Genius League Esports Analyst Assessment

November 5, 2022

[ ]:
```
!pip3 install numpy==1.20
!pip3 install --use-deprecated=legacy-resolver pycaret[full]
```

1 Determine if this dataset needs any preprocessing. If so, clean the dataset and document your steps. If not, explain how you came to that conclusion.

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from scipy import stats
```

[ ]:
```
df = pd.read_csv('starcraft_player_data.csv')
df
```

[ ]:
```
      GameID  LeagueIndex Age HoursPerWeek TotalHours       APM  \
0         52            5  27           10       3000  143.7180
1         55            5  23           10       5000  129.2322
2         56            4  30           10        200   69.9612
3         57            3  19           20        400  107.6016
4         58            3  32           10        500  122.8908
…        …            …  ..          …          …        …
3390   10089            8   ?            ?          ?  259.6296
3391   10090            8   ?            ?          ?  314.6700
3392   10092            8   ?            ?          ?  299.4282
3393   10094            8   ?            ?          ?  375.8664
3394   10095            8   ?            ?          ?  348.3576

      SelectByHotkeys  AssignToHotkeys  UniqueHotkeys  MinimapAttacks  \
0            0.003515         0.000220              7        0.000110
1            0.003304         0.000259              4        0.000294
2            0.001101         0.000336              4        0.000294
3            0.001034         0.000213              1        0.000053
4            0.001136         0.000327              2        0.000000
```

…	…	…	…	…
3390	0.020425	0.000743	9	0.000621
3391	0.028043	0.001157	10	0.000246
3392	0.028341	0.000860	7	0.000338
3393	0.036436	0.000594	5	0.000204
3394	0.029855	0.000811	4	0.000224

	MinimapRightClicks	NumberOfPACs	GapBetweenPACs	ActionLatency \
0	0.000392	0.004849	32.6677	40.8673
1	0.000432	0.004307	32.9194	42.3454
2	0.000461	0.002926	44.6475	75.3548
3	0.000543	0.003783	29.2203	53.7352
4	0.001329	0.002368	22.6885	62.0813
…	…	…	…	…
3390	0.000146	0.004555	18.6059	42.8342
3391	0.001083	0.004259	14.3023	36.1156
3392	0.000169	0.004439	12.4028	39.5156
3393	0.000780	0.004346	11.6910	34.8547
3394	0.001315	0.005566	20.0537	33.5142

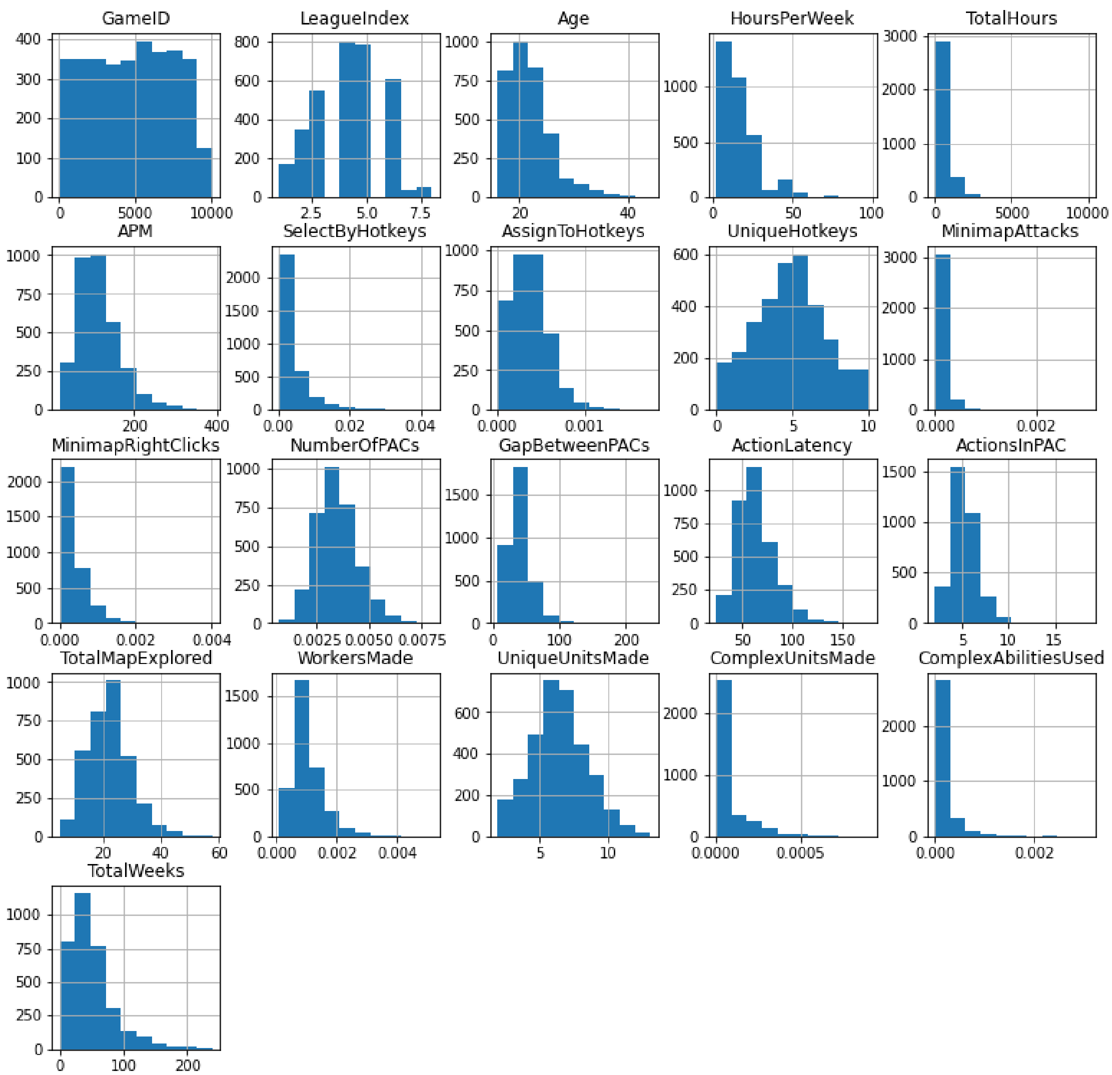
	ActionsInPAC	TotalMapExplored	WorkersMade	UniqueUnitsMade \
0	4.7508	28	0.001397	6
1	4.8434	22	0.001193	5
2	4.0430	22	0.000745	6
3	4.9155	19	0.000426	7
4	9.3740	15	0.001174	4
…	…	…	…	…
3390	6.2754	46	0.000877	5
3391	7.1965	16	0.000788	4
3392	6.3979	19	0.001260	4
3393	7.9615	15	0.000613	6
3394	6.3719	27	0.001566	7

	ComplexUnitsMade	ComplexAbilitiesUsed
0	0.000000	0.000000
1	0.000000	0.000208
2	0.000000	0.000189
3	0.000000	0.000384
4	0.000000	0.000019
…	…	…
3390	0.000000	0.000000
3391	0.000000	0.000000
3392	0.000000	0.000000
3393	0.000000	0.000631
3394	0.000457	0.000895

[3395 rows x 20 columns]

[20]:
```
df.hist(figsize = (13,13))
```

[20]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8709950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e871e510>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e86b4450>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e865c950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8691e50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8654390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e860b910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e85c0dd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e85c0e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8582450>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f30e84efd50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e84b3290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8469790>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e841fc90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e83e11d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f30e84196d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e83cfbd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8394110>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e834c610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8302b10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f30e82b9fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e827b550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e8231a50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e81e9f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f30e81ad490>]],
      dtype=object)
```



2 Multicollinearity has a negative impact on many popular ML models. Check if this dataset experiences any multicollinearity. If so, reduce the impact until an acceptable point.

The data will definitely require some preprocessing due to the presence of null values, indicated by ”?”s.

I will also convert column types, remove outliers, and remove high VIF columns.

[ ]:
```
## Cleaning the data
```

```
# Code below removes '?' values but that has consequence of removing League 8
data, which we need what few data points we have

# Code to split data between complete data and incomplete row values
# df_na = df.iloc[np.where(df.applymap(lambda x: x == '?'))]
df_below8 = df.drop(np.where(df.applymap(lambda x: x == '?'))[0])

# Fix column data types
df_below8['Age'] = pd.to_numeric(df_below8['Age'])
df_below8['HoursPerWeek'] = pd.to_numeric(df_below8['HoursPerWeek'])
df_below8['TotalHours'] = pd.to_numeric(df_below8['TotalHours'])


# Drop non-League 8 null values
df = df.drop(df.iloc[np.where(df[df.LeagueIndex<8].applymap(lambda x: x == '?
'))].index)

# To fix the null League 8 values, I will replace the metrics with the median
and top 75th percentile of League 7 players
df.loc[df.LeagueIndex==8,'Age'] = df_below8[df_below8.LeagueIndex==7].quantile(.
5).Age
df.loc[df.LeagueIndex==8,'HoursPerWeek'] = df_below8[df_below8.LeagueIndex==7].
quantile(.75).HoursPerWeek
df.loc[df.LeagueIndex==8,'TotalHours'] = df_below8[df_below8.LeagueIndex==7].
quantile(.75).TotalHours

# # Convert object type to integer columns
df['Age'] = pd.to_numeric(df['Age'])
df['HoursPerWeek'] = pd.to_numeric(df['HoursPerWeek'])
df['TotalHours'] = pd.to_numeric(df['TotalHours'])

df["TotalWeeks"] = df["TotalHours"]/df["HoursPerWeek"]
# Remove recorded playtime when hours per week > average hours awake, given 8
hours of sleep/day
df =df[df['HoursPerWeek']<112]
# Startcraft was released in 2010 and the dataset is cited to be from 2013. So,
we could set a cutoff of around 200 weeks.
# However, HoursPerWeek may be a more recent measurement average when the
survey was completed rather than all-time metric, so
# the hours per week may be higher or lower in reality, depending on the
player's consistency and interest.
# Thus, I will simply have 250 weeks as a cut off.
df = df[df['TotalWeeks']<250]
```

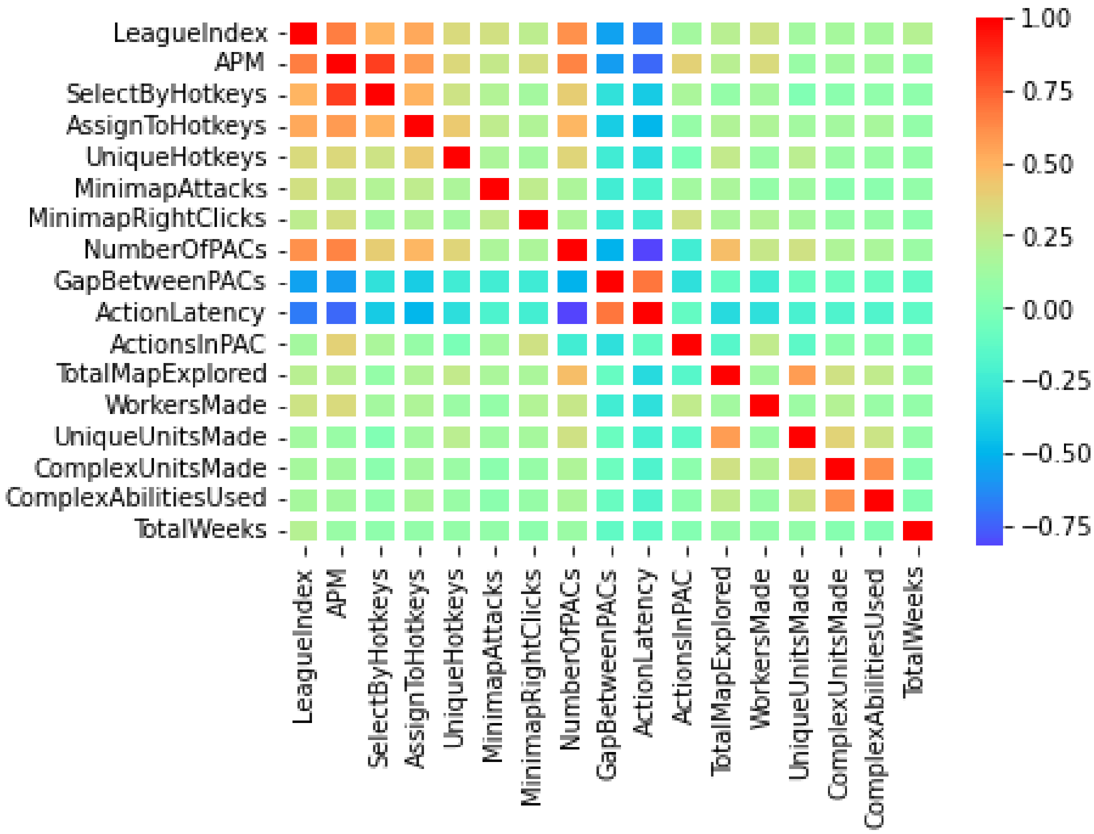
```
# Remove outliers
#Median Absolute Deviation
def find_outliers(df, variable_name):
    #Takes dataframe and checks every value's deviation per column
    columns = df.columns
    med = np.median(df, axis = 0)
    mad = np.abs(stats.median_absolute_deviation(df))
    threshold = 1.7
    outlier = []
    index=0
    for item in range(len(columns)):
        if columns[item] == variable_name:
            index == item
    for i, v in enumerate(df.loc[:,variable_name]):
        t = (v-med[index])/mad[index]
        if t > threshold:
            outlier.append(i)
        else:
            continue
    return outlier

outliers=[]
for col in df.columns:
  outliers.append(find_outliers(df, col))
outliers = sum(outliers, [])
outliers.sort(reverse=True)
for i in outliers:
  df = df[df.GameID != df.iloc[i].GameID]
# Effectively removes rows with abnormal TotalHours values
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:45:
DeprecationWarning: `median_absolute_deviation` is deprecated, use
`median_abs_deviation` instead!

To preserve the existing default behavior, use
`scipy.stats.median_abs_deviation(…, scale=1/1.4826)`.
The value 1.4826 is not numerically precise for scaling
with a normal distribution. For a numerically precise value, use
`scipy.stats.median_abs_deviation(…, scale='normal')`.
```

[ ]:
```
# #Correlation plot
sns.heatmap(df.drop(["GameID","Age","HoursPerWeek", "TotalHours"],axis=1).
 ↪corr(), linewidths=3, center=0, cmap='rainbow');
# Remove Age, HoursPerWeek, and TotalHours because they have no statistically␣
 ↪significant correlations
df.describe()
```

```
[ ]:          GameID  LeagueIndex     Age  HoursPerWeek  TotalHours     APM  \
     count   3343.00      3343.00 3343.00       3343.00     3343.00 3343.00
     mean    4803.39         4.17   21.66         16.36      647.05  116.81
     std     2721.90         1.52    4.18         11.74      539.95   51.92
     min       56.00         1.00   16.00          2.00        3.00   22.06
     25%     2462.50         3.00   19.00          8.00      300.00   79.59
     50%     4862.00         4.00   21.00         12.00      500.00  107.77
     75%     7113.00         5.00   24.00         24.00      800.00  142.49
     max    10095.00         8.00   44.00         98.00    10000.00  389.83

            SelectByHotkeys  AssignToHotkeys  UniqueHotkeys  MinimapAttacks  …  \
     count          3343.00          3343.00        3343.00         3343.00  …
     mean              0.00             0.00           4.37            0.00  …
     std               0.01             0.00           2.36            0.00  …
     min               0.00             0.00           0.00            0.00  …
     25%               0.00             0.00           3.00            0.00  …
     50%               0.00             0.00           4.00            0.00  …
     75%               0.01             0.00           6.00            0.00  …
     max               0.04             0.00          10.00            0.00  …

            NumberOfPACs  GapBetweenPACs  ActionLatency  ActionsInPAC  \
     count       3343.00         3343.00        3343.00       3343.00
     mean           0.00           40.44          63.86          5.27
     std            0.00           17.22          19.29          1.50
     min            0.00            6.67          24.09          2.04
     25%            0.00           28.97          50.50          4.27
     50%            0.00           36.85          61.06          5.09
     75%            0.00           48.32          73.83          6.03
     max            0.01          237.14         176.37         18.56

            TotalMapExplored  WorkersMade  UniqueUnitsMade  ComplexUnitsMade  \
     count           3343.00      3343.00          3343.00           3343.00
     mean              22.11         0.00             6.53              0.00
     std                7.43         0.00             1.86              0.00
     min                5.00         0.00             2.00              0.00
     25%               17.00         0.00             5.00              0.00
     50%               22.00         0.00             6.00              0.00
     75%               27.00         0.00             8.00              0.00
     max               58.00         0.01            13.00              0.00

            ComplexAbilitiesUsed  TotalWeeks
     count               3343.00     3343.00
     mean                   0.00       48.24
     std                    0.00       36.14
     min                    0.00        0.11
     25%                    0.00       25.00
     50%                    0.00       40.00
```

```
75%                 0.00       62.50
max                 0.00      240.00

[8 rows x 21 columns]
```



[ ]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
# the independent variables set (after dropping high VIF features 1 by 1)
X = df.drop(["GameID", "TotalWeeks","APM", "Age", "NumberOfPACs",␣
↪'ActionLatency', 'UniqueHotkeys', 'MinimapRightClicks', 'ComplexUnitsMade',␣
↪"UniqueUnitsMade", "ActionsInPAC", "TotalMapExplored", "AssignToHotkeys"],␣
↪axis = 1)
# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.
↪columns))]
vif_data
```

[ ]:

	feature	VIF
0	LeagueIndex	11.597965
1	HoursPerWeek	6.821954
2	TotalHours	6.595546
3	SelectByHotkeys	2.280508
4	MinimapAttacks	1.524177
5	GapBetweenPACs	3.753443
6	WorkersMade	5.065627
7	ComplexAbilitiesUsed	1.320425
8	TotalWeeks	5.700003

[ ]:
```
sns.heatmap(X.corr(), linewidths=3, center=0, cmap='rainbow')
```

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f30f027a950>

3 Determine what are the most important features that could help predict a player's rank?

The features most strongly correlated to LeagueIndex are APM, ActionLatency, and GapBetweenPACs.

Correlation is important because it measures the linear relationship between two factors. For instance, a higher APM and GapBetweenPACs is associated with a higher LeagueIndex. Conversely, a lower latency is associated with a higher league. TotalHours and SelectByHotkeys will also be shown as important features in the demo model I built.

These results are natural because these features are the building blocks of what enables a player to improve: low latency and total hours. If latency is higher, a player will not be able to showcase all their skills and will also have lower action speed. A minimum total hours is also necessary to practice enough and beat the learning curve. A natural indicator of skill, albeit biased. Meanwhile, players in higher leagues tend to be more skilled, i.e. faster actions (APM, Hotkeys). A more nuanced measure of skill (strategic/calculative skills) is not provided in the dataset. A skill indicator like win rate would be valuable, but players are matched against those in the same skill bracket. Because of this, the win rate will approach 50% when there is 1 winner and 1 loser per game. Therefore, action speed is the most prominent differentiator.

Raw speed is not normally an effective metric for competitive games, but StarCraft centers around multi-tasking, so performance will be stunted with fewer actions, regardless of strategy or skill.

4 Your team's Starcraft2 coaching staff loved your project! They think this is perfect for scouting rising stars. Using your discoveries from (3), create a function to find players who should be given a chance to become professionals. Explain why your set of players make sense.

By looking at the feature distribution grouped by each LeagueIndex, I saw a statistical difference between those in the professional league versus those in lower leagues. By taking the core stats of Professional League players, I believe I can filter the players and create a list of players with high potential.

League 7 players have an average playtime of 31 hours/week (10 hours more than League 6 players and almost double League 5 players), which represents the time commitment required. There is also a baseline of at least 1000 total hours of playtime, which is flexible but serves as a foundation for players' skills. I believe there is further error in the total hours played input because higher league players may create second accounts after playing for a while. Continuing by analyzing the minimum, first quartile, third quartile, and maximum values, I decided to filter as shown below:

APM >= 250, GapBetweenPACs <= 23, ActionLatency <= 40 For a smaller list, also filter by HoursPerWeek >= 25, TotalHours >= 1000.

[ ]:
```
pd.set_option('max_rows', 99999)
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

```
df[['HoursPerWeek','TotalWeeks', 'APM', 'GapBetweenPACs', 'ActionLatency',␣
↪'LeagueIndex']].groupby('LeagueIndex').describe().T
# *** Note:
# The professional league player's age, total hours, and hours per week were␣
↪unknown, but we can look at upper league players' stats for these and␣
↪compare.
```

```
[ ]: LeagueIndex              1      2      3      4      5      6      7      8
     HoursPerWeek   count 167.00 347.00 545.00 802.00 787.00 606.00  34.00  55.00
                    mean   13.13  13.30  14.03  14.14  16.37  20.82  32.47  42.00
                    std     9.41   9.59   9.74  10.20  11.39  12.49  20.44   0.00
                    min     2.00   2.00   2.00   2.00   2.00   2.00   6.00  42.00
                    25%     8.00   8.00   8.00   8.00   8.00  10.00  17.00  42.00
                    50%    12.00  10.00  12.00  12.00  12.00  20.00  28.00  42.00
                    75%    16.00  18.00  20.00  20.00  24.00  28.00  42.00  42.00
                    max    70.00  72.00  80.00  96.00  96.00  84.00  98.00  42.00
     TotalWeeks     count 167.00 347.00 545.00 802.00 787.00 606.00  34.00  55.00
                    mean   28.54  32.50  41.59  50.45  56.24  54.90  56.99  47.62
                    std    32.72  30.28  31.79  37.72  38.68  33.47  45.31   0.00
                    min     0.71   0.60   0.11   1.00   3.00   2.50   7.14  47.62
                    25%     6.25  10.57  18.33  25.00  30.00  33.33  36.90  47.62
                    50%    15.75  25.00  35.00  40.00  50.00  50.00  46.13  47.62
                    75%    37.50  44.07  52.14  64.82  70.00  69.69  58.04  47.62
                    max   170.00 200.00 200.00 237.50 240.00 225.00 238.10  47.62
     APM            count 167.00 347.00 545.00 802.00 787.00 606.00  34.00  55.00
                    mean   59.54  74.78  90.10 105.80 131.19 158.87 188.89 267.34
                    std    23.43  23.86  30.61  33.84  41.44  48.36  41.21  56.18
                    min    22.06  24.66  29.82  38.03  49.74  65.37 115.76 146.39
                    25%    43.32  57.21  69.05  80.71 102.80 125.39 167.33 222.23
                    50%    54.05  71.68  85.99 103.69 125.87 152.85 183.98 274.34
                    75%    72.50  89.21 104.87 123.70 152.40 187.27 212.44 313.28
                    max   172.95 179.62 226.66 249.02 372.64 389.83 298.80 375.87
     GapBetweenPACs count 167.00 347.00 545.00 802.00 787.00 606.00  34.00  55.00
                    mean   65.65  53.79  46.15  41.06  34.77  30.18  22.84  18.97
                    std    29.55  19.81  14.95  13.03  10.56   9.14   6.19   6.08
                    min     9.94   6.67  14.05  12.04  10.86  11.33  10.29   8.16
                    25%    47.33  39.63  35.23  31.62  27.37  23.71  18.55  15.10
                    50%    58.95  50.03  43.79  39.09  33.92  28.93  22.05  17.99
                    75%    76.26  64.83  54.04  48.63  41.01  35.08  26.04  21.77
                    max   237.14 156.62 122.80  94.15  84.61  71.50  38.79  35.41
     ActionLatency  count 167.00 347.00 545.00 802.00 787.00 606.00  34.00  55.00
                    mean   95.40  81.27  73.68  64.80  56.22  48.94  40.27  35.39
                    std    24.26  18.12  16.64  13.40  11.26  10.46   6.57   5.79
                    min    51.11  40.01  36.58  35.14  30.76  24.63  29.99  24.09
                    25%    77.88  68.48  62.44  55.54  48.14  41.73  34.74  31.28
                    50%    93.33  79.24  72.09  62.75  55.22  48.08  39.18  35.41
                    75%   107.82  90.18  83.07  72.17  62.86  55.30  45.98  38.58
```

```
max   173.56 176.37 150.30 129.85 103.38  88.32  52.31  54.56
```

[ ]:
```
df[(df["APM"]>= 200) & (df["GapBetweenPACs"]<= 23) & (df["ActionLatency"]<= 40)␣
↪& (df["LeagueIndex"]!= 8)].sort_values(by="APM", ascending=False).
↪reset_index().drop(['GameID','index'],axis=1)
```

[ ]:

	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys \
0	6	23.00	24.00	1400.00	311.01	0.04
1	6	19.00	20.00	960.00	301.77	0.03
2	7	23.00	42.00	2000.00	298.80	0.03
3	5	24.00	16.00	1000.00	292.54	0.02
4	6	17.00	10.00	700.00	290.12	0.02
5	6	18.00	28.00	1000.00	286.55	0.02
6	7	21.00	24.00	1000.00	286.45	0.02
7	6	24.00	36.00	1000.00	283.28	0.02
8	6	18.00	20.00	800.00	277.32	0.01
9	6	20.00	6.00	125.00	275.73	0.01
10	6	20.00	6.00	300.00	271.43	0.02
11	6	18.00	14.00	730.00	268.02	0.01
12	6	17.00	24.00	2500.00	263.38	0.02
13	6	16.00	20.00	800.00	260.53	0.01
14	6	16.00	28.00	1500.00	259.38	0.02
15	5	17.00	24.00	800.00	256.85	0.02
16	6	16.00	20.00	1095.00	244.97	0.02
17	6	20.00	28.00	2000.00	244.79	0.01
18	6	20.00	8.00	500.00	241.99	0.01
19	6	24.00	14.00	730.00	237.27	0.01
20	6	17.00	28.00	1600.00	236.68	0.01
21	7	18.00	98.00	700.00	236.03	0.02
22	5	20.00	12.00	400.00	231.74	0.01
23	5	24.00	6.00	700.00	226.35	0.01
24	7	23.00	36.00	1100.00	225.20	0.01
25	6	21.00	28.00	1095.00	219.44	0.01
26	5	21.00	12.00	600.00	217.95	0.01
27	6	22.00	42.00	3000.00	216.89	0.01
28	7	24.00	16.00	1250.00	216.69	0.01
29	6	19.00	16.00	700.00	215.96	0.01
30	6	19.00	20.00	800.00	215.96	0.01
31	6	17.00	24.00	850.00	215.71	0.01
32	5	21.00	28.00	960.00	215.64	0.02
33	6	21.00	16.00	1000.00	215.15	0.01
34	5	22.00	14.00	900.00	212.44	0.01
35	7	24.00	20.00	1000.00	211.97	0.01
36	6	18.00	14.00	500.00	211.24	0.01
37	7	18.00	42.00	2000.00	210.21	0.01
38	4	18.00	8.00	650.00	209.80	0.01
39	6	23.00	28.00	1000.00	208.95	0.01

40	6	20.00	16.00	1300.00	208.76	0.01
41	6	24.00	16.00	450.00	208.69	0.01
42	5	21.00	28.00	1400.00	208.03	0.00
43	6	17.00	14.00	300.00	207.58	0.01
44	6	17.00	30.00	1000.00	205.40	0.01
45	5	31.00	4.00	750.00	205.34	0.01
46	6	19.00	28.00	800.00	203.15	0.01
47	6	20.00	42.00	1500.00	202.01	0.01
48	7	17.00	56.00	1600.00	202.00	0.01
49	6	16.00	6.00	400.00	201.91	0.01
50	7	19.00	28.00	500.00	201.29	0.01

	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks \
0	0.00	7	0.00	0.00
1	0.00	4	0.00	0.00
2	0.00	4	0.00	0.00
3	0.00	5	0.00	0.00
4	0.00	5	0.00	0.00
5	0.00	10	0.00	0.00
6	0.00	6	0.00	0.00
7	0.00	8	0.00	0.00
8	0.00	6	0.00	0.00
9	0.00	6	0.00	0.00
10	0.00	7	0.00	0.00
11	0.00	7	0.00	0.00
12	0.00	6	0.00	0.00
13	0.00	7	0.00	0.00
14	0.00	5	0.00	0.00
15	0.00	4	0.00	0.00
16	0.00	4	0.00	0.00
17	0.00	6	0.00	0.00
18	0.00	6	0.00	0.00
19	0.00	5	0.00	0.00
20	0.00	7	0.00	0.00
21	0.00	10	0.00	0.00
22	0.00	6	0.00	0.00
23	0.00	6	0.00	0.00
24	0.00	8	0.00	0.00
25	0.00	10	0.00	0.00
26	0.00	3	0.00	0.00
27	0.00	10	0.00	0.00
28	0.00	6	0.00	0.00
29	0.00	10	0.00	0.00
30	0.00	7	0.00	0.00
31	0.00	5	0.00	0.00
32	0.00	5	0.00	0.00
33	0.00	9	0.00	0.00

34	0.00	6	0.00	0.00
35	0.00	8	0.00	0.00
36	0.00	5	0.00	0.00
37	0.00	6	0.00	0.00
38	0.00	4	0.00	0.00
39	0.00	2	0.00	0.00
40	0.00	8	0.00	0.00
41	0.00	7	0.00	0.00
42	0.00	2	0.00	0.00
43	0.00	5	0.00	0.00
44	0.00	9	0.00	0.00
45	0.00	5	0.00	0.00
46	0.00	6	0.00	0.00
47	0.00	7	0.00	0.00
48	0.00	4	0.00	0.00
49	0.00	7	0.00	0.00
50	0.00	9	0.00	0.00

	NumberOfPACs	GapBetweenPACs	ActionLatency	ActionsInPAC	\
0	0.01	12.79	36.99	4.49	
1	0.01	21.39	34.28	4.98	
2	0.00	16.86	33.73	6.14	
3	0.00	14.42	30.76	9.47	
4	0.01	17.93	27.56	5.53	
5	0.01	21.57	28.73	5.24	
6	0.01	20.42	30.98	5.72	
7	0.01	21.23	31.74	5.28	
8	0.01	16.69	24.63	5.19	
9	0.01	12.37	28.82	5.80	
10	0.00	16.41	33.53	6.78	
11	0.01	13.27	30.39	7.03	
12	0.01	18.15	31.11	4.92	
13	0.01	19.90	29.03	5.60	
14	0.00	20.62	37.45	6.12	
15	0.01	15.39	34.98	6.23	
16	0.01	21.75	38.85	4.90	
17	0.01	22.44	33.79	5.87	
18	0.01	17.91	29.94	5.27	
19	0.01	18.75	32.15	5.10	
20	0.01	19.27	29.08	6.59	
21	0.01	18.78	29.99	4.34	
22	0.00	12.99	31.01	6.61	
23	0.01	16.78	35.80	6.58	
24	0.01	21.41	38.49	4.91	
25	0.00	19.04	35.76	6.10	
26	0.01	16.45	34.17	5.41	
27	0.01	15.99	31.53	5.43	

28	0.01	21.61	32.62	5.01
29	0.00	15.39	34.68	6.24
30	0.01	20.20	35.54	6.60
31	0.01	22.24	35.59	5.25
32	0.01	22.63	33.02	4.09
33	0.01	21.21	31.82	4.29
34	0.01	20.92	30.96	6.01
35	0.00	10.29	34.58	7.24
36	0.01	14.57	37.57	5.09
37	0.01	18.36	34.04	5.41
38	0.01	16.48	36.24	5.84
39	0.01	13.96	29.21	4.68
40	0.01	21.23	33.80	5.50
41	0.00	20.70	37.34	9.73
42	0.00	17.48	39.07	9.50
43	0.01	20.52	37.99	4.95
44	0.01	20.78	31.66	5.15
45	0.00	21.51	39.83	7.14
46	0.00	13.53	35.75	5.55
47	0.00	17.77	36.98	5.66
48	0.01	20.10	33.94	3.77
49	0.01	12.66	31.89	5.39
50	0.01	19.22	37.73	5.14

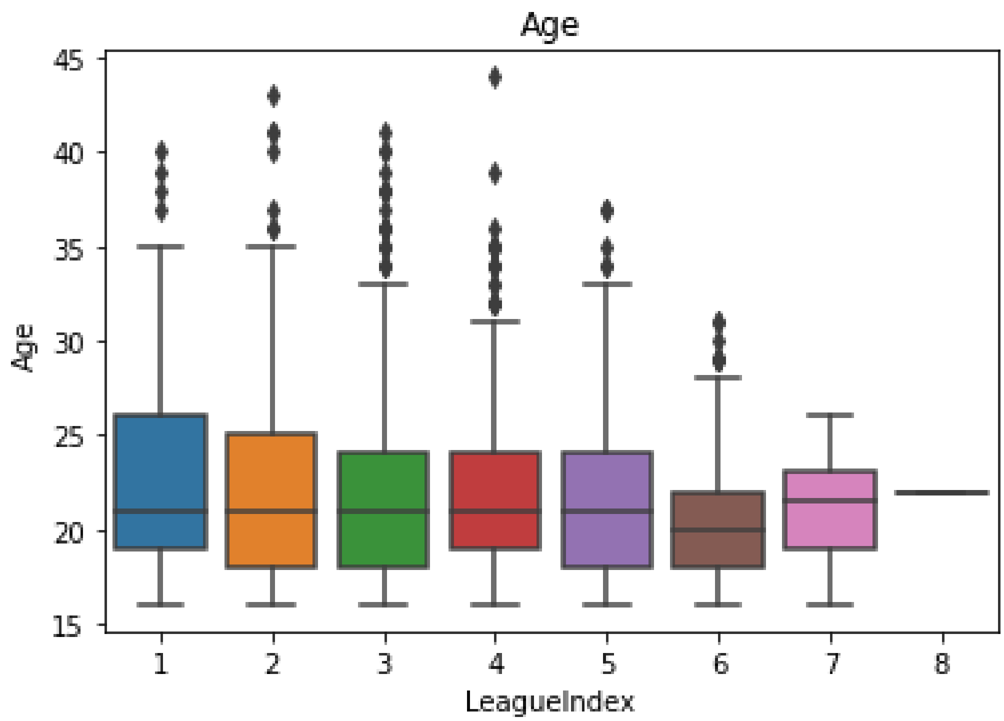
	TotalMapExplored	WorkersMade	UniqueUnitsMade	ComplexUnitsMade	\
0	19	0.00	7	0.00	
1	22	0.00	10	0.00	
2	14	0.00	4	0.00	
3	27	0.00	9	0.00	
4	29	0.00	10	0.00	
5	37	0.00	8	0.00	
6	30	0.00	6	0.00	
7	20	0.00	6	0.00	
8	27	0.00	10	0.00	
9	30	0.00	8	0.00	
10	23	0.00	9	0.00	
11	24	0.00	6	0.00	
12	33	0.00	8	0.00	
13	33	0.00	9	0.00	
14	28	0.00	5	0.00	
15	25	0.00	5	0.00	
16	23	0.00	6	0.00	
17	24	0.00	7	0.00	
18	32	0.00	5	0.00	
19	30	0.00	6	0.00	
20	30	0.00	9	0.00	
21	25	0.00	8	0.00	

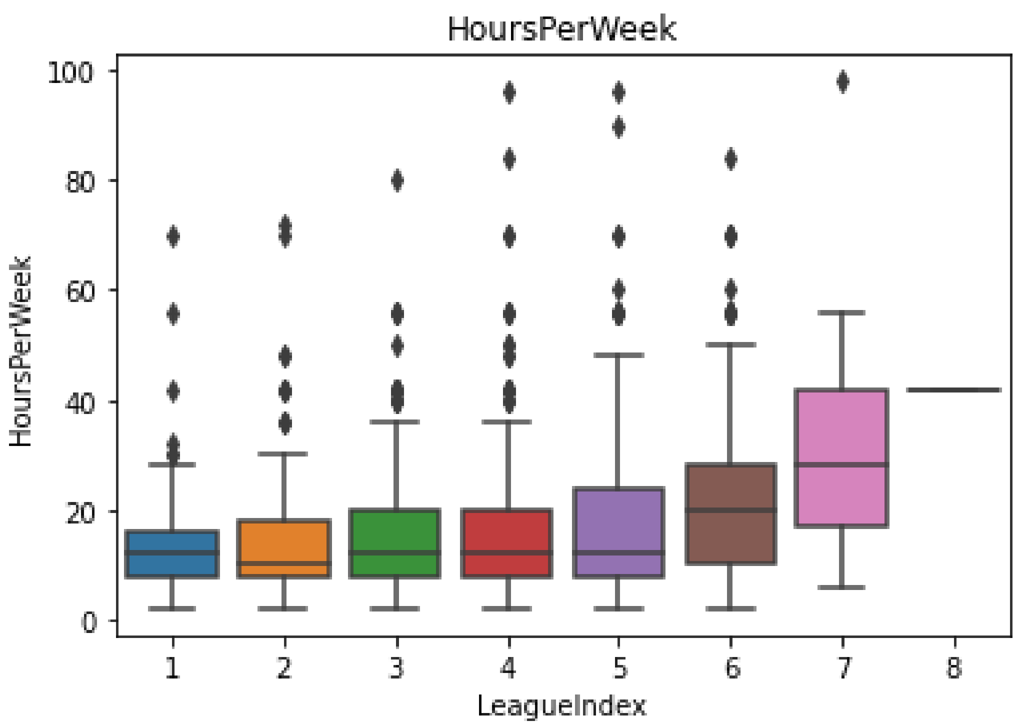
22	33	0.00	8	0.00
23	23	0.00	7	0.00
24	28	0.00	7	0.00
25	22	0.00	6	0.00
26	22	0.00	5	0.00
27	26	0.00	5	0.00
28	33	0.00	8	0.00
29	21	0.00	7	0.00
30	28	0.00	6	0.00
31	36	0.00	10	0.00
32	34	0.00	7	0.00
33	41	0.00	8	0.00
34	25	0.00	7	0.00
35	30	0.00	3	0.00
36	19	0.00	5	0.00
37	24	0.00	6	0.00
38	22	0.00	6	0.00
39	24	0.00	6	0.00
40	25	0.00	9	0.00
41	18	0.00	6	0.00
42	24	0.00	6	0.00
43	34	0.00	7	0.00
44	24	0.00	8	0.00
45	32	0.00	8	0.00
46	18	0.00	7	0.00
47	19	0.00	7	0.00
48	25	0.00	7	0.00
49	18	0.00	5	0.00
50	31	0.00	6	0.00

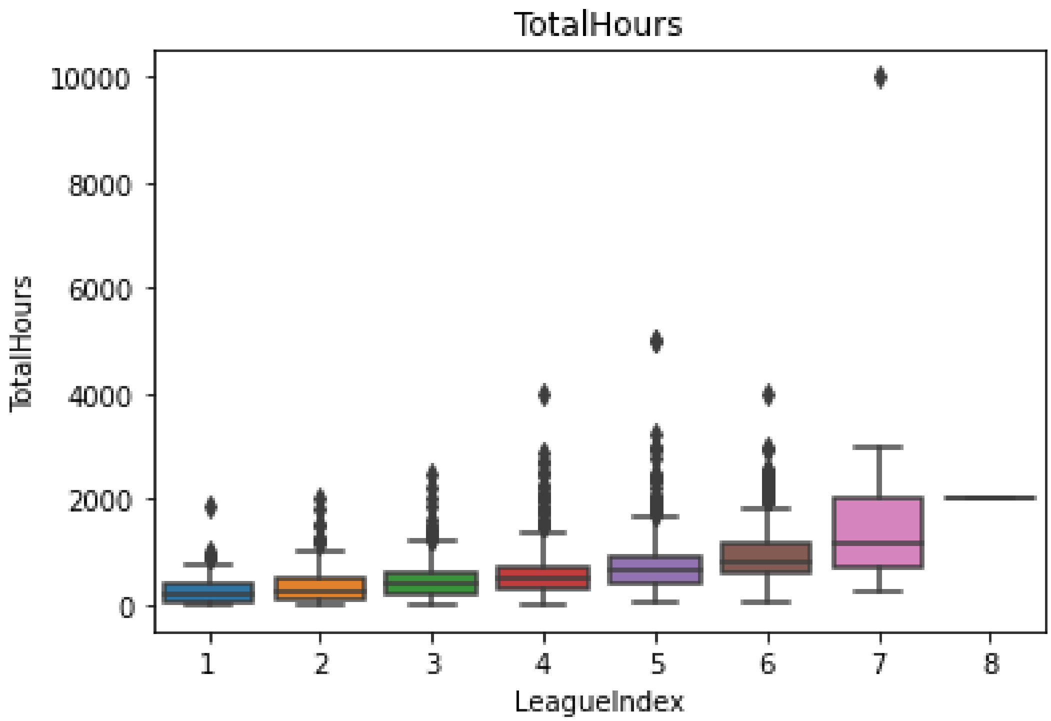
	ComplexAbilitiesUsed	TotalWeeks
0	0.00	58.33
1	0.00	48.00
2	0.00	47.62
3	0.00	62.50
4	0.00	70.00
5	0.00	35.71
6	0.00	41.67
7	0.00	27.78
8	0.00	40.00
9	0.00	20.83
10	0.00	50.00
11	0.00	52.14
12	0.00	104.17
13	0.00	40.00
14	0.00	53.57
15	0.00	33.33

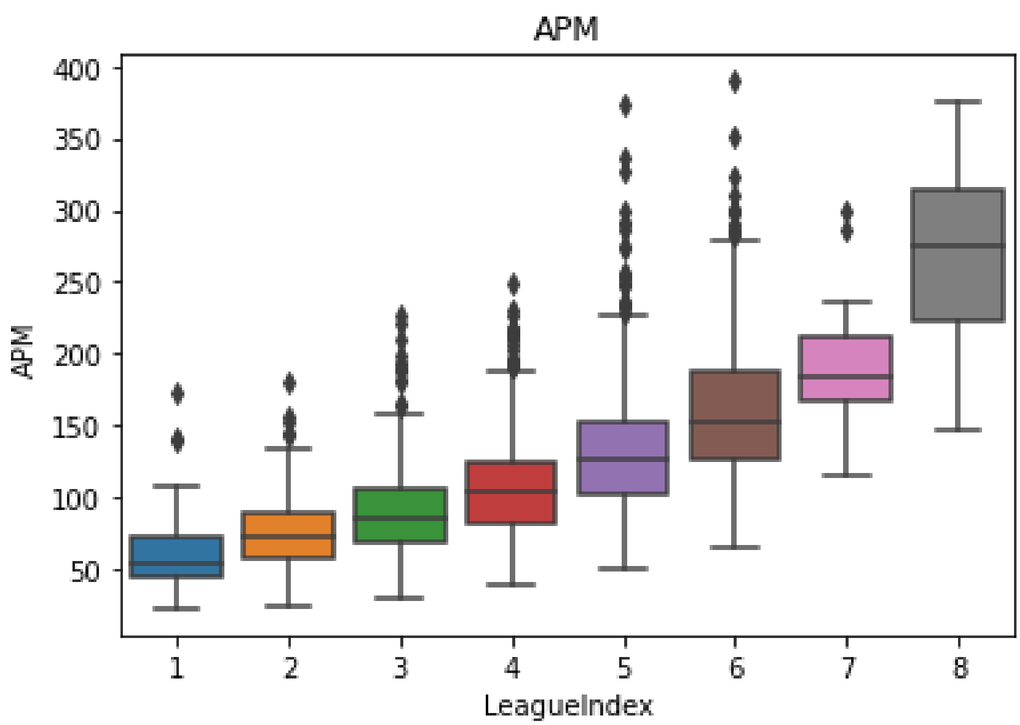
```
16                     0.00       54.75
17                     0.00       71.43
18                     0.00       62.50
19                     0.00       52.14
20                     0.00       57.14
21                     0.00        7.14
22                     0.00       33.33
23                     0.00      116.67
24                     0.00       30.56
25                     0.00       39.11
26                     0.00       50.00
27                     0.00       71.43
28                     0.00       78.12
29                     0.00       43.75
30                     0.00       40.00
31                     0.00       35.42
32                     0.00       34.29
33                     0.00       62.50
34                     0.00       64.29
35                     0.00       50.00
36                     0.00       35.71
37                     0.00       47.62
38                     0.00       81.25
39                     0.00       35.71
40                     0.00       81.25
41                     0.00       28.12
42                     0.00       50.00
43                     0.00       21.43
44                     0.00       33.33
45                     0.00      187.50
46                     0.00       28.57
47                     0.00       35.71
48                     0.00       28.57
49                     0.00       66.67
50                     0.00       17.86
```

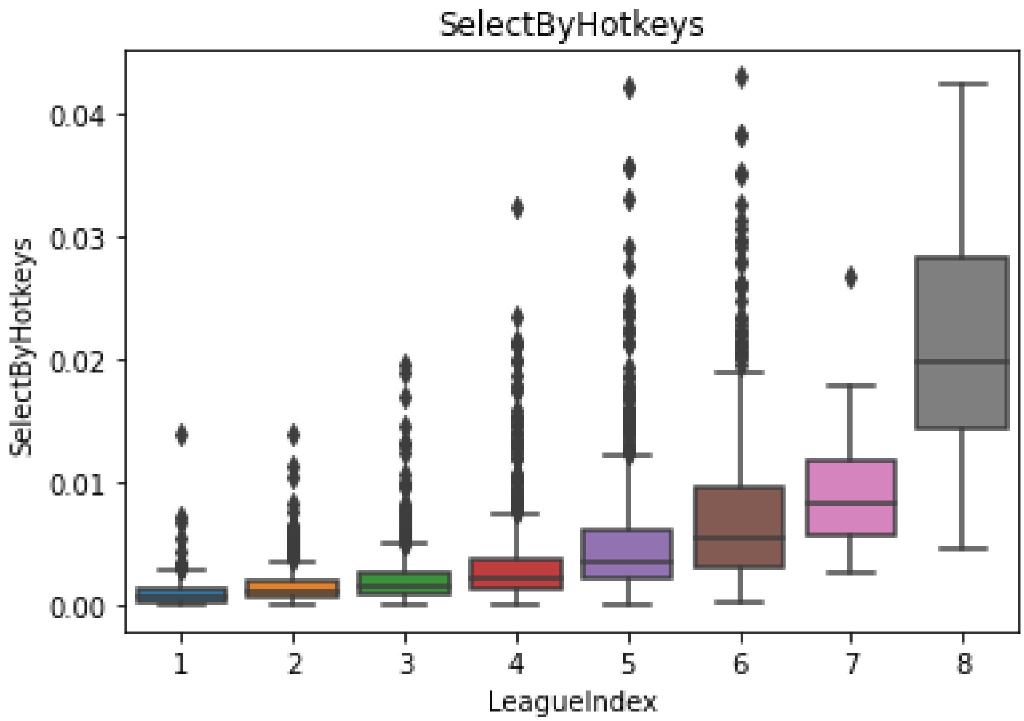
[ ]:
```
## Feature EDA in terms of LeagueIndex
for i, col in enumerate(df.drop(["GameID","LeagueIndex"], axis=1).columns):
    plt.figure(i)
    sns.boxplot(x='LeagueIndex', y=col, data=df).set(title=col)

# Comments:
# # Age looks to be declining slightly on average, until league 7 players␣
↪(possibly due to league 7's small sample size)
```

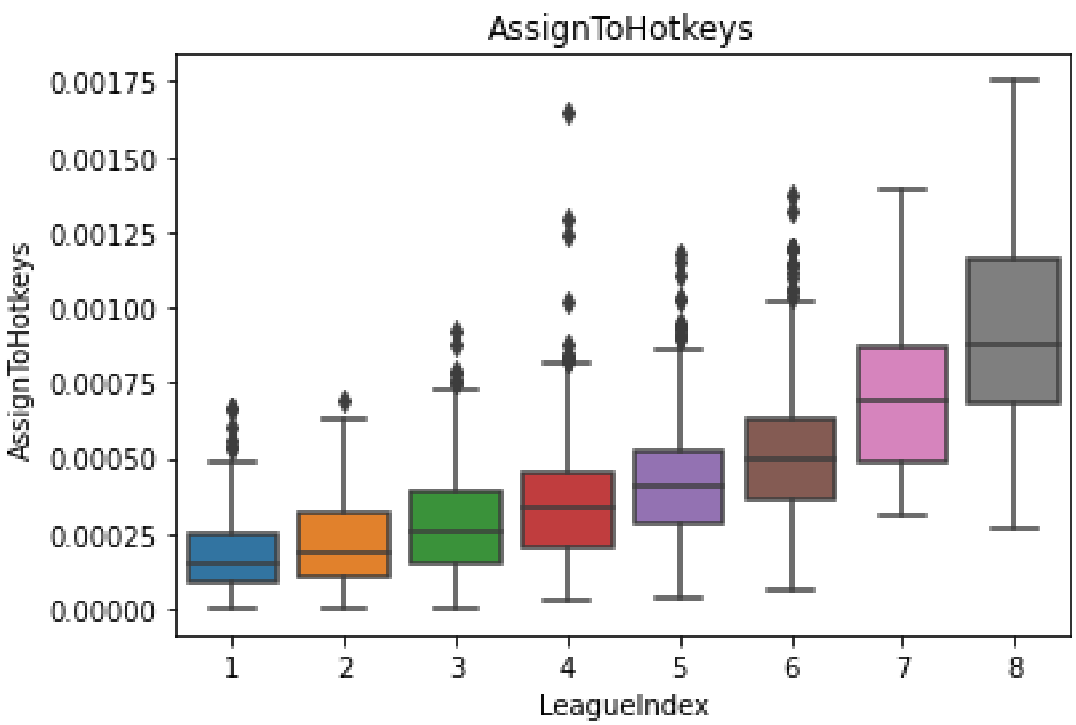

Age
Age
45
40
35
30
25
20
15
1
2
3
4
5
6
7
8
LeagueIndex

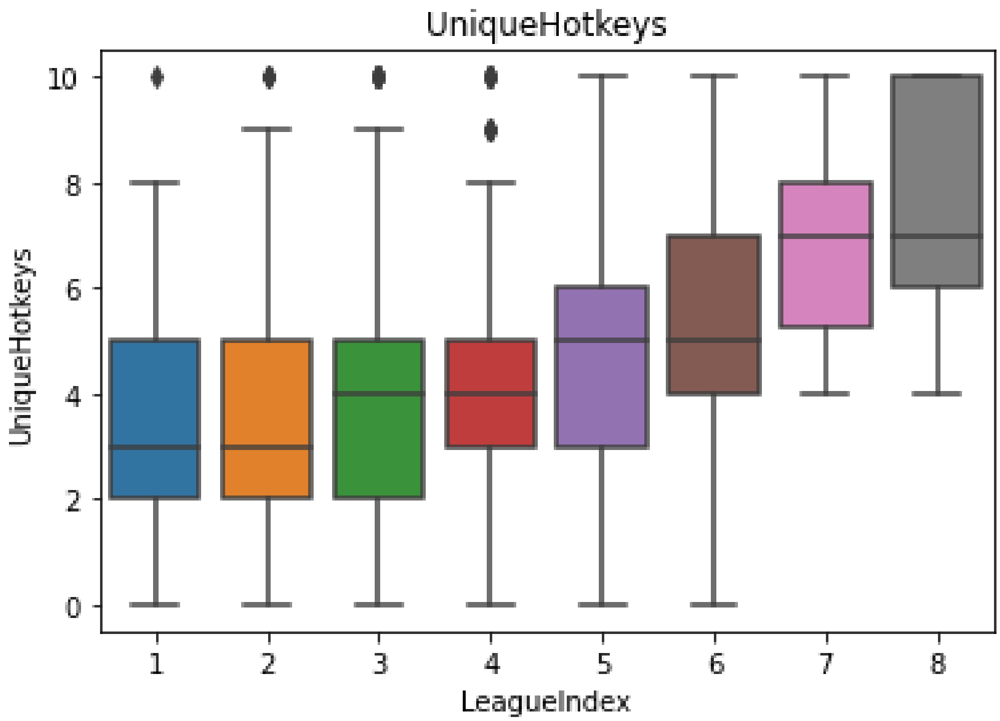

HoursPerWeek
HoursPerWeek
100
80
60
40
20
0
1
2
3
4
5
6
7
8
LeagueIndex

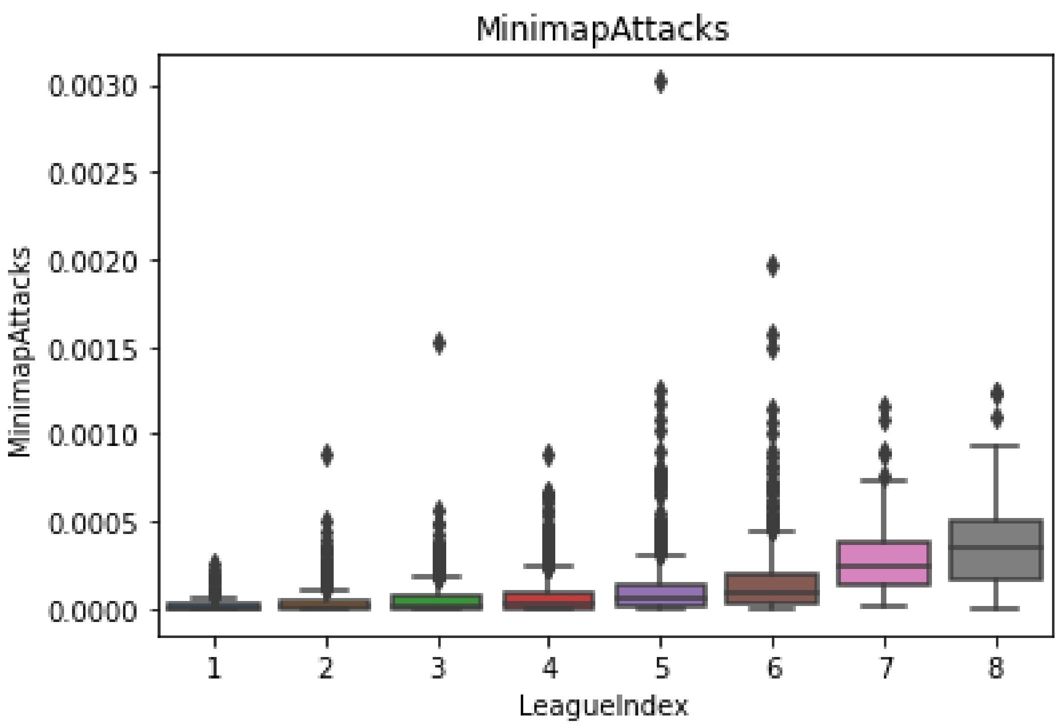

TotalHours
10000
8000
6000
4000
2000
0
TotalHours
1
2
3
4
5
6
7
8
LeagueIndex

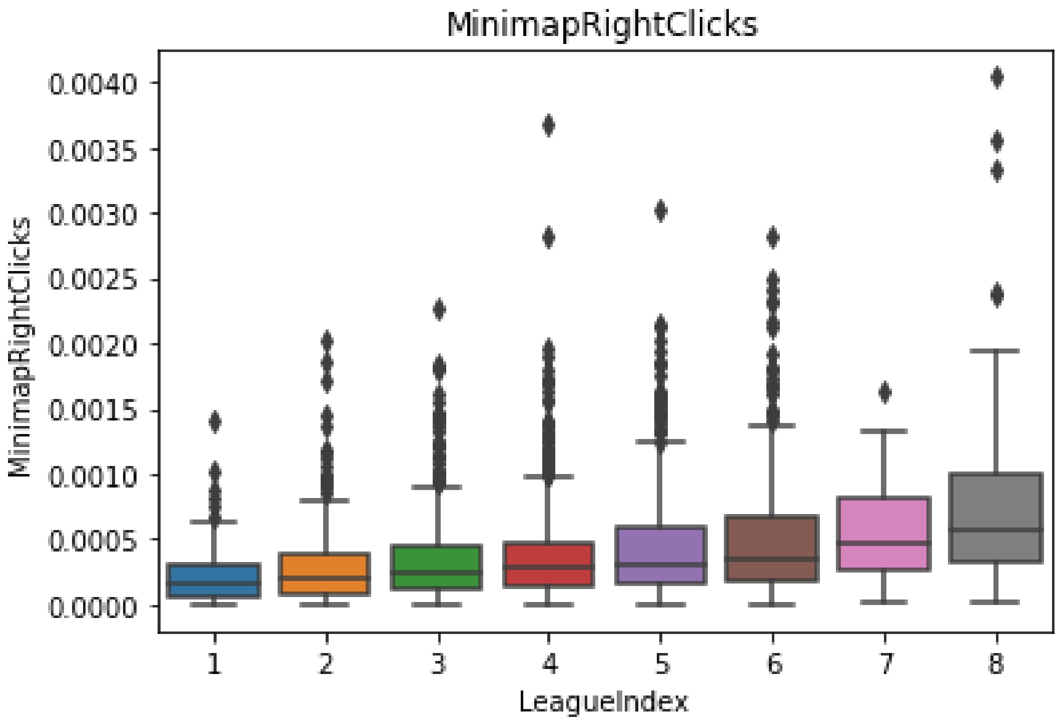

APM
400
350
300
250
200
150
100
50
APM
1
2
3
4
5
6
7
8
LeagueIndex

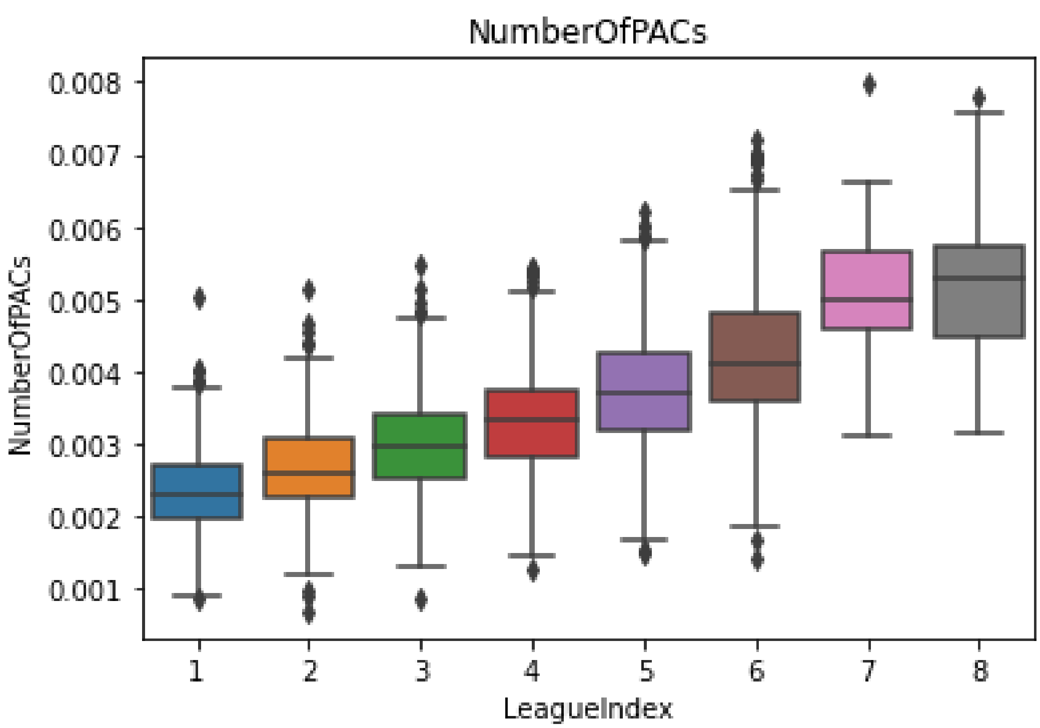

SelectByHotkeys
SelectByHotkeys
0.00
0.01
0.02
0.03
0.04
1
2
3
4
5
6
7
8
LeagueIndex

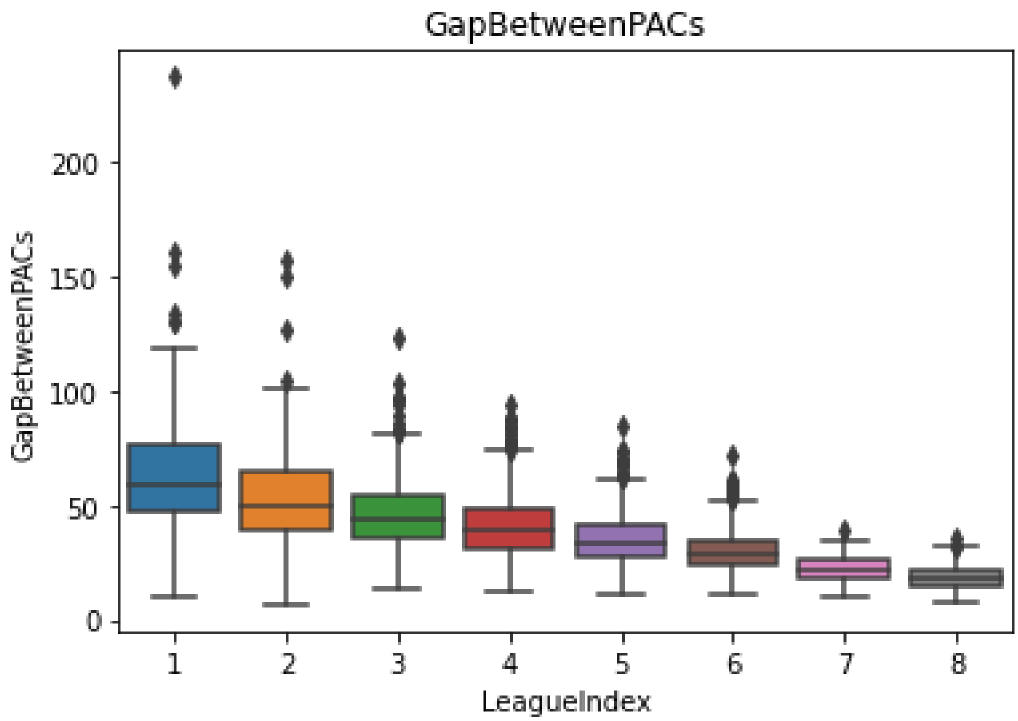

AssignToHotkeys
AssignToHotkeys
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175
1
2
3
4
5
6
7
8
LeagueIndex

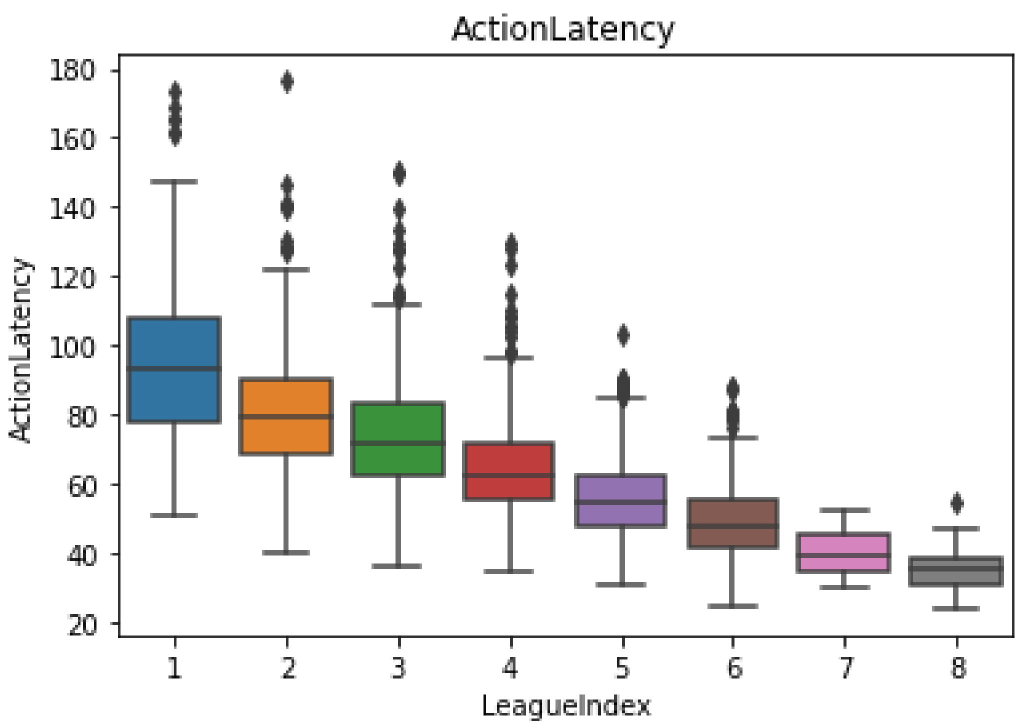

UniqueHotkeys
UniqueHotkeys
0
2
4
6
8
10
1
2
3
4
5
6
7
8
LeagueIndex

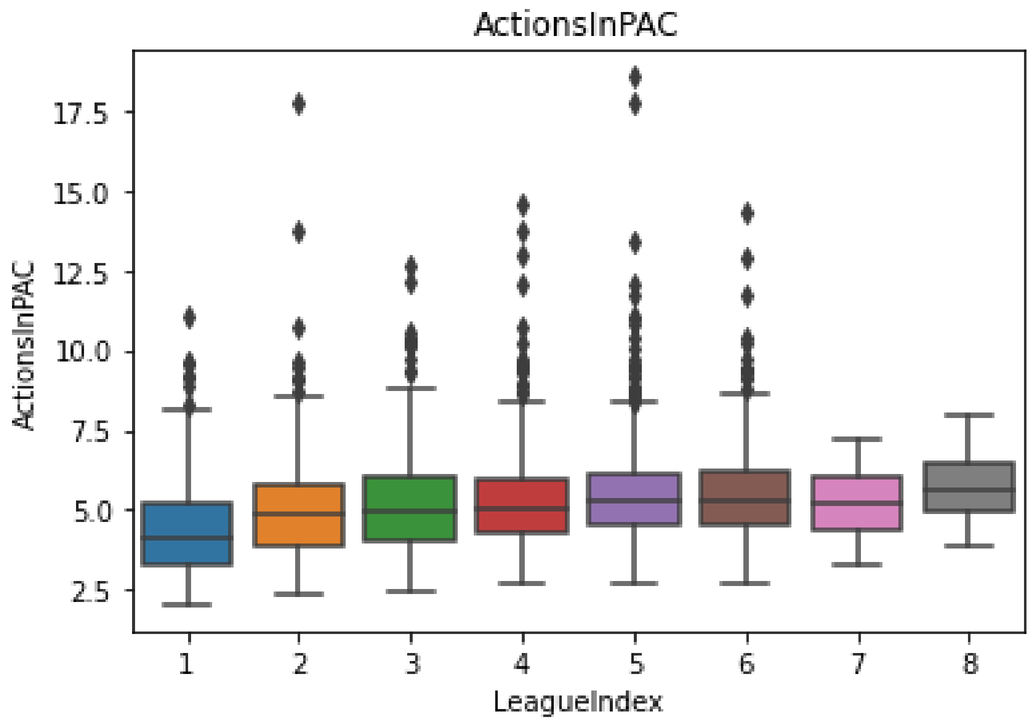

MinimapAttacks
MinimapAttacks
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
1
2
3
4
5
6
7
8
LeagueIndex

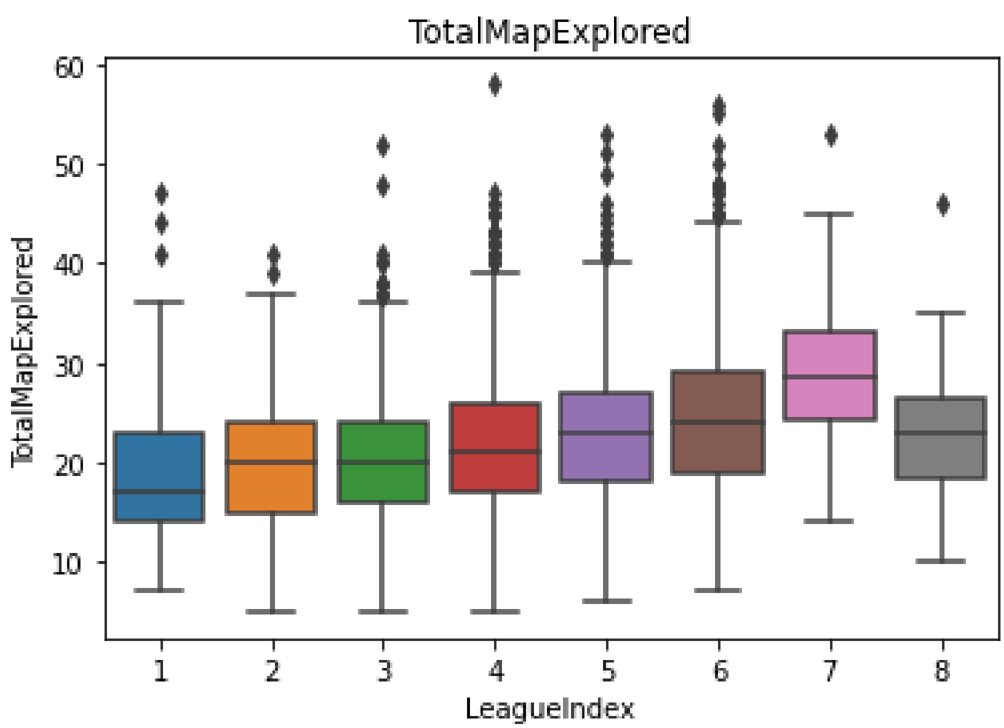

MinimapRightClicks
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
MinimapRightClicks
1
2
3
4
5
6
7
8
LeagueIndex

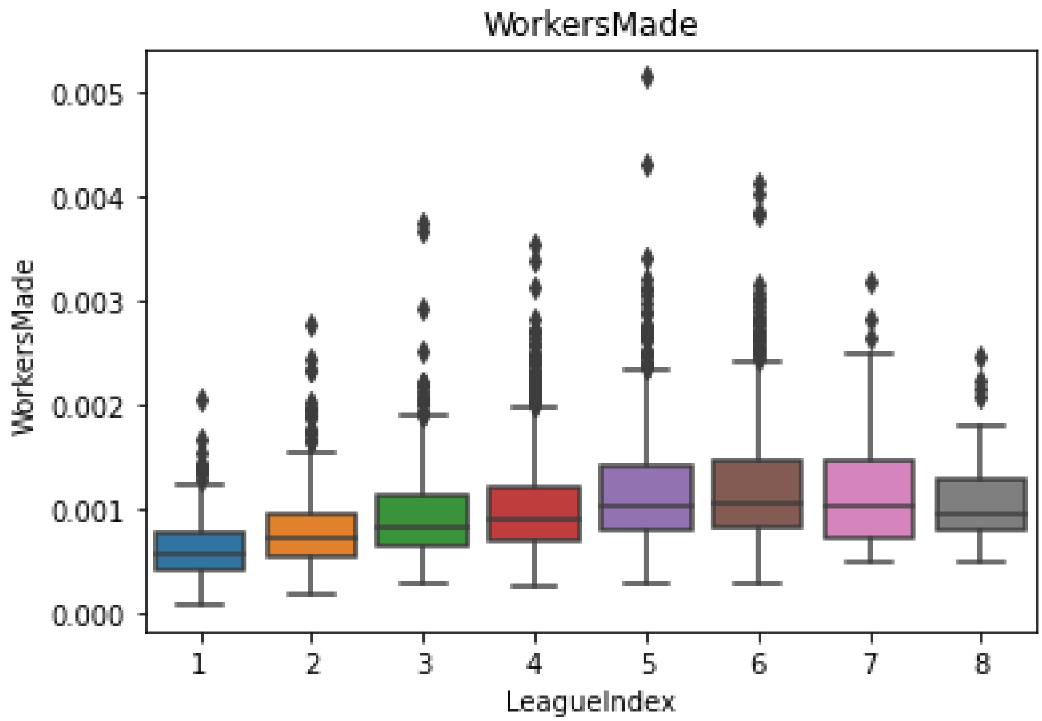

NumberOfPACs
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
NumberOfPACs
1
2
3
4
5
6
7
8
LeagueIndex

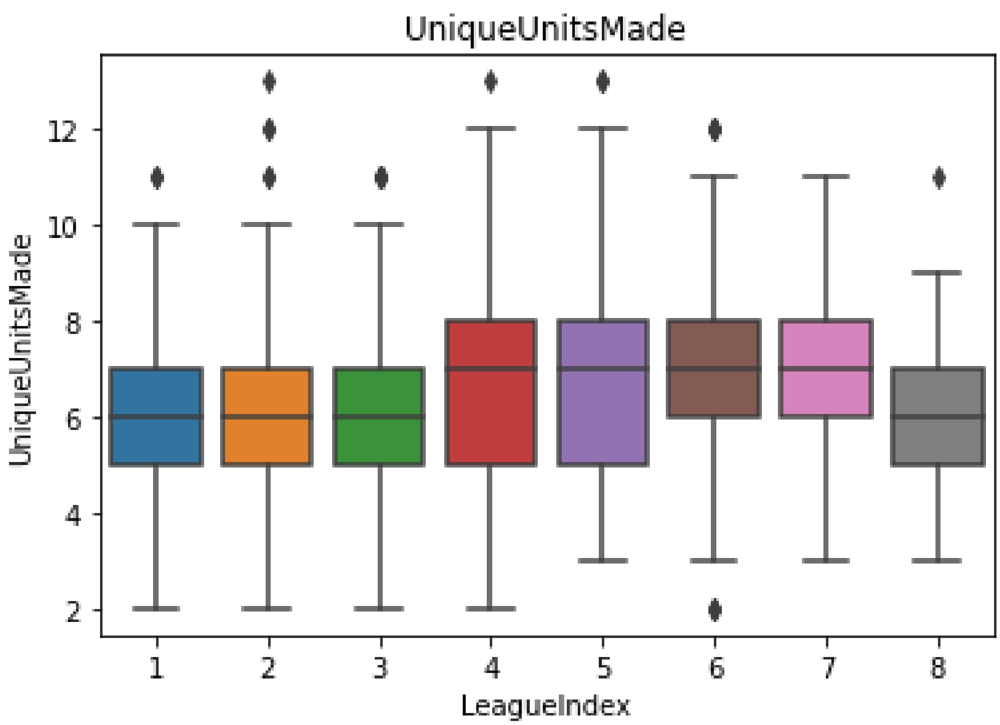

GapBetweenPACs
GapBetweenPACs
0
50
100
150
200
1
2
3
4
5
6
7
8
LeagueIndex

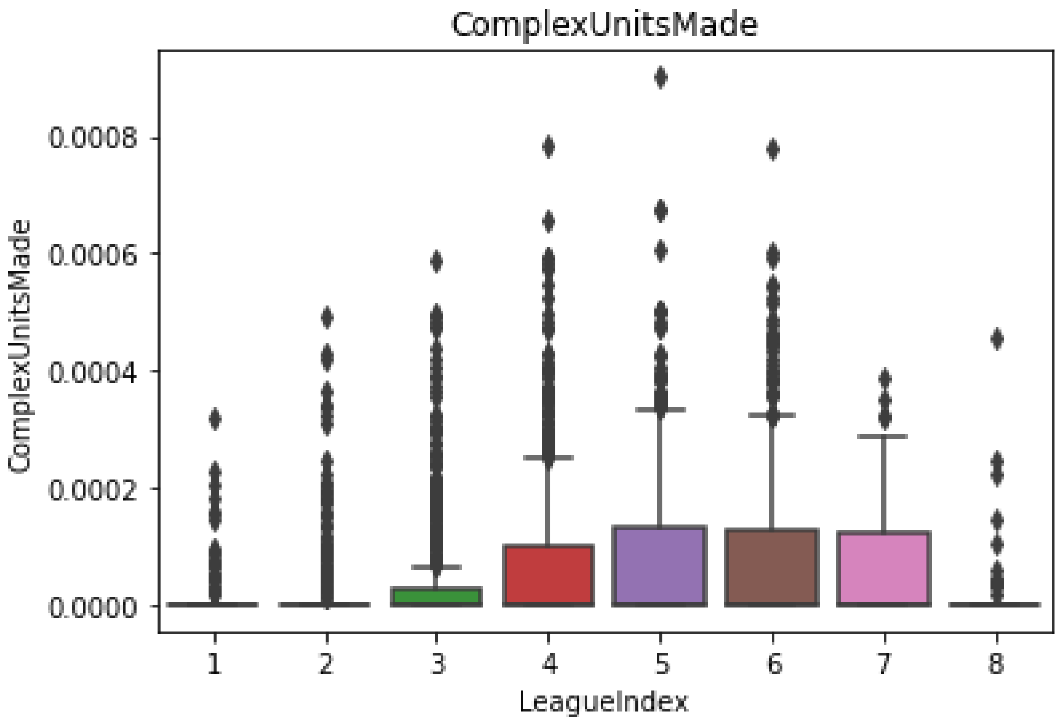

ActionLatency
ActionLatency
20
40
60
80
100
120
140
160
180
1
2
3
4
5
6
7
8
LeagueIndex

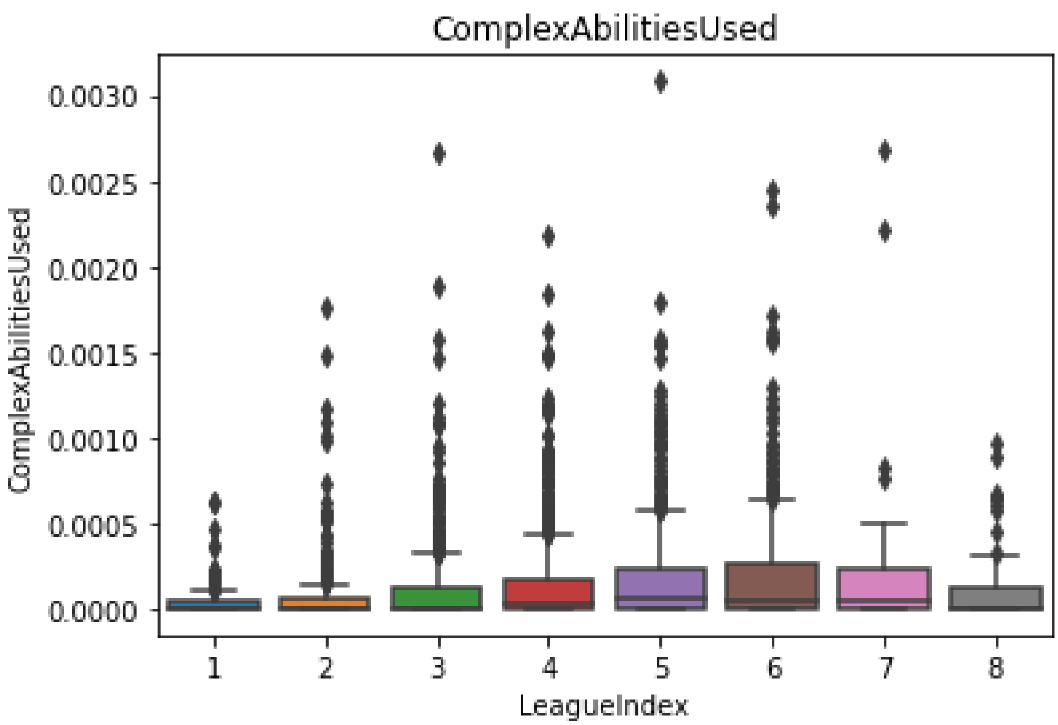

ActionsInPAC
ActionsInPAC
17.5
15.0
12.5
10.0
7.5
5.0
2.5
1 2 3 4 5 6 7 8
LeagueIndex

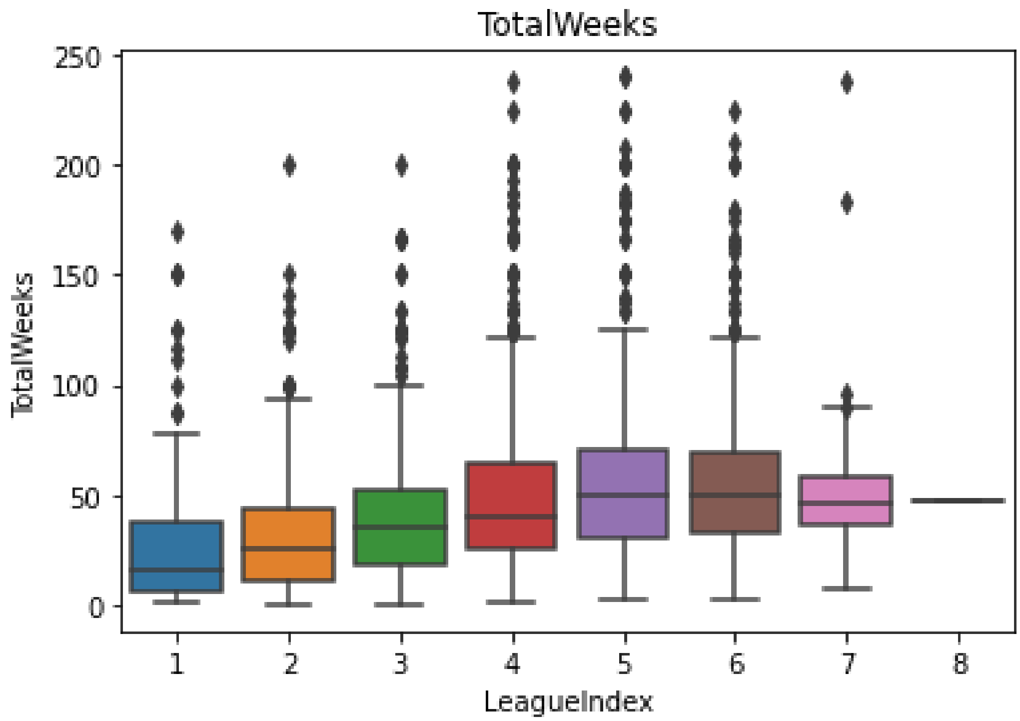

TotalMapExplored
TotalMapExplored
60
50
40
30
20
10
1 2 3 4 5 6 7 8
LeagueIndex


WorkersMade
0.005
0.004
0.003
0.002
0.001
0.000
WorkersMade
1
2
3
4
5
6
7
8
LeagueIndex


UniqueUnitsMade
12
10
8
6
4
2
UniqueUnitsMade
1
2
3
4
5
6
7
8
LeagueIndex


ComplexUnitsMade
ComplexUnitsMade
0.0000
0.0002
0.0004
0.0006
0.0008
1
2
3
4
5
6
7
8
LeagueIndex


ComplexAbilitiesUsed
ComplexAbilitiesUsed
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
1
2
3
4
5
6
7
8
LeagueIndex



5 Hypothetically, if you were to move forward with creating a fully-fledged model to predict LeagueIndex, what model(s) would you consider and why? (Don't actually implement anything!)

Linear regression and SVM may have trouble due to the hyper-overlapping of the players between leagues in terms of the other features in the dataset, as seen in the plotted pair plot. The regression would not accurately place a line through the scatter plot while differentating between leagues.

KNN, Decision Tree, and Random Forest models are popular picks because they are based on robust algorithms that systematically succeed, so I would probably start with them to pick out the league for the similar-looking players.

I did some preliminary research into trying to model the data already for Question 3, and I believe regression makes more sense in terms of accuracy and output than classification (although it depends on the goal).

6 Supplemental model work I did to check my answer for 3

[ ]:
```
# Create quick regression model
import pycaret
from pycaret.regression import *
stp = setup(data = df, target = 'LeagueIndex', train_size = 0.7,
            silent = True, session_id=1)
regression = compare_models(sort='RMSE')
```

```
                                    Model      MAE        MSE     RMSE  \
rf                Random Forest Regressor   0.7387     0.8655   0.9298
catboost               CatBoost Regressor   0.7366     0.8679   0.9308
gbr           Gradient Boosting Regressor   0.7397     0.8705   0.9324
lightgbm  Light Gradient Boosting Machine   0.7449     0.8935   0.9448
et                  Extra Trees Regressor   0.7539     0.8958   0.9458
ada                    AdaBoost Regressor   0.8137     1.0014   0.9997
omp           Orthogonal Matching Pursuit   0.8001     1.0075   1.0024
ridge                    Ridge Regression   0.8031     1.0191   1.0083
xgboost         Extreme Gradient Boosting   0.7984     1.0256   1.0122
br                         Bayesian Ridge   0.8105     1.0300   1.0137
en                            Elastic Net   0.8140     1.0340   1.0156
lasso                    Lasso Regression   0.8165     1.0375   1.0173
huber                     Huber Regressor   0.8984     1.2596   1.1209
knn                 K Neighbors Regressor   0.9911     1.5087   1.2260
lr                      Linear Regression   0.9820     1.5462   1.2385
dt                Decision Tree Regressor   1.0048     1.8290   1.3508
llar         Lasso Least Angle Regression   1.1832     2.0940   1.4462
dummy                     Dummy Regressor   1.1832     2.0940   1.4462
par          Passive Aggressive Regressor   1.5935     5.4453   1.9526
lar                Least Angle Regression  24.7343  8690.7077  30.3600

              R2   RMSLE    MAPE  TT (Sec)
rf        0.5840  0.2171  0.2502     2.057
catboost  0.5830  0.2171  0.2476     6.604
gbr       0.5816  0.2176  0.2502     0.783
lightgbm  0.5699  0.2194  0.2485     0.441
et        0.5697  0.2220  0.2592     1.401
ada       0.5196  0.2299  0.2713     0.299
omp       0.5170  0.2347  0.2774     0.018
ridge     0.5116  0.2352  0.2789     0.017
xgboost   0.5063  0.2372  0.2683     1.125
br        0.5063  0.2358  0.2809     0.022
en        0.5046  0.2366  0.2823     0.019
lasso     0.5030  0.2371  0.2834     0.019
huber     0.3951  0.2513  0.2996     0.108
knn       0.2783  0.2789  0.3428     0.070
lr        0.2601  0.2801  0.3233     0.665
```

```
dt           0.1188  0.3164  0.3215     0.041
llar        -0.0045  0.3318  0.4399     0.020
dummy       -0.0045  0.3318  0.4399     0.016
par         -1.4330  0.3844  0.4374     0.026
lar      -3643.0527  0.5748  7.8539     0.027


INFO:logs:create_model_container: 20
INFO:logs:master_model_container: 20
INFO:logs:display_container: 2
INFO:logs:RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=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=100, n_jobs=-1, oob_score=False,
                      random_state=1, verbose=0, warm_start=False)
INFO:logs:compare_models() succesfully
completed…
```

[ ]:
```
# Create quick regression model
import pycaret
from pycaret.regression import *
stp = setup(data = X, target = 'LeagueIndex', train_size = 0.7,
            silent = True, session_id=1)
regression = compare_models(sort='RMSE')
```

```
                                    Model     MAE      MSE    RMSE       R2  \
gbr           Gradient Boosting Regressor  0.7882   0.9669  0.9825   0.5357
catboost               CatBoost Regressor  0.7831   0.9691  0.9835   0.5348
rf                Random Forest Regressor  0.8004   0.9913  0.9950   0.5238
et                  Extra Trees Regressor  0.8016   0.9998  0.9992   0.5196
lightgbm  Light Gradient Boosting Machine  0.7999   1.0097  1.0041   0.5150
xgboost         Extreme Gradient Boosting  0.8456   1.1221  1.0584   0.4607
ada                    AdaBoost Regressor  0.8668   1.1300  1.0624   0.4576
lr                      Linear Regression  0.8515   1.1450  1.0685   0.4514
lar                Least Angle Regression  0.8515   1.1450  1.0685   0.4514
ridge                    Ridge Regression  0.9074   1.2908  1.1342   0.3819
br                         Bayesian Ridge  0.9111   1.2984  1.1375   0.3784
huber                     Huber Regressor  0.9110   1.3252  1.1489   0.3655
en                            Elastic Net  0.9421   1.3739  1.1700   0.3426
omp           Orthogonal Matching Pursuit  0.9406   1.3716  1.1700   0.3423
lasso                    Lasso Regression  0.9443   1.3767  1.1712   0.3413
knn                 K Neighbors Regressor  0.9716   1.4742  1.2131   0.2916
dt                Decision Tree Regressor  1.0583   1.9681  1.4015   0.0549
llar         Lasso Least Angle Regression  1.1832   2.0940  1.4462  -0.0045
dummy                     Dummy Regressor  1.1832   2.0940  1.4462  -0.0045
par          Passive Aggressive Regressor  3.1032  46.3379  4.3808 -21.0325
```

```
          RMSLE    MAPE  TT (Sec)
gbr       0.2280  0.2668     0.412
catboost  0.2276  0.2620     4.750
rf        0.2310  0.2716     1.530
et        0.2326  0.2741     0.881
lightgbm  0.2325  0.2689     0.181
xgboost   0.2447  0.2827     0.721
ada       0.2444  0.2940     0.200
lr        0.2485  0.2984     0.316
lar       0.2485  0.2984     0.017
ridge     0.2633  0.3234     0.014
br        0.2639  0.3247     0.017
huber     0.2658  0.3276     0.077
en        0.2689  0.3351     0.018
omp       0.2699  0.3343     0.014
lasso     0.2696  0.3361     0.018
knn       0.2755  0.3337     0.064
dt        0.3237  0.3404     0.031
llar      0.3318  0.4399     0.015
dummy     0.3318  0.4399     0.012
par       0.5537  0.8274     0.019
```

```
INFO:logs:create_model_container: 20
INFO:logs:master_model_container: 20
INFO:logs:display_container: 2
INFO:logs:GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0,
criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          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=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=1, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
INFO:logs:compare_models() succesfully
completed…
```

[ ]:
```
evaluate_model(regression)
```

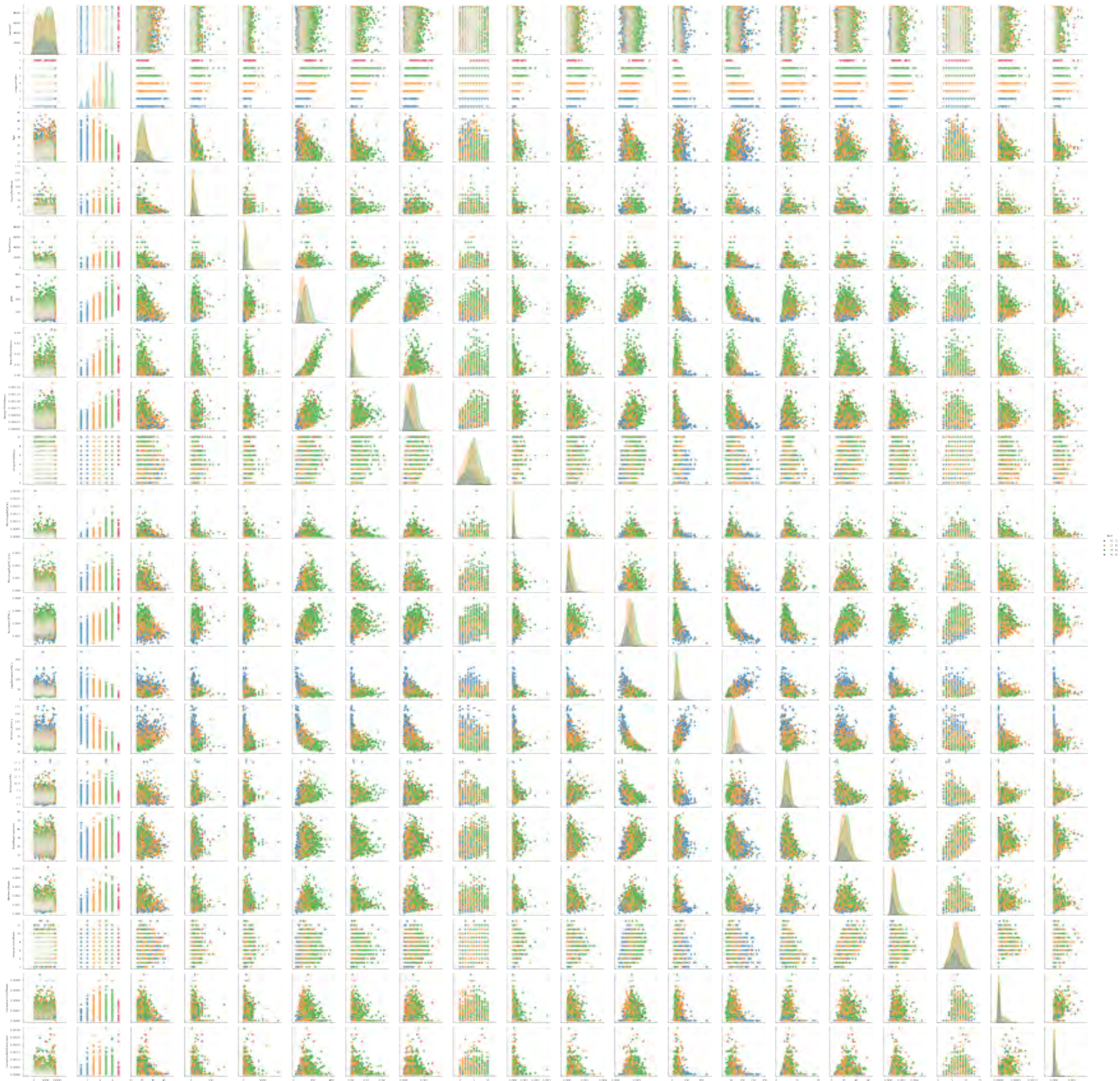
```
INFO:logs:Initializing evaluate_model()
INFO:logs:evaluate_model(estimator=RandomForestRegressor(bootstrap=True,
ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
```

```
                              min_samples_split=2, min_weight_fraction_leaf=0.0,
                              n_estimators=100, n_jobs=-1, oob_score=False,
                              random_state=1, verbose=0, warm_start=False), fold=None,
fit_kwargs=None, plot_kwargs=None, feature_name=None, groups=None,
use_train_data=False)
```

```
interactive(children=(ToggleButtons(description='Plot Type:', icons=('',), options=(('Hyperpara
```

[ ]:
```python
# Pair plot of df features
df['Skill'] = pd.cut(df['LeagueIndex'], [0, 2, 4, 6, 8] )
sns.pairplot(df, hue = 'Skill')
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f5cde72dc90>
```



[ ]:
```
# Interpret regression model weights

# !pip3 install shap
# import shap
# shap.initjs()
interpret_model(regression, plot='reason', observation=1)
```

```
INFO:logs:Initializing interpret_model()
INFO:logs:interpret_model(estimator=RandomForestRegressor(bootstrap=True,
ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=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=100, n_jobs=-1, oob_score=False,
                      random_state=1, verbose=0, warm_start=False),
use_train_data=False, X_new_sample=None, y_new_sample=None, feature=None,
kwargs={}, observation=32, plot=reason, save=False)
INFO:logs:Checking exceptions
INFO:logs:plot type: reason
INFO:logs:model type detected: type 2
INFO:logs:Creating TreeExplainer
INFO:logs:Compiling shap values

<IPython.core.display.HTML object>
```

[ ]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x7f1499f7ce10>
```

```
INFO:logs:Visual Rendered Successfully
INFO:logs:interpret_model() succesfully
completed…
```

[ ]:
```
# Create quick classification model
import pycaret
from pycaret.classification import *
stp = setup(data = df , target = 'LeagueIndex', train_size = 0.7,

            silent = True, session_id=1)
classification = compare_models()
```

```
                                    Model  Accuracy     AUC  Recall   Prec.  \
catboost              CatBoost Classifier    0.4022  0.7648  0.3287  0.3997
lr                    Logistic Regression    0.3997  0.7593  0.3153  0.4116
rf               Random Forest Classifier    0.3996  0.7660  0.3169  0.4001
lightgbm  Light Gradient Boosting Machine    0.3984  0.7580  0.3284  0.4027
gbc          Gradient Boosting Classifier    0.3872  0.7597  0.3271  0.3864
et                 Extra Trees Classifier    0.3817  0.7503  0.2913  0.3766
lda          Linear Discriminant Analysis    0.3670  0.7520  0.3314  0.3618
```

```
ridge                  Ridge Classifier    0.3533  0.0000  0.2454  0.3394
dt             Decision Tree Classifier    0.3113  0.5723  0.2815  0.3136
svm                 SVM - Linear Kernel    0.2818  0.0000  0.2178  0.2713
knn              K Neighbors Classifier    0.2671  0.6092  0.2095  0.2727
nb                          Naive Bayes    0.2667  0.6866  0.3328  0.2997
ada                Ada Boost Classifier    0.2633  0.6047  0.2816  0.2550
dummy                  Dummy Classifier    0.2436  0.5000  0.1429  0.0593
qda     Quadratic Discriminant Analysis    0.1544  0.4970  0.1428  0.1790

              F1   Kappa     MCC  TT (Sec)
catboost  0.3967  0.2523  0.2531    26.468
lr        0.3916  0.2444  0.2475     1.394
rf        0.3943  0.2466  0.2476     0.836
lightgbm  0.3929  0.2443  0.2455     1.092
gbc       0.3833  0.2346  0.2352     7.852
et        0.3734  0.2221  0.2235     0.684
lda       0.3571  0.2074  0.2099     0.032
ridge     0.3291  0.1755  0.1796     0.018
dt        0.3106  0.1470  0.1473     0.055
svm       0.1878  0.1045  0.1441     0.134
knn       0.2658  0.0868  0.0874     0.127
nb        0.2707  0.1256  0.1279     0.021
ada       0.2085  0.1326  0.1500     0.285
dummy     0.0954  0.0000  0.0000     0.017
qda       0.1513 -0.0057 -0.0059     0.027

INFO:logs:create_model_container: 17
INFO:logs:master_model_container: 17
INFO:logs:display_container: 2
INFO:logs:<catboost.core.CatBoostClassifier object at 0x7fcf2d7787d0>
INFO:logs:compare_models() succesfully
completed…
```

[ ]:
```
evaluate_model(classification)
```

```
                                         Parameters
nan_mode                                        Min
eval_metric                              MultiClass
iterations                                     1000
sampling_frequency                          PerTree
leaf_estimation_method                       Newton
grow_policy                           SymmetricTree
penalties_coefficient                             1
boosting_type                                 Plain
model_shrink_mode                          Constant
feature_border_type                     GreedyLogSum
bayesian_matrix_reg                0.10000000149011612
```

```
eval_fraction                                   0
force_unit_auto_pair_weights                False
l2_leaf_reg                                     3
random_strength                                 1
rsm                                             1
boost_from_average                          False
model_size_reg                                0.5
pool_metainfo_options                {'tags': {}}
use_best_model                              False
class_names                 [1, 2, 3, 4, 5, 6, 7]
random_seed                                     1
depth                                           6
posterior_sampling                          False
border_count                                  254
bagging_temperature                             1
classes_count                                   0
auto_class_weights                           None
sparse_features_conflict_fraction               0
leaf_estimation_backtracking       AnyImprovement
best_model_min_trees                            1
model_shrink_rate                               0
min_data_in_leaf                                1
loss_function                          MultiClass
learning_rate                 0.08261899650096893
score_function                             Cosine
task_type                                     CPU
leaf_estimation_iterations                      1
bootstrap_type                           Bayesian
max_leaves                                     64

INFO:logs:Visual Rendered Successfully
INFO:logs:plot_model() succesfully
completed…
```

[23]:
```
!jupyter nbconvert --to pdf 'Genius League Esports Analyst Assessment.ipynb'
```

```
[NbConvertApp] Converting notebook Genius League Esports Analyst
Assessment.ipynb to pdf
[NbConvertApp] Support files will be in Genius League Esports Analyst
Assessment_files/
[NbConvertApp] Making directory ./Genius League Esports Analyst Assessment_files
[NbConvertApp] Making directory ./Genius League Esports Analyst Assessment_files
[NbConvertApp] Making directory ./Genius League Esports Analyst Assessment_files
[NbConvertApp] Making directory ./Genius League Esports Analyst Assessment_files
[NbConvertApp] Making directory ./Genius League Esports Analyst Assessment_files
[NbConvertApp] Making directory ./Genius League Esports Analyst Assessment_files
[NbConvertApp] Making directory ./Genius League Esports Analyst Assessment_files
```