Which Team will Win in League of Legends?

Name(s): Trevan Nguyen

Website Link: https://sudosure.github.io/LeagueOfLegendsModel/

Code

In [1]:
```python
import pandas as pd
import numpy as np
import os

import plotly.express as px
pd.options.plotting.backend = 'plotly'

from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, QuantileTransformer, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_score
from sklearn.utils import resample
```

Framing the Problem

Prediction Problem: Predict if a team will win or lose a game based on post-game data.

Type: Classification (Binary)

Response Variable: Game result (Win/Loss)

Justification: Predicting the outcome of a League of Legends game can be useful for analyzing team performance and making strategic decisions. By predicting whether a team will win or lose based on post-game data, we can gain insights into factors that contribute to a team's success.

Evaluation Metric: Accuracy

Justification: Accuracy is appropriate for this classification problem as it measures the overall correctness of the predictions. We want to accurately classify whether a team will win or lose a game to evaluate the model's predictive performance.

In [2]:
```python
# Load League dataframe
lol_raw = pd.read_csv('2022_LoL_esports_match_data_from_OraclesElixir.csv',low_memory=False)
lol_raw.head()
```

Out[2]:

	gameid	datacompleteness	url	league	year	split	playoffs	date	game	patch	...	opp_csat15	golddiffat15	xpdiffat15	csdiffat15	killsat15	assistsat15	deathsat15	opp_killsat15	opp_assistsat15	opp_deathsat15
0	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08	1	12.01	...	121.0	391.0	345.0	14.0	0.0	1.0	0.0	0.0	1.0	0.0
1	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08	1	12.01	...	100.0	541.0	-275.0	-11.0	2.0	3.0	2.0	0.0	5.0	1.0
2	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08	1	12.01	...	119.0	-475.0	153.0	1.0	0.0	3.0	0.0	3.0	3.0	2.0
3	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08	1	12.01	...	149.0	-793.0	-1343.0	-34.0	2.0	1.0	2.0	3.0	3.0	0.0
4	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08	1	12.01	...	21.0	443.0	-497.0	7.0	1.0	2.0	2.0	0.0	6.0	2.0

5 rows × 123 columns

In [3]:
```python
# Get the columns that will be useful for our analysis
lol = lol_raw[['gameid','datacompleteness','side','position', 'result', \
               'kills','deaths','assists','pentakills','firstblood','team kpm', \
               'dragons','firstherald','firstbaron','firsttower', 'towers', 'damagetochampions','visionscore','totalgold']]

# Extract the team data and fill NaN values with 0.0 because they are missing because the value would have been 0.0 anyways
lol_teams = lol[lol['position']=='team'].fillna(0.0)
lol_teams.head()
```

Out[3]:

	gameid	datacompleteness	side	position	result	kills	deaths	assists	pentakills	firstblood	team kpm	dragons	firstherald	firstbaron	firsttower	towers	damagetochampions	visionscore	totalgold
10	ESPORTSTMNT01_2690210	complete	Blue	team	0	9	19	19	0.0	1.0	0.3152	1.0	1.0	0.0	1.0	3.0	56560.0	197.0	47070
11	ESPORTSTMNT01_2690210	complete	Red	team	1	19	9	62	0.0	0.0	0.6655	3.0	0.0	0.0	0.0	6.0	79912.0	205.0	52617
22	ESPORTSTMNT01_2690219	complete	Blue	team	0	3	16	7	0.0	0.0	0.0851	1.0	1.0	0.0	0.0	3.0	59579.0	277.0	57629
23	ESPORTSTMNT01_2690219	complete	Red	team	1	16	3	39	0.0	1.0	0.4541	4.0	0.0	1.0	1.0	11.0	74855.0	346.0	71004
34	8401-8401_game_1	partial	Blue	team	1	13	6	35	0.0	0.0	0.5714	2.0	0.0	0.0	0.0	8.0	40086.0	162.0	45468

Baseline Model

In [4]:
```python
# # Convert column ints into bools
# lol_teams['first_blood_bool'] = lol_teams['firstblood'].apply(lambda x: True if x == 1.0 else False)
# lol_teams['first_tower_bool'] = lol_teams['firsttower'].apply(lambda x: True if x == 1.0 else False)
# lol_teams['result_bool'] = lol_teams['result'].apply(lambda x: True if x == 1.0 else False)

# Select features and target variable
features = lol_teams[['firstblood', 'firsttower']]
target = lol_teams['result']

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

# Define the preprocessing steps for the categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), ['firstblood', 'firsttower'])
    ])

# Create the baseline model pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
])

# Train the model
pipeline.fit(X_train, y_train)

# Evaluate the model on the test set
accuracy = pipeline.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6600401606425703
```

Final Model

In [5]:
```python
# Select the features and the target variable
X = lol_teams[['firstblood','firsttower','damagetochampions','kills']]  # Features
y = lol_teams['result']  # Target variable

# 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)

# Define the feature transformation steps
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), [0])  # Categorical feature to be one-hot encoded
    ],
    remainder='passthrough'  # Pass through the remaining numerical features as-is
)

# Define the final model pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('feature_engineering', ColumnTransformer(
        transformers=[
            ('num1', StandardScaler(), [2]),  # StandardScaler on feature3
            ('num2', QuantileTransformer(), [3])  # QuantileTransformer on feature4
        ],
        remainder='passthrough'
    )),
    ('classifier', RandomForestClassifier())  # Random Forest classifier as an example
])

# Define the hyperparameters to tune
param_grid = {
    'classifier__n_estimators': [10, 20, 30],  # Number of trees in the forest
    'classifier__max_depth': [None, 5, 10],  # Maximum depth of the tree
}

# Perform grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and model
best_params = grid_search.best_params_
final_model = grid_search.best_estimator_

# Evaluate the best model on the testing data
accuracy = final_model.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
print("Best Hyperparameters:", best_params)
```

```
Accuracy: 0.85
Best Hyperparameters: {'classifier__max_depth': 5, 'classifier__n_estimators': 20}
```

Fairness Analysis

Null Hypothesis: The model is fair. The precision for the blue team and red team is roughly the same, and any differences are due to random chance.

Alternative Hypothesis: The model is unfair. The precision for the blue team is lower than the precision for the red team.

In [6]:
```
# Assume X_test and y_test are the test features and labels
X_blue_team = lol_teams[lol_teams['side'] == 'Blue']
y_blue_team = lol_teams[lol_teams['side'] == 'Blue']

X_red_team = lol_teams[lol_teams['side'] == 'Red']
y_red_team = lol_teams[lol_teams['side'] == 'Red']

# Make predictions for blue team and red team separately
y_pred_blue_team = final_model.predict(X_test[y_test == 1])
y_pred_red_team = final_model.predict(X_test[y_test == 0])

# Calculate the precision score for each group
precision_blue_team = precision_score(y_test[y_test == 1], y_pred_blue_team)
precision_red_team = precision_score(y_test[y_test == 0], y_pred_red_team)

# Define the number of permutations
num_permutations = 100

# Initialize an array to store the permutation test statistics
perm_scores = np.zeros(num_permutations)

# Perform the permutation test
for i in range(num_permutations):
    # Permute the labels within each group
    perm_y_blue_team = resample(y_test[y_test == 1])
    perm_y_red_team = resample(y_test[y_test == 0])

    # Concatenate the permuted labels with the original labels
    perm_y = np.concatenate((perm_y_blue_team, perm_y_red_team))

    # Make predictions on permuted labels
    perm_pred_blue_team = final_model.predict(X_test[y_test == 1])
    perm_pred_red_team = final_model.predict(X_test[y_test == 0])

    # Calculate the precision score for permuted groups
    perm_precision_blue_team = precision_score(perm_y_blue_team, perm_pred_blue_team)
    perm_precision_red_team = precision_score(perm_y_red_team, perm_pred_red_team)

    # Calculate the test statistic (difference in precision)
    perm_scores[i] = perm_precision_blue_team - perm_precision_red_team

# Calculate the observed test statistic (difference in precision)
observed_diff = precision_blue_team - precision_red_team

# Calculate the p-value
p_value = np.mean(perm_scores >= observed_diff)

print("Precision for blue team:", precision_blue_team)
print("Precision for red team:", precision_red_team)
print("Observed difference:", observed_diff)
print("p-value:", p_value)
```

```
Precision for blue team: 1.0
Precision for red team: 0.0
Observed difference: 1.0
p-value: 1.0
```