In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import KNNImputer
from sklearn.metrics import f1_score
```

Exploratory Data Analysis (EDA):

Step 1: Loaded the dataset and examined its structure and dimensions.

Step 2: Checked for missing values and handled them appropriately (e.g., imputation or removal).

Step 3: Explored the distribution and summary statistics of each feature.

Step 4: Visualized the relationships between variables using heat map.

In [2]:
```
# Load the dataset
data = pd.read_csv(r"C:\Users\Nimisha\OneDrive\Desktop\Assessment\starcraft_player_data.csv")

# Display the first few rows of the dataset
data.head()
```

Out[2]:

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks	NumberOfPACs	GapBetweenPACs	ActionLatency	ActionsInPA
0	52	5	27	10	3000	143.7180	0.003515	0.000220	7	0.000110	0.000392	0.004849	32.6677	40.8673	4.75
1	55	5	23	10	5000	129.2322	0.003304	0.000259	4	0.000294	0.000432	0.004307	32.9194	42.3454	4.84
2	56	4	30	10	200	69.9612	0.001101	0.000336	4	0.000294	0.000461	0.002926	44.6475	75.3548	4.04
3	57	3	19	20	400	107.6016	0.001034	0.000213	1	0.000053	0.000543	0.003783	29.2203	53.7352	4.91
4	58	3	32	10	500	122.8908	0.001136	0.000327	2	0.000000	0.001329	0.002368	22.6885	62.0813	9.37

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3395 entries, 0 to 3394
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   GameID                3395 non-null   int64  
 1   LeagueIndex           3395 non-null   int64  
 2   Age                   3395 non-null   object 
 3   HoursPerWeek          3395 non-null   object 
 4   TotalHours            3395 non-null   object 
 5   APM                   3395 non-null   float64
 6   SelectByHotkeys       3395 non-null   float64
 7   AssignToHotkeys       3395 non-null   float64
 8   UniqueHotkeys         3395 non-null   int64  
 9   MinimapAttacks        3395 non-null   float64
 10  MinimapRightClicks    3395 non-null   float64
 11  NumberOfPACs          3395 non-null   float64
 12  GapBetweenPACs        3395 non-null   float64
 13  ActionLatency         3395 non-null   float64
 14  ActionsInPAC          3395 non-null   float64
 15  TotalMapExplored      3395 non-null   int64  
 16  WorkersMade           3395 non-null   float64
 17  UniqueUnitsMade       3395 non-null   int64  
 18  ComplexUnitsMade      3395 non-null   float64
 19  ComplexAbilitiesUsed  3395 non-null   float64
dtypes: float64(12), int64(5), object(3)
memory usage: 530.6+ KB
```

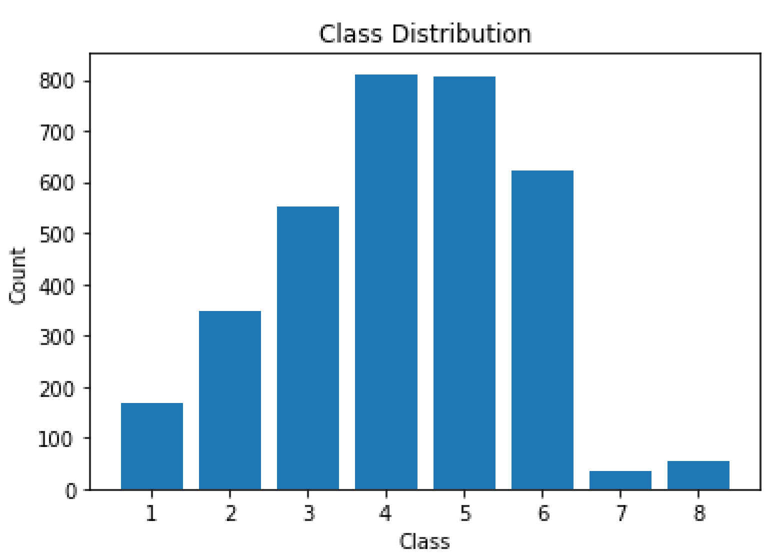
In [4]:
```
# Check the shape of the dataset
print("Shape of the dataset:", data.shape)

# Check for missing values
print("Missing values:\n", data.isna().sum())

# Summary statistics
print("Summary statistics:\n", data.describe())
```

```
Shape of the dataset: (3395, 20)
Missing values:
 GameID                  0
LeagueIndex             0
Age                     0
HoursPerWeek            0
TotalHours              0
APM                     0
SelectByHotkeys         0
AssignToHotkeys         0
UniqueHotkeys           0
MinimapAttacks          0
MinimapRightClicks      0
NumberOfPACs            0
GapBetweenPACs          0
ActionLatency           0
ActionsInPAC            0
TotalMapExplored        0
WorkersMade             0
UniqueUnitsMade         0
ComplexUnitsMade        0
ComplexAbilitiesUsed    0
dtype: int64
Summary statistics:
              GameID  LeagueIndex          APM  SelectByHotkeys  \
count   3395.000000  3395.000000  3395.000000      3395.000000   
mean    4805.012371     4.184094   117.046947         0.004299   
std     2719.944851     1.517327    51.945291         0.005284   
min       52.000000     1.000000    22.059600         0.000000   
25%     2464.500000     3.000000    79.900200         0.001258   
50%     4874.000000     4.000000   108.010200         0.002500   
75%     7108.500000     5.000000   142.790400         0.005133   
max    10095.000000     8.000000   389.831400         0.043088   

       AssignToHotkeys  UniqueHotkeys  MinimapAttacks  MinimapRightClicks  \
count      3395.000000    3395.000000     3395.000000         3395.000000   
mean          0.000374       4.364654        0.000098            0.000387   
std           0.000225       2.360333        0.000166            0.000377   
min           0.000000       0.000000        0.000000            0.000000   
25%           0.000204       3.000000        0.000000            0.000140   
50%           0.000353       4.000000        0.000040            0.000281   
75%           0.000499       6.000000        0.000119            0.000514   
max           0.001752      10.000000        0.003019            0.004041   

       NumberOfPACs  GapBetweenPACs  ActionLatency  ActionsInPAC  \
count   3395.000000     3395.000000    3395.000000   3395.000000   
mean       0.003463       40.361562      63.739403      5.272988   
std        0.000992       17.153570      19.238869      1.494835   
min        0.000679        6.666700      24.093600      2.038900   
25%        0.002754       28.957750      50.446600      4.272850   
50%        0.003395       36.723500      60.931800      5.095500   
75%        0.004027       48.290500      73.681300      6.033600   
max        0.007971      237.142900     176.372100     18.558100   

       TotalMapExplored  WorkersMade  UniqueUnitsMade  ComplexUnitsMade  \
count       3395.000000  3395.000000      3395.000000       3395.000000   
mean          22.131664     0.001032         6.534021          0.000059   
std            7.431719     0.000519         1.857697          0.000111   
min            5.000000     0.000077         2.000000          0.000000   
25%           17.000000     0.000683         5.000000          0.000000   
50%           22.000000     0.000905         6.000000          0.000000   
75%           27.000000     0.001259         8.000000          0.000086   
max           58.000000     0.005149        13.000000          0.000902   

       ComplexAbilitiesUsed  
count           3395.000000  
mean               0.000142  
std                0.000265  
min                0.000000  
25%                0.000000  
50%                0.000020  
75%                0.000181  
max                0.003084  
```

In [5]:
```
# Check the distribution of the target variable
print("Distribution of the target variable:\n", data['LeagueIndex'].value_counts())
```

```
Distribution of the target variable:
 4    811
5    806
6    621
3    553
2    347
1    167
8     55
7     35
Name: LeagueIndex, dtype: int64
```

In [6]:
```
class_counts = data['LeagueIndex'].value_counts()
plt.bar(class_counts.index, class_counts.values)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.show()
```



After looking at the dimension and structure of the dataset , I noticed a few important characteristics about the dataset:

1. There are 3 columns described as objects and those are Age, TotalHours and HoursPerWeek. I tried to find the null values in these columns but there are no null values. Instead, they have '?' so it needs to be either removed or imputed. First, we will simply remove all the '?' from the dataset.
2. This is a class imbalance problem which we will address later on. As we can see there are very few data points with LeagueIndex 7 and 8.

Conducted feature selection using correlation analysis and identified relevant features.

In [7]:
```
data['Age'].unique()
```

Out[7]:
```
array(['27', '23', '30', '19', '32', '21', '17', '20', '18', '16', '26',
       '38', '28', '25', '22', '29', '24', '35', '31', '33', '37', '40',
       '34', '43', '41', '36', '44', '39', '?'], dtype=object)
```

In [8]:
```
data['HoursPerWeek'].unique()
```

Out[8]:
```
array(['10', '20', '6', '8', '42', '14', '24', '16', '4', '12', '30',
       '28', '70', '2', '56', '36', '40', '18', '96', '50', '168', '48',
       '84', '0', '72', '112', '90', '32', '98', '140', '?', '80', '60'],
      dtype=object)
```

In [9]:
```
data['TotalHours'].unique()
```

Out[9]:
```
array(['3000', '5000', '200', '400', '500', '70', '240', '10000', '2708',
       '800', '6000', '190', '350', '1000', '1500', '2000', '120', '1100',
       '2520', '700', '160', '150', '250', '730', '230', '300', '100',
       '270', '1200', '30', '600', '540', '280', '1600', '50', '140',
       '900', '550', '625', '1300', '450', '750', '612', '180', '770',
       '720', '415', '1800', '2200', '480', '430', '639', '360', '1250',
       '365', '650', '233', '416', '1825', '780', '1260', '315', '10',
       '312', '110', '1700', '92', '2500', '1400', '220', '999', '303',
       '96', '184', '4000', '420', '60', '2400', '2160', '80', '25',
       '624', '176', '?', '35', '1163', '333', '75', '7', '40', '325',
       '90', '175', '88', '850', '26', '1650', '465', '235', '1350',
       '460', '848', '256', '130', '1466', '670', '711', '1030', '1080',
       '1460', '1050', '20000', '582', '2800', '553', '1008', '330',
       '936', '243', '1320', '425', '1145', '366', '2700', '830', '3',
       '125', '2300', '336', '24', '12', '72', '690', '320', '144', '20',
       '1155', '520', '865', '275', '548', '170', '898', '1170', '1148',
       '105', '575', '1850', '238', '820', '310', '85', '2942', '94',
       '2100', '224', '165', '577', '1440', '731', '727', '138', '45',
       '225', '95', '630', '1274', '1782', '610', '525', '2671', '2016',
       '123', '1095', '1000000', '2920', '640', '1344', '1940', '16',
       '410', '960', '740', '950', '551', '216', '840', '18000', '745',
       '530', '477', '1270', '36', '174', '2600', '1256', '9000', '1880',
       '288', '1150', '10260', '2190', '560', '25000', '128', '666',
       '854', '370', '65', '334', '755', '1024', '3257', '208', '1196',
       '1870', '990', '470', '699', '340', '2250', '255', '980', '620',
       '380', '196', '21', '153', '1098', '546', '433', '1560', '580',
       '77', '148', '2880', '364', '56'], dtype=object)
```

In [10]:
```
data[data['TotalHours']=='?']
```

Out[10]:

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks	NumberOfPACs	GapBetweenPACs	ActionLatency	ActionsI
358	1064	5	17	20	?	94.4724	0.003846	0.000783	3	0.000010	0.000135	0.004474	50.5455	54.9287	3
1841	5255	5	18	?	?	122.2470	0.006357	0.000433	3	0.000014	0.000257	0.003043	30.8929	62.2933	5
3340	10001	8	?	?	?	189.7404	0.004582	0.000655	4	0.000073	0.000618	0.006291	23.5130	32.5665	4
3341	10005	8	?	?	?	287.8128	0.029040	0.001041	9	0.000231	0.000656	0.005399	31.6416	36.1143	4
3342	10006	8	?	?	?	294.0996	0.029640	0.001076	6	0.000302	0.002374	0.006294	16.6393	36.8192	4
3343	10015	8	?	?	?	274.2552	0.018121	0.001264	8	0.000053	0.000975	0.007111	10.6419	24.3556	4
3344	10016	8	?	?	?	274.3404	0.023131	0.000739	8	0.000622	0.003552	0.005355	19.1568	36.3098	5
3345	10017	8	?	?	?	245.8188	0.010471	0.000841	10	0.000657	0.001314	0.005031	14.5518	36.7134	7
3346	10018	8	?	?	?	211.0722	0.013049	0.000940	10	0.000366	0.000909	0.003719	19.6169	38.9326	7
3347	10021	8	?	?	?	189.5778	0.007559	0.000487	10	0.000606	0.000566	0.005821	22.0317	36.7330	4
3348	10022	8	?	?	?	210.5088	0.007974	0.000867	7	0.000548	0.000638	0.006518	15.7856	30.7156	4
3349	10023	8	?	?	?	248.0118	0.014722	0.001752	7	0.000375	0.000110	0.004115	17.4656	34.2357	7
3350	10024	8	?	?	?	299.2290	0.026428	0.000951	10	0.000155	0.000929	0.005443	17.0835	33.7398	5
3351	10025	8	?	?	?	179.9982	0.009524	0.001052	6	0.000000	0.000125	0.003567	32.5628	39.5600	7
3352	10026	8	?	?	?	340.1982	0.028214	0.001242	8	0.000519	0.001163	0.006898	15.2852	26.6907	5
3353	10028	8	?	?	?	319.7148	0.037130	0.000820	5	0.000403	0.000619	0.005208	35.4127	44.0552	4
3354	10029	8	?	?	?	290.5914	0.027561	0.001750	6	0.000022	0.001949	0.005293	22.0126	36.0669	4
3355	10030	8	?	?	?	275.8632	0.019502	0.001449	10	0.000306	0.000386	0.007569	18.1407	24.0936	4
3356	10035	8	?	?	?	298.7916	0.023253	0.000659	4	0.000433	0.000330	0.005561	16.0743	29.2593	5
3357	10036	8	?	?	?	325.1154	0.029790	0.001338	10	0.000059	0.000357	0.005381	15.4571	40.3646	5
3358	10038	8	?	?	?	146.3892	0.006701	0.000400	10	0.000883	0.002384	0.003617	18.4444	47.3364	5
3359	10039	8	?	?	?	192.4554	0.014277	0.000466	4	0.000000	0.001591	0.003142	29.7500	35.7531	7
3360	10041	8	?	?	?	315.6936	0.028311	0.001160	10	0.001242	0.000628	0.005076	17.7035	32.6344	6
3361	10045	8	?	?	?	203.7726	0.008337	0.000573	5	0.000614	0.000757	0.005954	11.3597	31.1615	5
3362	10046	8	?	?	?	334.5240	0.017742	0.001548	6	0.000384	0.004041	0.007780	13.5401	28.2243	5
3363	10047	8	?	?	?	175.5936	0.012680	0.000934	9	0.000098	0.001010	0.005265	27.1322	43.7278	3
3364	10049	8	?	?	?	252.7206	0.019097	0.001522	6	0.000384	0.000569	0.004090	21.6151	38.2256	6
3365	10050	8	?	?	?	211.9188	0.019817	0.000633	4	0.000201	0.000201	0.003912	31.8222	54.5588	5
3366	10051	8	?	?	?	269.8998	0.024645	0.000642	10	0.000415	0.000491	0.004015	25.6352	43.3856	6
3367	10052	8	?	?	?	190.2396	0.008720	0.000879	10	0.000171	0.000342	0.004971	17.9901	35.9509	5
3368	10055	8	?	?	?	212.4972	0.014917	0.000767	10	0.000599	0.000273	0.005648	21.6687	41.2231	4
3369	10059	8	?	?	?	219.3894	0.005926	0.000741	6	0.000440	0.000709	0.005185	17.0456	30.5342	6
3370	10060	8	?	?	?	230.6694	0.010383	0.001242	10	0.000375	0.003328	0.006375	13.5028	31.4044	5
3371	10061	8	?	?	?	284.2296	0.016069	0.000711	9	0.000355	0.000548	0.006680	9.4756	29.6851	5
3372	10062	8	?	?	?	355.3518	0.037526	0.000600	7	0.001242	0.000514	0.004541	9.2871	41.9497	6
3373	10063	8	?	?	?	364.8504	0.042576	0.000996	8	0.000176	0.000146	0.004687	19.9499	41.1417	5
3374	10064	8	?	?	?	256.5888	0.019592	0.000580	8	0.000416	0.000357	0.005812	17.0462	34.3734	5
3375	10065	8	?	?	?	248.4012	0.016018	0.000874	9	0.000388	0.000372	0.005987	16.3144	30.2486	5
3376	10066	8	?	?	?	251.2284	0.022910	0.000946	5	0.001097	0.001173	0.005411	13.7404	35.7203	4
3377	10067	8	?	?	?	318.3000	0.034851	0.000933	7	0.000187	0.000023	0.005225	26.0987	32.4464	4
3378	10068	8	?	?	?	288.9198	0.029322	0.001569	6	0.000118	0.000219	0.005213	23.2857	32.8026	4
3379	10069	8	?	?	?	313.9080	0.019537	0.001214	4	0.000318	0.000607	0.005879	8.1642	26.0918	6
3380	10072	8	?	?	?	243.7134	0.017195	0.000711	6	0.000666	0.000426	0.005594	21.8795	30.5722	5
3381	10073	8	?	?	?	312.9804	0.026327	0.000266	6	0.000000	0.000207	0.005053	14.6118	30.7836	6
3382	10074	8	?	?	?	313.5762	0.030550	0.000560	5	0.000000	0.000206	0.004390	19.5405	35.4094	6
3383	10075	8	?	?	?	274.6194	0.022497	0.000707	6	0.000163	0.000082	0.004053	20.6757	32.7785	6
3384	10076	8	?	?	?	225.0678	0.014339	0.001627	7	0.000291	0.000911	0.005281	16.3502	33.2874	5
3385	10079	8	?	?	?	254.2188	0.016608	0.000788	6	0.000926	0.000330	0.005408	14.9191	35.9921	5
3386	10081	8	?	?	?	339.1524	0.033058	0.001017	10	0.000477	0.000509	0.004609	21.6389	37.1862	6
3387	10082	8	?	?	?	310.0416	0.026873	0.001278	10	0.000319	0.000479	0.005517	16.5446	33.8174	5
3388	10083	8	?	?	?	288.7608	0.024022	0.000628	6	0.000350	0.001051	0.005580	19.0108	30.0866	5
3389	10084	8	?	?	?	151.4046	0.009732	0.000949	6	0.000028	0.000156	0.004363	27.4658	43.8052	4
3390	10089	8	?	?	?	259.6296	0.020425	0.000743	9	0.000621	0.000146	0.004555	18.6059	42.8342	6
3391	10090	8	?	?	?	314.6700	0.028043	0.001157	10	0.000246	0.001083	0.004259	14.3023	36.1156	7
3392	10092	8	?	?	?	299.4282	0.028341	0.000860	7	0.000338	0.000169	0.004439	12.4028	39.5156	6
3393	10094	8	?	?	?	375.8664	0.036436	0.000594	5	0.000204	0.000780	0.004346	11.6910	34.8547	7
3394	10095	8	?	?	?	348.3576	0.029855	0.000811	4	0.000224	0.001315	0.005566	20.0537	33.5142	6

I checked all the three columns with '?' and figured out that TotalHours has the maximum '?' and if we drop its rows then our issue will be resolved because it combines the '?' rows of the other 2 columns as well.

In [11]:
```
data2 = data.drop(data[data['TotalHours'] == '?'].dropna().index)
```

In [12]:
```
data2.head()
```

Out[12]:

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks	NumberOfPACs	GapBetweenPACs	ActionLatency	ActionsInPA
0	52	5	27	10	3000	143.7180	0.003515	0.000220	7	0.000110	0.000392	0.004849	32.6677	40.8673	4.75
1	55	5	23	10	5000	129.2322	0.003304	0.000259	4	0.000294	0.000432	0.004307	32.9194	42.3454	4.84
2	56	4	30	10	200	69.9612	0.001101	0.000336	4	0.000294	0.000461	0.002926	44.6475	75.3548	4.04
3	57	3	19	20	400	107.6016	0.001034	0.000213	1	0.000053	0.000543	0.003783	29.2203	53.7352	4.91
4	58	3	32	10	500	122.8908	0.001136	0.000327	2	0.000000	0.001329	0.002368	22.6885	62.0813	9.37

In [13]:
```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3338 entries, 0 to 3339
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   GameID               3338 non-null   int64  
 1   LeagueIndex          3338 non-null   int64  
 2   Age                  3338 non-null   object 
 3   HoursPerWeek         3338 non-null   object 
 4   TotalHours           3338 non-null   object 
 5   APM                  3338 non-null   float64
 6   SelectByHotkeys      3338 non-null   float64
 7   AssignToHotkeys      3338 non-null   float64
 8   UniqueHotkeys        3338 non-null   int64  
 9   MinimapAttacks       3338 non-null   float64
 10  MinimapRightClicks   3338 non-null   float64
 11  NumberOfPACs         3338 non-null   float64
 12  GapBetweenPACs       3338 non-null   float64
 13  ActionLatency        3338 non-null   float64
 14  ActionsInPAC         3338 non-null   float64
 15  TotalMapExplored     3338 non-null   int64  
 16  WorkersMade          3338 non-null   float64
 17  UniqueUnitsMade      3338 non-null   int64  
 18  ComplexUnitsMade     3338 non-null   float64
 19  ComplexAbilitiesUsed  3338 non-null   float64
dtypes: float64(12), int64(5), object(3)
memory usage: 547.6+ KB
```

In [14]:
```
data2[data2['Age']=='?']
```

Out[14]:

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks	NumberOfPACs	GapBetweenPACs	ActionLatency	ActionsInPAC	T

In [15]:
```
data2[data2['HoursPerWeek']=='?']
```

Out[15]:

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks	NumberOfPACs	GapBetweenPACs	ActionLatency	ActionsInPAC	T

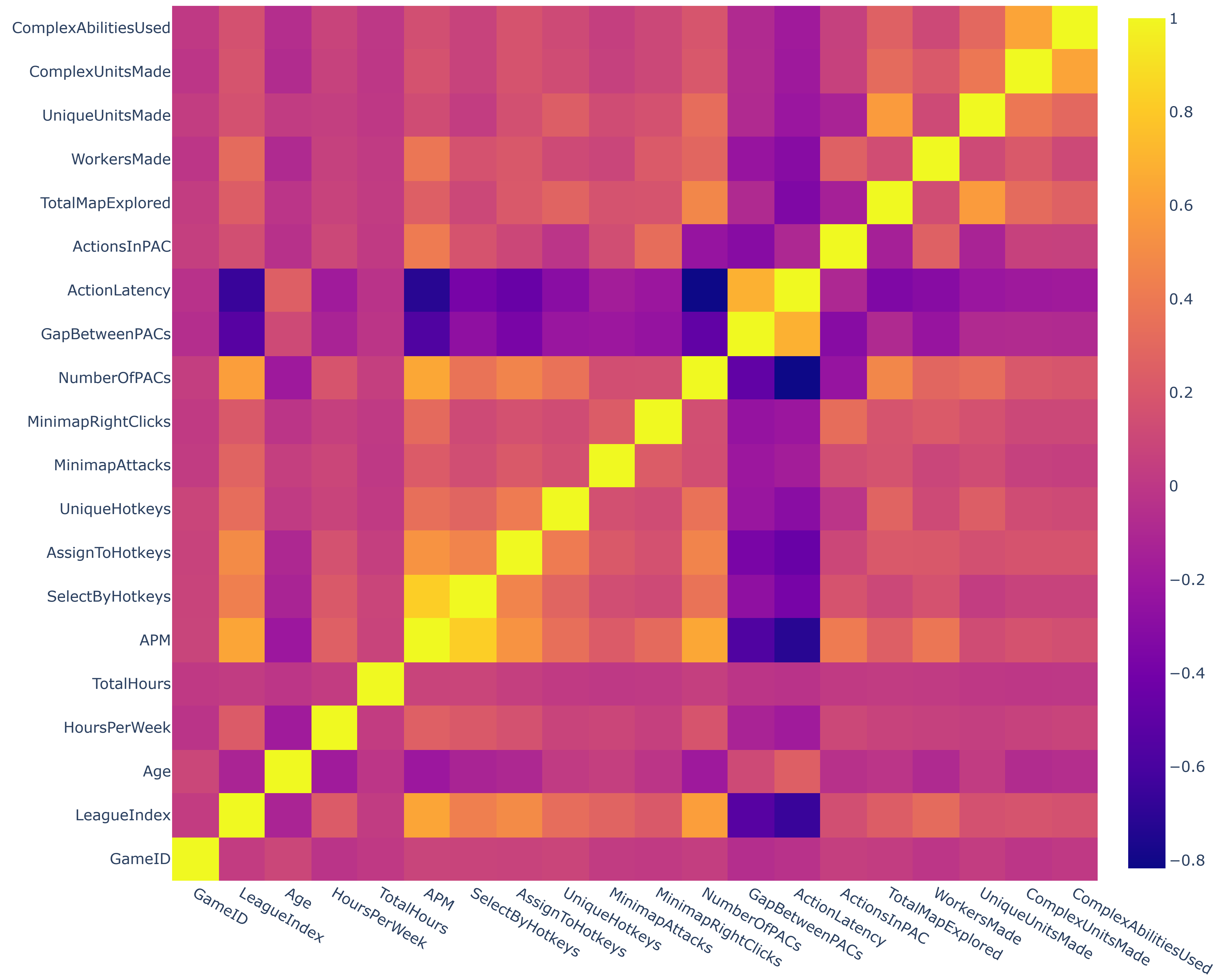
Then I converted all the 3 columns to integer type to find the correlation between the features.

In [16]:
```
#converting them into integer
data2['Age'] = data2['Age'].astype('int64')
data2['HoursPerWeek'] = data2['HoursPerWeek'].astype('int64')
data2['TotalHours'] = data2['TotalHours'].astype('int64')
```

In [17]:
```
data2.isna().sum()
```

Out[17]:
```
GameID                  0
LeagueIndex             0
Age                     0
HoursPerWeek            0
TotalHours              0
APM                     0
SelectByHotkeys         0
AssignToHotkeys         0
UniqueHotkeys           0
MinimapAttacks          0
MinimapRightClicks      0
NumberOfPACs            0
GapBetweenPACs          0
ActionLatency           0
ActionsInPAC            0
TotalMapExplored        0
WorkersMade             0
UniqueUnitsMade         0
ComplexUnitsMade        0
ComplexAbilitiesUsed    0
dtype: int64
```

In [18]:
```
#Then I checked correlation between columns to understand what impact does the other features have on target variable.
correl = data2.corr()

trace = go.Heatmap(z=correl.values,
                   x=correl.index.values,
                   y=correl.columns.values)
data=[trace]
layout = go.Layout(width=1000, height=900)
fig = go.Figure(data=data, layout=layout)
fig.show()
```



In [19]:
```
sorted_corr = correl['LeagueIndex'].sort_values(ascending=False)
sorted_corr
#found the two least correlated columns to LeagueIndex i.e. GameID and TotalHours
```

Out[19]:
```
LeagueIndex             1.000000
APM                     0.624171
NumberOfPACs            0.589193
AssignToHotkeys         0.487280
SelectByHotkeys         0.428637
UniqueHotkeys           0.322415
WorkersMade             0.310452
MinimapAttacks          0.270526
TotalMapExplored        0.230347
HoursPerWeek            0.217930
MinimapRightClicks      0.206380
ComplexUnitsMade        0.171190
ComplexAbilitiesUsed    0.156033
UniqueUnitsMade         0.151933
ActionsInPAC            0.140303
GameID                  0.024974
TotalHours              0.023884
Age                    -0.127518
GapBetweenPACs         -0.537536
ActionLatency          -0.659940
Name: LeagueIndex, dtype: float64
```

Data Preprocessing and Feature Engineering:

Step 1: Split the data into features (X) and the target variable (y) for rank prediction.

Step 2: Scaled the continuous variables using standardization or normalization.

In [20]:
```
# Split the dataset into features and target variable
X = data2.drop('LeagueIndex', axis=1)
y = data2['LeagueIndex']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

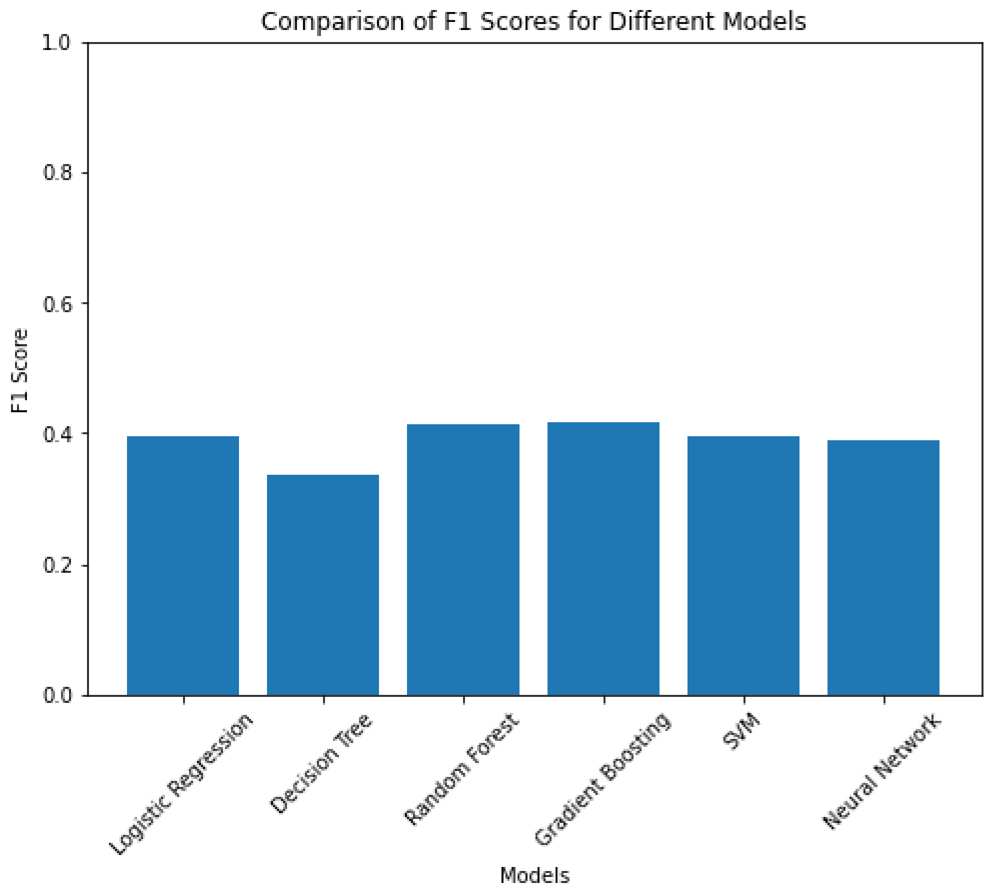
Model Selection, Training and Evaluation:

1. Selected appropriate models for rank prediction, such as logistic regression, decision trees, random forests, gradient boosting, SVM, or Neural Network.
2. Split the data into training and testing sets for model evaluation.
3. Trained the chosen models on the training set.
4. Evaluated the trained models on the testing set using suitable metrics like F1 score. I used F1 score to evaluate the performance instead of accuracy because this is a class imbalance problem.

In [21]:
```
# Create and train different models
models = [
    LogisticRegression(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    GradientBoostingClassifier(),
    SVC(),
    MLPClassifier()
]

model_names = [
    'Logistic Regression',
    'Decision Tree',
    'Random Forest',
    'Gradient Boosting',
    'SVM',
    'Neural Network'
]
scores = []
# Evaluate models and print accuracy
for model, name in zip(models, model_names):
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    f1score = f1_score(y_test, y_pred, average='weighted')
    print(f"{name} f1 Score: {f1score}")
    scores.append(f1score)
```

```
C:\Users\Nimisha\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)


Logistic Regression f1 Score: 0.39682245319806886
Decision Tree f1 Score: 0.3360993928974587
Random Forest f1 Score: 0.4150227632644157
Gradient Boosting f1 Score: 0.4171205649204047
SVM f1 Score: 0.39409664464830657
Neural Network f1 Score: 0.39057724741836686

C:\Users\Nimisha\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning:

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
```

In [22]:
```
# Plotting the F1 scores
plt.figure(figsize=(8, 6))
plt.bar(model_names, scores)
plt.xlabel('Models')
plt.ylabel('F1 Score')
plt.title('Comparison of F1 Scores for Different Models')
plt.xticks(rotation=45)
plt.ylim(0, 1)  # Set the y-axis limit
plt.show()
```



Class Imbalance Problem

Now we will address the class imbalance problem by class weighting. Assign higher weights to the minority class samples or lower weights to the majority class samples during model training. This gives more importance to the minority class during the learning process. I added weights and re-evaluated the decision tree classifier.

In [23]:
```
# Calculate class weights
class_weights = dict(zip(np.unique(y_train), np.bincount(y_train)))

# Create and train the decision tree classifier with class weights
dt_classifier = DecisionTreeClassifier(class_weight = class_weights)
dt_classifier.fit(X_train_scaled, y_train)

# Make predictions on the testing data
y_pred = dt_classifier.predict(X_test_scaled)

# Compute the weighted F1 score
f1score = f1_score(y_test, y_pred, average='weighted')
print("f1 Score:",f1score)
```

```
f1 Score: 0.31632318759935757
```

Removed least correlated columns

In [24]:
```
#Next, we remove the two least correlated columns to LeagueIndex.
data3 = data2.drop(columns=['GameID','TotalHours'])
```

In [25]:
```
# Split the dataset into features and target variable
X = data3.drop('LeagueIndex', axis=1)
y = data3['LeagueIndex']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train different models
models = [
    LogisticRegression(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    GradientBoostingClassifier(),
    SVC(),
    MLPClassifier()
]

model_names = [
    'Logistic Regression',
    'Decision Tree',
    'Random Forest',
    'Gradient Boosting',
    'SVM',
    'Neural Network'
]

# Evaluate models and print accuracy
for model, name in zip(models, model_names):
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    f1score = f1_score(y_test, y_pred, average="weighted")
    print(f"{name} F1 Score: {f1score}")
```

```
C:\Users\Nimisha\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)


Logistic Regression F1 Score: 0.39440696667818664
Decision Tree F1 Score: 0.33661780036159555
Random Forest F1 Score: 0.4178327084659385
Gradient Boosting F1 Score: 0.41165261719928964
SVM F1 Score: 0.3848394721153473
Neural Network F1 Score: 0.35787906559190974

C:\Users\Nimisha\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning:

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
```

K-Nearest Neighbors Classifier

In [26]:
```
knn_model = KNeighborsClassifier(n_neighbors=14)
knn_model.fit(X_train_scaled, y_train)
knn_pred = knn_model.predict(X_test_scaled)
f1score = f1_score(y_test, knn_pred, average="weighted")
print("KNN F1 Score:", f1score)
```

```
KNN F1 Score: 0.3559472932850343
```

Imputation using KNN

Now we will perform imputation. Instead of dropping all the rows with '?', we will fill the missing values through imputation.

In [27]:
```
sampledata = pd.read_csv(r"C:\Users\Nimisha\OneDrive\Desktop\Assessment\starcraft_player_data.csv")
```

In [28]:
```
sampledata[['Age','TotalHours','HoursPerWeek']] = sampledata[['Age','TotalHours','HoursPerWeek']].replace('?', None)
```

In [29]:
```
sampledata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3395 entries, 0 to 3394
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   GameID                3395 non-null   int64  
 1   LeagueIndex           3395 non-null   int64  
 2   Age                   3340 non-null   object 
 3   HoursPerWeek          3339 non-null   object 
 4   TotalHours            3338 non-null   object 
 5   APM                   3395 non-null   float64
 6   SelectByHotkeys       3395 non-null   float64
 7   AssignToHotkeys       3395 non-null   float64
 8   UniqueHotkeys         3395 non-null   int64  
 9   MinimapAttacks        3395 non-null   float64
 10  MinimapRightClicks    3395 non-null   float64
 11  NumberOfPACs          3395 non-null   float64
 12  GapBetweenPACs        3395 non-null   float64
 13  ActionLatency         3395 non-null   float64
 14  ActionsInPAC          3395 non-null   float64
 15  TotalMapExplored      3395 non-null   int64  
 16  WorkersMade           3395 non-null   float64
 17  UniqueUnitsMade       3395 non-null   int64  
 18  ComplexUnitsMade      3395 non-null   float64
 19  ComplexAbilitiesUsed  3395 non-null   float64
dtypes: float64(12), int64(5), object(3)
memory usage: 530.6+ KB
```

In [30]:
```
sampledata.isna().sum()
```

Out[30]:
```
GameID                  0
LeagueIndex             0
Age                    55
HoursPerWeek           56
TotalHours             57
APM                     0
SelectByHotkeys         0
AssignToHotkeys         0
UniqueHotkeys           0
MinimapAttacks          0
MinimapRightClicks      0
NumberOfPACs            0
GapBetweenPACs          0
ActionLatency           0
ActionsInPAC            0
TotalMapExplored        0
WorkersMade             0
UniqueUnitsMade         0
ComplexUnitsMade        0
ComplexAbilitiesUsed    0
dtype: int64
```

In [31]:
```
#imputing the values using knn
missingdata = sampledata[['Age','TotalHours','HoursPerWeek']]
```

In [32]:
```
k = 5
knn_imputer = KNNImputer(n_neighbors=k)
imputed_data = knn_imputer.fit_transform(missingdata)
```

In [33]:
```
df_imputed = pd.DataFrame(imputed_data, columns=missingdata.columns)
```

In [34]:
```
df_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3395 entries, 0 to 3394
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Age           3395 non-null   float64
 1   TotalHours    3395 non-null   float64
 2   HoursPerWeek  3395 non-null   float64
dtypes: float64(3)
memory usage: 79.7 KB
```

In [35]:
```
sampledata[['Age','TotalHours','HoursPerWeek']] = df_imputed[['Age','TotalHours','HoursPerWeek']]
```

In [36]:
```
sampledata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3395 entries, 0 to 3394
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   GameID                3395 non-null   int64  
 1   LeagueIndex           3395 non-null   int64  
 2   Age                   3395 non-null   float64
 3   HoursPerWeek          3395 non-null   float64
 4   TotalHours            3395 non-null   float64
 5   APM                   3395 non-null   float64
 6   SelectByHotkeys       3395 non-null   float64
 7   AssignToHotkeys       3395 non-null   float64
 8   UniqueHotkeys         3395 non-null   int64  
 9   MinimapAttacks        3395 non-null   float64
 10  MinimapRightClicks    3395 non-null   float64
 11  NumberOfPACs          3395 non-null   float64
 12  GapBetweenPACs        3395 non-null   float64
 13  ActionLatency         3395 non-null   float64
 14  ActionsInPAC          3395 non-null   float64
 15  TotalMapExplored      3395 non-null   int64  
 16  WorkersMade           3395 non-null   float64
 17  UniqueUnitsMade       3395 non-null   int64  
 18  ComplexUnitsMade      3395 non-null   float64
 19  ComplexAbilitiesUsed  3395 non-null   float64
dtypes: float64(15), int64(5)
memory usage: 530.6 KB
```

In [37]:
```
sampledata.isna().sum()
```

Out[37]:
```
GameID                  0
LeagueIndex             0
Age                     0
HoursPerWeek            0
TotalHours              0
APM                     0
SelectByHotkeys         0
AssignToHotkeys         0
UniqueHotkeys           0
MinimapAttacks          0
MinimapRightClicks      0
NumberOfPACs            0
GapBetweenPACs          0
ActionLatency           0
ActionsInPAC            0
TotalMapExplored        0
WorkersMade             0
UniqueUnitsMade         0
ComplexUnitsMade        0
ComplexAbilitiesUsed    0
dtype: int64
```

In [38]:
```
# Split the dataset into features and target variable
X = sampledata.drop('LeagueIndex', axis=1)
y = sampledata['LeagueIndex']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
X_train
X_test
y_train
y_test
```

Out[38]:
```
2076    6
2854    3
570     1
2821    6
744     5
       ..
2370    5
462     6
655     2
166     6
2625    6
Name: LeagueIndex, Length: 679, dtype: int64
```

In [39]:
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
f1score = f1_score(y_test, rf_pred,average= "weighted")
print("Random Forest F1 Score:", f1score)
```

```
Random Forest F1 Score: 0.4111486816805072
```

Finally, let's address the hypothetical scenario where stakeholders want to collect more data and seek guidance. Based on the EDA and model results, I would suggest the following:

1. Collect more samples for the minority classes: Since the dataset is imbalanced, collecting more data for the underrepresented rank levels can improve the model's performance.
2. Gather additional features: If there are relevant features that are not present in the current dataset, collecting additional data with those features can enhance the model's predictive power.
3. Monitor data quality: Ensure that the new data collection process maintains data quality standards, such as avoiding missing values, outliers, or inconsistencies.
4. Perform iterative model updates: As more data becomes available, it's beneficial to periodically update and retrain the model using the augmented dataset to capture any evolving patterns or changes in player performance.

These recommendations aim to enhance the predictive capabilities of the model and provide more accurate rank predictions.

In [ ]: