March 22, 2024

1 The Effect of Red Side on a Bot Laner's Experience

Name(s): Akshay Medidi, Pranav Kumarsubha

Website Link:https://redeyeszombiedragon.github.io/The-Red-Side-Experience/

[ ]:
```python
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.
```

1.1 Step 1: Introduction

[ ]:
```python
# TODO
'''
Looking at the dataset I specifically chose which was the 2024 lol esports␣
↪match data, I can think of a couple questions that I might be
interested in looking into. One question I was thinking of was whether red side␣
↪bot laners have an inherent advantage due to the map changes
in bot lane that were introduced in 2024. Another question I was thinking of␣
↪was if we could figure out which team is more likely to get first blood based on other columns (perhaps characters picked or players on the team).↪

The question I have opted to pursue is whether red side bot laners have an␣
↪inherrent advantage due to map changes. This is because it is
relevant to me as I play in the bot lane.
'''
```

[ ]: ' \nLooking at the dataset I specifically chose which was the 2024 lol esports match data, I can think of a couple questions that I might be\ninterested in looking into. One question I was thinking of was whether red side bot laners have an inherent advantage due to the map changes\nin bot lane that were introduced in 2024. Another question I was thinking of was if we could figure

out which team is more likely to get first\nblood based on other columns (perhaps characters picked or players on the team). \n\nThe question I have opted to pursue is whether red side bot laners have an inherrent advantage due to map changes. This is because it is \nrelevant to me as I play in the bot lane.\n'

1.2 Step 2: Data Cleaning and Exploratory Data Analysis

[ ]:
```
# TODO
# the first section of code here makes the dataset and pulls the relevant␣ ↪columns we
need for this section, also I made seperate dataframes # for red, blue and both sides
just for convenience later on
datapath = Path('OE Public Match Data') /␣
  ↪'2024_LoL_esports_match_data_from_OraclesElixir.csv'
league = pd.read_csv(datapath)
relevant_cols = league[['gameid', 'league', 'year', 'split','side', 'game',␣ ↪'position',
  'goldat10', 'xpat10', 'csat10', \
                                                'golddiffat10', 'xpdiffat10', 'csdiffat10',␣
  ↪'golddiffat15', 'xpdiffat15', 'csdiffat15']]
relevant_cols = relevant_cols.loc[relevant_cols['position'] == 'bot'].
  ↪loc[relevant_cols['year'] == 2024]
relevant_cols_red = relevant_cols.loc[relevant_cols['side'] == 'Red'].
↪loc[relevant_cols['position'] == 'bot'].loc[relevant_cols['year'] == 2024] relevant_cols_blue
= relevant_cols.loc[relevant_cols['side'] == 'Blue']. ↪loc[relevant_cols['position'] ==
'bot'].loc[relevant_cols['year'] == 2024]

# the second section of code was just me accounting for sections of the code␣ ↪that were
  missing and to fill them with NaN so they don't
# affects the analysis I do later
null_columns = relevant_cols.columns[relevant_cols.isnull().any()].tolist() for col in
null_columns:
     relevant_cols_red[col] = relevant_cols_red[col].fillna(np.nan)
     relevant_cols_blue[col] = relevant_cols_blue[col].fillna(np.nan)
     relevant_cols[col] = relevant_cols[col].fillna(np.nan)
display(relevant_cols.head())

# these 2 figs are the univariate analysis
fig1 = px.histogram(relevant_cols_red['golddiffat10'], title = 'RED SIDE BOT␣ ↪LANERS
  GOLD DIFF AT 10 IN 2024')
fig2 = px.histogram(relevant_cols_red['csat10'], title = 'RED SIDE BOT LANERS␣ ↪CS AT
  10 IN 2024')

fig1.write_html('uni.html', include_plotlyjs = 'cdn')

display(fig1)
display(fig2)
```



```
# these 2 figs are the bivariate analysis
fig3 = px.scatter(relevant_cols_red, x='golddiffat10', y='golddiffat15',␣ ↪title='RED
SIDE BOT LANERS GOLD DIFF AT 10 VS GOLD DIFF AT 15 IN 2024') display(fig3)
fig4 = px.scatter(relevant_cols_red, x='xpdiffat10', y='xpdiffat15', title='RED␣ ↪SIDE BOT
```

```
  LANERS XP DIFF AT 10 VS XP DIFF AT 15 IN 2024')
display(fig4)

fig3.write_html('bi.html', include_plotlyjs = 'cdn')

# this is me grouping the dataframe with both sides by side so I can see␣
↪whether there is an inherent difference in the sides stats when # averaged
grouped_means = relevant_cols.groupby('side').mean()
print(grouped_means[['goldat10', 'xpat10', 'csat10', 'golddiffat10',␣
↪'xpdiffat10', 'csdiffat10', 'golddiffat15', 'xpdiffat15', 'csdiffat15']])
```

```
                    gameid league year split … csdiffat10 golddiffat15 \
219 LOLTMNT06_13630 LEC 2024 Winter … 9.0 213.0 224 LOLTMNT06_13630
LEC 2024 Winter … -9.0 -213.0 231 LOLTMNT06_12701 LEC 2024 Winter … 9.0
639.0 236 LOLTMNT06_12701 LEC 2024 Winter … -9.0 -639.0 243
LOLTMNT06_13667 LEC 2024 Winter … 18.0 342.0

    xpdiffat15 csdiffat15
219 1319.0 17.0
224 -1319.0 -17.0
231 875.0 13.0
236 -875.0 -13.0
243 249.0 12.0

[5 rows x 16 columns]
      goldat10 xpat10 csat10 golddiffat10 … csdiffat10 golddiffat15 \ side …
Blue 3421.06 3190.1 77.76 42.9 … 0.22 62.74 Red 3378.16 3178.1 77.54 -42.9 …
-0.22 -62.74

      xpdiffat15 csdiffat15
side
Blue -8.12 -0.7
Red 8.12 0.7

[2 rows x 9 columns]
```


1.3 Step 3: Assessment of Missingness

[ ]:
```
# TODO
# none of the data is nmar, from what I can tell looking at the data, when the␣
↪datacompleteness columns is partial, you can see why some of
# the columns have missing data. Essentially, almost all of the points I saw␣
↪where missing with some information that can be drawn from looking # at other columns such as 'league' or 'year' or 'datacompleteness'.
```

```python
# for the column I want to check the missingness of, I have chosen the 'goldat10' column, and the column I think it depends on is 'league' and a # column I don't think it depends on is 'side'

# filter the dataset to get the columns for the missingness permutations we will do, also convert the league column to bool
filtered = league[['league', 'year', 'split', 'goldat10', 'side']].loc[league['year'] == 2024].fillna(np.nan)
filtered['league'] = filtered['league'].apply(lambda x: True if x == 'LDL' else False)

# new column that sees if we have missing values in a row
filtered['goldat10_isnull'] = filtered['goldat10'].isnull()
filtered

# the following sections are for the permutation test, i used TVD as the test stat
def calc_test_stat(df, column):
    return df.groupby(column)['goldat10_isnull'].value_counts(normalize=True).diff().abs().sum() / 2

def perform_permutation_test_and_plot(df, column):
    # observed test stat
    observed = calc_test_stat(df, column)

    # permutation test
    permuted_test_stats = []
    for _ in range(1000):
        permuted_data = df.copy()
        permuted_data['goldat10_isnull'] = permuted_data['goldat10_isnull'].sample(frac=1).reset_index(drop=True)
        permuted_test_stat = calc_test_stat(permuted_data, column)
        permuted_test_stats.append(permuted_test_stat)

    # p-val calculation
    p_value = np.mean([observed < permuted_test_stat for permuted_test_stat in permuted_test_stats])
```



```python
    # plots for the website
    fig = px.histogram(permuted_test_stats, nbins=20, title=f'Empirical Distribution of the TVD ({column})')
    fig.add_vline(x=observed, line_width=3, line_color="red")
    fig.update_layout(xaxis_title='TVD', yaxis_title='Probability') fig.show()

    # print the p-value and observed test stat
    print(f"P-value for '{column}': {p_value}")
```

```
    print(f"Observed Test Statistic for '{column}': {observed}")

    return fig

perform_permutation_test_and_plot(filtered, 'league')
perform_permutation_test_and_plot(filtered, 'side')
miss = perform_permutation_test_and_plot(filtered, 'league')
miss.write_html('miss.html', include_plotlyjs = 'cdn')


for column in ['league', 'side']:
    fig = px.histogram(filtered, x=column, color='goldat10_isnull',␣
↪barmode='group', \
                                         title=f"Distribution of '{column}' when 'goldat10' is␣
↪missing and not missing")
    fig.update_layout(xaxis_title=column, yaxis_title='Count')
    display(fig)
```

```
P-value for 'league': 1.0
Observed Test Statistic for 'league': 0.8493385595296423

P-value for 'side': 0.533
Observed Test Statistic for 'side': 0.9732824427480917

P-value for 'league': 1.0
Observed Test Statistic for 'league': 0.8493385595296423
```

1.4 Step 4: Hypothesis Testing

```
[ ]: # TODO

# Hypotheses: Null: There is no inherent advantage through gold diff and cs␣ ↪diff for red
  side and blue side bot laners.
# Alternate: There is an inherent advantage for blue side bot laners over red␣ ↪side bot
  laners.

def tvd(dist1, dist2):
    return np.sum(np.abs(dist1 - dist2)) / 2
```



```
def perm_test(xs, ys, num_rounds=1000):
    n, k = len(xs), 0
    diff = tvd(np.histogram(xs)[0], np.histogram(ys)[0])
    zs = np.concatenate([xs, ys])
    for j in range(num_rounds):
        np.random.shuffle(zs)
        k += diff < tvd(np.histogram(zs[:n])[0], np.histogram(zs[n:])[0]) return k /
```

```
    num_rounds

# Assuming 'relevant_cols' is a pandas DataFrame
blue_side = relevant_cols[relevant_cols['side'] == 'Blue']['goldat10'].dropna(). ↪values
red_side = relevant_cols[relevant_cols['side'] == 'Red']['goldat10'].dropna(). ↪values

p_val = perm_test(blue_side, red_side)

print(f"P-value: {p_val}")
```

```
P-value: 0.708
```

1.5 Step 5: Framing a Prediction Problem

```
[ ]: # TODO

# The prediction problem I would like to do would be to use the picks for both␣ ↪bot laners and the sides they are on to see if we can predict
# a number for what the golddiff at 10 would be?
# This is a regression problem as we want to predict a quantitative variable␣ ↪using other features in our dataset.
```

1.6 Step 6: Baseline Model

```
[ ]: # TODO

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import numpy as np

# clean up any data in addition to what we alr did
features = relevant_cols[['goldat10', 'xpat10']]
target = relevant_cols['golddiffat15'].dropna()
features_filtered = features.loc[target.index]
```



```
# train test split
X_train, X_test, y_train, y_test = train_test_split(features_filtered, target,␣ ↪test_size=0.2)

# pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()), # Feature scaling
    ('regressor', LinearRegression()) # Linear regression model
```

```
])

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

# predict
y_pred = pipeline.predict(X_test)

# evaluation stats
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

rmse, r2
```

[ ]: (894.1517322425155, 0.4193662703479494)

1.7 Step 7: Final Model

[ ]:

```
# TODO

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, QuantileTransformer, PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV

# pipeline transformers for new features
pipeline_steps = [
    ('preprocessor', ColumnTransformer(transformers=[
        ('num', Pipeline(steps=[
            ('scaler', StandardScaler()),
            ('poly', PolynomialFeatures(degree=2))]), ['goldat10', 'xpat10']), ('quant',
        QuantileTransformer(), ['goldat10'])])),
    ('model', RandomForestRegressor())
]
```



```
pipeline = Pipeline(steps=pipeline_steps)

# hyperparameters for the grid search
param_grid = {
    'model__n_estimators': [100, 200],
    'model__max_depth': [5, 10, None]
}
```

```
grid_search = GridSearchCV(pipeline, param_grid, cv=5,␣
↪scoring='neg_mean_squared_error')

# train the model
grid_search.fit(X_train, y_train)

# eval the model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# eval stats
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

best_params = grid_search.best_params_

rmse, r2, best_params
```

[ ]:
```
(891.3922409847274,
 0.4229445921053301,
 {'model__max_depth': 5, 'model__n_estimators': 100})
```

1.8 Step 8: Fairness Analysis

[ ]:
```
# TODO

#Null: Our model is fair. The RMSE's for both red and blue side is the same and␣
↪any differences are due to chance.
#Alternate: Our model is not fair. The RMSE for blue is less than red.

X_red = relevant_cols_red.drop('golddiffat15', axis=1)
y_red = relevant_cols_red['golddiffat15']

X_blue = relevant_cols_blue.drop('golddiffat15', axis=1)
y_blue = relevant_cols_blue['golddiffat15']

mean_red = X_red.mean()
mean_blue = X_blue.mean()

X_red = X_red.fillna(mean_red)
```



```
X_blue = X_blue.fillna(mean_blue)

mean_y_red = y_red.mean()
mean_y_blue = y_blue.mean()
```

```
y_red = y_red.fillna(mean_y_red)
y_blue = y_blue.fillna(mean_y_blue)

# rmse using our earlier models
y_pred_red = best_model.predict(X_red)
y_pred_blue = best_model.predict(X_blue)

rmse_red = np.sqrt(mean_squared_error(y_red, y_pred_red))
rmse_blue = np.sqrt(mean_squared_error(y_blue, y_pred_blue))

# permutation tests
n_permutations = 1000
diffs = []

for _ in range(n_permutations):
    # concatenate and shuffle preds
    y_pred = np.concatenate([y_pred_red, y_pred_blue])
    np.random.shuffle(y_pred)

    # split the preds into the two groups
    y_pred_red_perm = y_pred[:len(y_pred_red)]
    y_pred_blue_perm = y_pred[len(y_pred_red):]

    # rmse calculations
    rmse_red_perm = np.sqrt(mean_squared_error(y_red, y_pred_red_perm))
    rmse_blue_perm = np.sqrt(mean_squared_error(y_blue, y_pred_blue_perm))

    # diff between the rmses
    diffs.append(rmse_red_perm - rmse_blue_perm)

# observed
obs_diff = rmse_red - rmse_blue

# p-value
p_value = (np.sum(np.abs(diffs) >= np.abs(obs_diff)) + 1) / (n_permutations + 1) rmse_red,

rmse_blue, p_value
```

C:\Users\aksha\AppData\Local\Temp\ipykernel_1492\1769803685.py:12:
FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid

columns before calling the reduction.

```
C:\Users\aksha\AppData\Local\Temp\ipykernel_1492\1769803685.py:13:
FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is
deprecated; in a future version this will raise TypeError. Select only valid columns before
calling the reduction.
```

[ ]: (766.9088018177355, 759.6312191522248, 0.7332667332667333)