EG Data Assessment_Zhenming Mo

May 29, 2023

1 EDA

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
df = pd.read_csv('starcraft_player_data.csv')
```

[3]:
```
df
```

[3]:
```
      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]
```

[4]:
```
# Check null values in each column
df.isnull().sum()
```

[4]:
```
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
```

[5]:
```
# Check the data type in each column
df.dtypes
```

[5]:
```
GameID                    int64
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
```

[6]:
```
# Remove rows containing non-numeric values
for column in df.columns:
    df = df[pd.to_numeric(df[column], errors='coerce').notnull()]
df
```

[6]:

```
      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
…        …            …  ..           …         …         …
3335    9261            4  20            8        400  158.1390
3336    9264            5  16           56       1500  186.1320
3337    9265            4  21            8        100  121.6992
3338    9270            3  20           28        400  134.2848
3339    9271            4  22            6        400   88.8246

      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
…                 …               …              …              …
3335         0.013829         0.000504              7        0.000217
3336         0.006951         0.000360              6        0.000083
3337         0.002956         0.000241              8        0.000055
3338         0.005424         0.000182              5        0.000000
3339         0.000844         0.000108              2        0.000000

      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
…                    …            …              …              …
3335            0.000313      0.003583         36.3990        66.2718
3336            0.000166      0.005414         22.8615        34.7417
3337            0.000208      0.003690         35.5833        57.9585
3338            0.000480      0.003205         18.2927        62.4615
3339            0.000341      0.003099         45.1512        63.4435

      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
…              …                …           …               …
3335        4.5097                30     0.001035                7
```

```
3336        4.9309                  38      0.001343                 7
3337        5.4154                  23      0.002014                 7
3338        6.0202                  18      0.000934                 5
3339        5.1913                  20      0.000476                 8

      ComplexUnitsMade  ComplexAbilitiesUsed
0                  0.0              0.000000
1                  0.0              0.000208
2                  0.0              0.000189
3                  0.0              0.000384
4                  0.0              0.000019
…                    …                     …
3335               0.0              0.000287
3336               0.0              0.000388
3337               0.0              0.000000
3338               0.0              0.000000
3339               0.0              0.000054

[3338 rows x 20 columns]
```

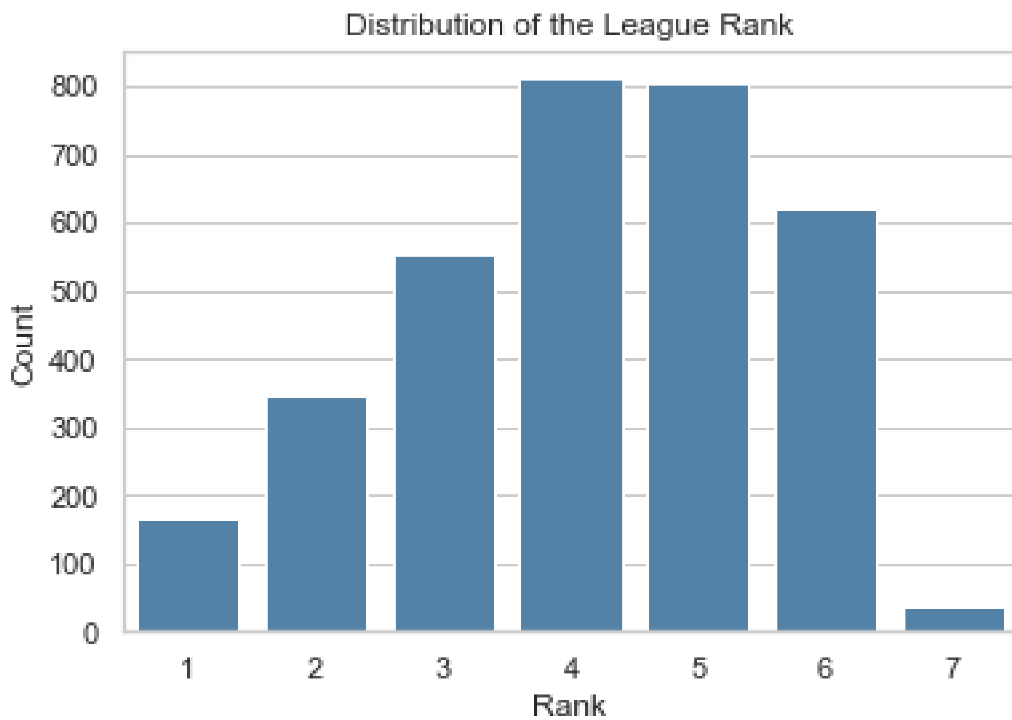
[7]:
```
# Convert object to int64 for model building
df['Age'] = df['Age'].astype('int64')
df['HoursPerWeek'] = df['HoursPerWeek'].astype('int64')
df['TotalHours'] = df['TotalHours'].astype('int64')
```

[8]:
```
# Drop GameID
df = df.drop('GameID', axis=1)
```

1.0.1 The data set has no missing or null values. However, there are columns containing non-numeric values. Since all features are supposed to contain either integers or continuous numeric values, I removed rows that contain non-numeric values.

[9]:
```
# Find the distribution of the league rank
import seaborn as sns
import matplotlib.pyplot as plt

league_counts = df['LeagueIndex'].value_counts().sort_index()
sns.set(style="whitegrid")
sns.barplot(x=league_counts.index, y=league_counts.values, color='steelblue')
plt.xlabel('Rank')
plt.ylabel('Count')
plt.title('Distribution of the League Rank')
plt.show()
```



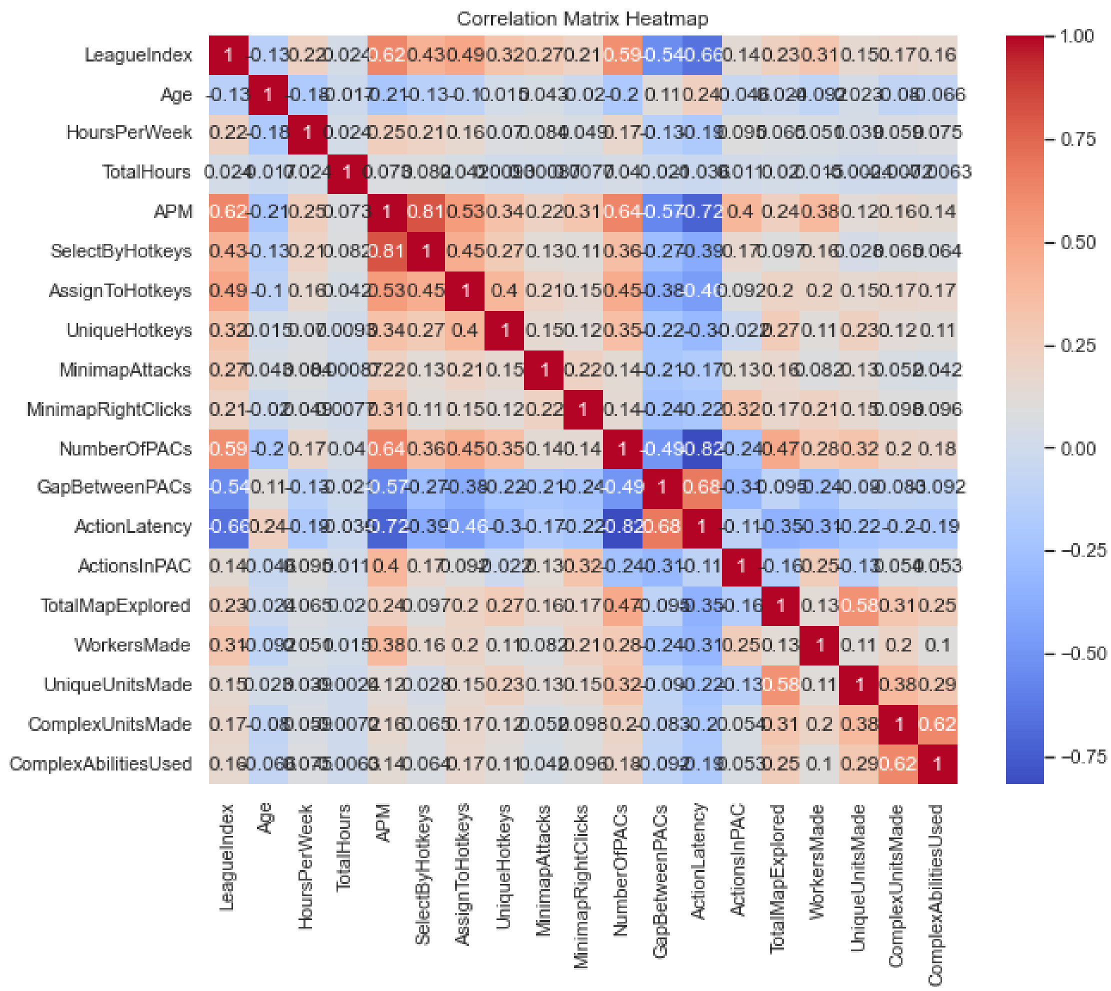
[10]:
```
# Count the number of the differnt ranks
index_counts = df['LeagueIndex'].value_counts()
print(index_counts)
```

```
4    811
5    804
6    621
3    553
2    347
1    167
7     35
Name: LeagueIndex, dtype: int64
```

1.0.2 The distribution of our target variable, LeagueIndex, shows that our data set is highly imbalanced. In other words, significantly more players are ranked under Gold, Platinum, Diamond, and Master while less players are ranked under Bronze, Silver, or GrandMaster. What's more, no player is ranked under Professional in the dataset, which means the model I build based on this dataset will not provide any support to predict someone who is ranked at Professional.

[11]:
```
# Check the correlation matrix heat map
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=True)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



[12]:
```python
# Find the top positively correlated features with LeagueIndex
league_index_correlation = correlation_matrix['LeagueIndex']
positively_correlated = league_index_correlation[league_index_correlation > 0]
top_positively_correlated = positively_correlated.
    ↪sort_values(ascending=False)[1:]
print(top_positively_correlated)
```

```
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
TotalHours              0.023884
Name: LeagueIndex, dtype: float64
```

[13]:
```python
# Find the top negatively correlated features with LeagueIndex
negatively_correlated = league_index_correlation[league_index_correlation < 0]
top_negatively_correlated = negatively_correlated.sort_values(ascending=True)
print(top_negatively_correlated)
```

```
ActionLatency    -0.659940
GapBetweenPACs   -0.537536
Age              -0.127518
Name: LeagueIndex, dtype: float64
```

1.0.3 The top positively and negatively correlated features with LeagueIndex are shown above. This provides information on which features are more likely to contribute to the prediction of LeagueIndex in the model. The higher the absolute value of the correlation between a feature and LeagueIndex, the greater the likelihood that the feature will play a significant role in predicting LeagueIndex.

1.0.4 At this point, I would assume APM, NumberOfPACs, ActionLatency, and GapBetweenPACs as more significant features in predicting the rank.

[14]:
```python
# Pairs of features with top correlation
corr_df = correlation_matrix.stack().reset_index()
corr_df.columns = ['Feature1', 'Feature2', 'Correlation']
corr_df = corr_df[corr_df['Feature1'] != corr_df['Feature2']]
corr_df['AbsCorrelation'] = np.abs(corr_df['Correlation'])
sorted_corr_df = corr_df.sort_values(by='AbsCorrelation', ascending=False)

print(sorted_corr_df.head(40))
```

```
                Feature1           Feature2  Correlation  AbsCorrelation
202         NumberOfPACs      ActionLatency    -0.817162        0.817162
238        ActionLatency       NumberOfPACs    -0.817162        0.817162
99         SelectByHotkeys               APM     0.814624        0.814624
81                   APM    SelectByHotkeys     0.814624        0.814624
232        ActionLatency                APM    -0.722253        0.722253
88                   APM      ActionLatency    -0.722253        0.722253
221       GapBetweenPACs      ActionLatency     0.680483        0.680483
```

239	ActionLatency	GapBetweenPACs	0.680483	0.680483
228	ActionLatency	LeagueIndex	-0.659940	0.659940
12	LeagueIndex	ActionLatency	-0.659940	0.659940
194	NumberOfPACs	APM	0.635248	0.635248
86	APM	NumberOfPACs	0.635248	0.635248
4	LeagueIndex	APM	0.624171	0.624171
76	APM	LeagueIndex	0.624171	0.624171
341	ComplexUnitsMade	ComplexAbilitiesUsed	0.620551	0.620551
359	ComplexAbilitiesUsed	ComplexUnitsMade	0.620551	0.620551
10	LeagueIndex	NumberOfPACs	0.589193	0.589193
190	NumberOfPACs	LeagueIndex	0.589193	0.589193
318	UniqueUnitsMade	TotalMapExplored	0.575231	0.575231
282	TotalMapExplored	UniqueUnitsMade	0.575231	0.575231
213	GapBetweenPACs	APM	-0.567396	0.567396
87	APM	GapBetweenPACs	-0.567396	0.567396
209	GapBetweenPACs	LeagueIndex	-0.537536	0.537536
11	LeagueIndex	GapBetweenPACs	-0.537536	0.537536
82	APM	AssignToHotkeys	0.534134	0.534134
118	AssignToHotkeys	APM	0.534134	0.534134
219	GapBetweenPACs	NumberOfPACs	-0.491407	0.491407
201	NumberOfPACs	GapBetweenPACs	-0.491407	0.491407
114	AssignToHotkeys	LeagueIndex	0.487280	0.487280
6	LeagueIndex	AssignToHotkeys	0.487280	0.487280
204	NumberOfPACs	TotalMapExplored	0.470955	0.470955
276	TotalMapExplored	NumberOfPACs	0.470955	0.470955
234	ActionLatency	AssignToHotkeys	-0.461496	0.461496
126	AssignToHotkeys	ActionLatency	-0.461496	0.461496
124	AssignToHotkeys	NumberOfPACs	0.454480	0.454480
196	NumberOfPACs	AssignToHotkeys	0.454480	0.454480
119	AssignToHotkeys	SelectByHotkeys	0.450342	0.450342
101	SelectByHotkeys	AssignToHotkeys	0.450342	0.450342
95	SelectByHotkeys	LeagueIndex	0.428637	0.428637
5	LeagueIndex	SelectByHotkeys	0.428637	0.428637

1.0.5 The pairs of features with top correlation above shows that many features in the dataset is highly correlated, which means it might be necessary for us to combine them through feature engineering or discard one of them, depending on the potential improvement to the model.

1.0.6 For instance, NumberOfPACs is highly correlated with ActionLatency, and SelectHotkeys is also highly correlated with APM. There are many reasons why this could happen. First of all, since we have limited amount of data, this correlation might happen simply by chance. The correlation will decrease as more data are collected. Secondly, these features are indeed inherently correlated and need to be modified for modeling.

[15]:
```
# Feature Engineering
# df['PACLatency'] = df['NumberOfPACs'] * df['ActionLatency']
# df['SelectAction'] = df['APM'] * df['SelectByHotkeys']
# df['PACActions'] = df['NumberOfPACs'] * df['ActionsInPAC']
```

1.0.7 Above is the feature engineering I will attempt for modeling based on my interpretation on the features.

[16]:
```
# Find the number of outliers in each column
import numpy as np

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers_count = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()
print(outliers_count)
```

```
LeagueIndex              0
Age                     98
HoursPerWeek           171
TotalHours             169
APM                     74
SelectByHotkeys        261
AssignToHotkeys         44
UniqueHotkeys            0
MinimapAttacks         271
MinimapRightClicks     181
NumberOfPACs            49
GapBetweenPACs         108
ActionLatency           87
ActionsInPAC            85
TotalMapExplored        44
WorkersMade            142
UniqueUnitsMade          4
ComplexUnitsMade       345
ComplexAbilitiesUsed   294
```

```
dtype: int64
```

1.0.8 There are outliers in almost every feature. Some features even have many outliers. However, it might not be reasonable to remove outliers considering that there is only 3338 rows of data, which are expected to applied on predicting 7 ordinal values. Removing outliers of any features might affect the representiveness of the dataset.

2 Modeling and Evaluation

[17]:
```
# Separate the features and target
X = df.drop('LeagueIndex', axis=1).drop('NumberOfPACs', axis=1).
 ↪drop('SelectByHotkeys', axis=1)
y = df['LeagueIndex']
```

[18]:
```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
```

2.0.1 The target variable, LeagueIndex, is already ordinal. Hence, there is no need to encode it before fitting the model.

[19]:
```
# Scale the features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

2.0.2 There is a big difference in the magnitude between features due to different units, so we need to scale the feature data before we fit the model.

[20]:
```
# Apply SMOTE to the training set
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled,␣
 ↪y_train)
```

2.0.3 Since the dataset is imbalanced, we need to apply SMOTE to the training set before we fit the model.

2.1 Model 1: Random Forest

[21]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

rf = RandomForestClassifier()
rf.fit(X_train_resampled, y_train_resampled)
rf_pred = rf.predict(X_test_scaled)

print(classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           1       0.33      0.45      0.38        33
           2       0.27      0.30      0.29        70
           3       0.33      0.38      0.35       111
           4       0.41      0.36      0.39       162
           5       0.45      0.34      0.38       161
           6       0.58      0.70      0.64       124
           7       0.00      0.00      0.00         7

    accuracy                           0.42       668
   macro avg       0.34      0.36      0.35       668
weighted avg       0.41      0.42      0.41       668
```

[22]:
```
importance_rf = rf.feature_importances_
feature_importance_rf = pd.DataFrame({'Feature': X.columns, 'Importance':
 ↪importance_rf})
feature_importance_rf = feature_importance_rf.sort_values(by='Importance',
 ↪ascending=False)
print(feature_importance_rf.to_string(index=False))
```

```
           Feature  Importance
     ActionLatency    0.125753
               APM    0.109553
        TotalHours    0.079082
      GapBetweenPACs  0.077726
     AssignToHotkeys  0.077337
      MinimapAttacks  0.064887
         WorkersMade  0.059638
        ActionsInPAC  0.058174
  MinimapRightClicks  0.053374
       UniqueHotkeys  0.052011
        HoursPerWeek  0.049057
     TotalMapExplored 0.047149
```

```
                 Age    0.046341
     UniqueUnitsMade    0.041393
ComplexAbilitiesUsed    0.035710
    ComplexUnitsMade    0.022815
```

[23]:

```python
# Hyperparameter tuning
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

rf = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
 ↪n_jobs=-1, verbose=2)


grid_search.fit(X_train_resampled, y_train_resampled)


best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")

best_rf = RandomForestClassifier(**best_params)
best_rf.fit(X_train_resampled, y_train_resampled)

rf_prediction = best_rf.predict(X_test_scaled)
print(classification_report(y_test, rf_prediction))
```

```
Fitting 5 folds for each of 450 candidates, totalling 2250 fits
Best parameters: {'bootstrap': False, 'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 500}
              precision    recall  f1-score   support

           1       0.34      0.39      0.37        33
           2       0.32      0.33      0.33        70
           3       0.31      0.35      0.33       111
           4       0.43      0.41      0.42       162
           5       0.41      0.34      0.37       161
           6       0.58      0.68      0.62       124
           7       0.00      0.00      0.00         7

    accuracy                           0.42       668
```

```
   macro avg       0.34      0.36      0.35       668
weighted avg       0.41      0.42      0.41       668
```

2.2 Model 2: Neural network

[25]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=1000, random_state=42)
mlp.fit(X_train_resampled, y_train_resampled)
mlp_pred = mlp.predict(X_test_scaled)

print(classification_report(y_test, mlp_pred))
```

```
              precision    recall  f1-score   support

           1       0.17      0.21      0.19        33
           2       0.22      0.29      0.25        70
           3       0.32      0.33      0.33       111
           4       0.31      0.24      0.27       162
           5       0.32      0.34      0.33       161
           6       0.45      0.43      0.44       124
           7       0.50      0.29      0.36         7

    accuracy                           0.32       668
   macro avg       0.33      0.30      0.31       668
weighted avg       0.32      0.32      0.32       668

C:\Users\10022\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

2.3 Model 3: Support Vector Machine

[26]:
```
from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train_resampled, y_train_resampled)
svm_pred = svm.predict(X_test_scaled)

print(classification_report(y_test, svm_pred))
```

```
              precision    recall  f1-score   support

           1       0.28      0.55      0.37        33
           2       0.28      0.40      0.33        70
```

```
           3       0.34      0.30      0.32       111
           4       0.43      0.30      0.35       162
           5       0.41      0.39      0.40       161
           6       0.56      0.61      0.59       124
           7       0.11      0.14      0.12         7

    accuracy                           0.40       668
   macro avg       0.34      0.38      0.35       668
weighted avg       0.41      0.40      0.40       668
```

2.4 Model 4: Gradient Boosting

[27]:
```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier()
gb.fit(X_train_resampled, y_train_resampled)
gb_pred = gb.predict(X_test_scaled)

print(classification_report(y_test, gb_pred))
```

```
              precision    recall  f1-score   support

           1       0.33      0.42      0.37        33
           2       0.35      0.40      0.37        70
           3       0.33      0.38      0.35       111
           4       0.47      0.41      0.44       162
           5       0.44      0.37      0.40       161
           6       0.54      0.60      0.57       124
           7       0.14      0.14      0.14         7

    accuracy                           0.43       668
   macro avg       0.37      0.39      0.38       668
weighted avg       0.43      0.43      0.43       668
```

2.5 Conclusion

2.5.1 Based on the outputs from 4 models trained above, we can conclude that Random Forest has the highest accuracy in predicting the player's rank. This is in part because Random Forest is robust to outliers and can handle imbalanced data better than other models. However, an accuracy of 42% after tuning is still very low, which means the model is not qualified for application.

2.5.2 According to the feature importance table from Random Forest, we can tell that the difference between each feature importance is not significant, even after I attempted to drop the highly correlated features and performed possible feature engineering documented in the EDA. We need more independant features to improve the performance of the model.

2.5.3 According to the precision, recall, and f1-score from the report, we can tell that Random Forest has a higher prediction power when the given rank has a larger sample size. Although the model has a generally poor prediction power over all ranks, it is completely not able to predict rank GrandMaster as the performance of all evaluation metrics for GrandMaster are extremely poor, which are close to 0. Other models have the same problem because we have very limited amount of data from players ranked at GrandMaster. What's more, other than GrandMaster, the model also has a poor performance on predicting players ranked under Bronze and Silver due to a lack of data.

3 Guidance for Stakeholders

3.0.1 In case my stakeholders come to me and say that they can collect more data, I will give them the following guidance:

1. If we have enough resources, collect more data from players under all ranks. If resources are limited, focus on collecting data from players who are ranked under GrandMaster, Professional, Bronze, and Silver because data under these ranks are highly limited.

2. Try to collect more data on other related features. For instance, based on my previous gaming experience and experience in Starcraft, I think the time spent being supply block, the number of base, the frequency of different faction used, and the previous rank might all help to explain the skill level of a player, which could possibly be used to predict the rank.

```
[ ]:
```