PUBG Finish Placement Prediction

July 19, 2021

Import Packages

[1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
sns.set

pd.set_option('display.max_columns', 500)
```

Load Data

[2]:
```
raw_training_data = pd.read_csv(r'C:\Users\amind\Downloads\train_V2.csv')
raw_test_data = pd.read_csv(r'C:\Users\amind\Downloads\test_V2.csv')


# Reference: memory usage reduction code from https://www.kaggle.com/nms2016145/
↪gbr-lightgbm-test
def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum()

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).
↪max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.
↪int16).max:
```

```
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.
↪int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.
↪int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.
↪float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
↪float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum()

    return df

raw_training_data = reduce_mem_usage(raw_training_data)
raw_test_data = reduce_mem_usage(raw_test_data)


training_data = raw_training_data
test_data = raw_test_data
```

Explore Data

[3]:
```
# Split Numerical and Categorical Variables
numerical_data = training_data[['assists', 'boosts', 'damageDealt', 'DBNOs',␣
↪'headshotKills', 'heals', 'kills',
       'killStreaks', 'longestKill', 'matchDuration','revives', 'rideDistance',␣
↪'roadKills', 'swimDistance',
       'teamKills', 'vehicleDestroys', 'walkDistance', 'weaponsAcquired']]

categorical_data = training_data[['killPlace', 'killPoints', 'matchType',␣
↪'maxPlace', 'numGroups', 'rankPoints',
        'winPoints', 'winPlacePerc']]
```

[4]:
```
numerical_data.describe().apply(lambda x: x.apply('{0:.3f}'.format))
```

[4]:
```
            assists       boosts  damageDealt        DBNOs headshotKills  \
count   4446966.000  4446966.000  4446966.000  4446966.000   4446966.000
```

```
mean        0.234        1.107         nan        0.658         0.227
std         0.589        1.716         nan        1.146         0.602
min         0.000        0.000       0.000        0.000         0.000
25%         0.000        0.000       0.000        0.000         0.000
50%         0.000        0.000      84.250        0.000         0.000
75%         0.000        2.000     186.000        1.000         0.000
max        22.000       33.000    6616.000       53.000        64.000

            heals        kills  killStreaks  longestKill matchDuration  \
count 4446966.000  4446966.000  4446966.000  4446966.000   4446966.000
mean        1.370        0.925        0.544          nan      1579.506
std         2.680        1.558        0.711          nan       258.740
min         0.000        0.000        0.000        0.000         9.000
25%         0.000        0.000        0.000        0.000      1367.000
50%         0.000        0.000        0.000        0.000      1438.000
75%         2.000        1.000        1.000       21.312      1851.000
max        80.000       72.000       20.000     1094.000      2237.000

          revives rideDistance    roadKills swimDistance    teamKills  \
count 4446966.000  4446966.000  4446966.000  4446966.000  4446966.000
mean        0.165          nan        0.003          nan        0.024
std         0.472          nan        0.073          nan        0.167
min         0.000        0.000        0.000        0.000        0.000
25%         0.000        0.000        0.000        0.000        0.000
50%         0.000        0.000        0.000        0.000        0.000
75%         0.000        0.191        0.000        0.000        0.000
max        39.000    40704.000       18.000     3824.000       12.000

      vehicleDestroys walkDistance weaponsAcquired
count     4446966.000  4446966.000     4446966.000
mean            0.008          nan           3.660
std             0.093          nan           2.457
min             0.000        0.000           0.000
25%             0.000      155.125           2.000
50%             0.000      685.500           3.000
75%             0.000     1976.000           5.000
max             5.000    25776.000         236.000
```

[5]:
```python
categorical_data.describe().apply(lambda x: x.apply('{0:.3f}'.format))
```

[5]:
```
        killPlace   killPoints     maxPlace    numGroups   rankPoints  \
count 4446966.000  4446966.000  4446966.000  4446966.000  4446966.000
mean       47.599      505.006       44.505       43.008      892.010
std        27.463      627.505       23.828       23.289      736.648
min         1.000        0.000        1.000        1.000       -1.000
25%        24.000        0.000       28.000       27.000       -1.000
50%        47.000        0.000       30.000       30.000     1443.000
```

```
75%         71.000     1172.000       49.000       47.000      1500.000
max        101.000     2170.000      100.000      100.000      5910.000

         winPoints winPlacePerc
count  4446966.000  4446965.000
mean       606.460          nan
std        739.700        0.000
min          0.000        0.000
25%          0.000        0.200
50%          0.000        0.458
75%       1495.000        0.741
max       2013.000        1.000
```

[6]:
```python
# Find how many observations for each match type
training_data['matchType'].value_counts()
```

[6]:
```
squad-fpp           1756186
duo-fpp              996691
squad                626526
solo-fpp             536762
duo                  313591
solo                 181943
normal-squad-fpp      17174
crashfpp               6287
normal-duo-fpp         5489
flaretpp               2505
normal-solo-fpp        1682
flarefpp                718
normal-squad            516
crashtpp                371
normal-solo             326
normal-duo              199
Name: matchType, dtype: int64
```

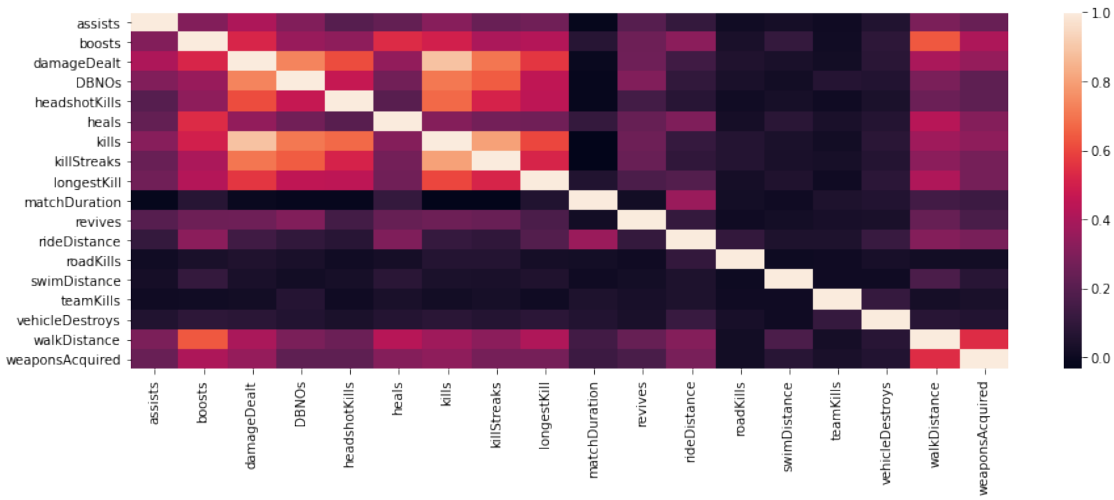
[7]:
```python
# Add "normal - ..." matchTypes to the larger 'matchType' options
training_data = training_data.replace(to_replace='normal-duo', value='duo')
training_data = training_data.replace(to_replace='normal-solo', value='solo')
training_data = training_data.replace(to_replace='normal-squad', value='squad')
training_data = training_data.replace(to_replace='normal-solo-fpp',␣
 ↪value='solo-fpp')
training_data = training_data.replace(to_replace='normal-duo-fpp',␣
 ↪value='duo-fpp')
training_data = training_data.replace(to_replace='normal-squad-fpp',␣
 ↪value='squad-fpp')

test_data = test_data.replace(to_replace='normal-duo', value='duo')
test_data = test_data.replace(to_replace='normal-solo', value='solo')
```

```python
test_data = test_data.replace(to_replace='normal-squad', value='squad')
test_data = test_data.replace(to_replace='normal-solo-fpp', value='solo-fpp')
test_data = test_data.replace(to_replace='normal-duo-fpp', value='duo-fpp')
test_data = test_data.replace(to_replace='normal-squad-fpp', value='squad-fpp')
```

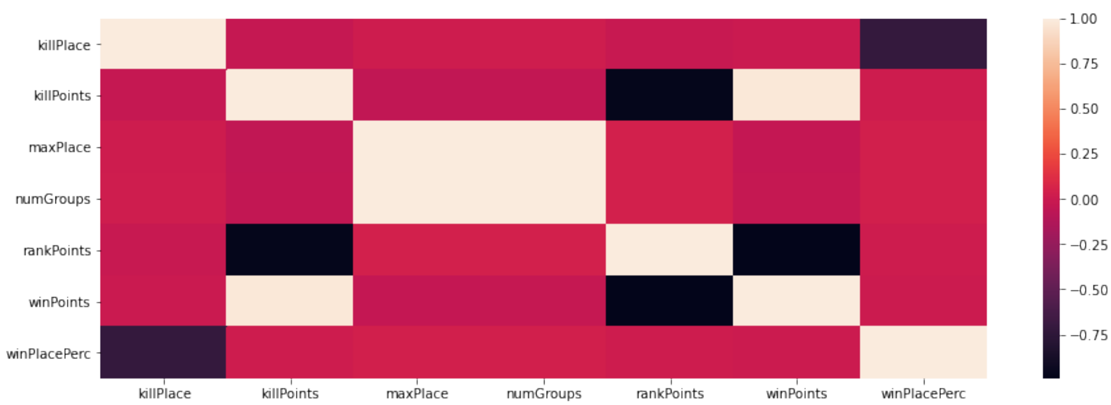
[8]:
```python
# Find correlation for numerical data
fig_dims = (15, 5)
fig, ax = plt.subplots(figsize=fig_dims)
sns.heatmap(ax=ax, data=numerical_data.corr())
```

[8]: <AxesSubplot:>



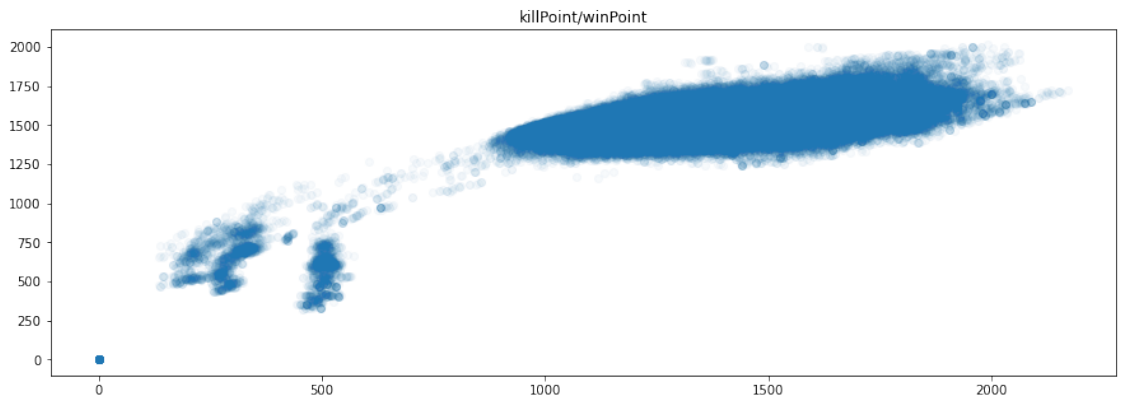
[9]:
```python
# Find correlation for categorical data
fig_dims = (15, 5)
fig, ax = plt.subplots(figsize=fig_dims)
sns.heatmap(ax=ax, data=categorical_data.corr())
```

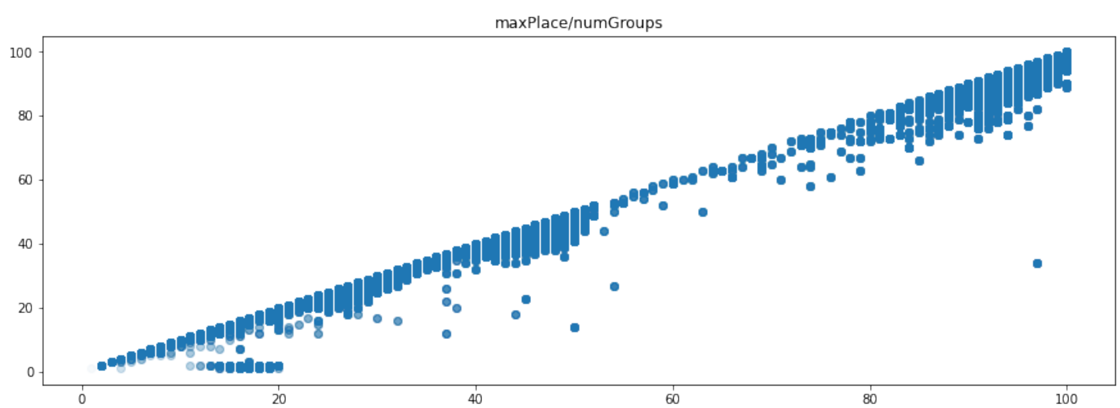
[9]: <AxesSubplot:>



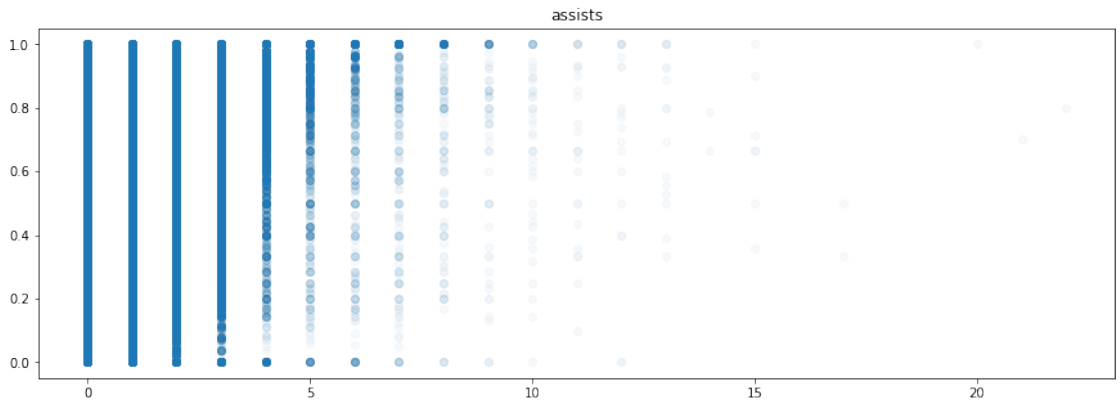
[10]:

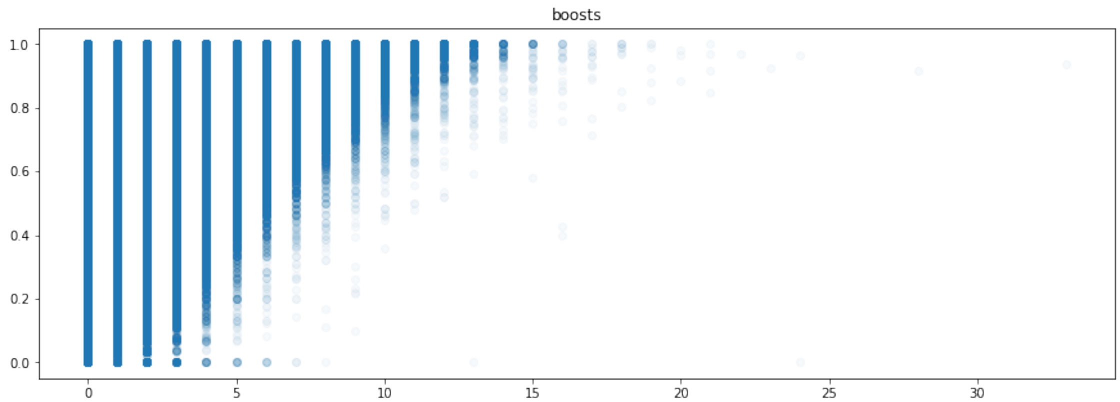
```python
# Look into high correlation between 'killPoints'/'winPoints' and 'maxPlace'/
↪'numGroups'
for i in range(1):
    plt.figure(figsize=(15,5))
    plt.scatter(x=training_data['killPoints'], y=training_data['winPoints'],␣
↪alpha=0.03)
    plt.title('killPoint/winPoint')
    plt.show()
    plt.figure(figsize=(15,5))
    plt.scatter(x=training_data['maxPlace'], y=training_data['numGroups'],␣
↪alpha=0.03)
    plt.title('maxPlace/numGroups')
    plt.show()
```

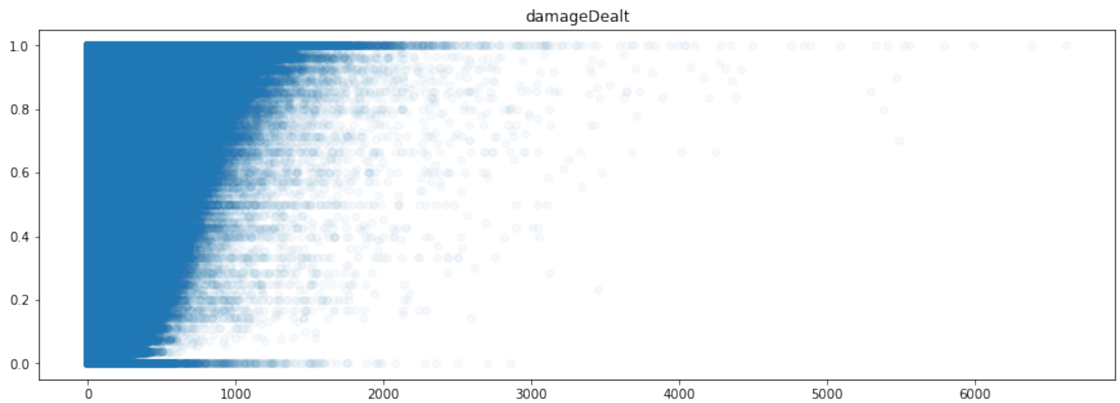


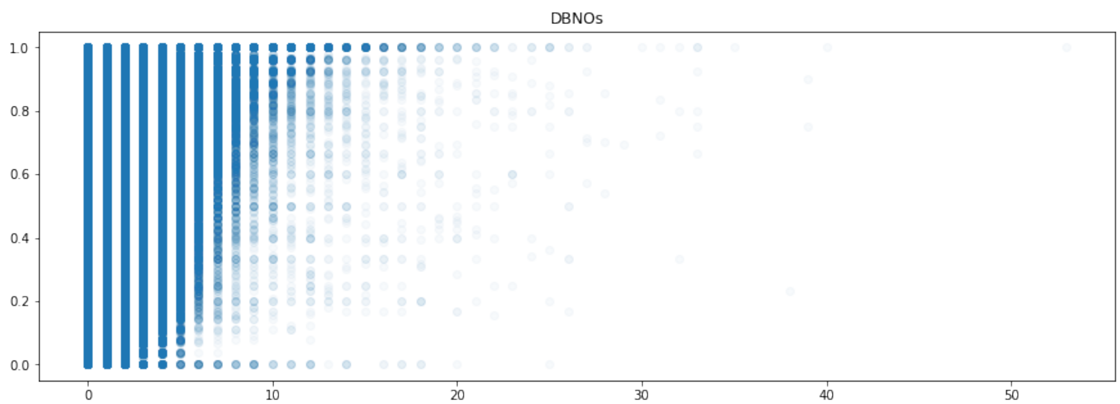


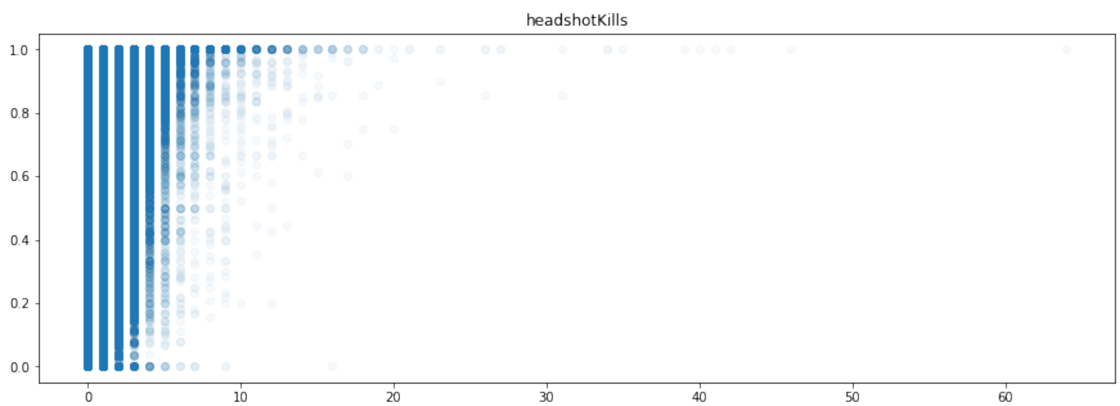
```
[11]: # Look into how 'winPlacePerc' varies with numerical inputs
for i in range(len(numerical_data.columns)):
    var_name = str(numerical_data.columns[i])
    plt.figure(figsize=(15, 5))
    plt.scatter(x=training_data[var_name], y=training_data['winPlacePerc'],␣
 ↪alpha=0.03)
    plt.title(var_name)
    plt.show()
```

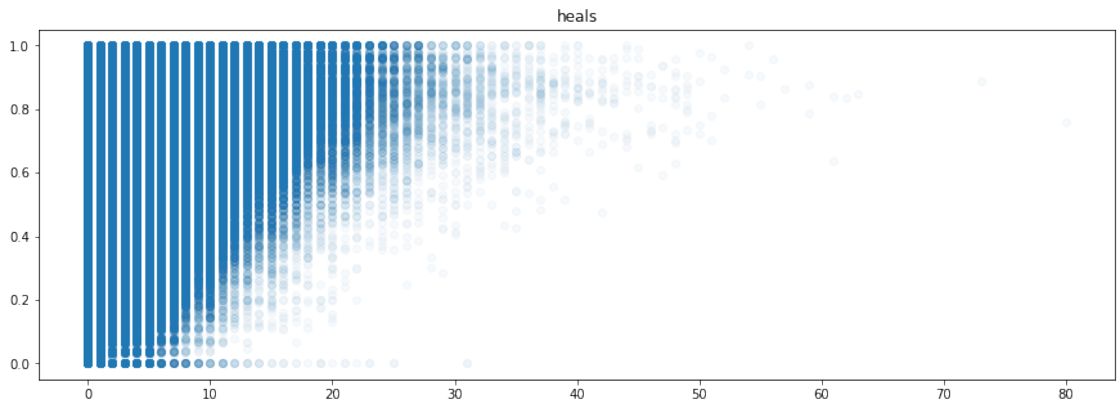


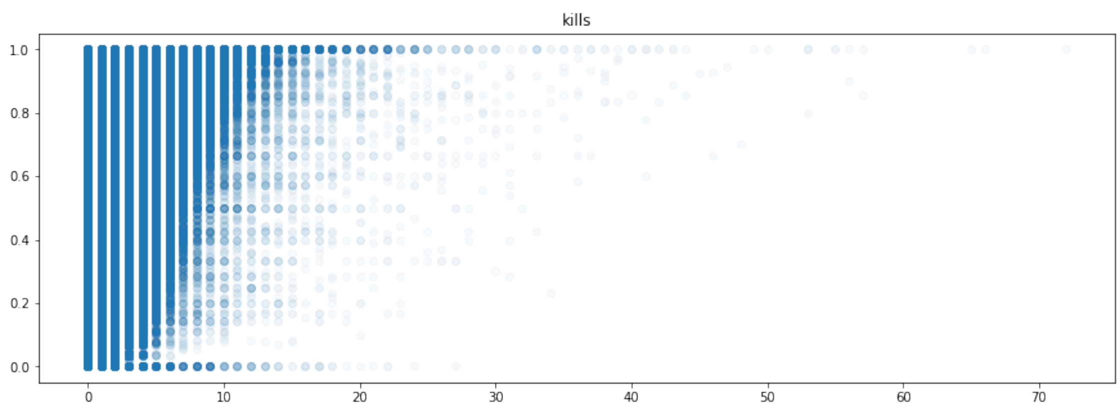


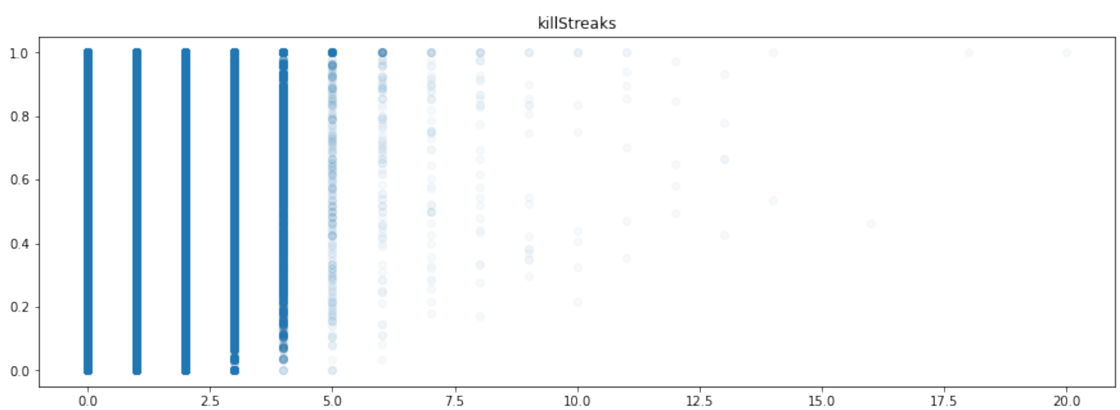

damageDealt
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000
6000

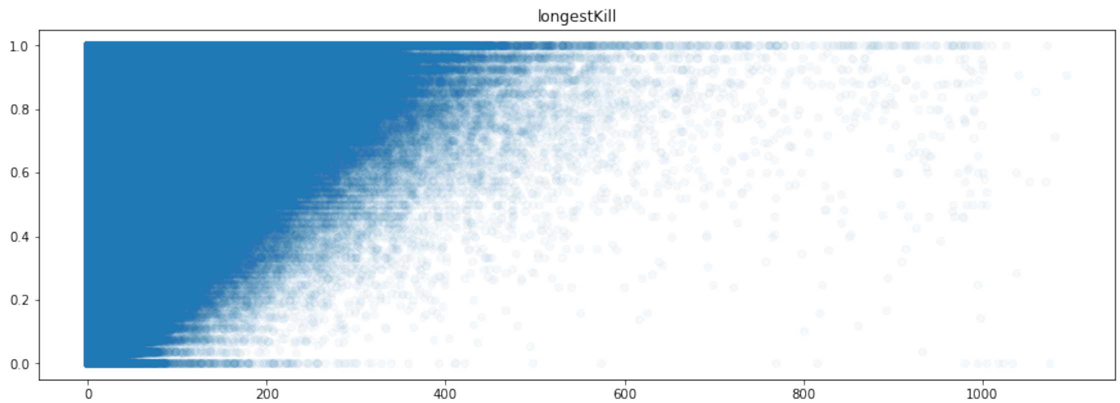

DBNOs
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

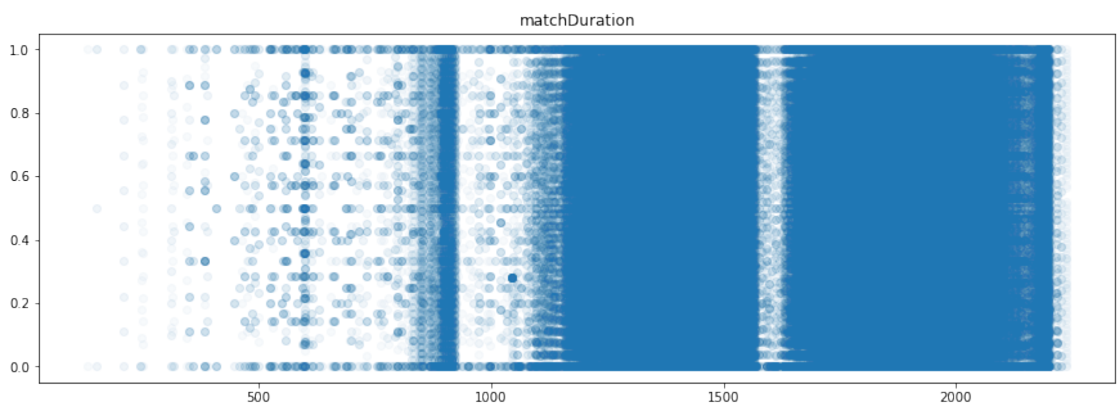

headshotKills
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60

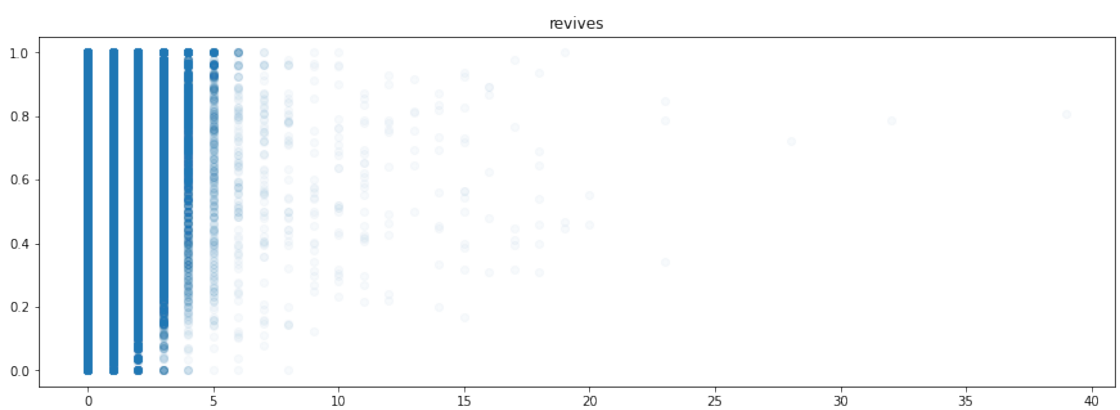

heals

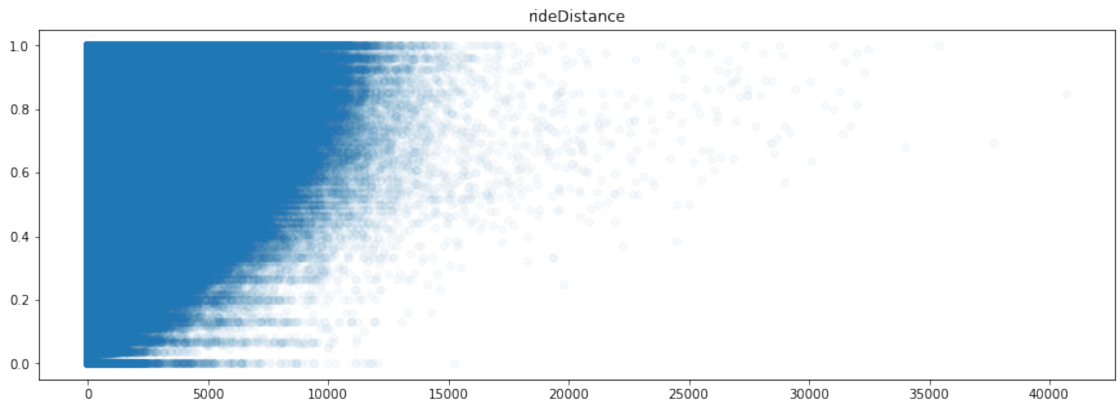

kills

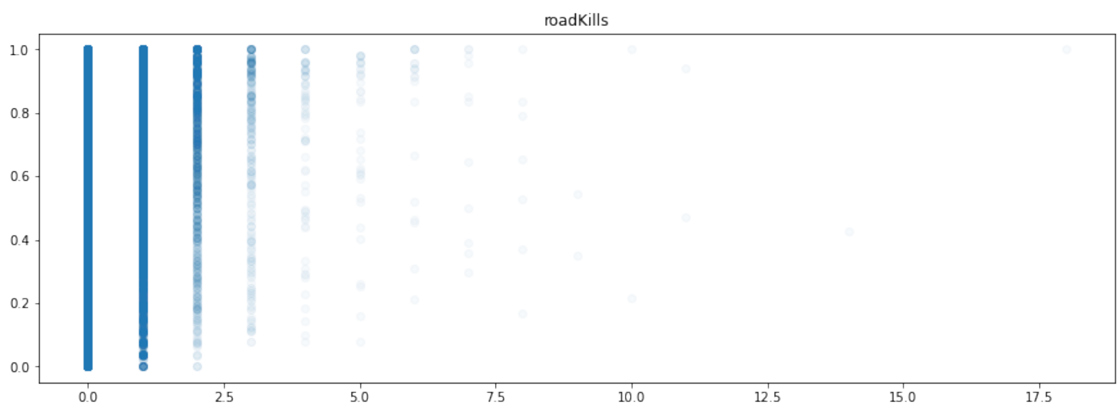

killStreaks

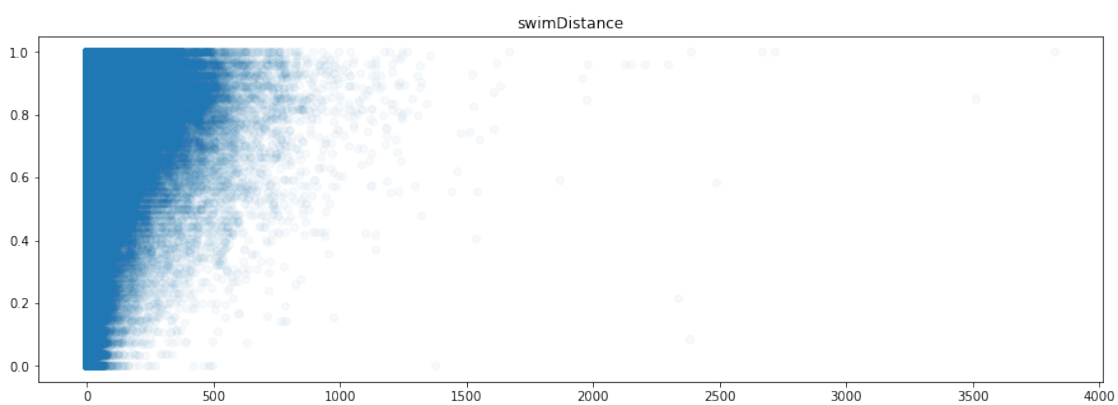

longestKill
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000

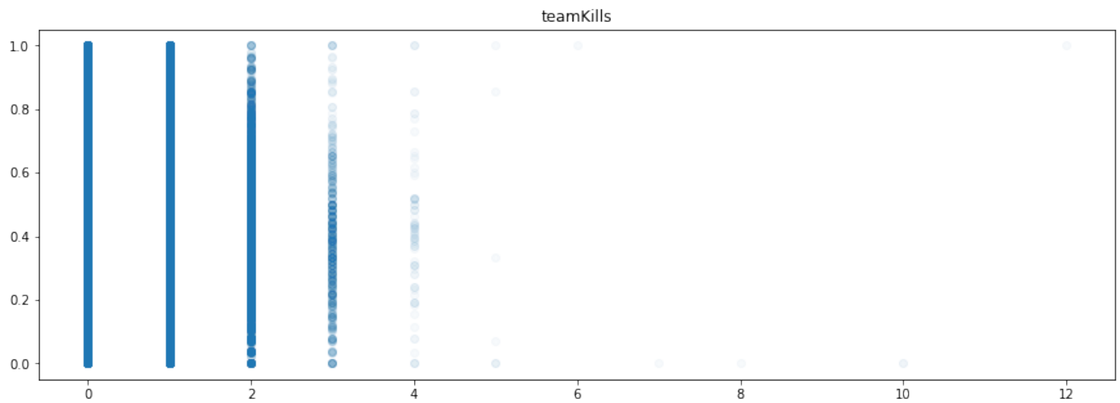

matchDuration
1.0
0.8
0.6
0.4
0.2
0.0
500
1000
1500
2000

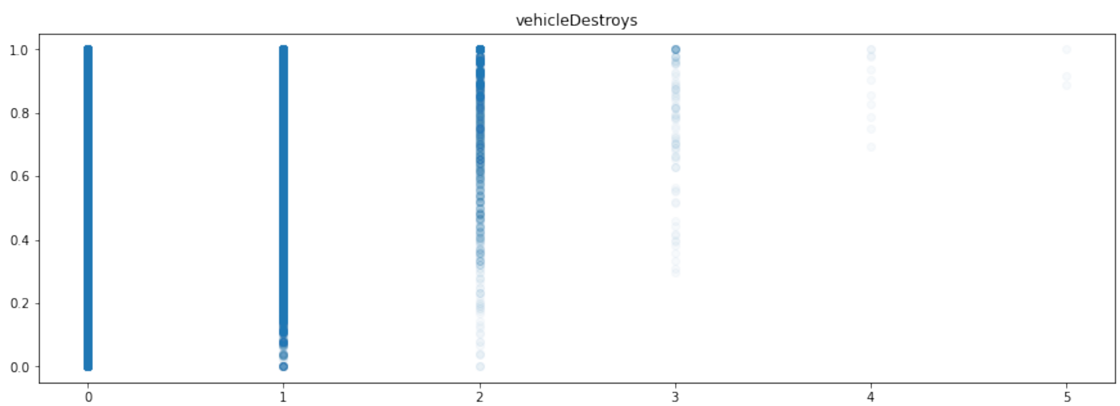

revives
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35
40

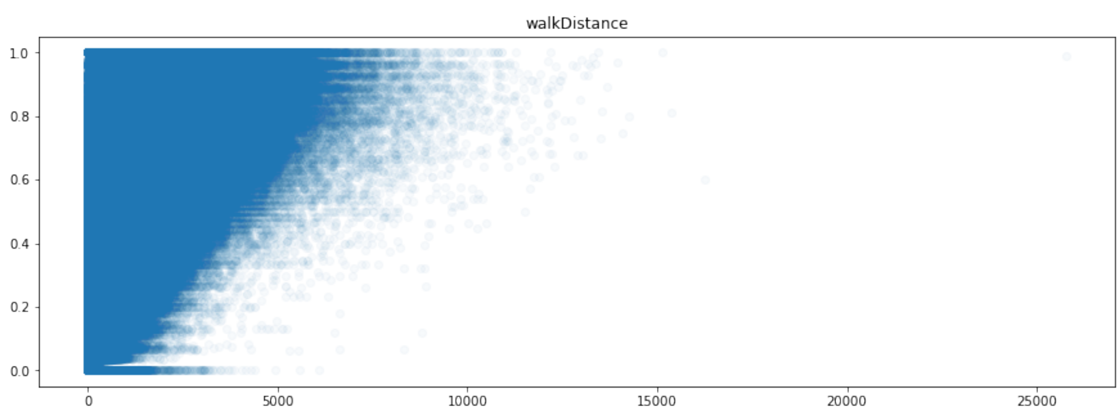

rideDistance
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
30000
35000
40000

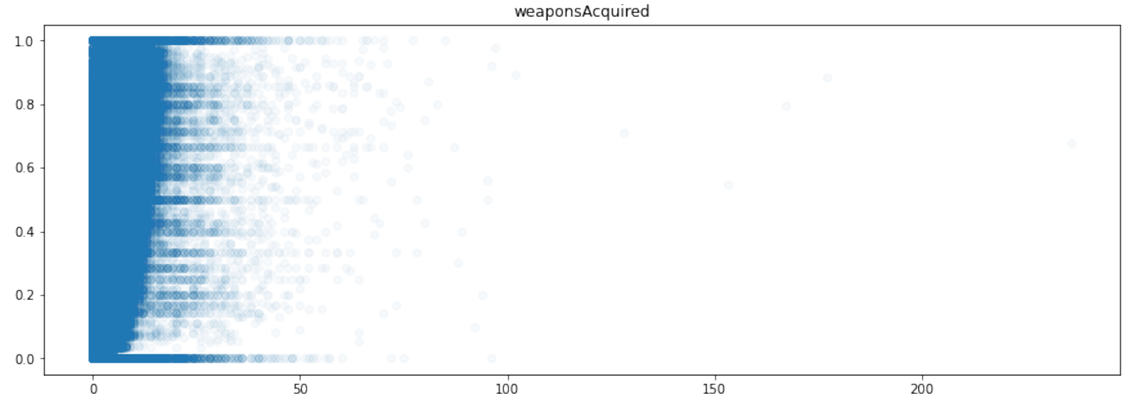

roadKills
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


swimDistance
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500
3000
3500
4000

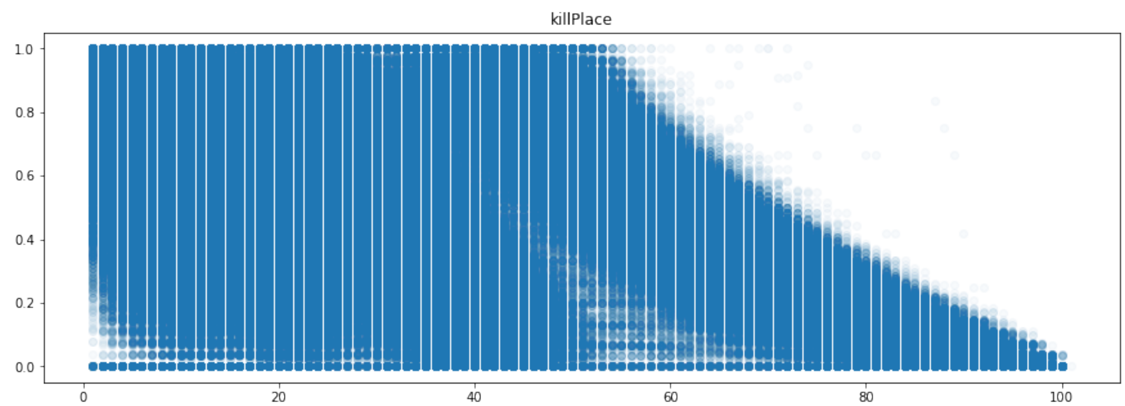

teamKills
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12

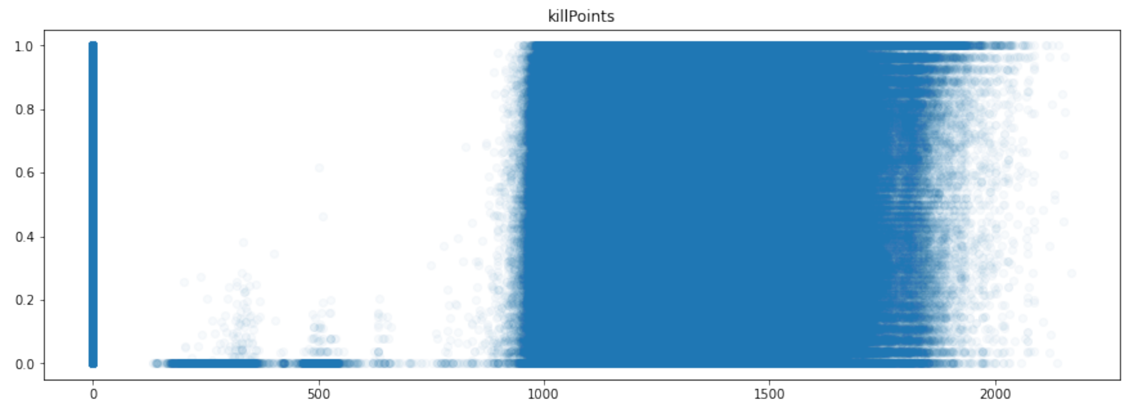

vehicleDestroys
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5

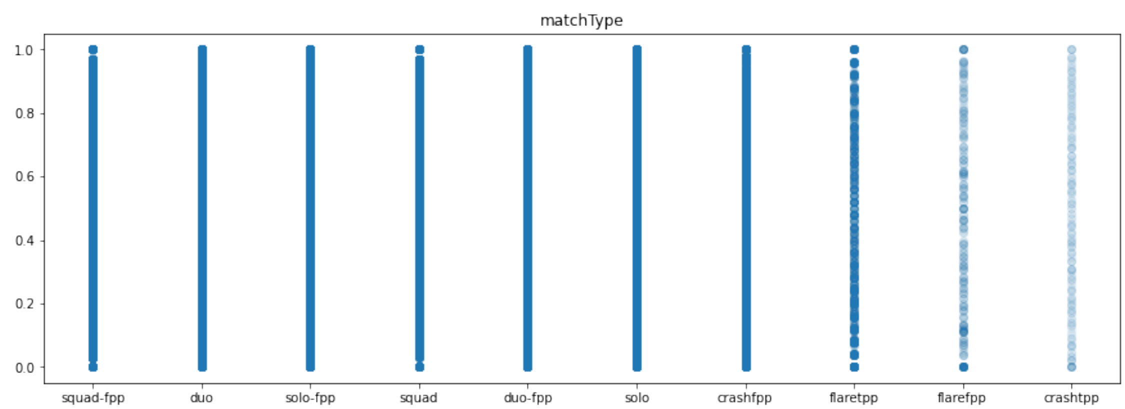

walkDistance
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000

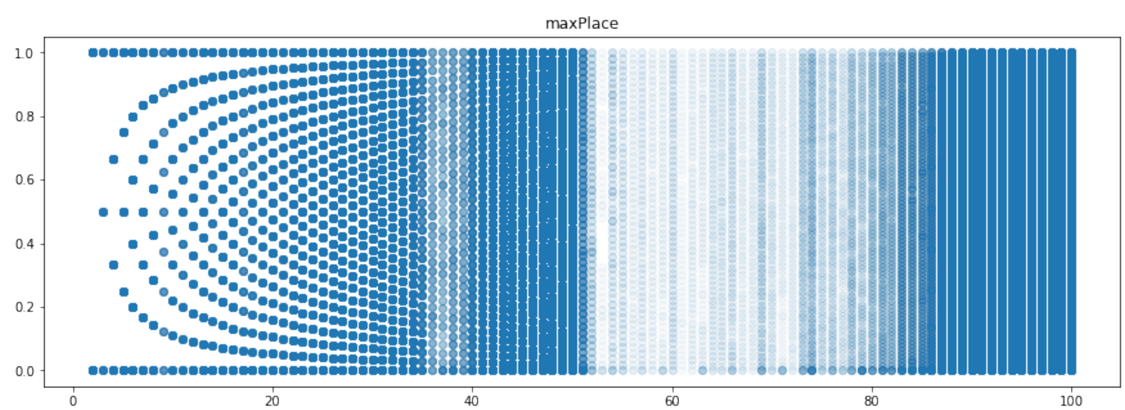


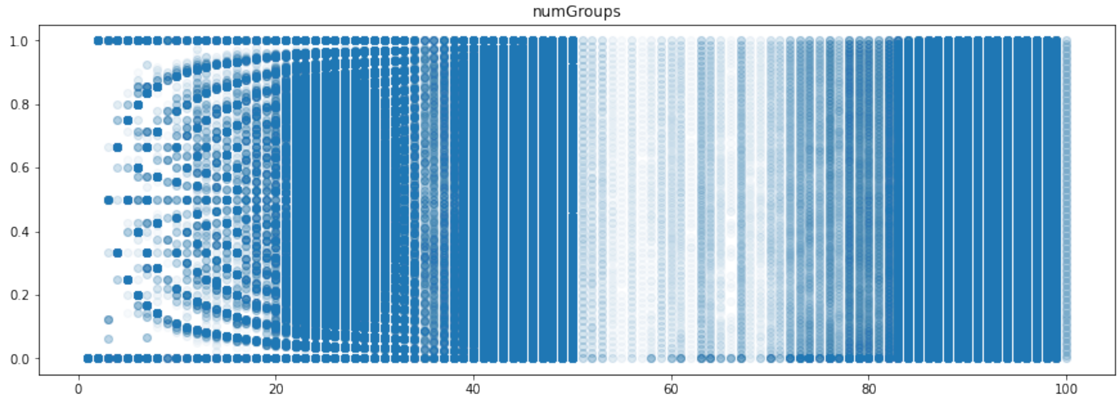
[12]:
```python
# Look into how 'winPlacePerc' varies with categorical inputs
for i in range(len(categorical_data.columns)):
    var_name = str(categorical_data.columns[i])
    plt.figure(figsize=(15, 5))
    plt.scatter(x=training_data[var_name], y=training_data['winPlacePerc'],␣
 ↪alpha=0.03)
    plt.title(var_name)
    plt.show()
```

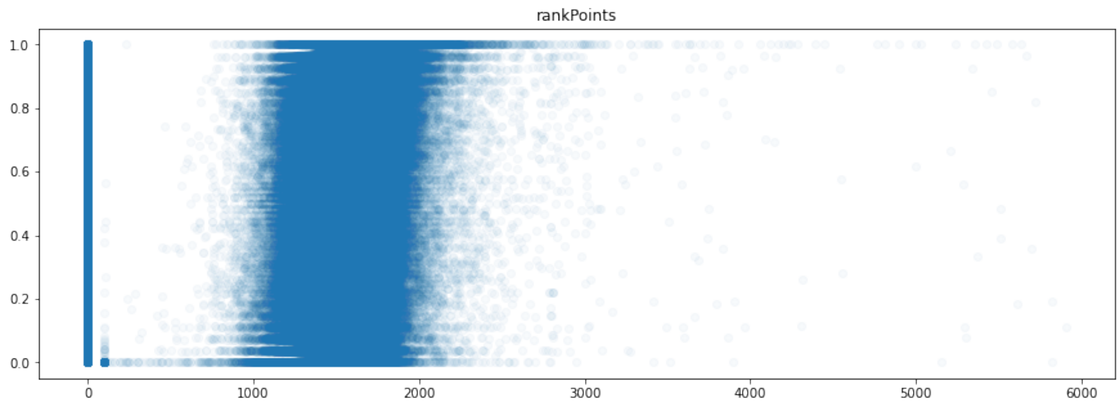


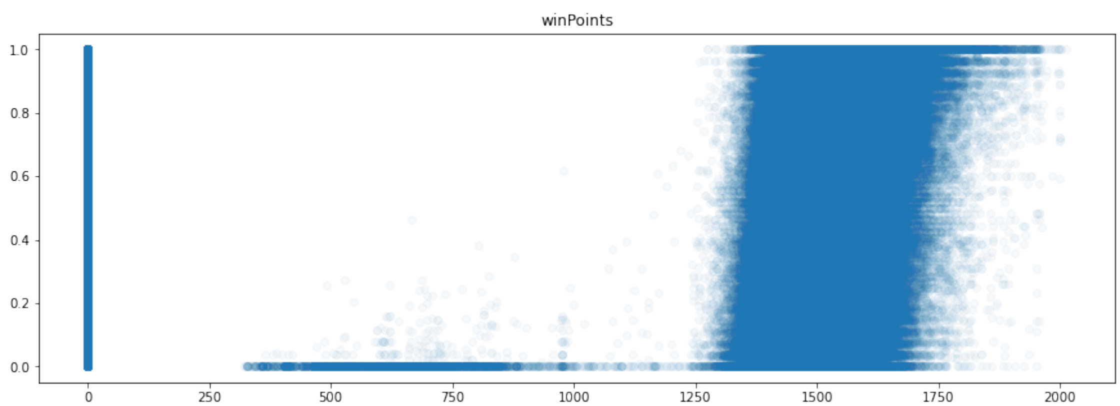

killPoints
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000

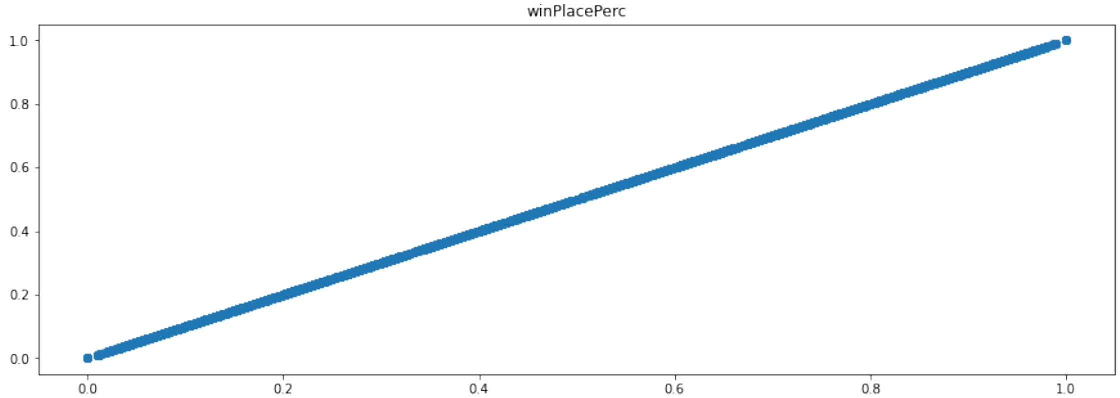

matchType
1.0
0.8
0.6
0.4
0.2
0.0
squad-fpp
duo
solo-fpp
squad
duo-fpp
solo
crashfpp
flaretpp
flarefpp
crashtpp


maxPlace
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

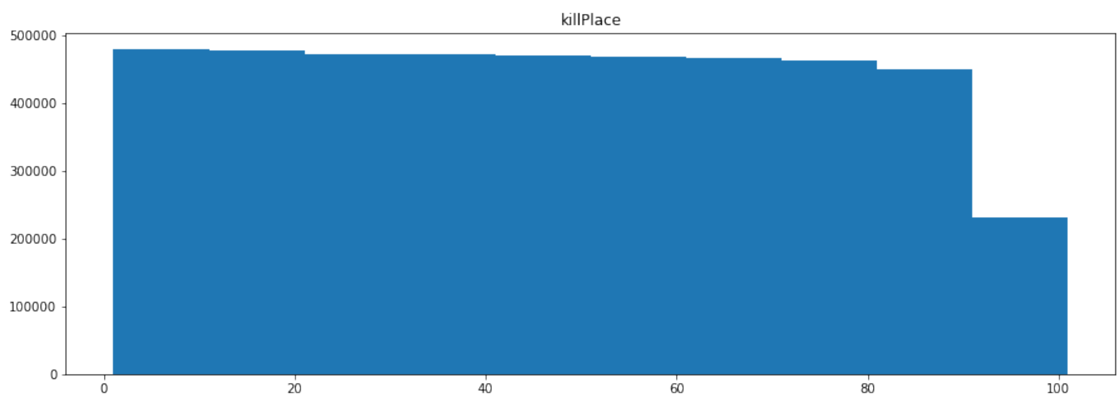

numGroups
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

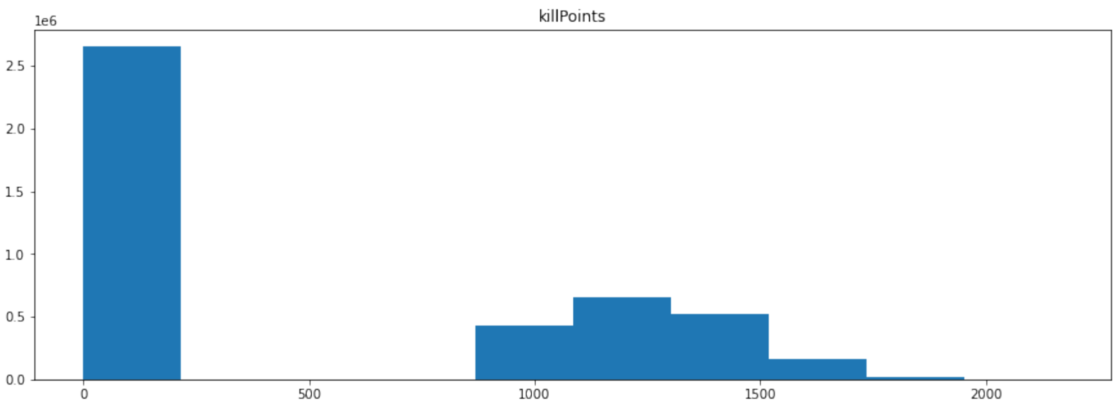

rankPoints
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000
6000

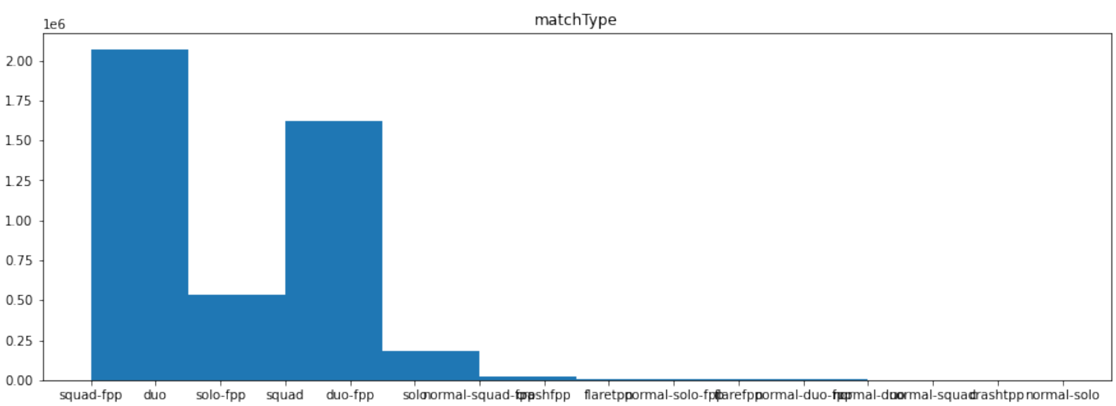

winPoints
1.0
0.8
0.6
0.4
0.2
0.0
0
250
500
750
1000
1250
1500
1750
2000

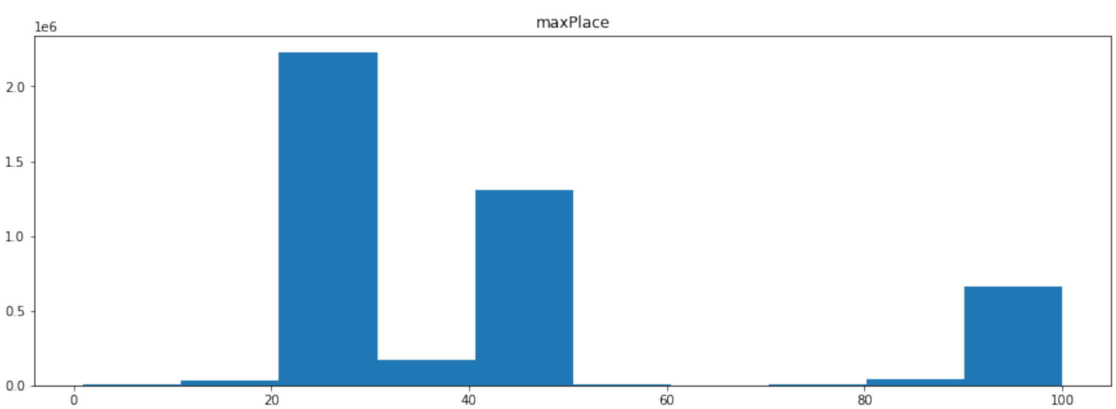


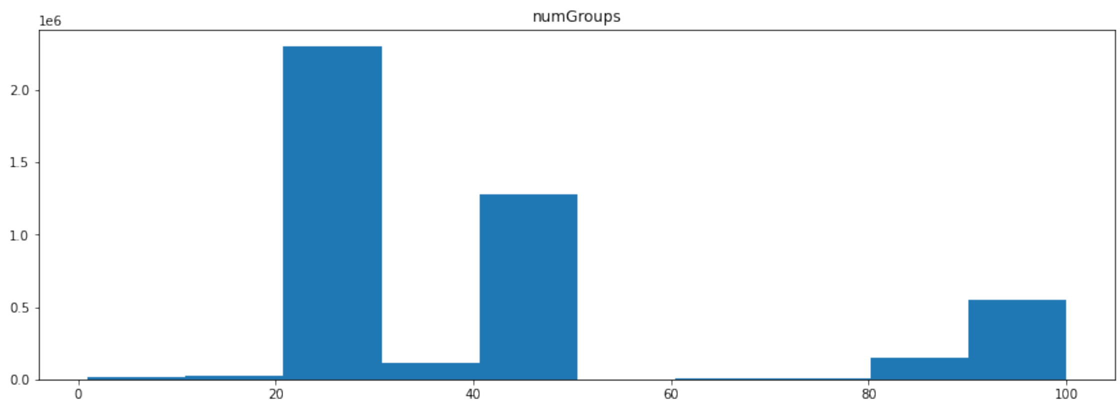
[13]:
```
# See how catergorical data is skewed
for i in range(len(categorical_data.columns)):
    var_name = str(categorical_data.columns[i])
    plt.figure(figsize=(15,5))
    plt.hist(categorical_data[var_name])
    plt.title(var_name)
    plt.show()
```

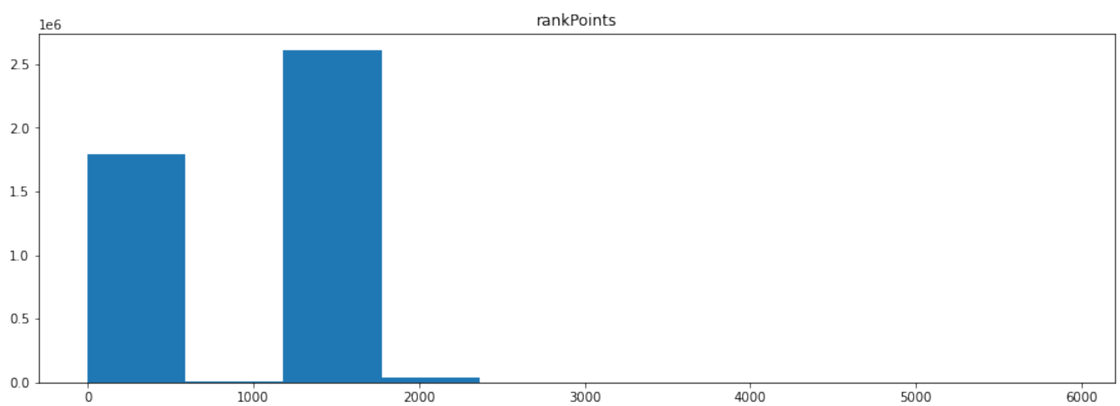


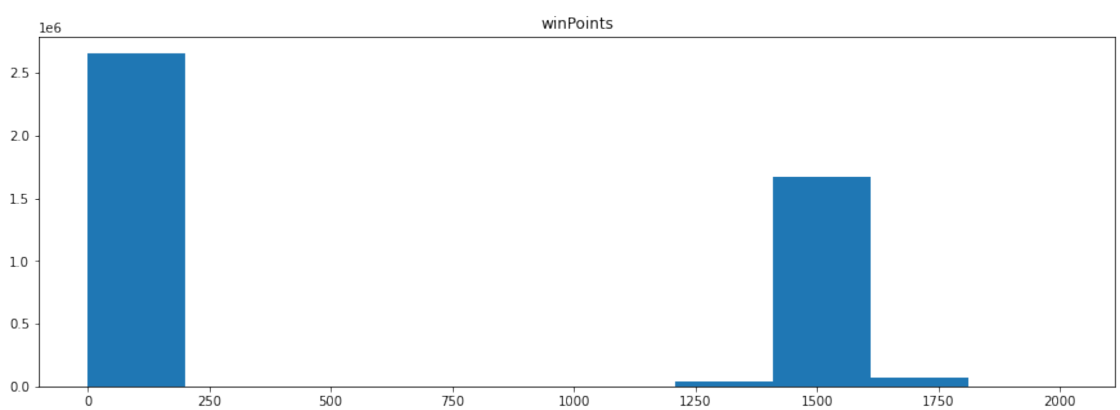

killPoints
1e6
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000

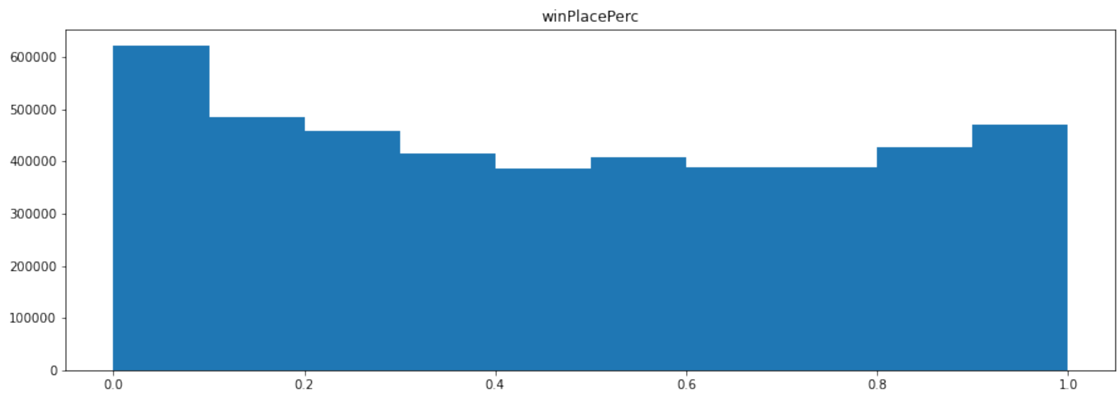

matchType
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
squad-fpp
duo
solo-fpp
squad
duo-fpp
solo
normal-squad-fpp
crashfpp
flaretpp
normal-solo-fpp
flarefpp
normal-duo-fpp
normal-duo
normal-squad
crashtpp
normal-solo


maxPlace
1e6
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100


numGroups
1e6
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100


rankPoints
1e6
2.5
2.0
1.5
1.0
0.5
0.0
0
1000
2000
3000
4000
5000
6000


winPoints
1e6
2.5
2.0
1.5
1.0
0.5
0.0
0
250
500
750
1000
1250
1500
1750
2000



[14]:
```
# Find observations with only one numGroups
empty_games = training_data.loc[(training_data['numGroups'] == 1)]
empty_games['winPlacePerc'].value_counts()
```

[14]:
```
0.0    1146
Name: winPlacePerc, dtype: int64
```

Preprocess Data

[15]:
```
# Drop single observation missing 'winPlacePerc' from training data
training_data = training_data.dropna()

# Drop 'maxPlace' since it highly correlated with 'numGroups'
training_data = training_data.drop('maxPlace', axis=1)
test_data = test_data.drop('maxPlace', axis=1)

# Drop 'rankPoints' as this ranking is inconsistent and is being deprecated in␣
↪the API's next version
training_data = training_data.drop('rankPoints', axis=1)
test_data = test_data.drop('rankPoints', axis=1)

# Dop 'Id', 'groupId', 'matchId'
training_data = training_data.drop('Id', axis=1)
training_data = training_data.drop('groupId', axis=1)
training_data = training_data.drop('matchId', axis=1)
test_data = test_data.drop('Id', axis=1)
test_data = test_data.drop('groupId', axis=1)
test_data = test_data.drop('matchId', axis=1)
```

[16]:
```
# Standardize inputs
scaled_training_data = training_data
scaled_test_data = test_data
```

```
def standardize_data(df):
    for i in range(len(df.columns)):
        var_name = str(df.columns[i])
        if var_name != 'matchType':
            scaler = MinMaxScaler()
            scaled_training_data[var_name] = scaler.fit_transform(np.
 ↪array(training_data[var_name]).reshape(4446965,1))
    return scaled_training_data

scaled_training_data = standardize_data(training_data)
scaled_test_data = standardize_data(test_data)
```

[17]:
```
# Get dummies for 'matchType'
final_training_data = pd.get_dummies(data=scaled_training_data,␣
 ↪drop_first=True, columns=['matchType'])
final_test_data = pd.get_dummies(data=scaled_test_data, drop_first=True,␣
 ↪columns=['matchType'])
```

[18]:
```
# split data
Y = final_training_data.pop('winPlacePerc')
X = final_training_data

train_x, test_x, train_y, test_y = train_test_split(X, Y, test_size=0.20,␣
 ↪random_state=1)
```

[19]:
```
train_x.to_csv('train_x.csv', index=False)
test_x.to_csv('test_x.csv', index=False)
train_y.to_csv('train_y.csv', index=False)
test_y.to_csv('test_y.csv', index=False)
```