League of Legends: Vision and Dragons

Name(s): Jared Wang, Shaun Israni

Website Link: (your website link)

In [58]:
```
import pandas as pd
import numpy as np
from pathlib import Path

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

from dsc80_utils import * # Feel free to uncomment and use this.
```

Step 1: Introduction

General Introduction

League of Legends is a popular multiplayer game, in which two teams of five players compete against one another, attempting to gain an advantage over the opposing team, and ultimately, reaching and destroying their base. Outside of standard play, there is a large competetive scene, where preorganized groups compete in organized tournaments. Of competetive play, the largest and most popular of such would be Worlds, a yearly competition in which select, qualified teams from different regions around the world come together and compete for the chance to win the world title.

There are various ways that a team could get ahead in a game, with one of such methods being through slaying dragons. This objective (that of slaying the dragon/ dragons), could be vital to the state of the game, as upon completing such objectives, the team is granted benefits, such as increased movement speed, reduced skill cooldowns, and more.

Another key aspect of strategy in League of Legends is vision. Teams place "wards" on the map, effectively lighting up previously hidden areas of the map, helping teams track the positions of enemies.

This increased visibility can be especially important when contesting dragons, as it provides critical information on the opposing team's position and intentions, enabling better coordination and decision-making around these high-value objectives.

Our central research question is: How strongly is a team's vision control, measured by the number of wards they place (and wards per minute), associated with the number of dragons they secure in a Worlds game?

This question matters both to players and to analysts. If we find a strong relationship, it would support the idea that investing in vision is a key part of objective control strategy at the highest level of play. On the other hand, if the relationship is weak, it could suggest that other factors (like early-game lane pressure, jungle pathing, or team-fight execution) are more important than raw ward count when it comes to securing dragons. Throughout the rest of this project, we'll explore this relationship through exploratory data analysis, hypothesis testing, evaluating missingness, and predictive modeling.

Introduction of Columns

For the rest of this project, we focus on a few key columns that are most relevant to our question:

gameid: A unique identifier for each match, used to group the two teams in the same game.

league: The competetive region in which the game is taking place.

teamname: The name or identifier of the team.

gamelength: The duration of the game (in seconds), used to normalize columns

wpm: A derived feature equal to wards_placed divided by game length in minutes, which measures how actively a team uses vision over time.

wcpm: The number of enemy wards the team destroyed, divided by game length. This captures how effectively the team denies the enemy's vision.

controlwardsbought: The total number of control wards the team purchased. Control wards are special wards that reveal and disable enemy wards, so higher values indicate a stronger focus on long-term, persistent vision.

visionscore: Riot's (Creator of League of Legends) composite vision metric that rewards placing useful wards, clearing enemy wards, and providing vision of unseen enemies and objectives. A higher vision score reflects better overall vision control, not just raw ward counts.

dragons: The number of elemental dragons a team secured in that game.

cwpm: Number of control wards placed per minute, normalized using gamelength

dragons_per_minute: Number of dragons secured, normalized using gamelength

supvis: Vision score given by Riot for only the support. Useful as supports typically are tasked with the role of gaining and controling vision (though other teammates may help)

supcont: Number of control wards placed by the support

Step 2: Data Cleaning and Exploratory Data Analysis

The columns that we kept for the purpose of analysis consisted of:

```
['gamelength', 'league', 'wpm', 'wcpm', 'controlwardsbought',
'visionscore',
 'dragons', 'cwpm', 'dragons_per_minute', 'supvis', 'supcont']
```

Of the 12 years of Worlds data collected, we decided to exclude years 2014-2018 from our analysis, as during that time frame, control wards were not a part of the game. And only after 2018, control wards have not been updated.

We Concatenated the remaining 7 years into a two seperate DataFrames, being individual_data_df and team_data_df. From the total 883488 rows, we extracted the support related data (support vision score and support control wards) and added them as columns to the team data, we were left with a sibgle DataFrame, indexed by gameid and teamname, of 147186 rows.

From the data, we had missingness in columns, wpm, wcpm, cwpm, supvis and supcount, which we imputed using the median of their respective columns.

In [59]:
```
# First Worlds DataFrame
df2019 = pd.read_csv(r"C:\Users\jwang\OneDrive\Documents\GitHub\dsc80-2025-fa\proje

# Seperating the data by TeamWide data, and IndividualBased data
to_concat_ind = df2019[df2019["position"] != "team"]
to_concat_team = df2019[df2019["position"] == "team"]

# Concating all years worth of data into a single DataFrame, for TeamWide and Indiv
for i in range(20, 26):
    worldsDf = pd.read_csv(rf"C:\Users\jwang\OneDrive\Documents\GitHub\dsc80-2025-f

    player_data_df = worldsDf[worldsDf["position"] != "team"]
    team_data_df = worldsDf[worldsDf["position"] == "team"]
    to_concat_ind = pd.concat([to_concat_ind, player_data_df]).reset_index(drop=Tru
    to_concat_team = pd.concat([to_concat_team, team_data_df]).reset_index(drop=Tru

# Renaming of DataFrames
individual_data_df = to_concat_ind
team_data_df = to_concat_team
```

In [60]:
```
team_data_df = team_data_df[team_data_df["wpm"] != 0]
```

In [61]:
```
# for i in range(19, 26):
#     df = pd.read_csv(rf"C:\Users\jwang\OneDrive\Documents\GitHub\dsc80-2025-fa\pr
#     df["visionscore"] = pd.to_numeric(df["visionscore"], errors="coerce")
#     print(f"20{i}", df["visionscore"].isna().sum())
```

In [62]:
```
# get only the rows which belong to the support role, extracting necessary columns
sup_df = individual_data_df[individual_data_df["position"] == "sup"][["gameid", "te
```

```
# impute missingness through median
sup_df["controlwardsbought"] = sup_df["controlwardsbought"].fillna(sup_df["controlw
sup_df["visionscore"] = sup_df["visionscore"].fillna(sup_df["visionscore"].median()

# store supvis and supcont as variables to be assigned into teamwide data
sup_vision = sup_df["visionscore"]
sup_cont = sup_df["controlwardsbought"]
```

In [63]:
```
team_data_df["dragons"].isna().sum()
```

Out[63]:
```
np.int64(0)
```

In [64]:
```
# renaming of team_data
ward_dragon_info = team_data_df[["gameid", "teamname", "league", "gamelength", "wpm

# normalizing control wards into cwpm
ward_dragon_info["cwpm"] = ward_dragon_info["controlwardsbought"] / (ward_dragon_in

# impute missingness in ward related data
ward_dragon_info["wpm"] = ward_dragon_info["wpm"].fillna(ward_dragon_info["wpm"].me
ward_dragon_info["wcpm"] = ward_dragon_info["wcpm"].fillna(ward_dragon_info["wcpm"]
ward_dragon_info["cwpm"] = ward_dragon_info["cwpm"].fillna(ward_dragon_info["cwpm"]

# ward_dragon_info.duplicated(subset=["gameid", "teamname"]).sum()

# grouping team data by gameid and teamname
grouped_by_gameid = ward_dragon_info.set_index(["gameid", "teamname"])
grouped_by_gameid

# normalized dragons into dragons per minute
grouped_by_gameid["dragons_per_minute"] = grouped_by_gameid["dragons"] / grouped_by

# assign the values from the previously stored columns into the DataFrame
grouped_by_gameid['supvis'] = sup_vision
grouped_by_gameid['supcont'] = sup_cont
```

In [65]:
```
grouped_by_gameid.head(5)
```

Out[65]:

gameid	teamname	league	gamelength	wpm	wcpm	...	cwpm	drag
ESPORTSTMNT01/1030526	**Flamengo MDL**	CBLOL	1770	2.88	1.46	...	1.02	
	KaBuM! Esports	CBLOL	1770	2.98	1.12	...	1.46	
ESPORTSTMNT01/1040501	**Vivo Keyd**	CBLOL	2362	3.76	0.99	...	1.50	
	Uppercut esports	CBLOL	2362	3.40	1.40	...	1.32	
ESPORTSTMNT01/1040511	**ProGaming Esports**	CBLOL	2128	3.27	1.18	...	1.30	

5 rows × 11 columns

In [66]:
```
import plotly.graph_objects as go
```

In [68]:
```
# Distrubtion of game length
fig = px.histogram(grouped_by_gameid, x="gamelength", title="Histogram Game Length"
fig.write_html("histogram_game_length.html")
```

From this graph, we can see that the average length of a game follows a normal distribution, but is slightly right skewed.

In [69]:
```
# Distribution of Wards per Minute
fig = px.histogram(grouped_by_gameid, x="wpm", title="Histogram of Wards per Minute
fig
```

In [70]:
```
# Distribution of Dragons
fig = px.histogram(grouped_by_gameid, x="dragons", title="Histogram of Dragons")
fig
```

In [71]:
```
# histogram of Wards per Minute and Dragon Count
fig = px.histogram(
    grouped_by_gameid,
```

```
    x="dragons",
    y="wpm",
    histfunc="avg",
    title="Distribution of WPM by Dragon Count"
)
fig.write_html("distributionWPMDC.html")
```

This histogram shows that as game length increases, so does the number of dragons secured

In [72]:
```
# Scatterplot of Vision score to Dragons
fig = px.scatter(
    grouped_by_gameid,
    x="visionscore",
    y="dragons",
    title="Dragons Secured vs Vision Score",
    labels={"visionscore": "Vision score", "dragons": "Dragons secured"}
)
fig.show()
```

From this scatter plot, we see the general trend where as Vision Score increases, so does the number of Dragons Secured.

In [73]:
```
# Cut the DataFrame into quartiles
toCut = grouped_by_gameid.reset_index()
toCut["wpm_quartile"] = pd.qcut(
    toCut["wpm"],
    q=4,
    labels=["Low", "Med-Low", "Med-High", "High"]
)
```

```
# Aggregate team-level statistics by WPM quartile
# Computes mean wpm, mean dragons, mean dragons per minute, mean vision score, and
agg_by_wpm = (
    toCut
    .groupby("wpm_quartile")
    .agg(
        mean_wpm=("wpm", "mean"),
        mean_dragons=("dragons", "mean"),
        mean_dragons_per_min=("dragons_per_minute", "mean"),
        mean_visionscore=("visionscore", "mean"),
        n_games=("gameid", "count")
    ).reset_index()
)

agg_by_wpm.to_html()
```

```
C:\Users\jwang\AppData\Local\Temp\ipykernel_34084\1846542633.py:13: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

Out[73]:
```
'<table border="1" class="dataframe">\n  <thead>\n    <tr style="text-align: right;">\n      <th></th>\n      <th>wpm_quartile</th>\n      <th>mean_wpm</th>\n      <th>mean_dragons</th>\n      <th>mean_dragons_per_min</th>\n      <th>mean_visionscore</th>\n      <th>n_games</th>\n    </tr>\n  </thead>\n  <tbody>\n    <tr>\n      <th>0</th>\n      <td>Low</td>\n      <td>2.47</td>\n      <td>1.82</td>\n      <td>1.04e-03</td>\n      <td>179.49</td>\n      <td>36391</td>\n    </tr>\n    <tr>\n      <th>1</th>\n      <td>Med-Low</td>\n      <td>2.94</td>\n      <td>2.18</td>\n      <td>1.16e-03</td>\n      <td>222.54</td>\n      <td>36426</td>\n    </tr>\n    <tr>\n      <th>2</th>\n      <td>Med-High</td>\n      <td>3.28</td>\n      <td>2.35</td>\n      <td>1.20e-03</td>\n      <td>251.08</td>\n      <td>36353</td>\n    </tr>\n    <tr>\n      <th>3</th>\n      <td>High</td>\n      <td>3.87</td>\n      <td>2.55</td>\n      <td>1.25e-03</td>\n      <td>292.85</td>\n      <td>36380</td>\n    </tr>\n  </tbody>\n</table>'
```

Step 3: Assessment of Missingness

In [74]:
```
# Work from the original file so this cell is self-contained
worlds = pd.read_csv(
    r"C:\Users\jwang\OneDrive\Documents\GitHub\dsc80-2025-fa\projects\proj04\Leauge
    low_memory=False
)

# Keep only complete games and team rows
teams = worlds[(worlds["datacompleteness"] == "complete") &
               (worlds["position"] == "team")].copy()

# Create gamelength in minutes
teams["gamelength_min"] = teams["gamelength"] / 60

# Our column with non-trivial missingness
col = "goldat25"
```

```
print("Total team rows:", len(teams))
print(f"Missing {col}:", teams[col].isna().sum())

# Indicator for whether goldat25 is missing
teams["gold25_missing"] = teams[col].isna()

teams[["gameid", "teamname", "gamelength_min", "side", col, "gold25_missing"]].head
```

```
Total team rows: 18276
Missing goldat25: 680
```

Out[74]:

	gameid	teamname	gamelength_min	side	goldat25	gold25_missing
10	LOLTMNT03_179647	IziDream	26.53	Blue	39226.0	False
11	LOLTMNT03_179647	Team Valiant	26.53	Red	46192.0	False
22	LOLTMNT06_96134	Esprit Shōnen	32.03	Blue	47876.0	False
23	LOLTMNT06_96134	Skillcamp	32.03	Red	39499.0	False
34	LOLTMNT06_95160	Karmine Corp Blue Stars	29.70	Blue	42735.0	False

In [75]:

```
def diff_in_means(y, group):
    return y[group].mean() - y[~group].mean()

def permutation_test_diff_means(y, group, reps=1000, seed=0):
    rng = np.random.default_rng(seed)
    group = np.array(group)
    stats = np.empty(reps)
    for i in range(reps):
        shuffled = rng.permutation(group)
        stats[i] = diff_in_means(y, shuffled)
    return stats
```

In [76]:

```
# Numeric array for game length and Boolean array for missingness
glen = teams["gamelength_min"].values
missing = teams["gold25_missing"].values

# Observed test statistic
obs_glen = diff_in_means(glen, missing)
print("Observed difference in means (gamelength_min):", obs_glen)

# Permutation test
perm_stats_glen = permutation_test_diff_means(glen, missing, reps=2000, seed=1)

# Two-sided p-value
pval_glen = np.mean(np.abs(perm_stats_glen) >= abs(obs_glen))
print("Approximate p-value:", pval_glen)
```

```
Observed difference in means (gamelength_min): -10.079634564896665
Approximate p-value: 0.0
```

In [77]:
```
fig = px.histogram(
    x=perm_stats_glen,
    nbins=40,
    title="Null Distribution of Difference in Mean Game Length\n(missing vs non-mis
    labels={"x": "difference in mean gamelength_min"}
)
fig.add_vline(x=obs_glen, line_color="red", line_width=3,
              annotation_text="observed stat", annotation_position="top left")
fig.show()
```

In [78]:
```
side_blue = (teams["side"] == "Blue").astype(int).values

# Observed difference in mean "side_blue" (i.e., prop Blue)
obs_side = diff_in_means(side_blue, missing)
print("Observed difference in proportion Blue:", obs_side)

# Permutation test
perm_stats_side = permutation_test_diff_means(side_blue, missing, reps=2000, seed=2
pval_side = np.mean(np.abs(perm_stats_side) >= abs(obs_side))
print("Approximate p-value:", pval_side)

# Check actual proportions for intuition
prop_table = (
    teams.groupby(["gold25_missing", "side"]).size()
         .unstack()
)
print(prop_table)
```

```
Observed difference in proportion Blue: 0.0
Approximate p-value: 1.0
side            Blue   Red
gold25_missing
False           8798  8798
True             340   340
```

In [79]:
```
display(teams.head())
```

	gameid	datacompleteness	url	league	...	opp_assistsat25	opp_deathsat25
10	LOLTMNT03_179647	complete	NaN	LFL2	...	26.0	3.0
11	LOLTMNT03_179647	complete	NaN	LFL2	...	5.0	10.0
22	LOLTMNT06_96134	complete	NaN	LFL2	...	12.0	15.0
23	LOLTMNT06_96134	complete	NaN	LFL2	...	35.0	8.0
34	LOLTMNT06_95160	complete	NaN	LFL2	...	32.0	14.0

5 rows × 166 columns

In [80]:
```
# Plot A: conditional distribution of gamelength_min by missingness of goldat25

fig_len = px.histogram(
    teams,
    x="gamelength_min",
    color="gold25_missing",
    nbins=40,
    barmode="overlay",
    opacity=0.6,
    title="Game Length When goldat25 Is Missing vs Not Missing",
    labels={
        "gamelength_min": "Game length (minutes)",
        "gold25_missing": "goldat25 is missing?"
    }
)

fig_len.update_layout(
    legend_title_text="goldat25 missing?",
)

# Show in notebook (as an iframe)
fig_len.show()

# Save as HTML for embedding on the website
fig_len.write_html(
    "gold_missing_gamelength.html",
)
```

In [81]:
```
# Plot B: null distribution of difference in mean gamelength_min

fig_null = px.histogram(
    x=perm_stats_glen,
    nbins=40,
    title="Null Distribution of Difference in Mean Game Length\n(missing vs non-mis
    labels={"x": "difference in mean gamelength_min"}
)

fig_null.add_vline(
    x=obs_glen,
    line_color="red",
    line_width=3,
    annotation_text="observed stat",
    annotation_position="top left"
)

# Show in notebook (as an iframe)
fig_null.show()

# Save as HTML for embedding on the website
fig_null.write_html(
    "gold_missing_null_diff_glen.html"
)
```

Step 4: Hypothesis Testing

Hypothesis Test: Do higher warding rates lead to more dragons?

Null hypothesis (H_0): The average number of dragons secured is the same for high-WPM and low-WPM teams. Any observed difference in mean dragons is due to random variation.

Alternative hypothesis (H_1): High-WPM teams secure more dragons on average than low-WPM teams. (One-sided: mean_dragons_high > mean_dragons_low.)

In [82]:
```
# Create a high-vs-low WPM grouping based on the median
median_wpm = team_data_df["wpm"].median()
median_wpm

team_data_df["high_wpm"] = team_data_df["wpm"] >= median_wpm

team_data_df["high_wpm"].value_counts()
```

Out[82]:
```
high_wpm
False    72794
True     72756
Name: count, dtype: int64
```

In [83]:
```
# Define the test statistic and permutation function
def diff_in_means(y, group):
    y = np.asarray(y)
    group = np.asarray(group)
    return y[group].mean() - y[~group].mean()
```

```
def perm_test_diff_means(y, group, reps=5000, seed=0):
    rng = np.random.default_rng(seed)
    y = np.asarray(y)
    group = np.asarray(group)
    stats = np.empty(reps)
    for i in range(reps):
        shuffled = rng.permutation(group)
        stats[i] = diff_in_means(y, shuffled)
    return stats
```

In [84]:
```
# Dragons and group indicator
dragons = team_data_df["dragons"].values
high_wpm = team_data_df["high_wpm"].values

# Observed difference in mean dragons (high WPM - low WPM)
obs_diff = diff_in_means(dragons, high_wpm)
obs_diff
```

Out[84]: np.float64(0.4531584019421817)

In [85]:
```
# Permutation test (one-sided and two-sided p-values)
perm_stats = perm_test_diff_means(dragons, high_wpm, reps=5000, seed=1)

# One-sided p-value: H1 is "high WPM > low WPM"
pval_one_sided = np.mean(perm_stats >= obs_diff)

# Two-sided p-value (for completeness)
pval_two_sided = np.mean(np.abs(perm_stats) >= abs(obs_diff))

obs_diff, pval_one_sided, pval_two_sided
```

Out[85]: (np.float64(0.4531584019421817), np.float64(0.0), np.float64(0.0))

In [86]:
```
# Plot the null distribution (optional but great for website)
fig = px.histogram(
    x=perm_stats,
    nbins=40,
    title="Null Distribution of Difference in Mean Dragons\n(high WPM vs low WPM)",
    labels={"x": "difference in mean dragons (high - low WPM)"}
)
fig.add_vline(
    x=obs_diff,
    line_color="red",
    line_width=3,
    annotation_text="observed stat",
    annotation_position="top left"
)
fig.show()

fig.write_html("plotc.html")
```

Step 5: Framing a Prediction Problem

Since we saw a positive coorelation between the wards per minute and the number of dragons secured, we wanted to see if there was any relationship between other vision related metrics. More specifically, we wanted to ask, can we predict if a team ended up securing more dragons than their opponent based solely on vision related data?

Our prediction model was focused on teamwide data, however, with an emphasis on the support role. The model that we build for this falls under Classification, where we Biranized the dragons column, assigning the value 1 if a team secured more dragons than their opponents, and 0 otherwise within a game. To address ties, we assigned the value 0 to both teams, as neither team has “more” dragons than the other.

We split our data into two parts, consisting of 70% training, and 30% test data. To evaluate this model, we considered accuracy as our primary metric, as our data was relatively balanced, skewed slightly by out assigning of 0 to ties. In our DataFrame, we had a Brianized ratio of 58% 0’s, and 44% 1’s.

In [111…

```
max_dragons = grouped_by_gameid.groupby(level="gameid")["dragons"].transform("max")
tie_mask = grouped_by_gameid.groupby(level="gameid")["dragons"].transform("nunique"

grouped_by_gameid["dragons_binary"] = (
    (grouped_by_gameid["dragons"] == max_dragons) & ~tie_mask
).astype(int)
```

```
dropped = grouped_by_gameid.reset_index(drop=True)
dropped["dragons_binary"].value_counts()
```

Out[111…
```
dragons_binary
0    82025
1    63525
Name: count, dtype: int64
```

In [112…
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = (
    train_test_split(grouped_by_gameid[['wpm', 'wcpm', 'cwpm']], grouped_by_gameid[
)
```

In [113…
```
fig = (
    X_train.assign(Outcome=y_train.astype(str))
           .plot(kind='scatter',
                 x='wpm',
                 y='wcpm',
                 color='Outcome',
                 color_discrete_map={'0': 'orange', '1': 'blue'},
                 title='Relationship between WPM, CWPM, and Dragon Control')
)
fig
```

Step 6: Baseline Model

In [96]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.pipeline import Pipeline
from sklearn.metrics import f1_score
```

In our baseline model, we used a Random Forest Classifier with the features: `wpm`, `wcpm`, `cwpm`. While all three of these features were quantitative, only two of them were provided in the original Data Tables. As mentioned earlier, we standardized `controlwardsplaced`, dividing by the length of the game (in minutes), creating the new column `cwpm`.

After fitting a baseline model, our model resulted in a Training Accuracy of `0.9992`. Out model, however, had a Test Accuracy of `0.5469`, much lower than the Training Accuracy, suggesting that our model was likely over fitting the data.

In [101…
```
# Training/ Test split created with base training parameters
X_train, X_test, y_train, y_test = (
    train_test_split(grouped_by_gameid[['wpm', 'wcpm', 'cwpm']], grouped_by_gameid[
)

# Random Forest Pipeline with base training parameters
baseline_model = Pipeline([
    ("clf", RandomForestClassifier())
])

baseline_model.fit(X_train, y_train)

train_score = baseline_model.score(X_train, y_train)
test_score = baseline_model.score(X_test, y_test)

print("Train accuracy:", train_score)
print("Test accuracy:", test_score)
```

```
Train accuracy: 0.9992344309761005
Test accuracy: 0.5438680865681896
```

Step 7: Final Model

In our final model, we added three new features to the data: `visionscore`, `supvis`, and `supcont`. We chose to add these features to our model because in the game of Leauge of Legends, the supports typically have the largest impact on vision score, having dedicated items allowing for them to place more wards and clear wards more easily. Additionally, supports are often tasked with setting up for objectives, which is often associated with the use of control wards.

In [102…
```
from sklearn.model_selection import GridSearchCV
```

In [103…
```
# Test/Training split made with updated parameters
# Rather than only capturing team wide data, we incorporated support related statis
# This is because they are typically the player who contribute most to vision on a
X_train, X_test, y_train, y_test = (
    train_test_split(grouped_by_gameid[['wpm', 'wcpm', 'cwpm', 'visionscore', 'supv
)
```

In [ ]:
```
# # Base model with fixed parameters
# clf = RandomForestClassifier(
#     bootstrap=True,
#     oob_score=True,
#     n_jobs=-1
# )

# # Parameter grid to search
# param_grid = {
#     "max_depth": [5, 7, 9, None],
#     "min_samples_split": [2, 30, 45],
#     "min_samples_leaf": [2, 4, 8],
#     "n_estimators": [200, 500],
# }

# # Grid search setup
# grid = GridSearchCV(
#     estimator=clf,
#     param_grid=param_grid,
#     cv=3,
#     scoring="accuracy",
#     n_jobs=-1,
#     verbose=1
# )

# # Fit
# grid.fit(X_train, y_train)

# # Best model
# best_model = grid.best_estimator_

# print("Best params:", best_model.best_params_)
# print("Train score:", best_model.score(X_train, y_train))
# print("Validation score:", best_model.best_score_)

# # Fitting 3 folds for each of 72 candidates, totalling 216 fits
# # Best params: {'max_depth': 9, 'min_samples_leaf': 2, 'min_samples_split': 4, 'n
# # Train score: 0.6279530843598174
# # Validation score: 0.608404900950418
```

In [105…
```
# Random Forest Pipeline using parameters found through GridSearch

final_pipeline = Pipeline([
    ("rf", RandomForestClassifier(
        bootstrap=True,
        oob_score=True,
        n_jobs=-1,
        max_depth=9,
        min_samples_split=4,
        min_samples_leaf=2,
        n_estimators=500
    ))
])

final_pipeline.fit(X_train, y_train)
```

```
print("Train score:", final_pipeline.score(X_train, y_train))
print("Test score:", final_pipeline.score(X_test, y_test))
```

```
Train score: 0.6279530843598174
Test score: 0.608404900950418
```

Step 8: Fairness Analysis

In [106…
```
# New DataFrame, consisting of only games from the NLC and LCK
# Convert these leagues into binary values
keep_two_regions = grouped_by_gameid[(grouped_by_gameid["league"] == "NLC") | (grou
keep_two_regions["league"] = keep_two_regions["league"].apply(lambda x: 1 if x == "
```

In [107…
```
# Create the Train / test split
X_train, X_test, y_train, y_test, groups_train, groups_test = train_test_split(
    keep_two_regions[['wpm', 'wcpm', 'cwpm', 'visionscore', 'supvis', 'supcont']],
)

# Fit to the pipeline
final_pipeline.fit(X_train, y_train)

preds = final_pipeline.predict(X_test)

acc_LCK = (preds[groups_test == 1] == y_test[groups_test == 1]).mean()
acc_NLC = (preds[groups_test == 0] == y_test[groups_test == 0]).mean()

# Observed Difference between LCK and NLC prediction accuracies
observed_diff = acc_LCK - acc_NLC
```

In [108…
```
perm_diffs = []

for _ in range(10000):
    permuted_groups = np.random.permutation(groups_test)

    LCK_permuted = (preds[permuted_groups == 1] == y_test[permuted_groups == 1]).me
    NLC_permuted = (preds[permuted_groups == 0] == y_test[permuted_groups == 0]).me

    perm_diffs.append(LCK_permuted - NLC_permuted)
```

In [109…
```
p_value = np.mean(np.array(perm_diffs) >= observed_diff)
```

In [110…
```
p_value
```

Out[110…
```
np.float64(0.0609)
```

Null hypothesis: The model is fair. The accuracy calculated for players from the Korean Region (LCK) is the same as the accuracy calculated from those in the North American Region (NLC).

Null hypothesis: The model is unfair. The accuracy calculated for players from the Korean Region (LCK) is not the same as the accuracy calculated from those in the North American

Region (NLC).