to_show][0:11]
```

Out[8]:

	Id	kills	killsNorm	damageDealt	damageDealtNorm	maxPlace	\
0	7f96b2f878858a	0	0.00	0.000	0.00000	28	
1	eef90569b9d03c	0	0.00	91.470	99.70230	26	
2	1eaf90ac73de72	0	0.00	68.000	69.36000	50	
3	4616d365dd2853	0	0.00	32.900	35.86100	31	
4	315c96c26c9aac	1	1.03	100.000	103.00000	97	
5	ff79c12f326506	1	1.05	100.000	105.00000	28	
6	95959be0e21ca3	0	0.00	0.000	0.00000	28	
7	311b84c6ff4390	0	0.00	8.538	8.87952	96	
8	1a68204ccf9891	0	0.00	51.600	53.14800	28	

```
9   e5bb5a43587253      0       0.00       37.270          38.38810        29
10  2b574d43972813      0       0.00       28.380          28.66380        29

    maxPlaceNorm  matchDuration  matchDurationNorm
0          29.12           1306            1358.24
1          28.34           1777            1936.93
2          51.00           1318            1344.36
3          33.79           1436            1565.24
4          99.91           1424            1466.72
5          29.40           1395            1464.75
6          28.84           1316            1355.48
7          99.84           1967            2045.68
8          28.84           1375            1416.25
9          29.87           1930            1987.90
10         29.29           1811            1829.11
```

2.0.3 Heals and Boosts

We create a feature called 'healsandboosts' by adding heals and boosts. (duh!) We are not sure if this has additional predictive value, but we can always delete it later if the feature importance according to our random forest model is too low.

```
In [9]: # Create new feature healsandboosts
        train['healsandboosts'] = train['heals'] + train['boosts']
        train[['heals', 'boosts', 'healsandboosts']].tail()
```

```
Out[9]:          heals  boosts  healsandboosts
        4446961      0       0               0
        4446962      0       1               1
        4446963      0       0               0
        4446964      2       4               6
        4446965      1       2               3
```

2.0.4 Killing without moving

We try to identify cheaters by checking if people are getting kills without moving. We first identify the totalDistance travelled by a player and then set a boolean value to True if someone got kills without moving a single inch. We will remove cheaters in our outlier detection section.

```
In [10]: # Create feature totalDistance
         train['totalDistance'] = train['rideDistance'] + train['walkDistance'] + train['swimDis
         # Create feature killsWithoutMoving
         train['killsWithoutMoving'] = ((train['kills'] > 0) & (train['totalDistance'] == 0))
```

The feature headshot_rate will also help us to catch cheaters.

```
In [11]: # Create headshot_rate feature
         train['headshot_rate'] = train['headshotKills'] / train['kills']
         train['headshot_rate'] = train['headshot_rate'].fillna(0)
```

3 Outlier Detection

Some rows in our dataset have weird characteristics. The players could be cheaters, maniacs or just anomalies. Removing these outliers will likely improve results.

Inspiration for this section comes from this amazing Kaggle Kernel.

Kills without movement

This is perhaps the most obvious sign of cheating in the game. It is already fishy if a player hasn't moved during the whole game, but the player could be AFK and got killed. However, if the player managed to get kills without moving it is most likely a cheater.

```
In [12]: # Check players who kills without moving
         display(train[train['killsWithoutMoving'] == True].shape)
         train[train['killsWithoutMoving'] == True].head(10)
```

```
(1535, 38)
```

Out[12]:

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

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	winPlacePerc	\
1824	593.00	0	0	3	18	...	0.8571	
6673	346.60	0	0	6	33	...	0.6000	
11892	1750.00	0	4	5	3	...	0.8947	
14631	157.80	0	0	0	69	...	0.0000	
15591	100.00	0	1	0	37	...	0.3000	
20881	506.10	4	1	3	7	...	0.8000	
23298	1124.00	0	4	1	7	...	0.6000	
24640	529.90	0	2	11	12	...	0.8571	
25659	128.90	0	1	6	53	...	0.2857	
30079	85.56	0	0	0	46	...	0.8571	

	playersJoined	killsNorm	damageDealtNorm	maxPlaceNorm	\
1824	58	8.52	842.0600	21.30	
6673	42	4.74	547.6280	17.38	
11892	21	35.80	3132.5000	35.80	
14631	73	1.27	200.4060	24.13	
15591	42	1.58	158.0000	17.38	
20881	44	9.36	789.5160	9.36	
23298	48	18.24	1708.4800	9.12	

```
24640             57      10.01         757.7570         21.45
25659             61       2.78         179.1710         11.12
30079             53       1.47         125.7732         22.05

      matchDurationNorm  healsandboosts  totalDistance  killsWithoutMoving  \
1824             842.06               3            0.0                True
6673            2834.52               6            0.0                True
11892           1607.42               5            0.0                True
14631           1014.73               0            0.0                True
15591           2834.52               0            0.0                True
20881            909.48               3            0.0                True
23298            836.00               1            0.0                True
24640            856.57              11            0.0                True
25659           1017.48               7            0.0                True
30079           1051.05               0            0.0                True

       headshot_rate
1824        0.000000
6673        0.000000
11892       0.200000
14631       0.000000
15591       1.000000
20881       0.166667
23298       0.333333
24640       0.285714
25659       0.500000
30079       0.000000

[10 rows x 38 columns]
```

Got the suckers!

```
In [13]: # Remove outliers
         train.drop(train[train['killsWithoutMoving'] == True].index, inplace=True)
```

Anomalies in roadKills

```
In [14]: # Players who got more than 10 roadKills
         train[train['roadKills'] > 10]
```

```
Out[14]:                    Id         groupId         matchId  assists  boosts  \
2733926  c3e444f7d1289f  489dd6d1f2b3bb  4797482205aaa4        0       0
2767999  34193085975338  bd7d50fa305700  a22354d036b3d6        0       0
2890740  a3438934e3e535  1081c315a80d14  fe744430ac0070        0       8
3524413  9d9d044f81de72  8be97e1ba792e3  859e2c2db5b125        0       3

         damageDealt  DBNOs  headshotKills  heals  killPlace  ...  \
2733926       1246.0      0              0      0          1  ...
2767999       1102.0      0              0      0          1  ...
```

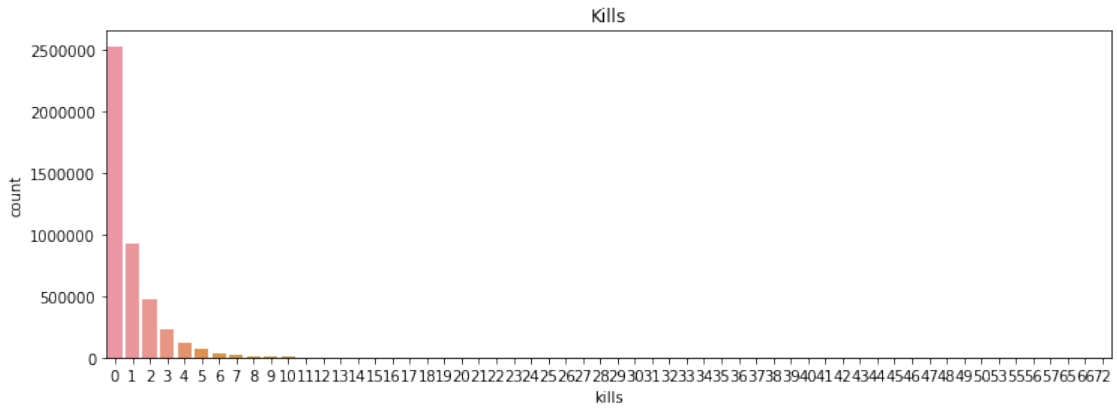
```
2890740       2074.0      0              1     11          1  ...
3524413       1866.0      0              5      7          1  ...

         winPlacePerc  playersJoined  killsNorm  damageDealtNorm  \
2733926        0.4286             92      15.12          1345.68
2767999        0.4713             88      12.32          1234.24
2890740        1.0000             38      32.40          3359.88
3524413        0.9398             84      20.88          2164.56

         maxPlaceNorm matchDurationNorm  healsandboosts  totalDistance  \
2733926         99.36           1572.48               0       1282.302
2767999         98.56           2179.52               0       4934.600
2890740         61.56           3191.40              19       5876.000
3524413         97.44           2233.00              10       7853.000

         killsWithoutMoving  headshot_rate
2733926               False       0.000000
2767999               False       0.000000
2890740               False       0.050000
3524413               False       0.277778

[4 rows x 38 columns]
```

```
In [15]: # Drop roadKill 'cheaters'
         train.drop(train[train['roadKills'] > 10].index, inplace=True)
```

Anomalies in aim (More than 30 kills)

Let's plot the total kills for every player first. It doesn't look like there are too many outliers.

```
In [16]: # Plot the distribution of kills
         plt.figure(figsize=(12,4))
         sns.countplot(data=train, x=train['kills']).set_title('Kills')
         plt.show()
```



Let's take a closer look.

In [17]:
```
# Players who got more than 30 kills
display(train[train['kills'] > 30].shape)
train[train['kills'] > 30].head(10)
```

```
(95, 38)
```

Out[17]:

	Id	groupId	matchId	assists	boosts	\
57978	9d8253e21ccbbd	ef7135ed856cd8	37f05e2a01015f	9	0	
87793	45f76442384931	b3627758941d34	37f05e2a01015f	8	0	
156599	746aa7eabf7c86	5723e7d8250da3	f900de1ec39fa5	21	0	
160254	15622257cb44e2	1a513eeecfe724	db413c7c48292c	1	0	
180189	1355613d43e2d0	f863cd38c61dbf	39c442628f5df5	5	0	
334400	810f2379261545	7f3e493ee71534	f900de1ec39fa5	20	0	
353128	f3e9746e3ff151	4bc1f00f07b304	a9e84c456cc859	2	0	
457829	265e23756baa0b	9d94424171c2a1	664dee9ed8f646	3	0	
488335	31a0682922ef45	275a27a3ee4cc8	3037f74ef8a3a3	2	0	
662650	dd424a8b74bd49	ac9dea6d62f2e6	8a728def0644be	9	0	

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
57978	3725.0	0	7	0	2	...	
87793	3087.0	0	8	27	3	...	
156599	5479.0	0	12	7	4	...	
160254	4033.0	0	40	0	1	...	
180189	3171.0	0	6	15	1	...	
334400	6616.0	0	13	5	1	...	
353128	3834.0	0	9	5	1	...	
457829	2907.0	0	27	2	1	...	
488335	3055.0	0	9	0	1	...	
662650	3454.0	38	9	4	1	...	

	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	maxPlaceNorm	\
57978	0.8571	16	64.40	6854.00	14.72	
87793	1.0000	16	57.04	5680.08	14.72	
156599	0.7000	11	90.72	10355.31	20.79	
160254	1.0000	62	57.96	5565.54	11.04	
180189	1.0000	11	66.15	5993.19	17.01	
334400	1.0000	11	122.85	12504.24	20.79	
353128	1.0000	13	72.93	7169.58	24.31	
457829	1.0000	38	53.46	4709.34	8.10	
488335	1.0000	20	59.40	5499.00	32.40	
662650	0.2308	54	49.64	5042.84	20.44	

	matchDurationNorm	healsandboosts	totalDistance	killsWithoutMoving	\
57978	3308.32	0	48.82	False	
87793	3308.32	27	780.70	False	
156599	3398.22	7	23.71	False	
160254	1164.72	0	718.30	False	

```
180189          3394.44              15          71.51              False
334400          3398.22               5        1036.00              False
353128          3356.65               5         124.20              False
457829          1339.74               2         382.40              False
488335          1605.60               0          35.30              False
662650          1749.08               4         111.10              False

        headshot_rate
57978        0.200000
87793        0.258065
156599       0.250000
160254       0.952381
180189       0.171429
334400       0.200000
353128       0.230769
457829       0.818182
488335       0.272727
662650       0.264706

[10 rows x 38 columns]
```

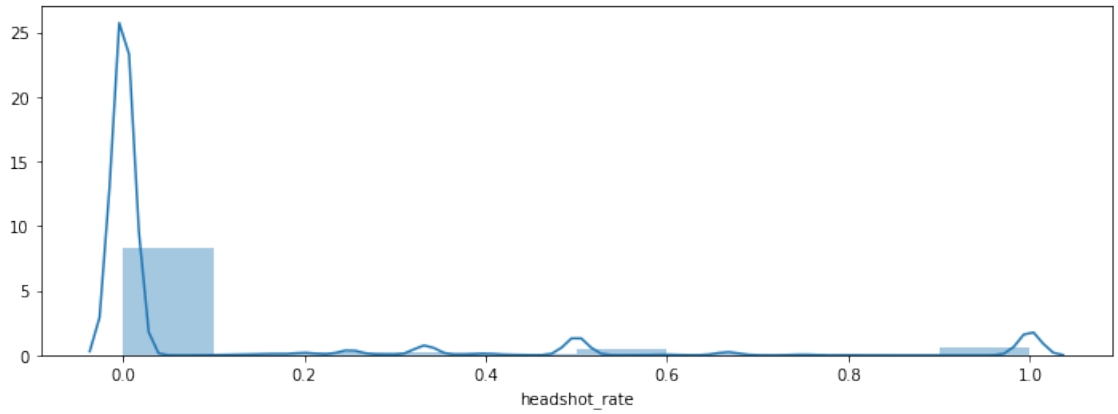
```
In [18]: # Remove outliers
         train.drop(train[train['kills'] > 30].index, inplace=True)
```

What do you think? Should we remove all these outliers from our dataset?

Anomalies in aim part 2 (100% headshot rate)

Again, we first take a look at the whole dataset and create a new feature 'headshot_rate'. We see that the most players score in the 0 to 10% region. However, there are a few anomalies that have a headshot_rate of 100% percent with more than 9 kills!

```
In [19]: # Plot the distribution of headshot_rate
         plt.figure(figsize=(12,4))
         sns.distplot(train['headshot_rate'], bins=10)
         plt.show()
```



In [20]:
```
# Players who made a minimum of 10 kills and have a headshot_rate of 100%
display(train[(train['headshot_rate'] == 1) & (train['kills'] > 9)].shape)
train[(train['headshot_rate'] == 1) & (train['kills'] > 9)].head(10)
```

```
(24, 38)
```

Out[20]:

	Id	groupId	matchId	assists	boosts	\
281570	ab9d7168570927	add05ebde0214c	e016a873339c7b	2	3	
346124	044d18fc42fc75	fc1dbc2df6a887	628107d4c41084	3	5	
871244	e668a25f5488e3	5ba8feabfb2a23	f6e6581e03ba4f	0	4	
908815	566d8218b705aa	a9b056478d71b2	3a41552d553583	2	5	
963463	1bd6fd288df4f0	90584ffa22fe15	ba2de992ec7bb8	2	6	
1079403	1c245ed99b5f96	e42d09a9b8463a	5cec236bce68eb	0	5	
1167959	c4f80d4be5c561	b4a7892189b5dd	c7f7733ebbd447	0	4	
1348164	474a641f0a4bcb	2fdad3ca6fb3c0	114499c82f35d7	1	5	
1380385	202ce6a55119c5	2df66861f597b4	496700c29a5d44	1	4	
1483199	9d483f7cbb34d4	db5867bc814191	69495e3c478eb9	0	10	

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
281570	1212.0	8	10	0	1	...	
346124	1620.0	13	11	3	1	...	
871244	1365.0	9	13	0	1	...	
908815	1535.0	10	10	3	1	...	
963463	1355.0	12	10	2	1	...	
1079403	1218.0	8	11	3	1	...	
1167959	1065.0	6	10	1	1	...	
1348164	1319.0	11	12	1	1	...	
1380385	1150.0	4	11	1	1	...	
1483199	1478.0	8	13	2	1	...	

	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	\
281570	0.8462	93	10.70	1296.84	
346124	1.0000	96	11.44	1684.80	
871244	1.0000	98	13.26	1392.30	
908815	0.9630	95	10.50	1611.75	
963463	1.0000	96	10.40	1409.20	
1079403	1.0000	95	11.55	1278.90	
1167959	1.0000	97	10.30	1096.95	
1348164	1.0000	93	12.84	1411.33	
1380385	1.0000	88	12.32	1288.00	
1483199	1.0000	96	13.52	1537.12	

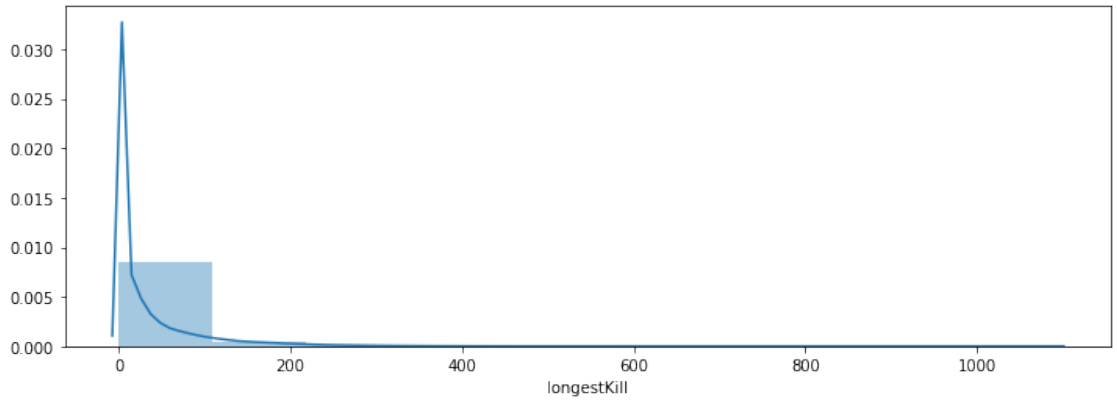
	maxPlaceNorm	matchDurationNorm	healsandboosts	totalDistance	\
281570	28.89	1522.61	3	2939.0	
346124	28.08	1796.08	8	8142.0	
871244	27.54	1280.10	4	2105.0	
908815	29.40	1929.90	8	7948.0	

```
963463          28.08          1473.68              8          3476.0
1079403         29.40          1912.05              8          8178.0
1167959         27.81          1283.38              5          2858.9
1348164         29.96          1851.10              6          5963.0
1380385         51.52          1397.76              5          3108.0
1483199         50.96          1434.16             12          2479.8

        killsWithoutMoving  headshot_rate
281570               False            1.0
346124               False            1.0
871244               False            1.0
908815               False            1.0
963463               False            1.0
1079403              False            1.0
1167959              False            1.0
1348164              False            1.0
1380385              False            1.0
1483199              False            1.0

[10 rows x 38 columns]
```

Anomalies in aim part 3 (Longest kill)

Most kills are made from a distance of 100 meters or closer. There are however some outliers who make a kill from more than 1km away. This is probably done by cheaters.

```
In [21]: # Plot the distribution of longestKill
         plt.figure(figsize=(12,4))
         sns.distplot(train['longestKill'], bins=10)
         plt.show()
```



Let's take a look at the players who make these shots.

```
In [22]: # 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, 38)
```

Out[22]:

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

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
202281	783.9	5	1	1	5	...	
240005	1284.0	8	5	7	18	...	
324313	1028.0	0	0	0	9	...	
656553	1410.0	17	5	0	3	...	
803632	196.8	0	0	0	51	...	
895411	1012.0	11	5	0	5	...	
1172437	329.3	0	0	2	45	...	
1209416	1640.0	0	7	0	1	...	
1642712	2103.0	0	4	11	11	...	
2015559	1302.0	0	6	5	15	...	

	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	\
202281	0.9231	88	4.48	877.968	
240005	0.5385	29	18.81	2195.640	
324313	1.0000	51	14.90	1531.720	
656553	0.6000	41	25.44	2241.900	
803632	0.0000	61	1.39	273.552	
895411	0.9091	86	11.40	1153.680	
1172437	0.2857	58	4.26	467.606	
1209416	0.9412	52	22.20	2427.200	
1642712	0.5000	28	39.56	3617.160	
2015559	0.6000	42	17.38	2057.160	

	maxPlaceNorm	matchDurationNorm	healsandboosts	totalDistance	\
202281	30.24	2087.68	4	3775.20	
240005	23.94	2236.68	7	48.87	
324313	19.37	1040.02	0	2981.00	
656553	9.54	1734.69	0	29.21	
803632	11.12	654.69	0	3159.00	
895411	13.68	1163.94	0	569.50	
1172437	11.36	825.02	3	832.50	
1209416	76.96	1827.80	6	2848.00	

```
1642712         25.80           3092.56              13         235.30
2015559         17.38           2834.52               8         133.20

         killsWithoutMoving  headshot_rate
202281                False       0.250000
240005                False       0.454545
324313                False       0.000000
656553                False       0.312500
803632                False       0.000000
895411                False       0.500000
1172437               False       0.000000
1209416               False       0.466667
1642712               False       0.173913
2015559               False       0.545455

[10 rows x 38 columns]
```

```
In [23]: # Remove outliers
         train.drop(train[train['longestKill'] >= 1000].index, inplace=True)
```

Anomalies in travelling (rideDistance, walkDistance and swimDistance)
Let's check out anomalies in Distance travelled.

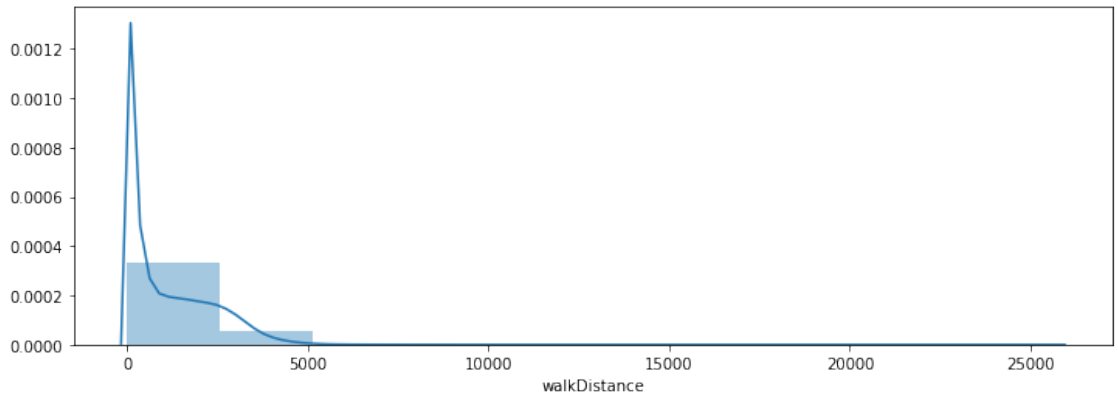
```
In [24]: # Summary statistics for the Distance features
         train[['walkDistance', 'rideDistance', 'swimDistance', 'totalDistance']].describe()
```

Out[24]:

	walkDistance	rideDistance	swimDistance	totalDistance
count	4.445311e+06	4.445311e+06	4.445311e+06	4.445311e+06
mean	1.154628e+03	6.063272e+02	4.510977e+00	1.765466e+03
std	1.183514e+03	1.498567e+03	3.050773e+01	2.183257e+03
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.554000e+02	0.000000e+00	0.000000e+00	1.584000e+02
50%	6.864000e+02	0.000000e+00	0.000000e+00	7.893000e+02
75%	1.977000e+03	2.606500e-01	0.000000e+00	2.729000e+03
max	2.578000e+04	4.071000e+04	3.823000e+03	4.127010e+04

walkDistance

```
In [25]: # Plot the distribution of walkDistance
         plt.figure(figsize=(12,4))
         sns.distplot(train['walkDistance'], bins=10)
         plt.show()
```



In [26]:
```
# walkDistance anomalies
display(train[train['walkDistance'] >= 10000].shape)
train[train['walkDistance'] >= 10000].head(10)
```

```
(219, 38)
```

Out[26]:

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

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
23026	0.00	0	0	0	44	...	
34344	23.22	0	0	1	34	...	
49312	117.20	1	0	1	24	...	
68590	32.34	0	0	1	46	...	
94400	73.08	0	0	3	27	...	
125103	37.73	0	0	7	47	...	
136421	0.00	0	0	1	46	...	
136476	0.00	0	0	0	41	...	
154080	0.00	0	0	13	46	...	
154128	52.16	0	0	7	25	...	

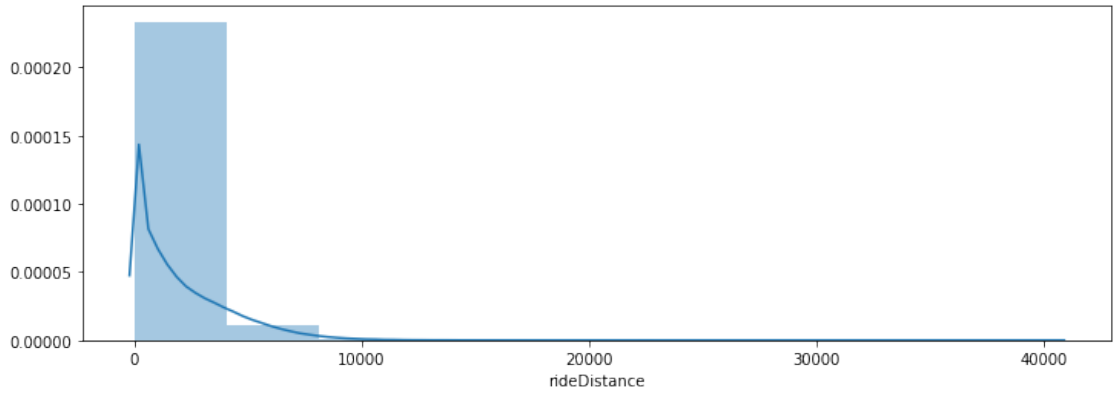
	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	maxPlaceNorm	\
23026	0.8163	99	0.00	0.0000	99.99	

```
34344         0.9474             65       0.00          31.3470         27.00
49312         0.9130             94       1.06         124.2320         49.82
68590         0.8333             96       0.00          33.6336         50.96
94400         0.8194             73       0.00          92.8116         92.71
125103        0.7340             95       0.00          39.6165         99.75
136421        0.6957             94       0.00           0.0000         49.82
136476        0.9333             91       0.00           0.0000         99.19
154080        0.8602             94       0.00           0.0000         99.64
154128        0.8936             95       1.05          54.7680         50.40

       matchDurationNorm  healsandboosts  totalDistance  killsWithoutMoving  \
23026            1925.06               1     13540.3032               False
34344            2668.95               4     10070.9073               False
49312            2323.52               5     12446.7588               False
68590            1909.44               2     12483.6200               False
94400            2293.62               4     11490.6300               False
125103           2054.85              11     12828.7978               False
136421           2091.38               2     12223.8100               False
136476           2028.49               3     14918.2000               False
154080           2038.38              16     12636.7000               False
154128           1927.80              11     10889.8614               False

        headshot_rate
23026             0.0
34344             0.0
49312             0.0
68590             0.0
94400             0.0
125103            0.0
136421            0.0
136476            0.0
154080            0.0
154128            0.0

[10 rows x 38 columns]
```

In [27]:
```
# Remove outliers
train.drop(train[train['walkDistance'] >= 10000].index, inplace=True)
```

rideDistance

In [28]:
```
# Plot the distribution of rideDistance
plt.figure(figsize=(12,4))
sns.distplot(train['rideDistance'], bins=10)
plt.show()
```



In [29]:
```
# rideDistance anomalies
display(train[train['rideDistance'] >= 20000].shape)
train[train['rideDistance'] >= 20000].head(10)
```

```
(150, 38)
```

Out[29]:

	Id	groupId	matchId	assists	boosts	\
28588	6260f7c49dc16f	b24589f02eedd7	6ebea3b4f55b4a	0	0	
63015	adb7dae4d0c10a	8ede98a241f30a	8b36eac66378e4	0	0	
70507	ca6fa339064d67	f7bb2e30c3461f	3bfd8d66edbeff	0	0	
72763	198e5894e68ff4	ccf47c82abb11f	d92bf8e696b61d	0	0	
95276	c3fabfce7589ae	15529e25aa4a74	d055504340e5f4	0	7	
140097	9944fbbea2b91e	18b4d5f4bb1906	d9d4a3e50cae75	1	0	
297186	88904c200175b6	012a61a01e146e	7a270c25e9b70c	0	1	
371098	f7071357f6b762	f3ee20821f4627	ac47c86bf385bf	0	0	
403647	c65da7b3fceef5	814d1b3736e276	ff9f570b555d48	0	2	
426708	149e224a2330ae	6d8cb80b3de8ff	f8b8e2643f60ee	0	2	

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
28588	99.20	0	0	1	30	...	
63015	0.00	0	0	0	55	...	
70507	100.00	0	0	0	26	...	
72763	0.00	0	0	0	46	...	
95276	778.20	0	1	2	2	...	
140097	12.55	0	0	0	53	...	
297186	0.00	0	0	1	47	...	
371098	72.92	1	0	0	45	...	
403647	0.00	0	0	3	54	...	
426708	0.00	0	0	2	43	...	

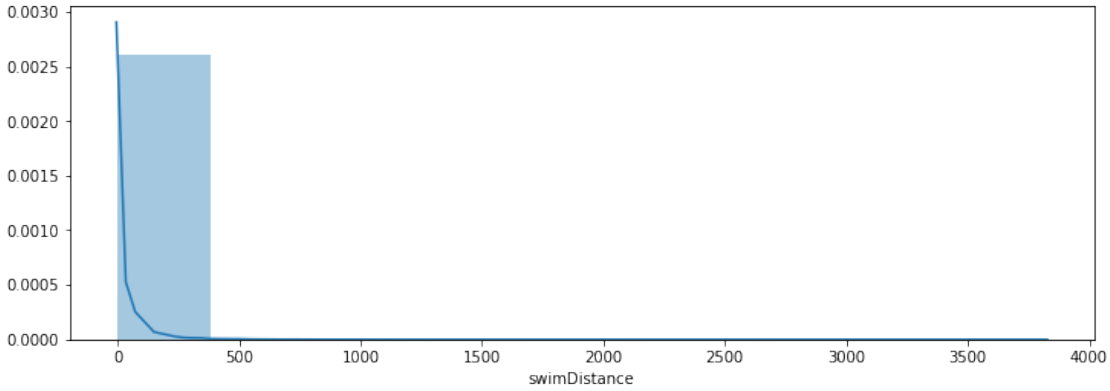
	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	maxPlaceNorm	\
28588	0.6421	96	1.04	103.1680	99.84	

```
63015         0.5376              94       0.00           0.0000          99.64
70507         0.8878              99       1.01         101.0000          99.99
72763         0.7917              97       0.00           0.0000          99.91
95276         0.9785              94       7.42         824.8920          99.64
140097        0.5000              89       0.00          13.9305          98.79
297186        0.7447              96       0.00           0.0000          49.92
371098        0.6875              96       0.00          75.8368          50.96
403647        0.6739              94       0.00           0.0000          49.82
426708        0.8171              83       0.00           0.0000          97.11

        matchDurationNorm  healsandboosts  totalDistance  killsWithoutMoving  \
28588             1969.76               1       26306.60               False
63015             2004.46               0       22065.40               False
70507             1947.28               0       28917.50               False
72763             1861.21               0       21197.20               False
95276             1986.44               9       26733.20               False
140097            2107.89               0       21293.23               False
297186            1995.76               2       29267.30               False
371098            1953.12               0       21942.10               False
403647            1930.26               5       21198.20               False
426708            2348.19               4       32362.10               False

        headshot_rate
28588        0.000000
63015        0.000000
70507        0.000000
72763        0.000000
95276        0.142857
140097       0.000000
297186       0.000000
371098       0.000000
403647       0.000000
426708       0.000000

[10 rows x 38 columns]
```

In [30]:
```
# Remove outliers
train.drop(train[train['rideDistance'] >= 20000].index, inplace=True)
```

swimDistance

In [31]:
```
# Plot the distribution of swimDistance
plt.figure(figsize=(12,4))
sns.distplot(train['swimDistance'], bins=10)
plt.show()
```



In [32]:
```
# Players who swam more than 2 km
train[train['swimDistance'] >= 2000]
```

Out[32]:

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

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
177973	78.12	1	0	1	47	...	
274258	53.32	0	0	16	39	...	
1005337	503.00	4	3	1	6	...	
1195818	352.30	3	1	6	4	...	
1227362	589.20	3	1	1	46	...	
1889163	0.00	0	0	0	47	...	
2065940	49.59	0	0	5	48	...	
2327586	474.40	2	0	0	7	...	
2784855	843.80	5	5	2	2	...	
3359439	0.00	0	0	0	77	...	
3513522	109.80	0	0	18	40	...	
4132225	0.00	0	0	0	83	...	

	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	\
177973	0.9592	98	0.00	79.6824	

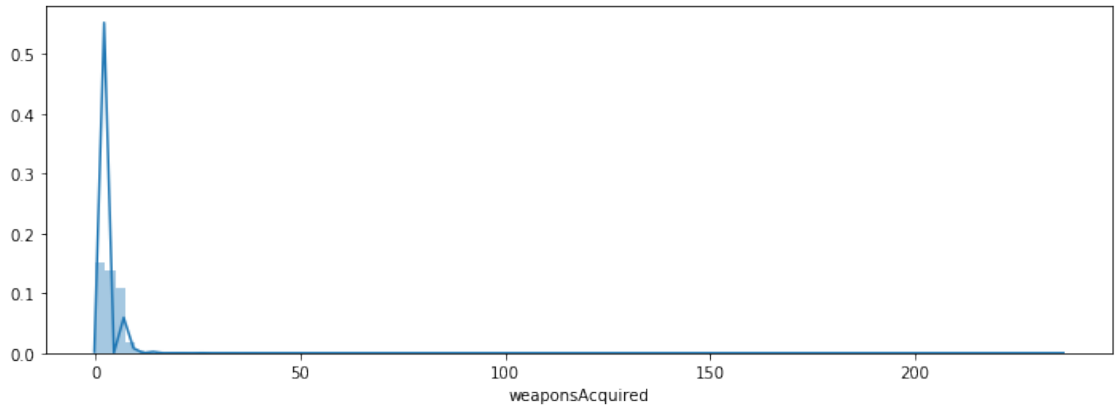
```
274258         0.9600              94       0.00          56.5192
1005337        1.0000              88       3.36         563.3600
1195818        1.0000              98       5.10         359.3460
1227362        1.0000              86       2.28         671.6880
1889163        0.5833              87       0.00           0.0000
2065940        0.8511              96       0.00          51.5736
2327586        1.0000              91       3.27         517.0960
2784855        0.9600              94       7.42         894.4280
3359439        0.2143              96       0.00           0.0000
3513522        0.9600              94       0.00         116.3880
4132225        0.0851              95       0.00           0.0000

         maxPlaceNorm matchDurationNorm  healsandboosts  totalDistance  \
177973          51.00           1426.98               6       3297.000
274258          27.56           2319.28              20      10113.000
1005337         29.12           2124.64               2      10740.000
1195818         49.98           1423.92              13       3083.100
1227362         13.68           1163.94               2       4818.300
1889163         28.25           1567.31               0       5314.000
2065940         49.92           1434.16               8       9899.000
2327586         29.43           1318.90               1       2394.546
2784855         27.56           2319.28               6       9926.000
3359439         30.16           1426.88               0       4088.000
3513522         27.56           2319.28              22       9809.000
4132225         50.40           1470.00               0       3916.000

         killsWithoutMoving  headshot_rate
177973                False       0.000000
274258                False       0.000000
1005337               False       1.000000
1195818               False       0.200000
1227362               False       0.500000
1889163               False       0.000000
2065940               False       0.000000
2327586               False       0.000000
2784855               False       0.714286
3359439               False       0.000000
3513522               False       0.000000
4132225               False       0.000000

[12 rows x 38 columns]
```

```
In [33]: # Remove outliers
         train.drop(train[train['swimDistance'] >= 2000].index, inplace=True)
```

Anomalies in supplies (weaponsAcquired)

Most people acquire between 0 and 10 weapons in a game, but you also see some people acquire more than 80 weapons! Let's check these guys out.

In [34]:
```
# Plot the distribution of weaponsAcquired
plt.figure(figsize=(12,4))
sns.distplot(train['weaponsAcquired'], bins=100)
plt.show()
```



In [35]:
```
# Players who acquired more than 80 weapons
display(train[train['weaponsAcquired'] >= 80].shape)
train[train['weaponsAcquired'] >= 80].head()
```

```
(19, 38)
```

Out[35]:

	Id	groupId	matchId	assists	boosts	\
233643	7c8c83f5f97d0f	b33b210a52a2f8	2e8a0917a71c43	0	0	
588387	c58e3e0c2ba678	3d3e6100c07ff0	d04dbb98249f76	0	1	
1437471	8f0c855d23e4cd	679c3316056de8	fbaf1b3ae1d884	1	0	
1449293	db54cf45b9ed1c	898fccaeeb041d	484b4ae51fe80f	0	0	
1592744	634a224c53444e	75fa7591d1538c	f900de1ec39fa5	9	0	

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
233643	67.11	0	0	0	44	...	
588387	175.30	1	0	2	48	...	
1437471	100.00	0	0	0	24	...	
1449293	0.00	0	0	0	54	...	
1592744	1726.00	0	3	0	9	...	

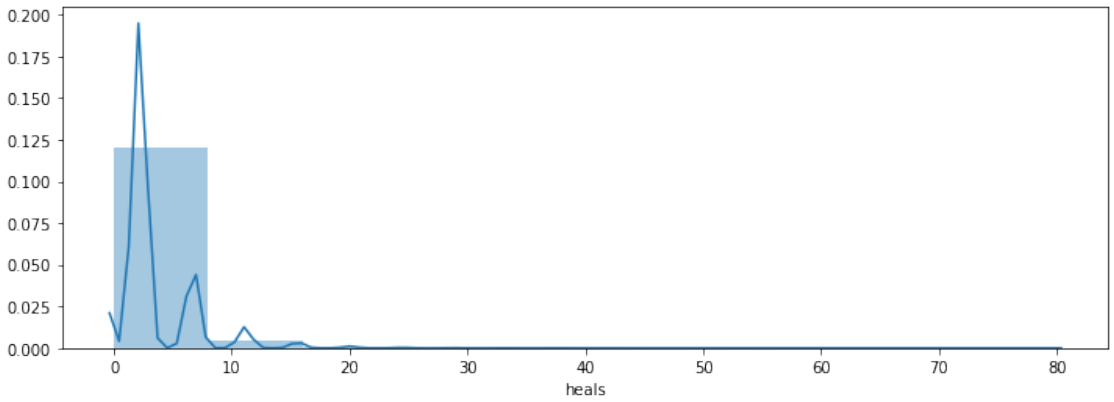
	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	\
233643	0.7111	91	0.00	73.1499	
588387	0.7500	98	0.00	178.8060	
1437471	0.8929	97	1.03	103.0000	
1449293	0.5600	80	0.00	0.0000	
1592744	0.2000	11	43.47	3262.1400	

```
         maxPlaceNorm matchDurationNorm  healsandboosts  totalDistance  \
233643          50.14           2072.09               0        3187.00
588387          29.58           1399.44               3        1687.00
1437471         29.87           1895.20               0        5299.21
1449293         31.20           1596.00               0         653.10
1592744         20.79           3398.22               0        2888.80

         killsWithoutMoving  headshot_rate
233643                False       0.000000
588387                False       0.000000
1437471               False       0.000000
1449293               False       0.000000
1592744               False       0.130435

[5 rows x 38 columns]
```

```
In [36]: # Remove outliers
         train.drop(train[train['weaponsAcquired'] >= 80].index, inplace=True)
```

Anomalies in supplies part 2 (heals)
Most players us 5 healing items or less. We can again recognize some weird anomalies

```
In [37]: # Distribution of heals
         plt.figure(figsize=(12,4))
         sns.distplot(train['heals'], bins=10)
         plt.show()
```




```
In [38]: # 40 or more healing items used
         display(train[train['heals'] >= 40].shape)
         train[train['heals'] >= 40].head(10)
```

```
(135, 38)
```

Out[38]:

	Id	groupId	matchId	assists	boosts	\
18405	63ab976895d860	927eeba5614c4f	69473402649f11	0	2	
54463	069ddee7c9d26a	58ab5a1ce8e06f	942416b6caf21e	1	4	
126439	c45bd6917146e2	81ab9f863957cb	4335664c6716fa	0	2	
259351	86910c38335c2f	2738398928d28c	7d2911e944bfaa	0	10	
268747	a007734fbc6ebf	5bf702dfa1e5d4	ad6b5669d33a2c	0	5	
269098	a0891dbc2950ea	dde848d90491ba	b4fd3348551b73	0	2	
284195	91a2fb00455eb3	f639b09774c5b1	65b73c71653822	0	3	
300204	1f4f2efc86bfcb	3d668492d1fca9	d3638466a43d38	0	6	
349908	7725ad71ad2ff7	4b2a7cf86d1546	cfa2775c9ef944	3	0	
375156	d64866c78ebcb0	aa0f089ae6430c	4dbc4ebba33ec6	0	7	

	damageDealt	DBNOs	headshotKills	heals	killPlace	...	\
18405	0.0	0	0	47	43	...	
54463	182.0	0	1	43	21	...	
126439	0.0	0	0	52	49	...	
259351	0.0	0	0	42	45	...	
268747	0.0	0	0	48	43	...	
269098	0.0	0	0	42	44	...	
284195	123.0	0	0	40	52	...	
300204	175.0	2	1	47	25	...	
349908	2348.0	0	8	41	9	...	
375156	278.5	3	1	44	3	...	

	winPlacePerc	playersJoined	killsNorm	damageDealtNorm	maxPlaceNorm	\
18405	0.9368	96	0.00	0.00	99.84	
54463	0.9615	93	1.07	194.74	28.89	
126439	0.8333	97	0.00	0.00	99.91	
259351	0.8646	97	0.00	0.00	99.91	
268747	0.8370	93	0.00	0.00	99.51	
269098	0.9259	97	0.00	0.00	28.84	
284195	0.8276	99	0.00	124.23	30.30	
300204	0.9355	95	1.05	183.75	33.60	
349908	0.8889	41	42.93	3733.32	58.83	
375156	0.9630	94	4.24	295.21	29.68	

	matchDurationNorm	healsandboosts	totalDistance	killsWithoutMoving	\
18405	1868.88	49	6854.000	False	
54463	1639.24	47	3083.400	False	
126439	1415.22	54	1343.443	False	
259351	1822.07	52	7444.000	False	
268747	2009.46	53	5816.000	False	
269098	1333.85	44	2439.000	False	
284195	1984.65	43	4848.000	False	
300204	1425.90	53	3415.600	False	
349908	2857.23	41	268.800	False	
375156	1915.42	51	4927.000	False	

```
        headshot_rate
18405        0.000000
54463        1.000000
126439       0.000000
259351       0.000000
268747       0.000000
269098       0.000000
284195       0.000000
300204       1.000000
349908       0.296296
375156       0.250000

[10 rows x 38 columns]
```

```
In [39]: # Remove outliers
         train.drop(train[train['heals'] >= 40].index, inplace=True)
```

4 Categorical Variables

We will one hot encode the 'matchType' feature to use it in our Random Forest model.

```
In [41]: print('There are {} different Match types in the dataset.'.format(train['matchType'].nu
```

```
There are 16 different Match types in the dataset.
```

```
In [42]: # 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[42]:    matchType_crashfpp  matchType_crashtpp  matchType_duo  matchType_duo-fpp  \
         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

            matchType_flarefpp  matchType_flaretpp  matchType_normal-duo  \
         0                   0                   0                     0
         1                   0                   0                     0
         2                   0                   0                     0
         3                   0                   0                     0
         4                   0                   0                     0

            matchType_normal-duo-fpp  matchType_normal-solo  matchType_normal-solo-fpp  \
```

```
   0                        0                        0                        0
   1                        0                        0                        0
   2                        0                        0                        0
   3                        0                        0                        0
   4                        0                        0                        0

   matchType_normal-squad  matchType_normal-squad-fpp  matchType_solo  \
0                       0                           0               0
1                       0                           0               0
2                       0                           0               0
3                       0                           0               0
4                       0                           0               0

   matchType_solo-fpp  matchType_squad  matchType_squad-fpp
0                   0                0                    1
1                   0                0                    1
2                   0                0                    0
3                   0                0                    1
4                   1                0                    0
```

There are a lot of groupId's and matchId's so one-hot encoding them is computational suicide. We will turn them into category codes. That way we can still benefit from correlations between groups and matches in our Random Forest algorithm.

```
In [43]: # 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[43]:    groupId_cat  matchId_cat
         0       613591        30085
         1       827582        32751
         2       843273         3143
         3      1340072        45260
         4      1757338        20531
```

```
In [44]: # Drop Id column, because it probably won't be useful for our Machine Learning algorith
         # because the test set contains different Id's
         train.drop(columns = ['Id'], inplace=True)
```

5 Preparation for Machine Learning

5.1 Sampling

We will take a sample of 500000 rows from our training set for easy debugging and exploration.

```
In [45]: # Take sample for debugging and exploration
         sample = 500000
         df_sample = train.sample(sample)
```

5.2 Split target variable, validation data, etc.

```
In [46]: # Split sample into training data and target variable
         df = df_sample.drop(columns = ['winPlacePerc']) #all columns except target
         y = df_sample['winPlacePerc'] # Only target variable
```

```
In [47]: # Function for splitting training and validation data
         def split_vals(a, n : int):
             return a[:n].copy(), a[n:].copy()
         val_perc = 0.12 # % to use for validation set
         n_valid = int(val_perc * sample)
         n_trn = len(df)-n_valid
         # Split data
         raw_train, raw_valid = split_vals(df_sample, n_trn)
         X_train, X_valid = split_vals(df, n_trn)
         y_train, y_valid = split_vals(y, 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:  (440000, 51) Sample target shape:  (440000,) Sample validation shape:  (600
```

5.3 Set metrics (MSE)

MSE is the metric that is used for this competition. The scikit-learn library already programmed this metric for us so we don't have to implement it from scratch.

```
In [48]: # Metric used for the PUBG competition (Mean Absolute Error (MAE))
         from sklearn.metrics import mean_squared_error

         # 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 = ['mse train: ', mean_squared_error(m.predict(X_train), y_train),
                    'mse val: ', mean_squared_error(m.predict(X_valid), y_valid)]
             if hasattr(m, 'oob_score_'): res.append(m.oob_score_)
             print(res)
```

5.4 First basic Random Forest Model

```
In [49]: # Train basic model
         m1 = RandomForestRegressor(n_estimators=40, min_samples_leaf=3, max_features='sqrt',
                                    n_jobs=-1)
         m1.fit(X_train, y_train)
         print_score(m1)
```

```
['mse train: ', 0.003154876980091164, 'mse val: ', 0.007871685441863036]
```

6 Feature Importance

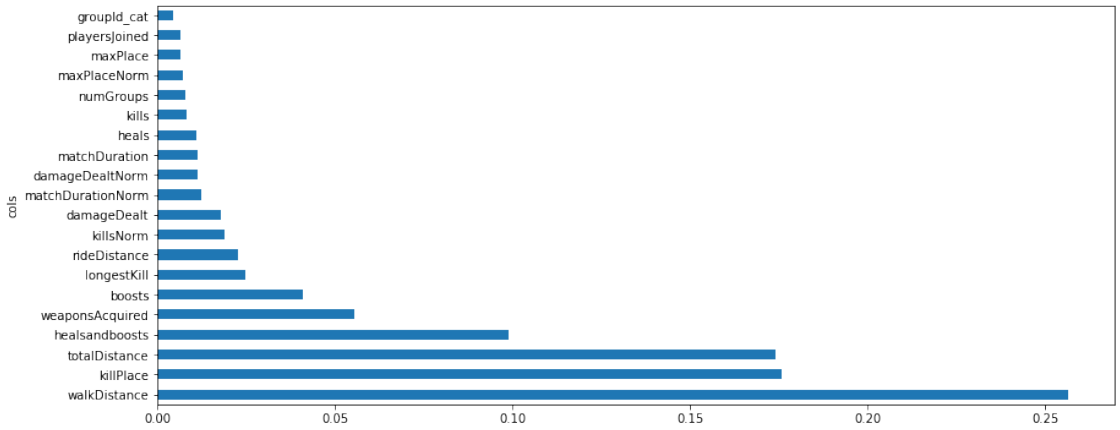
The fastai library gives us an easy way to analyze feature importances from a random forest algorithm with just one line of code!

```
In [50]: # What are the most predictive features according to our basic random forest model
         fi = rf_feat_importance(m1, df); fi[:10]
```

```
Out[50]:
                    cols       imp
21       walkDistance  0.256609
6           killPlace  0.175831
30     totalDistance  0.174133
29     healsandboosts  0.098687
22   weaponsAcquired  0.055212
1              boosts  0.040988
10       longestKill  0.024760
16      rideDistance  0.022468
25         killsNorm  0.018716
2        damageDealt  0.017939
```

```
In [51]: # Plot a feature importance graph for the 20 most important features
         plot1 = fi[:20].plot('cols', 'imp', figsize=(14,6), legend=False, kind = 'barh')
         plot1
```

```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x7fce5d09aa58>
```



```
In [52]: # Use this code if you want to save the figure
         #fig = plot1.get_figure()
         #fig.savefig("Feature_importances(AllFeatures).png")
```

```
In [53]: # Keep only significant features
         to_keep = fi[fi.imp>0.005].cols
         print('Significant features: ', len(to_keep))
         to_keep
```

```
Significant features:  19
```

```
Out[53]: 21         walkDistance
         6             killPlace
         30        totalDistance
         29       healsandboosts
         22      weaponsAcquired
         1                boosts
         10          longestKill
         16         rideDistance
         25            killsNorm
         2           damageDealt
         28    matchDurationNorm
         26      damageDealtNorm
         11        matchDuration
         5                 heals
         8                 kills
         13            numGroups
         27         maxPlaceNorm
         12             maxPlace
         24        playersJoined
         Name: cols, dtype: object
```

```
In [54]: # Make a DataFrame with only significant features
         df_keep = df[to_keep].copy()
         X_train, X_valid = split_vals(df_keep, n_trn)
```

6.1 Second Random Forest Model

This time we use only the top features to train a random forest model. This often improves results a little bit.

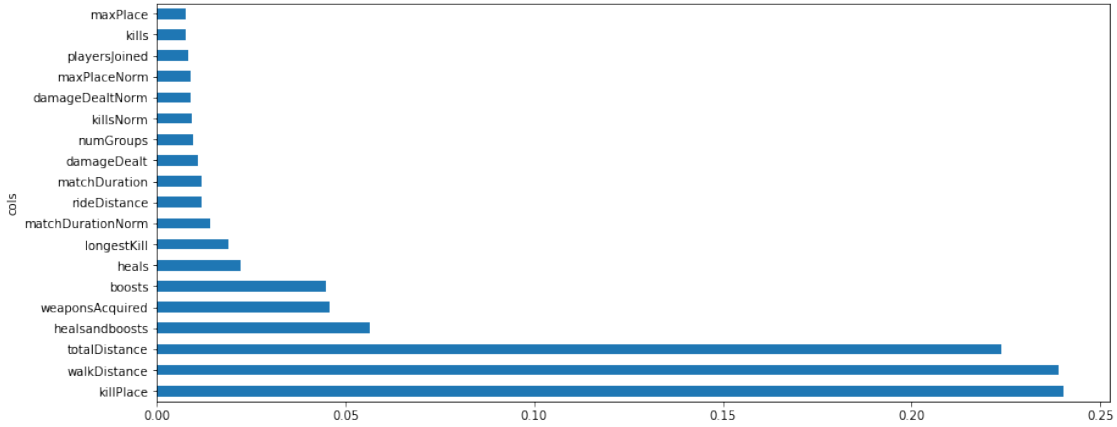
```
In [55]: # Train model on top features
         m2 = RandomForestRegressor(n_estimators=80, min_samples_leaf=3, max_features='sqrt',
                                    n_jobs=-1)
         m2.fit(X_train, y_train)
         print_score(m2)
```

```
['mse train: ', 0.0028149753464996267, 'mse val: ', 0.0073181510066403005]
```

Feature importance for top features

```
In [56]: # Get feature importances of our top features
         fi_to_keep = rf_feat_importance(m2, df_keep)
         plot2 = fi_to_keep.plot('cols', 'imp', figsize=(14,6), legend=False, kind = 'barh')
         plot2

         # Use this code if you want to save the figure
         #fig = plot2.get_figure()
         #fig.savefig("Feature_importances(TopFeatures).png")
```

```
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x7fce41ac54e0>
```

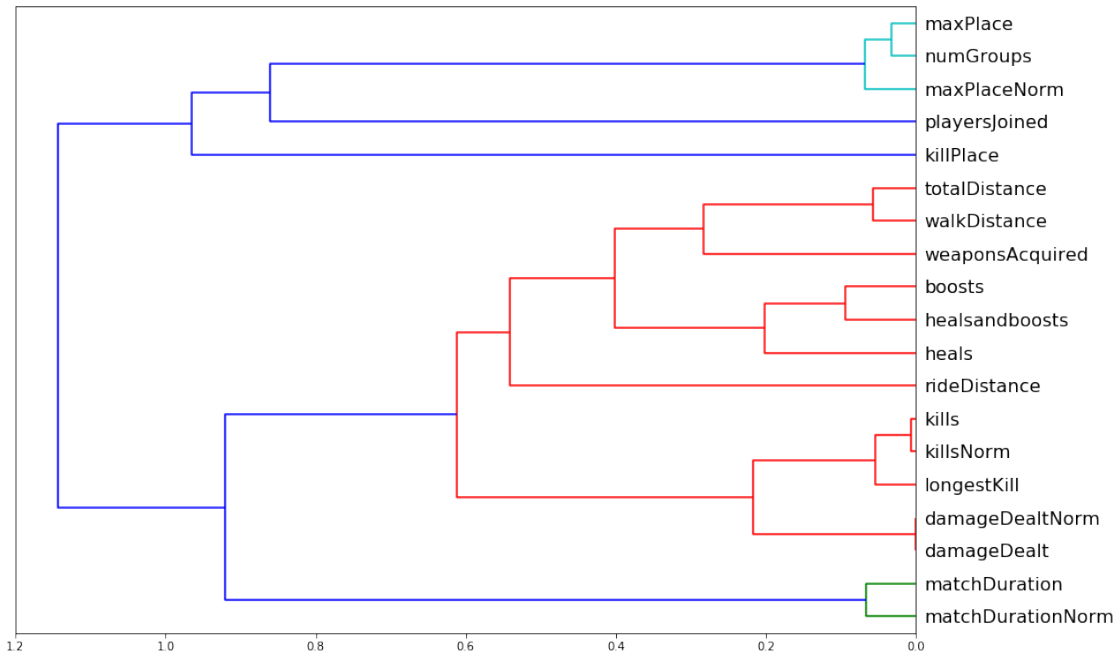


6.2 Correlations

Dendrogram (to view correlation of features)

```
In [57]: # Create a Dendrogram to view highly correlated features
         corr = np.round(scipy.stats.spearmanr(df_keep).correlation, 4)
         corr_condensed = hc.distance.squareform(1-corr)
         z = hc.linkage(corr_condensed, method='average')
         fig = plt.figure(figsize=(14,10))
         dendrogram = hc.dendrogram(z, labels=df_keep.columns, orientation='left', leaf_font_siz
         plt.plot()
```

```
Out[57]: []
```



7 Final Random Forest Model

In [63]:
```
# Prepare data
val_perc_full = 0.12 # % to use for validation set
n_valid_full = int(val_perc_full * len(train))
n_trn_full = len(train)-n_valid_full
df_full = train.drop(columns = ['winPlacePerc']) # all columns except target
y = train['winPlacePerc'] # target variable
df_full = df_full[to_keep] # Keep only relevant features
X_train, X_valid = split_vals(df_full, n_trn_full)
y_train, y_valid = split_vals(y, n_trn_full)

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

```
Sample train shape:  (3911403, 19) Sample target shape:  (3911403,) Sample validation shape:  (5
```

In [64]:
```
# Train final model
# You should get better results by increasing n_estimators
# and by playing around with the parameters
m3 = RandomForestRegressor(n_estimators=200, min_samples_leaf=3, max_features=0.5,
                           n_jobs=-1)
```

```
        m3.fit(X_train, y_train)
        print_score(m3)
```

```
['mse train: ', 0.0020285283698793792, 'mse val: ', 0.006280851841971277]
```