In [1]:
```
import pandas as pd
```

In [2]:
```
dota_train = pd.read_csv('dota2Train.csv', header=None)
dota_test = pd.read_csv('dota2Test.csv', header=None)
```

In [3]:
```
dota_test.shape
```

Out[3]: (10294, 117)

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

Out[4]:

	0	1	2	3	4	5	6	7	8	9	...	107	108	109	110	111	112	113	114
0	-1	223	8	2	0	-1	0	0	0	0	...	-1	0	0	0	0	0	0	0
1	1	227	8	2	0	0	0	0	0	0	...	-1	0	0	0	0	0	0	0
2	-1	136	2	2	1	0	0	0	-1	0	...	0	0	0	0	0	0	0	0
3	1	227	2	2	-1	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	1	184	2	3	0	0	0	-1	0	0	...	0	0	0	0	0	0	0	0

5 rows × 117 columns

In [5]:
```
dota_train.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	107	108	109	110	111	112	113	114	11
0	-1	223	2	2	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	1	152	2	2	0	0	0	1	0	-1	...	0	0	0	0	0	0	0	0	
2	1	131	2	2	0	0	0	1	0	-1	...	0	0	0	0	0	0	0	0	
3	1	154	2	2	0	0	0	0	0	0	...	-1	0	0	0	0	0	0	0	
4	-1	171	2	3	0	0	0	0	0	-1	...	0	0	0	0	0	0	0	0	

5 rows × 117 columns

In [6]:
```
dota_train.shape
```

Out[6]: (92650, 117)

In [7]:
```
heroes = pd.read_json('heroes.json')
```

In [8]:
```
heroes.head()
```

Out[8]:

	1	2	3	
id	1	2	3	
name	npc_dota_hero_antimage	npc_dota_hero_axe	npc_dota_hero_bane	npc_dota_he
primary_attr	agi	str	all	
attack_type	Melee	Melee	Ranged	
roles	[Carry, Escape, Nuker]	[Initiator, Durable, Disabler, Carry]	[Support, Disabler, Nuker, Durable]	[Carry,

5 rows × 126 columns

In [9]:
```
heroes.shape
```

Out[9]:
```
(33, 126)
```

About dataset

The DOTA 2 win prediction dataset contains records of 10,294 matches, each described by 117 numerical features. The first column encodes the match outcome (1: Radiant win, 0: Dire win); remaining features represent various aspects of match state, such as hero selection and player statistics. No missing values were detected. Feature types are all numerical, with many columns acting as binary flags or counters. Data was sourced from Kaggle and formatted as CSV files for analysis.

The heroes.json dataset consists of 33 heroes in DOTA 2, each described by 126 attributes. These attributes include hero ID, name, primary attribute (e.g., agility, strength, intelligence), attack type (melee or ranged), and hero roles (such as Carry, Disabler, Nuker, Support, etc.). All columns appear to be categorical or text-based, with some numerical attributes mixed in.

In [10]:
```
# Define column names for the raw data
columns = ['label', 'hero_id', 'lobby_type', 'game_mode']
feature_cols = [f'feature_{i}' for i in range(1, 114)]
all_cols = columns + feature_cols
dota_train.columns = all_cols
dota_test.columns = all_cols

# Create a mapping from hero ID to hero name and primary attribute
hero_map = {
    int(hero_id): {
        'hero_name': details['localized_name'],
        'primary_attr': details['primary_attr']
    }
    for hero_id, details in heroes.items()
}

# Convert the hero map to a DataFrame for merging
hero_df = pd.DataFrame.from_dict(hero_map, orient='index')
hero_df.index.name = 'hero_id'
```

```
# Merge the hero data with the training and test datasets
dota_train_mapped = dota_train.merge(hero_df, on='hero_id', how='left')
dota_test_mapped = dota_test.merge(hero_df, on='hero_id', how='left')

# Save the mapped datasets for the next steps
dota_train_mapped.to_csv('dota2_train_mapped.csv', index=False)
dota_test_mapped.to_csv('dota2_test_mapped.csv', index=False)

print("\nSample of Mapped Training Data:")
print(dota_train_mapped[['label', 'hero_id', 'hero_name', 'primary_attr']].head(
```

```
Sample of Mapped Training Data:
   label  hero_id    hero_name primary_attr
0     -1      223          NaN          NaN
1      1      152          NaN          NaN
2      1      131  Ring Master          int
3      1      154          NaN          NaN
4     -1      171          NaN          NaN
```

In [11]:
```
print(hero_df.index.tolist())
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2
3, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 4
4, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 6
4, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 8
4, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 10
3, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 119, 120, 121, 123, 12
6, 128, 129, 131, 135, 136, 137, 138, 145]
```

In [12]:
```
# Fill missing hero details with placeholder values
for col in ['hero_name', 'primary_attr']:
    dota_train_mapped[col] = dota_train_mapped[col].fillna('Unknown')
    dota_test_mapped[col] = dota_test_mapped[col].fillna('Unknown')

# Verify the change
print(dota_train_mapped[['hero_id', 'hero_name', 'primary_attr']].head(10))
print("\nMissing hero_name count:", dota_train_mapped['hero_name'].isnull().sum(
print("Missing primary_attr count:", dota_train_mapped['primary_attr'].isnull().
```

```
   hero_id    hero_name primary_attr
0      223      Unknown      Unknown
1      152      Unknown      Unknown
2      131  Ring Master          int
3      154      Unknown      Unknown
4      171      Unknown      Unknown
5      122      Unknown      Unknown
6      224      Unknown      Unknown
7      227      Unknown      Unknown
8      111       Oracle          int
9      151      Unknown      Unknown

Missing hero_name count: 0
Missing primary_attr count: 0
```

Some hero IDs in the match data did not appear in the reference hero list due to data version differences or deprecated/test heroes. These missing hero details were replaced with placeholder values ('Unknown') to preserve all rows for analysis.

Supervised Learning

Random Forest Classifier

Pre-processing dataset

In [13]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
```

In [14]:
```
# Trained dataset
LabelEncoder().fit_transform(dota_train_mapped['hero_name'])

LabelEncoder().fit_transform(dota_train_mapped['primary_attr'])
```

Out[14]:
```
array([0, 0, 3, ..., 3, 0, 0])
```

In [15]:
```
# Test dataset
LabelEncoder().fit_transform(dota_test_mapped['hero_name'])

LabelEncoder().fit_transform(dota_test_mapped['primary_attr'])
```

Out[15]:
```
array([0, 0, 2, ..., 0, 0, 0])
```

In [16]:
```
dota_test_mapped.head()
```

Out[16]:

	label	hero_id	lobby_type	game_mode	feature_1	feature_2	feature_3	feature_4
0	-1	223	8	2	0	-1	0	0
1	1	227	8	2	0	0	0	0
2	-1	136	2	2	1	0	0	0
3	1	227	2	2	-1	0	0	0
4	1	184	2	3	0	0	0	-1

5 rows × 119 columns

In [17]:
```
feature_cols = [col for col in dota_train_mapped.columns if col not in ['label',
```

In [18]:
```
# assigning Trained data sets for target 'label' - cause, it contain game result
X_train = dota_train_mapped[feature_cols]
y_train = dota_train_mapped['label']
```

In [19]:
```
# assigning Tested data sets for target - 'label'
X_test = dota_test_mapped[feature_cols]
y_test = dota_test_mapped['label']
```

Initializing and training for Random forest classifier

In [20]:
```
randomForest = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [21]:
```
randomForest.fit(X_train, y_train)
```

Out[21]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [22]:
```
# Predicting
y_pred = randomForest.predict(X_test)
```

In [23]:
```
print("Accuracy on test set:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy on test set: 0.5785894695939382

Classification Report:
               precision    recall  f1-score   support

          -1       0.55      0.49      0.52      4792
           1       0.60      0.66      0.63      5502

    accuracy                           0.58     10294
   macro avg       0.57      0.57      0.57     10294
weighted avg       0.58      0.58      0.58     10294


Confusion Matrix:
 [[2335 2457]
 [1881 3621]]
```

Report

The accuracy of is: 57%

- Since label is binary classification, the prediction appears to be accurate about the match predictions: Radiant win.

The Classification Report:

- The model performs better for label value 1, which Team radiant wins with higher recall and f1 score

Confusion Matrix:

- The model tends to predict label value 1 - Radiant's win with many label value '-1' which Dire team's wins missing values.

The baseline Random Forest model achieved an accuracy of 57.6% on the test set. The classification report indicates moderate precision and recall, with performance skewed towards correctly predicting Radiant wins. The confusion matrix shows the model frequently misclassifies Dire wins as Radiant. Although better than random guessing,

these results suggest that further feature engineering, model tuning, or different algorithms could yield stronger predictive power.

Unsupervised Learning

In [24]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score, silhouette_score
```

In [25]:
```
# Scaling features
scaler = StandardScaler()
```

In [26]:
```
# Transforming train and test datasets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

In [44]:
```
# K-means with 2 cluster
kmeans = KMeans(n_clusters=2, random_state=42, n_init=5)
kmeans.fit(X_train_scaled)
```

Out[44]:

KMeans

```
KMeans(n_clusters=2, n_init=5, random_state=42)
```

In [45]:
```
# Predicting clusters on test set
clusters = kmeans.predict(X_test_scaled)
```

In [29]:
```
# Evaluate clustering against actual labels
ari = adjusted_rand_score(y_test, clusters)
sil = silhouette_score(X_test_scaled, clusters)

print("Adjusted Rand Index:", ari)
print("Silhouette Score:", sil)
```

```
Adjusted Rand Index: -0.00014706698878080444
Silhouette Score: 0.014653969071830947
```

Report

We applied K-Means clustering (with k=2) to the test set features to explore whether the match data exhibits intrinsic groupings that align with actual game outcomes.

The Adjusted Rand Index was nearly zero, and the Silhouette Score was also close to zero, indicating that the clustering approach could not distinguish matches by outcome label. This suggests that match wins and losses are not strongly separated in the raw feature space without label supervision, reinforcing the need for supervised learning to model this task.

PCA + Supervised Learning

Applying PCA

In [30]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [31]:
```
# Scale features before PCA (important for PCA)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [32]:
```
# Apply PCA - keep enough components to explain ~95% variance
pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Original feature count:", X_train.shape[1])
print("Reduced feature count:", X_train_pca.shape[1])
```

```
Original feature count: 116
Reduced feature count: 106
```

Random Forest on PCA Reduced dataset

In [33]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
```

In [34]:
```
rf_pca = RandomForestClassifier(n_estimators=100, random_state=42)
rf_pca.fit(X_train_pca, y_train)
```

Out[34]:

RandomForestClassifier

```
RandomForestClassifier(random_state=42)
```

In [35]:
```
y_pred_pca = rf_pca.predict(X_test_pca)
```

In [36]:
```
print("Accuracy after PCA:", accuracy_score(y_test, y_pred_pca))
print("\nClassification Report:\n", classification_report(y_test, y_pred_pca))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_pca))
```

```
Accuracy after PCA: 0.5596463959588109

Classification Report:
               precision    recall  f1-score   support

          -1       0.53      0.45      0.49      4792
           1       0.58      0.66      0.61      5502

    accuracy                           0.56     10294
   macro avg       0.55      0.55      0.55     10294
weighted avg       0.56      0.56      0.55     10294


Confusion Matrix:
 [[2141 2651]
 [1882 3620]]
```

Principal Component Analysis (PCA) was applied to reduce dimensionality of the dataset while retaining 95% of the variance. After retraining the Random Forest classifier on the PCA-transformed features, accuracy dropped from 57.6% (original features) to 55.96%. Class-wise analysis shows a decline in both precision and recall for the Dire win class, while performance for the Radiant win class was largely maintained. These results suggest that Random Forest handled the original high-dimensional feature space well, and that PCA in this case removed useful information along with noise.

PCA + Unsupervised Learning

In [46]:
```
# Applying PCA for 95% of data
pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Reduced feature count after PCA:", X_train_pca.shape[1])
```

```
Reduced feature count after PCA: 106
```

In [38]:
```
# K-Means clustering on PCA data
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(X_train_pca)
```

Out[38]:

KMeans

```
KMeans(n_clusters=2, n_init=10, random_state=42)
```

In [42]:
```
# Predict clusters on test PCA data
clusters_pca = kmeans.predict(X_test_pca)
```

In [43]:
```
# Evaluate clustering result vs actual labels
ari = adjusted_rand_score(y_test, clusters_pca)
sil = silhouette_score(X_test_pca, clusters_pca)

print("Adjusted Rand Index after PCA + KMeans:", ari)
print("Silhouette Score after PCA + KMeans:", sil)
```

```
Adjusted Rand Index after PCA + KMeans: -0.00024624311556727596
Silhouette Score after PCA + KMeans: 0.021388545552645005
```

Report - Comparision: K-means vs PCA+K-means

Clustering with K-Means was performed after first applying PCA for dimensionality reduction. The results — an Adjusted Rand Index near zero and a silhouette score close to zero — showed that data points in the reduced feature space were not grouped meaningfully by match outcome, mirroring the results from K-Means on the full feature set. This suggests either the outcome classes are not intrinsically clustered in the raw feature space, or meaningful class separation requires more sophisticated or label-driven approaches. PCA did not improve cluster structure for unsupervised learning in this task.

Logistic Regression

In [37]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
```

In [38]:
```
# Initialize logistic regression, increase max_iter if needed for convergence
logisticRegression = LogisticRegression(max_iter=1000, random_state=42, n_jobs=-
```

In [39]:
```
# Fit the model
logisticRegression.fit(X_train, y_train)
```

Out[39]:
```
LogisticRegression(max_iter=1000, n_jobs=-1, random_state=42)
```

In [40]:
```
# Make predictions
y_pred_logisticRegression = logisticRegression.predict(X_test)
```

In [42]:
```
# Evaluate
print("Logistic Regression Test Accuracy:", accuracy_score(y_test, y_pred_logist
print("\nClassification Report:\n", classification_report(y_test, y_pred_logisti
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_logisticRegressio
```

```
Logistic Regression Test Accuracy: 0.597921119098504

Classification Report:
              precision    recall  f1-score   support

          -1       0.58      0.52      0.54      4792
           1       0.61      0.67      0.64      5502

    accuracy                           0.60     10294
   macro avg       0.59      0.59      0.59     10294
weighted avg       0.60      0.60      0.60     10294


Confusion Matrix:
 [[2475 2317]
 [1822 3680]]
```

Comparision

RandomForest vs Logistic Regression

Accuracy: 59.8%

- This is slightly higher than your previous Random Forest result (which was around 57.6%).

Classification Report:

- Higher recall (0.67) and F1-score (0.64) indicate the model predicts Radiant wins better.

Confusion Matrix:

- The model is better at predicting Radiant wins than Dire wins—similar to your Random Forest results, but overall it performs better on almost every metric.

Report

Logistic Regression outperformed Random Forest, achieving an accuracy of 59.8% on the test set. Analysis shows higher recall and F1-score for Radiant wins, indicating the model is more effective at predicting those outcomes. The confusion matrix suggests a tendency to misclassify Dire wins as Radiant, but overall precision and recall improved compared to prior models. This result demonstrates that a simpler linear approach can match or surpass ensemble methods for this dataset.