ASSESSMENT: Evil Geniuses x Genius League: Data Scientist Internship

Author: Uditi Namdev

Importing Libraries

This work is completed as a part of an assessment for Evil Geniuses x Genius League: Data Scientist Internship. The company provided us with csv file of player game data: starcraft_player_data.csv which consists of Starcraft player performance data in ranked games. The objective of this assessment is to predict a player's rank using the information provided in the dataset. I started by importing all the libraries and packages needed for this task.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn as sk
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import AdaBoostClassifier
import xgboost as xgb
from xgboost import XGBClassifier
import plotly.express as px
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

Data Exploration and Cleaning

In [3]:
```
df = pd.read_csv('starcraft_player_data.csv', index_col="GameID")
```

In [4]:
```
df.head()
```

Out[4]:

	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	Min
GameID										
52	5	27	10	3000	143.7180	0.003515	0.000220	7	0.000110	
55	5	23	10	5000	129.2322	0.003304	0.000259	4	0.000294	
56	4	30	10	200	69.9612	0.001101	0.000336	4	0.000294	
57	3	19	20	400	107.6016	0.001034	0.000213	1	0.000053	
58	3	32	10	500	122.8908	0.001136	0.000327	2	0.000000	

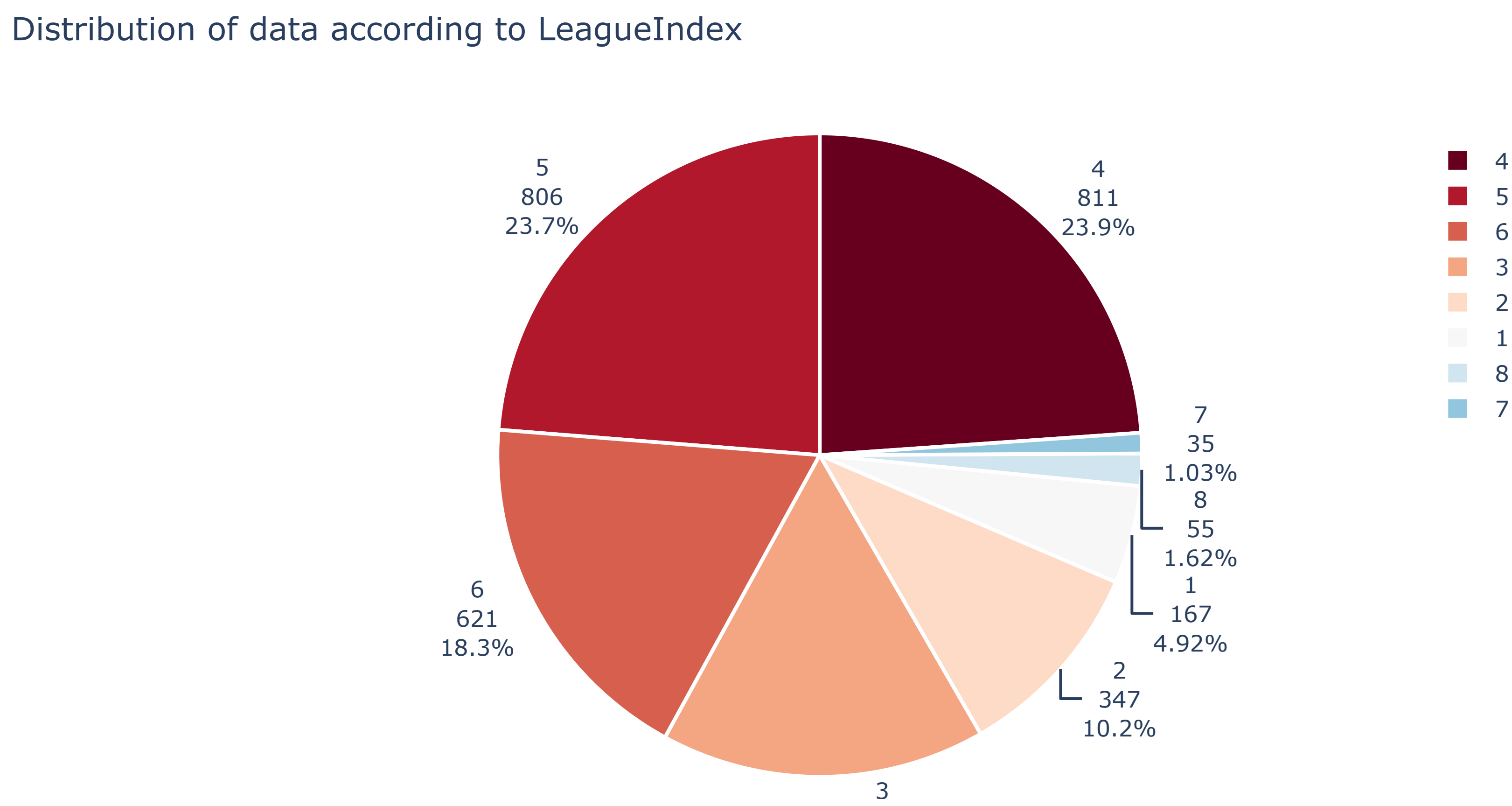
I attempt to observe if the dataset is balanced or imbalanced and notice that the counts of 'LeagueIndex' values which are 7 and 8 differ by a huge margin from the rest of the classes.

In [5]:
```
df['LeagueIndex'].value_counts()
```

Out[5]:
```
4    811
5    806
6    621
3    553
2    347
1    167
8     55
7     35
Name: LeagueIndex, dtype: int64
```

In [6]:

```
classes = [1,2,3,4,5,6,7,8]
num_rows = [167,347,553,811,806,621,35,55]

fig = px.pie(values=num_rows, names=classes,
             color_discrete_sequence=px.colors.sequential.RdBu, title='Distribution of data according to

fig.update_traces(textposition='outside',
                  textinfo='percent+label+value',
                  marker=dict(line=dict(color='#FFFFFF', width=2)),
                  textfont_size=12)

fig.show()
```



I also check if there are any null values in the dataset.

In [7]:
```
df.isnull().sum().sort_values(ascending=False)
```

Out[7]:
```
LeagueIndex             0
NumberOfPACs            0
ComplexUnitsMade        0
UniqueUnitsMade         0
WorkersMade             0
TotalMapExplored        0
ActionsInPAC            0
ActionLatency           0
GapBetweenPACs          0
MinimapRightClicks      0
Age                     0
MinimapAttacks          0
UniqueHotkeys           0
AssignToHotkeys         0
SelectByHotkeys         0
APM                     0
TotalHours              0
HoursPerWeek            0
ComplexAbilitiesUsed    0
dtype: int64
```

I check for the datatypes of all columns and notice that Age, HoursPerWeek, TotalHours are Object datatype and hence can contain some undesirable value other than the required int/float values.

In [8]:
```
df.dtypes
```

Out[8]:
```
LeagueIndex               int64
Age                      object
HoursPerWeek             object
TotalHours               object
APM                     float64
SelectByHotkeys         float64
AssignToHotkeys         float64
UniqueHotkeys             int64
MinimapAttacks          float64
MinimapRightClicks      float64
NumberOfPACs            float64
GapBetweenPACs          float64
ActionLatency           float64
ActionsInPAC            float64
TotalMapExplored          int64
WorkersMade             float64
UniqueUnitsMade           int64
ComplexUnitsMade        float64
ComplexAbilitiesUsed    float64
dtype: object
```

In [9]:
```
df['Age'].value_counts()
```

Out[9]:
```
20    357
21    344
18    325
22    314
19    313
23    259
16    256
17    248
24    225
25    168
26    136
27    111
28     73
?      55
29     52
30     32
31     29
32     21
35     17
33     15
34     15
36      8
38      5
37      5
40      4
41      3
39      3
43      1
44      1
Name: Age, dtype: int64
```

There is an unknown character '?' in our dataset. This will hamper the analysis so it is important to take care of these values by either imputation by mean or median of that column or by dropping those rows which contain these values. Initially I thought that dropping these values is a good idea and hence I dropped all such rows that contained '?'. I proceeded with the analysis in this fashion. But post analysis I realized that this method of dropping the values deleted an entire class of 'LeagueIndex' = 8. Therefore, I concluded that simply dropping the values is a bad approach to tackle this problem.

To overcome this, I had to come up with a way to retain all classes and replace the '?' with a value that was representative of the class. So, I decided to replace all '?' with the mean value of the class 'LeagueIndex' = 7 of the respective columns. I chose 'LeagueIndex' = 7 because it is more similar to the class 'LeagueIndex' = 8 compared to the

In [10]:
```
df[df.eq("?").any(axis=1)].head()
```

Out[10]:

GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	Min
1064	5	17	20	?	94.4724	0.003846	0.000783	3	0.000010	
5255	5	18	?	?	122.2470	0.006357	0.000433	3	0.000014	
10001	8	?	?	?	189.7404	0.004582	0.000655	4	0.000073	
10005	8	?	?	?	287.8128	0.029040	0.001041	9	0.000231	
10006	8	?	?	?	294.0996	0.029640	0.001076	6	0.000302	

In [11]:
```
df_rep = df.copy()
```

In [12]:
```
df_rep.replace('?', 0, inplace=True)
```

In [13]:
```
df_rep[df_rep.eq("?").any(axis=1)].head()
```

Out[13]:

GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	Minima

In [14]:
```
df_rep['Age'] = df_rep['Age'].astype(int)
df_rep['HoursPerWeek'] = df_rep['HoursPerWeek'].astype(int)
df_rep['TotalHours'] = df_rep['TotalHours'].astype(int)
```

In [15]:
```
target_value = 7
cols = ['Age', 'HoursPerWeek', 'TotalHours']
for col in cols:
    imputation_value = df_rep.loc[df_rep['LeagueIndex'] == target_value, col].mean()
    print(imputation_value)
    df_rep[col].replace(0, int(imputation_value), inplace=True)
```

```
21.17142857142857
31.714285714285715
1581.0285714285715
```

I wanted to check how different values in the dataset are distributed, to identify if there were any outliers. Therefore, I first describe the dataset and analyzed it as follows.

In [16]:
```
df_rep.describe().T
```

Out[16]:

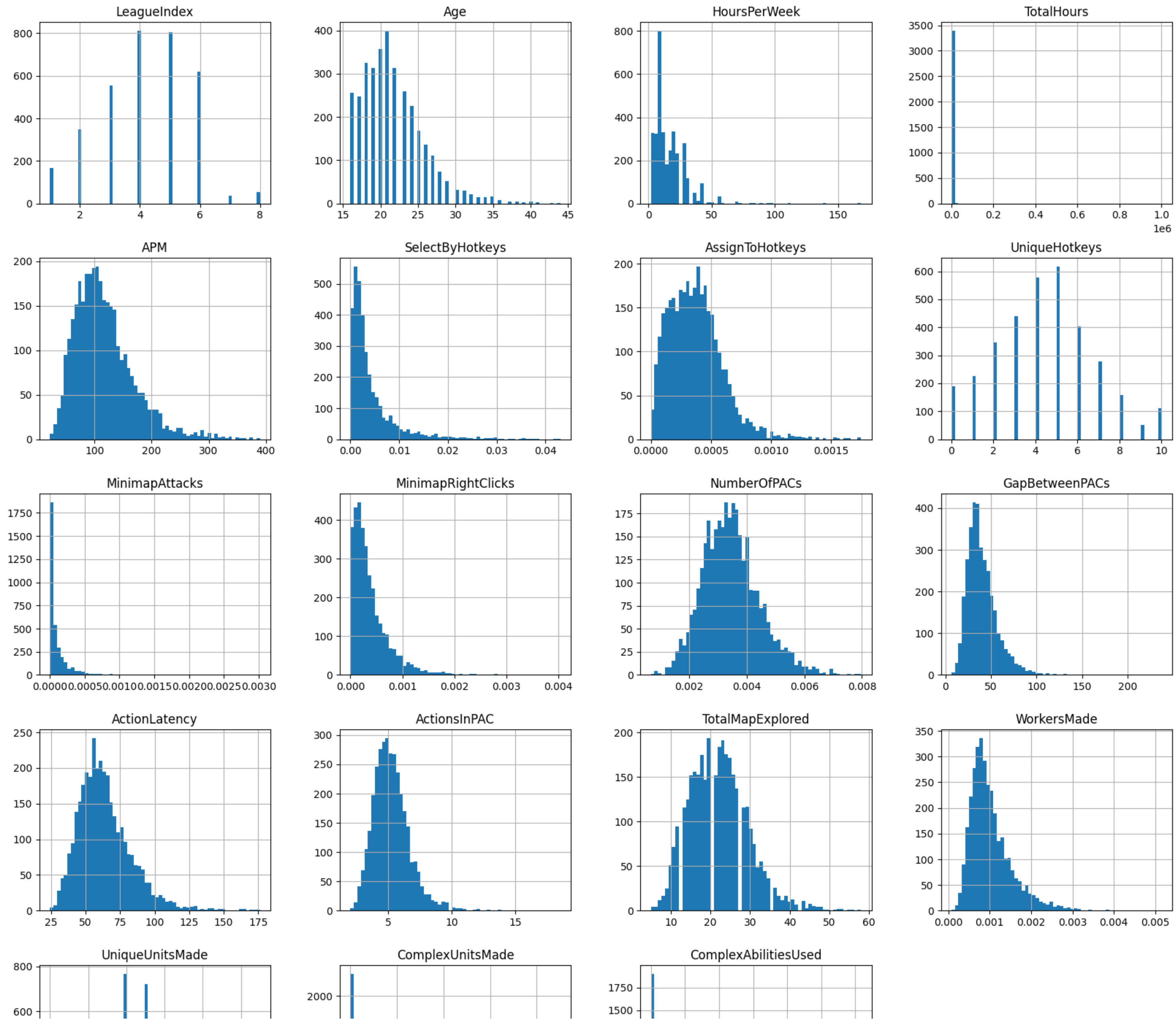
	count	mean	std	min	25%	50%	75%	max
LeagueIndex	3395.0	4.184094	1.517327	1.000000	3.000000	4.000000	5.000000	8.000000
Age	3395.0	21.637408	4.172921	16.000000	19.000000	21.000000	24.000000	44.000000
HoursPerWeek	3395.0	16.168778	12.018012	2.000000	8.000000	12.000000	24.000000	168.000000
TotalHours	3395.0	970.840943	17172.279988	3.000000	300.000000	500.000000	800.000000	1000000.000000
APM	3395.0	117.046947	51.945291	22.059600	79.900200	108.010200	142.790400	389.831400
SelectByHotkeys	3395.0	0.004299	0.005284	0.000000	0.001258	0.002500	0.005133	0.043088
AssignToHotkeys	3395.0	0.000374	0.000225	0.000000	0.000204	0.000353	0.000499	0.001752
UniqueHotkeys	3395.0	4.364654	2.360333	0.000000	3.000000	4.000000	6.000000	10.000000
MinimapAttacks	3395.0	0.000098	0.000166	0.000000	0.000000	0.000040	0.000119	0.003019
MinimapRightClicks	3395.0	0.000387	0.000377	0.000000	0.000140	0.000281	0.000514	0.004041
NumberOfPACs	3395.0	0.003463	0.000992	0.000679	0.002754	0.003395	0.004027	0.007971
GapBetweenPACs	3395.0	40.361562	17.153570	6.666700	28.957750	36.723500	48.290500	237.142900
ActionLatency	3395.0	63.739403	19.238869	24.093600	50.446600	60.931800	73.681300	176.372100
ActionsInPAC	3395.0	5.272988	1.494835	2.038900	4.272850	5.095500	6.033600	18.558100
TotalMapExplored	3395.0	22.131664	7.431719	5.000000	17.000000	22.000000	27.000000	58.000000
WorkersMade	3395.0	0.001032	0.000519	0.000077	0.000683	0.000905	0.001259	0.005149
UniqueUnitsMade	3395.0	6.534021	1.857697	2.000000	5.000000	6.000000	8.000000	13.000000
ComplexUnitsMade	3395.0	0.000059	0.000111	0.000000	0.000000	0.000000	0.000086	0.000902
ComplexAbilitiesUsed	3395.0	0.000142	0.000265	0.000000	0.000000	0.000020	0.000181	0.003084

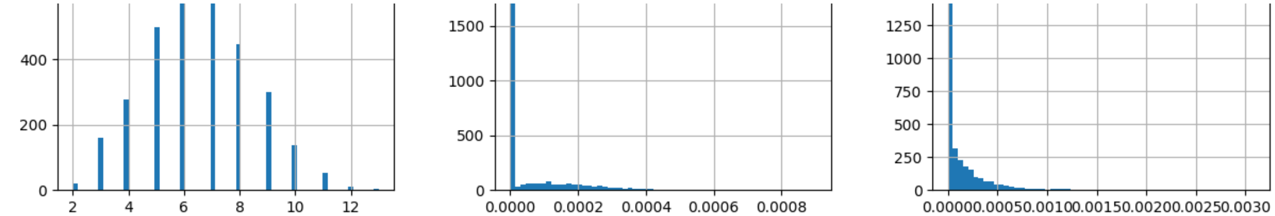
Histogram Plot

I found that there were some values in the dataset which were not in range with the values of the column. For example, the maximum value of TotalHours is 1000000 which is 114 years. Did my grandparents play this game with their friends? :) I found similar discrepancies with HoursPerWeek and other columns.

To fix these outliers, I first plotted the distribution of each column of the data. I found most of them were right-skewed and hence contained outliers.

In [17]:
```
df_rep.hist(bins=60, figsize=(20, 20));
```


LeagueIndex
Age
HoursPerWeek
TotalHours
1e6
APM
SelectByHotkeys
AssignToHotkeys
UniqueHotkeys
MinimapAttacks
MinimapRightClicks
NumberOfPACs
GapBetweenPACs
ActionLatency
ActionsInPAC
TotalMapExplored
WorkersMade
UniqueUnitsMade
ComplexUnitsMade
ComplexAbilitiesUsed



be between -1 and 1.

In [18]:

```
for column in df_rep.columns:
    print(column,' ', df_rep[column].skew())
```

```
LeagueIndex   -0.16335848835584843
Age   1.1686487040977005
HoursPerWeek   2.56248195046941
TotalHours   58.05062784127176
APM   1.2044502448138221
SelectByHotkeys   2.9653288739808947
AssignToHotkeys   1.1413170799516998
UniqueHotkeys   0.20827949675211357
MinimapAttacks   4.8191624735702385
MinimapRightClicks   2.5638199908682635
NumberOfPACs   0.5503794030392924
GapBetweenPACs   1.9082825762546316
ActionLatency   1.1516554826634637
ActionsInPAC   1.5990491180205209
TotalMapExplored   0.6299973302794324
WorkersMade   1.661404465029941
UniqueUnitsMade   0.18832426776415156
ComplexUnitsMade   2.3014040322600615
ComplexAbilitiesUsed   3.778854560804163
```

It can be seen that columns like TotalHours, MinimapAttacks, etc have skewness > 1 meaning they are right-skewed. Ideally all columns that have skewness greater than 1 should be fixed to get a Normal distribution but as this means that a lot of the columns had to be modified, I chose to be liberal with the threshold for skewness and fixed it to 2.

Outliers Detection and Fixation

In [19]:
```
outlier_cols = []
for column in df_rep.columns:
    if (df_rep[column].skew()>2) or (df_rep[column].skew()<-2):
        outlier_cols.append(column)

print(outlier_cols)
```

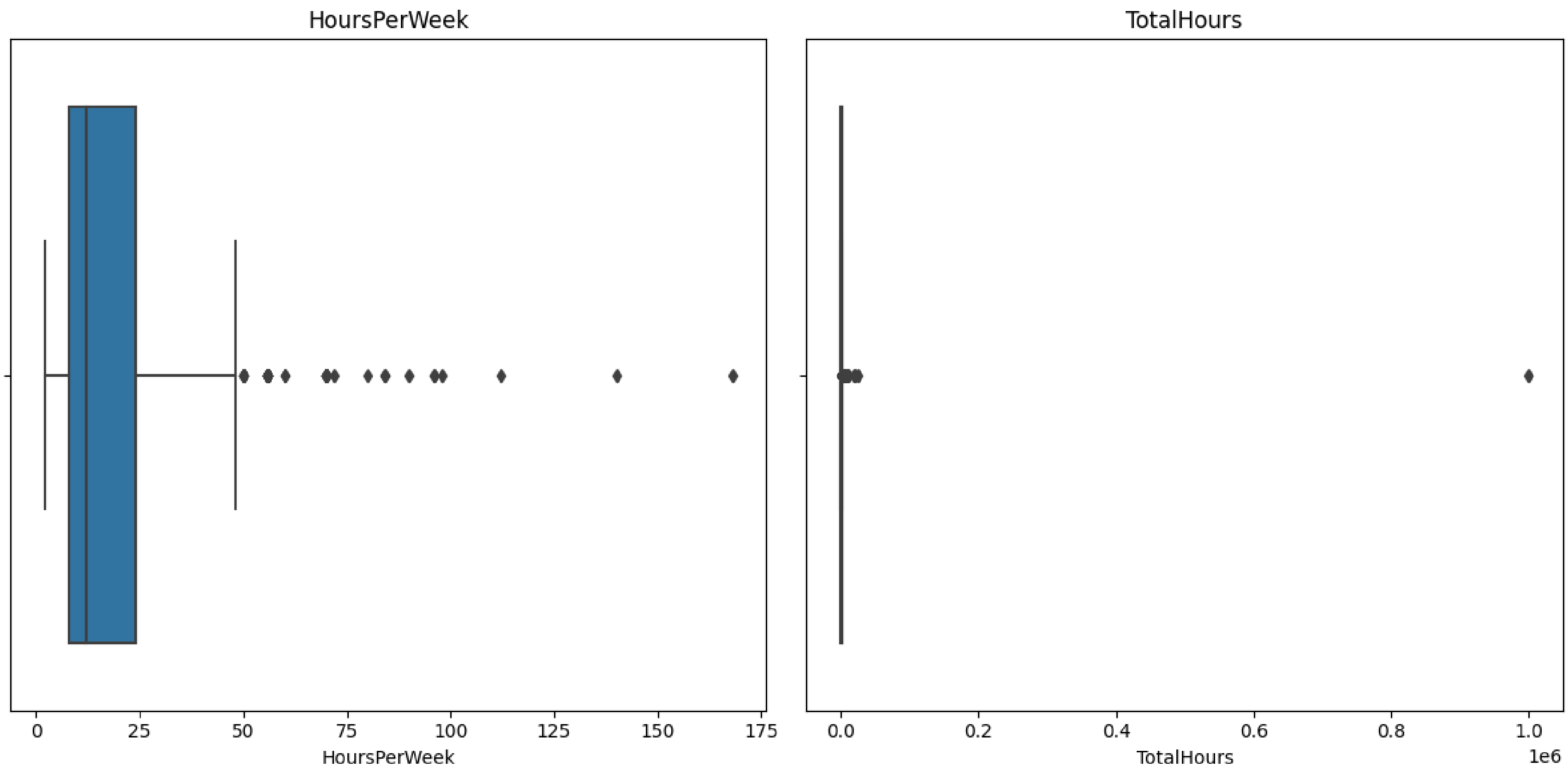
```
['HoursPerWeek', 'TotalHours', 'SelectByHotkeys', 'MinimapAttacks', 'MinimapRightClicks', 'ComplexUnit
sMade', 'ComplexAbilitiesUsed']
```

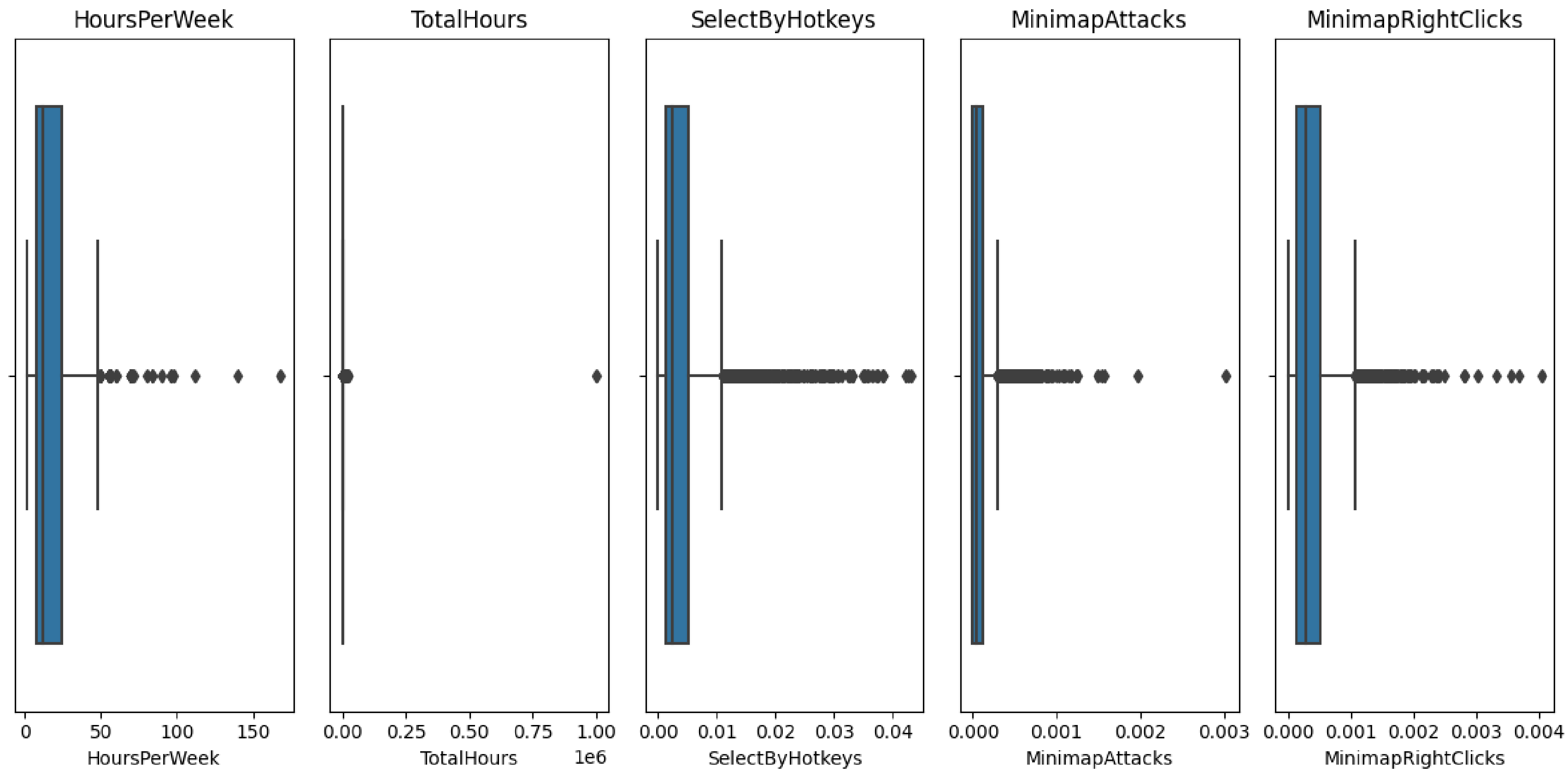
To visualize the distribution before fixing the outliers, I defined a function to generate box-plots.

In [20]:
```
def plot_boxplot(num_plots):
    num_rows = (num_plots // 3) + (num_plots % 2 > 0)
    num_cols = min(num_plots, 2)

    fig, axes = plt.subplots(1, num_plots, figsize=(12, 6))

    for i, column in enumerate(outlier_cols[:num_plots]):
        sns.boxplot(data=df_rep, x=column, ax=axes[i])
        axes[i].set_title(column)

    plt.tight_layout()

    plt.show()
```

Plotting boxplots before fixing outliers:

In [21]:
```
plot_boxplot(len(outlier_cols[:2]))
plot_boxplot(len(outlier_cols[2:]))
```





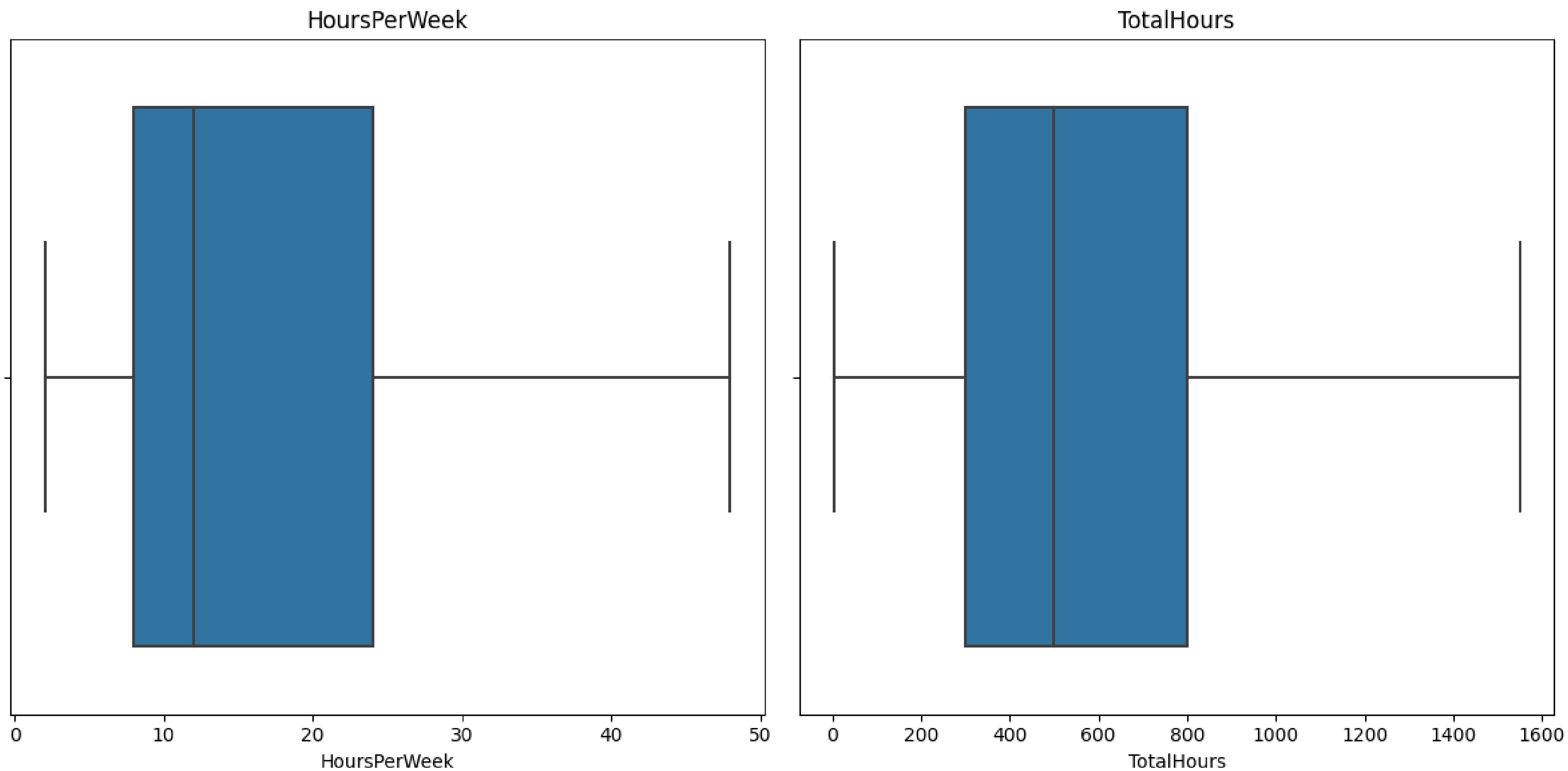
To fix the outlier, we apply the concept of flooring and capping. Flooring and capping is a technique used to fix outliers by setting a lower and upper limit, respectively, to the extreme values in a dataset. It involves replacing values below the floor and above the cap with the floor and cap values, effectively limiting the range of the variable.

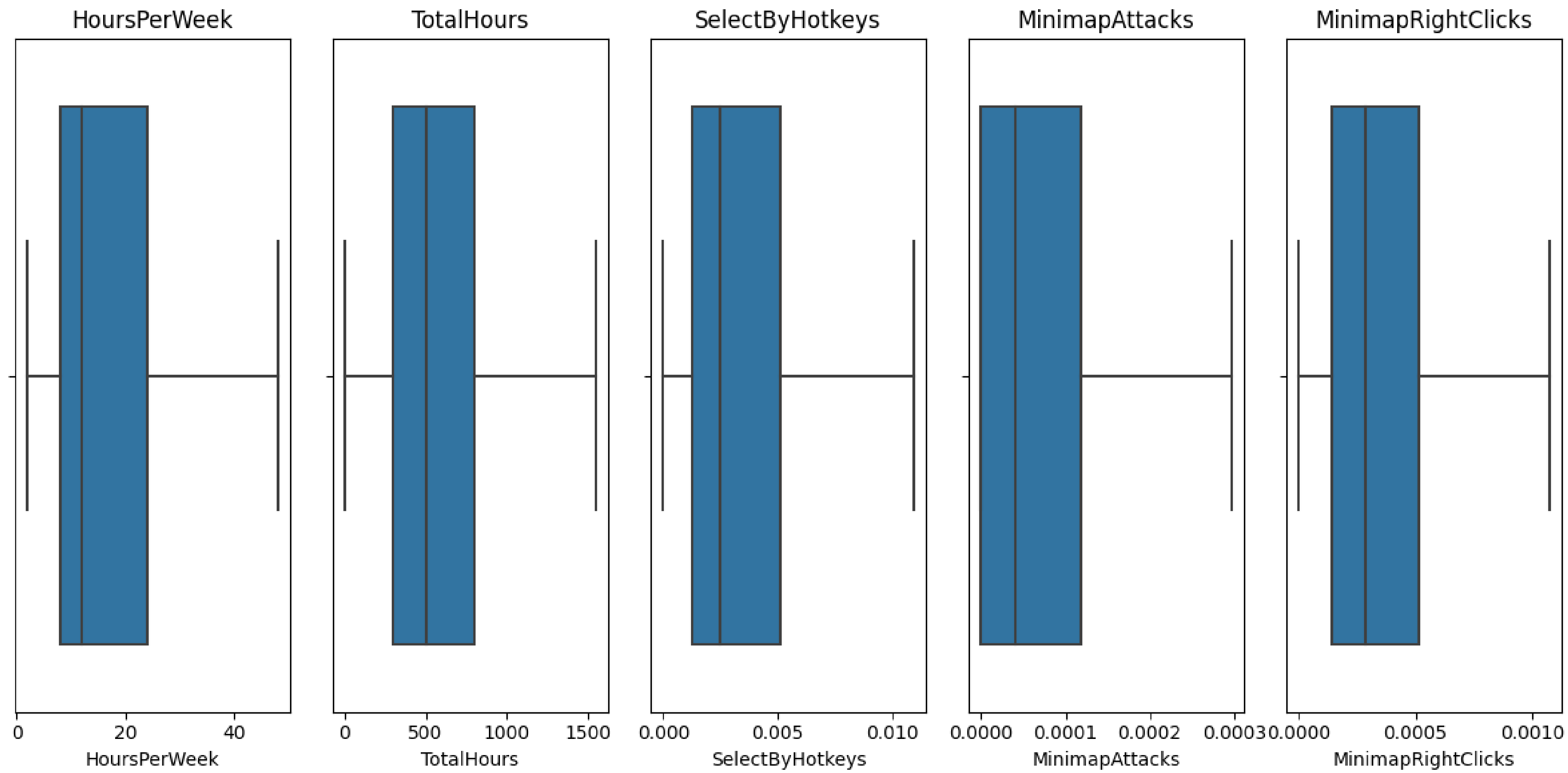
In [22]:
```
def outlier_fixation(column):
    Q1 = df_rep[column].quantile(0.25)
    Q3 = df_rep[column].quantile(0.75)
    IQR = Q3 - Q1
    whisker_width = 1.5
    lower_whisker = Q1 - (whisker_width*IQR)
    upper_whisker = Q3 + (whisker_width*IQR)
    df_rep[column]=np.where(df_rep[column]>upper_whisker,upper_whisker,np.where(df_rep[column]<lower_whi
```

In [23]:
```
for col in outlier_cols:
    outlier_fixation(col)
```

Plotting boxplots after fixing outliers:
(The difference in the boxplots are evident)

In [24]:
```
plot_boxplot(len(outlier_cols[:2]))
plot_boxplot(len(outlier_cols[2:]))
```



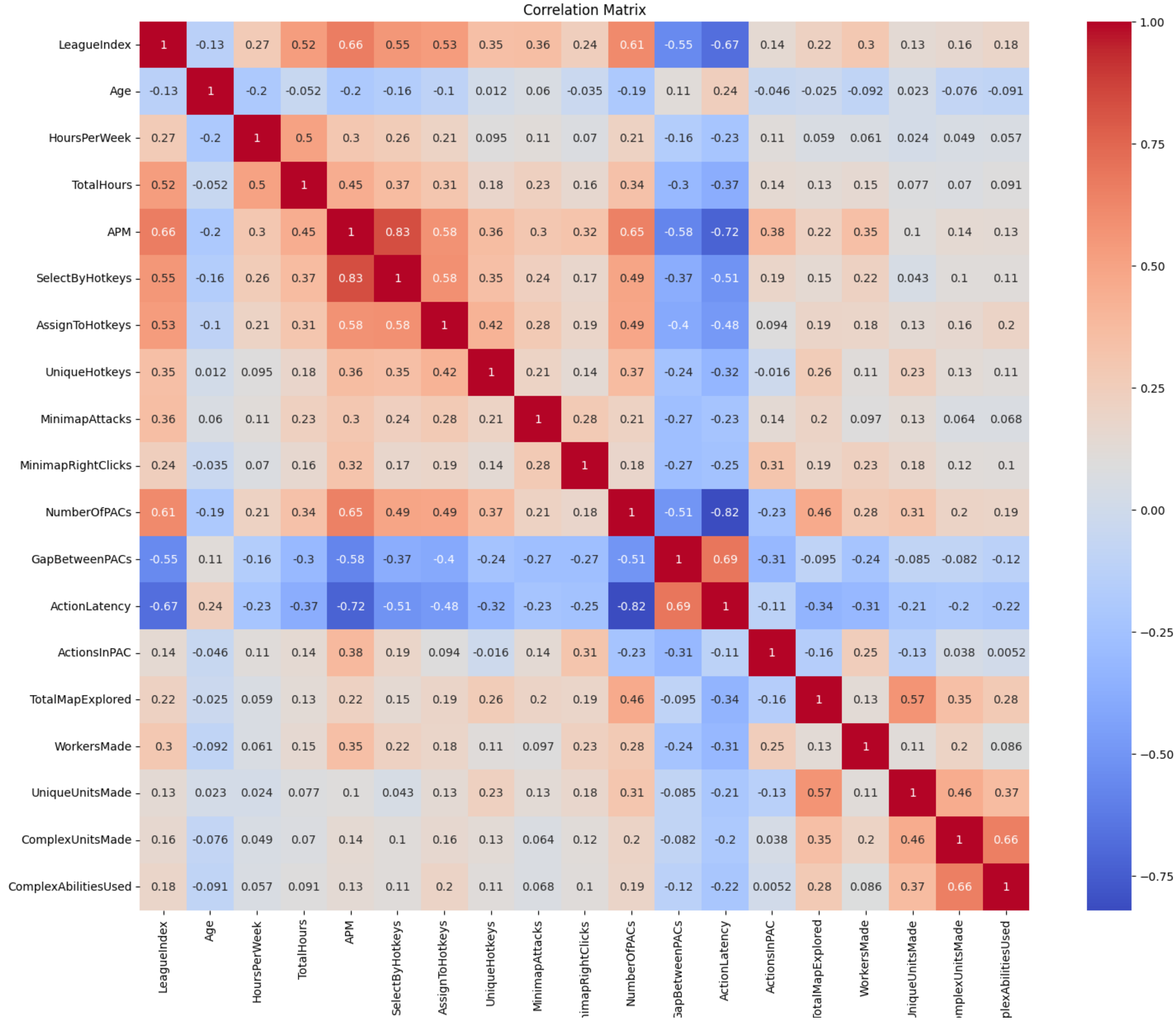


Correlation Matrix of all features

Before proceeding with modelling phase, I wanted to check how correlated each feature is with the target variable: 'LeagueIndex'.

In [25]:
```
correlation_matrix = df_rep.corr()
```

In [26]:
```
plt.figure(figsize=(18, 14))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=True)
plt.title('Correlation Matrix')
plt.show()
```


Correlation Matrix
LeagueIndex 1 -0.13 0.27 0.52 0.66 0.55 0.53 0.35 0.36 0.24 0.61 -0.55 -0.67 0.14 0.22 0.3 0.13 0.16 0.18
Age -0.13 1 -0.2 -0.052 -0.2 -0.16 -0.1 0.012 0.06 -0.035 -0.19 0.11 0.24 -0.046 -0.025 -0.092 0.023 -0.076 -0.091
HoursPerWeek 0.27 -0.2 1 0.5 0.3 0.26 0.21 0.095 0.11 0.07 0.21 -0.16 -0.23 0.11 0.059 0.061 0.024 0.049 0.057
TotalHours 0.52 -0.052 0.5 1 0.45 0.37 0.31 0.18 0.23 0.16 0.34 -0.3 -0.37 0.14 0.13 0.15 0.077 0.07 0.091
APM 0.66 -0.2 0.3 0.45 1 0.83 0.58 0.36 0.3 0.32 0.65 -0.58 -0.72 0.38 0.22 0.35 0.1 0.14 0.13
SelectByHotkeys 0.55 -0.16 0.26 0.37 0.83 1 0.58 0.35 0.24 0.17 0.49 -0.37 -0.51 0.19 0.15 0.22 0.043 0.1 0.11
AssignToHotkeys 0.53 -0.1 0.21 0.31 0.58 0.58 1 0.42 0.28 0.19 0.49 -0.4 -0.48 0.094 0.19 0.18 0.13 0.16 0.2
UniqueHotkeys 0.35 0.012 0.095 0.18 0.36 0.35 0.42 1 0.21 0.14 0.37 -0.24 -0.32 -0.016 0.26 0.11 0.23 0.13 0.11
MinimapAttacks 0.36 0.06 0.11 0.23 0.3 0.24 0.28 0.21 1 0.28 0.21 -0.27 -0.23 0.14 0.2 0.097 0.13 0.064 0.068
MinimapRightClicks 0.24 -0.035 0.07 0.16 0.32 0.17 0.19 0.14 0.28 1 0.18 -0.27 -0.25 0.31 0.19 0.23 0.18 0.12 0.1
NumberOfPACs 0.61 -0.19 0.21 0.34 0.65 0.49 0.49 0.37 0.21 0.18 1 -0.51 -0.82 -0.23 0.46 0.28 0.31 0.2 0.19
GapBetweenPACs -0.55 0.11 -0.16 -0.3 -0.58 -0.37 -0.4 -0.24 -0.27 -0.27 -0.51 1 0.69 -0.31 -0.095 -0.24 -0.085 -0.082 -0.12
ActionLatency -0.67 0.24 -0.23 -0.37 -0.72 -0.51 -0.48 -0.32 -0.23 -0.25 -0.82 0.69 1 -0.11 -0.34 -0.31 -0.21 -0.2 -0.22
ActionsInPAC 0.14 -0.046 0.11 0.14 0.38 0.19 0.094 -0.016 0.14 0.31 -0.23 -0.31 -0.11 1 -0.16 0.25 -0.13 0.038 0.0052
TotalMapExplored 0.22 -0.025 0.059 0.13 0.22 0.15 0.19 0.26 0.2 0.19 0.46 -0.095 -0.34 -0.16 1 0.13 0.57 0.35 0.28
WorkersMade 0.3 -0.092 0.061 0.15 0.35 0.22 0.18 0.11 0.097 0.23 0.28 -0.24 -0.31 0.25 0.13 1 0.11 0.2 0.086
UniqueUnitsMade 0.13 0.023 0.024 0.077 0.1 0.043 0.13 0.23 0.13 0.18 0.31 -0.085 -0.21 -0.13 0.57 0.11 1 0.46 0.37
ComplexUnitsMade 0.16 -0.076 0.049 0.07 0.14 0.1 0.16 0.13 0.064 0.12 0.2 -0.082 -0.2 0.038 0.35 0.2 0.46 1 0.66
ComplexAbilitiesUsed 0.18 -0.091 0.057 0.091 0.13 0.11 0.2 0.11 0.068 0.1 0.19 -0.12 -0.22 0.0052 0.28 0.086 0.37 0.66 1
LeagueIndex
Age
HoursPerWeek
TotalHours
APM
SelectByHotkeys
AssignToHotkeys
UniqueHotkeys
MinimapAttacks
imapRightClicks
NumberOfPACs
apBetweenPACs
ActionLatency
ActionsInPAC
otalMapExplored
WorkersMade
niqueUnitsMade
mplexUnitsMade
plexAbilitiesUsed
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

There is clear correlation of certain columns and I thought it was better to only use those columns for the training which had a correlation value greater than the threshold. After observing the matrix, I decided the threshold to be 0.3 and found all the important columns.

In [27]:
```
threshold = 0.3
target_column = 'LeagueIndex'
correlated_columns = correlation_matrix[correlation_matrix[target_column].abs() >= threshold].index.toli
noncorrelated_columns = correlation_matrix[correlation_matrix[target_column].abs() < threshold].index.tc
```

In [28]:
```
correlated_columns
```

Out[28]:
```
['LeagueIndex',
 'TotalHours',
 'APM',
 'SelectByHotkeys',
 'AssignToHotkeys',
 'UniqueHotkeys',
 'MinimapAttacks',
 'NumberOfPACs',
 'GapBetweenPACs',
 'ActionLatency']
```

In [29]:
```
noncorrelated_columns
```

Out[29]:
```
['Age',
 'HoursPerWeek',
 'MinimapRightClicks',
 'ActionsInPAC',
 'TotalMapExplored',
 'WorkersMade',
 'UniqueUnitsMade',
 'ComplexUnitsMade',
 'ComplexAbilitiesUsed']
```

Data Modeling and Evaluation

In [30]:
```
dataset = df_rep
```

Splitting the dataset into test and train sets after dropping the uncorrelated columns.

In [31]:
```
X = dataset.drop(noncorrelated_columns, axis=1)
X = X.drop('LeagueIndex', axis=1)
names = X.columns
X = X.to_numpy()
Y = dataset['LeagueIndex'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=42, shuffle=True, test_size=0.2,
```

Scaling the data to reduce the variability in the values.

In [32]:
```
scl = sk.preprocessing.StandardScaler()
X_train_scaled = scl.fit_transform(X_train)
X_test_scaled = scl.transform(X_test)
```

I defined a function to train different ML models and generate a classification report consisting of the various metrics like Accuracy, MSE, RMSE, R^2.

In [33]:
```
def train_and_evaluate_classifier(classifier, X_train, y_train, X_test, y_test):
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    MSE = np.mean((y_test - y_pred)**2)
    RMSE = np.sqrt(MSE)
    R_squared = 1 - np.sum((y_test - y_pred)**2) / np.sum((y_test - np.mean(y_test))**2)
    print("Classification Report for",classifier)
    print(classification_report(y_test, y_pred))
    print('MSE: ', MSE)
    print('RMSE: ', RMSE)
    print('R²: ', R_squared)
    print('================================================================')
    print(' ')
```

In [34]:
```
lgr_clf = LogisticRegression(max_iter=10000)
rf_clf = RandomForestClassifier(n_estimators=1000)
ada_clf = AdaBoostClassifier()
xgb_clf = xgb.XGBClassifier()

train_and_evaluate_classifier(lgr_clf, X_train_scaled, y_train, X_test_scaled, y_test)
train_and_evaluate_classifier(rf_clf, X_train, y_train, X_test, y_test)
train_and_evaluate_classifier(ada_clf, X_train_scaled, y_train, X_test_scaled, y_test)
train_and_evaluate_classifier(xgb_clf, X_train_scaled, y_train, X_test_scaled, y_test)
```

```
Classification Report for LogisticRegression(max_iter=10000)
              precision    recall  f1-score   support

           1       0.54      0.42      0.47        33
           2       0.34      0.29      0.31        70
           3       0.34      0.23      0.27       111
           4       0.40      0.57      0.47       162
           5       0.42      0.40      0.41       161
           6       0.59      0.62      0.61       124
           7       0.00      0.00      0.00         7
           8       1.00      0.55      0.71        11

    accuracy                           0.44       679
   macro avg       0.45      0.38      0.41       679
weighted avg       0.44      0.44      0.43       679

MSE:  0.9646539027982327
RMSE:  0.982167960584254
R²:  0.5804925671596741
===============================================================

Classification Report for RandomForestClassifier(n_estimators=1000)
              precision    recall  f1-score   support

           1       0.50      0.36      0.42        33
           2       0.40      0.31      0.35        70
           3       0.38      0.39      0.38       111
           4       0.40      0.46      0.43       162
           5       0.40      0.40      0.40       161
           6       0.57      0.61      0.59       124
           7       0.00      0.00      0.00         7
           8       1.00      0.64      0.78        11

    accuracy                           0.44       679
   macro avg       0.46      0.40      0.42       679
weighted avg       0.44      0.44      0.44       679

MSE:  1.0456553755522828
RMSE:  1.0225729194303372
R²:  0.545266752188349
===============================================================

Classification Report for AdaBoostClassifier()
```

```
              precision    recall  f1-score   support

           1       0.50      0.03      0.06        33
           2       0.21      0.83      0.33        70
           3       0.20      0.05      0.09       111
           4       0.37      0.40      0.38       162
           5       0.37      0.14      0.20       161
           6       0.50      0.48      0.49       124
           7       0.20      0.14      0.17         7
           8       0.88      0.64      0.74        11

    accuracy                           0.32       679
   macro avg       0.40      0.34      0.31       679
weighted avg       0.36      0.32      0.29       679

MSE:  1.9322533136966127
RMSE:  1.390055147717749
R²:  0.1597041955931181
===============================================================

Classification Report for XGBClassifier(objective='multi:softprob')
              precision    recall  f1-score   support

           1       0.41      0.39      0.40        33
           2       0.36      0.26      0.30        70
           3       0.30      0.29      0.29       111
           4       0.37      0.41      0.39       162
           5       0.40      0.41      0.40       161
           6       0.59      0.64      0.61       124
           7       0.00      0.00      0.00         7
           8       0.88      0.64      0.74        11

    accuracy                           0.42       679
   macro avg       0.41      0.38      0.39       679
weighted avg       0.41      0.42      0.41       679

MSE:  1.1325478645066274
RMSE:  1.0642123211590004
R²:  0.5074790597645639
===============================================================
```

I observed that the accuracy was not significant and I believe this is because the dataset is highly imbalanced especially for the 'LeagueIndex' = 7 and 8. To make it balanced, I did oversampling using Synthetic Minority Oversampling Technique (SMOTE) which is a statistical technique for increasing the number of cases in the dataset in a balanced way.

In [35]:
```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, Y)
```

This makes all the classes balanced and the number of instances in each class is equal to the maximum no. of instances in any class.

In [36]:
```
unique, counts = np.unique(y_resampled, return_counts=True)
class_distribution = dict(zip(unique, counts))
print("Class Distribution:", class_distribution)
```

```
Class Distribution: {1: 811, 2: 811, 3: 811, 4: 811, 5: 811, 6: 811, 7: 811, 8: 811}
```

The splitting of data is carried out again using the sampled data.

In [37]:
```
X_resampled_train, X_resampled_test, y_resampled_train, y_resampled_test = train_test_split(X_resampled,
                                                    random_state=42,
                                                    shuffle=True,
                                                    test_size=0.2, stratify = y_resampled)
scl = sk.preprocessing.StandardScaler()
X_resampled_train_scaled = scl.fit_transform(X_resampled_train)
X_resampled_test_scaled = scl.transform(X_resampled_test)
```

Hyperparameter Tuning

I also wanted to do hyperparameter tuning and hence I proceeded with GridSearchCV but this did not have any major impact on the accuracy score of the AdaBoost model.

In [38]:
```
ada_clf = AdaBoostClassifier()

param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.1, 0.5, 1.0]
}

grid_search = GridSearchCV(ada_clf, param_grid, cv=5)
grid_search.fit(X_resampled_train_scaled, y_resampled_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
Best Parameters: {'learning_rate': 0.5, 'n_estimators': 100}
Best Score: 0.37341040462427744
```

In [39]:
```
xgb_clf = XGBClassifier()

param_grid = {
    'n_estimators': [200, 500, 1000],
    'learning_rate': [0.5, 1.0]
}

grid_search = GridSearchCV(xgb_clf, param_grid, cv=5)
grid_search.fit(X_resampled_train_scaled, y_resampled_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
Best Parameters: {'learning_rate': 0.5, 'n_estimators': 1000}
Best Score: 0.6263969171483621
```

Classification Report after Oversampling

In [40]:
```
lgr_clf = LogisticRegression(max_iter=10000)
rf_clf = RandomForestClassifier(n_estimators=1000)
ada_clf = AdaBoostClassifier(learning_rate=0.1, n_estimators=200)
xgb_clf = xgb.XGBClassifier(learning_rate=0.5, n_estimators=1000)

train_and_evaluate_classifier(lgr_clf, X_resampled_train_scaled, y_resampled_train, X_resampled_test_sca
train_and_evaluate_classifier(rf_clf, X_resampled_train, y_resampled_train, X_resampled_test, y_resample
train_and_evaluate_classifier(ada_clf, X_resampled_train_scaled, y_resampled_train, X_resampled_test_sca
train_and_evaluate_classifier(xgb_clf, X_resampled_train_scaled, y_resampled_train, X_resampled_test_sca
```

```
Classification Report for LogisticRegression(max_iter=10000)
              precision    recall  f1-score   support

           1       0.57      0.69      0.62       162
           2       0.32      0.28      0.30       162
           3       0.31      0.31      0.31       162
           4       0.30      0.29      0.30       163
           5       0.35      0.34      0.34       162
           6       0.39      0.37      0.38       162
           7       0.71      0.73      0.72       163
           8       0.86      0.90      0.88       162

    accuracy                           0.49      1298
   macro avg       0.48      0.49      0.48      1298
weighted avg       0.48      0.49      0.48      1298

MSE:  1.0431432973805854
RMSE:  1.0213438683325933
R²:  0.8011892500304014
===============================================================

Classification Report for RandomForestClassifier(n_estimators=1000)
              precision    recall  f1-score   support

           1       0.81      0.91      0.85       162
           2       0.71      0.73      0.72       162
           3       0.56      0.56      0.56       162
           4       0.36      0.33      0.34       163
           5       0.40      0.34      0.37       162
           6       0.63      0.62      0.63       162
           7       0.94      1.00      0.97       163
           8       0.97      1.00      0.98       162

    accuracy                           0.69      1298
   macro avg       0.67      0.69      0.68      1298
weighted avg       0.67      0.69      0.68      1298

MSE:  0.6949152542372882
RMSE:  0.8336157713463008
R²:  0.8675573881295584
===============================================================

Classification Report for AdaBoostClassifier(learning_rate=0.1, n_estimators=200)
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       162
           2       0.30      0.51      0.38       162
           3       0.24      0.20      0.22       162
           4       0.17      0.19      0.18       163
           5       0.35      0.40      0.37       162
           6       0.40      0.51      0.45       162
           7       0.45      0.84      0.59       163
           8       0.90      0.06      0.10       162

    accuracy                           0.34      1298
   macro avg       0.35      0.34      0.29      1298
weighted avg       0.35      0.34      0.29      1298

MSE:  1.4822804314329738
RMSE:  1.2174893968462206
R²:  0.7174949165867743
===============================================================

Classification Report for XGBClassifier(learning_rate=0.5, n_estimators=1000, objective='multi:softpro
b')
              precision    recall  f1-score   support

           1       0.80      0.83      0.81       162
           2       0.62      0.63      0.63       162
           3       0.48      0.49      0.48       162
           4       0.29      0.28      0.29       163
           5       0.36      0.35      0.35       162
           6       0.60      0.55      0.57       162
           7       0.94      0.99      0.97       163
           8       0.96      0.99      0.98       162

    accuracy                           0.64      1298
   macro avg       0.63      0.64      0.64      1298
weighted avg       0.63      0.64      0.64      1298

MSE:  0.8929121725731896
RMSE:  0.9449403010630828
R²:  0.8298215219979581
===============================================================
```

Getting the classification report after sampling the data. This improved the accuracy of Random Forest Classifier from **44%** to **69%** and of XGBClassifier from **42%** to **64%** which is a major change in this case. This implies that there is a possibility of making better predictions if we can collect more and nearly equal amount of data for all the classes.

Hypothetical Question

Question: After seeing your work, your stakeholders come to you and say that they can collect more data, but want your guidance before starting. How would you advise them based on your EDA and model results?

Answer:

Based on the insights I have gained from my analysis, it is clear that artificially generating data has significantly improved the accuracy of our model. However, we must consider that the artificially generated data may not always accurately represent real-life scenarios. Nevertheless, the importance of having a balanced dataset cannot be ignored, as it greatly enhances the performance of the models.

Considering the current situation, it is advisable to gather more data for players whose LeagueIndex lacks sufficient representation. By doing so, we can ensure a more comprehensive and reliable dataset, leading to the development of robust models with improved predictive accuracy.

Before collecting additional data, it would be beneficial to carefully consider the specific attributes and factors that contribute to a player's rank. I have identifed those key variables that influence the prediction which can help in the targeted collection of relevant data. Those variables are the 'correlated columns'.

Additionally, it is essential to ensure the quality and consistency of the collected data. Clear guidelines and standards should be established to maintain data integrity and minimize biases. Furthermore, exploring different data collection methods, such as surveys or additional game metrics, could provide valuable insights into the players' performance and help augment the existing dataset.

Regular monitoring and evaluation of the model's performance should be conducted as new data is added. This will enable us to assess the impact of the additional data and determine if any further adjustments or fine-tuning of the model are necessary.

To conclude, based on the analysis, I would recommend collecting more data for players with insufficient representation,

To explain the key findings to the non-technical stakeholders, I have prepared a PPT.

Thank you!