# Final Project

April 20, 2025

## 1 Forecasting Victory: 2024 League of Legends Worlds Matches Predictions

**Name(s)**: Jiahao Cheng

**Website Link**: https://cjhjw.github.io/EECS398-Final-Project/

[145]:
```
import pandas as pd
import numpy as np

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

from lec_utils import * # Feel free to uncomment and use this. It'll make your␣
↪plotly graphs look like ours in lecture!
```

### 1.1 Introduction

[146]:
```
data = pd.read_csv('2024_LoL_esports_match_data_from_OraclesElixir.csv')
# Each 12 consecutive records correspond to one match
data.head(12)
```

[146]:
```
                 gameid datacompleteness  \
0   10660-10660_game_1          partial
1   10660-10660_game_1          partial
2   10660-10660_game_1          partial
3   10660-10660_game_1          partial
4   10660-10660_game_1          partial
5   10660-10660_game_1          partial
6   10660-10660_game_1          partial
7   10660-10660_game_1          partial
8   10660-10660_game_1          partial
9   10660-10660_game_1          partial
10  10660-10660_game_1          partial
11  10660-10660_game_1          partial

                                            url league  …  deathsat25  \
0   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …         NaN
```

```
1   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
2   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
3   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
4   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
5   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
6   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
7   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
8   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
9   https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
10  https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN
11  https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …          NaN

   opp_killsat25  opp_assistsat25 opp_deathsat25
0            NaN              NaN            NaN
1            NaN              NaN            NaN
2            NaN              NaN            NaN
3            NaN              NaN            NaN
4            NaN              NaN            NaN
5            NaN              NaN            NaN
6            NaN              NaN            NaN
7            NaN              NaN            NaN
8            NaN              NaN            NaN
9            NaN              NaN            NaN
10           NaN              NaN            NaN
11           NaN              NaN            NaN

[12 rows x 161 columns]
```

[147]:
```
# The raw data contains 117,576 records (rows) and 161 features (columns).
data.shape
```

[147]:
```
(117576, 161)
```

[148]:
```
# The final 2 records provide team-level overviews for both sides.
data.iloc[10:12]
```

[148]:
```
                  gameid datacompleteness  \
10  10660-10660_game_1          partial
11  10660-10660_game_1          partial

                                             url league  …  deathsat25  \
10  https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …         NaN
11  https://lpl.qq.com/es/stats.shtml?bmid=10660   DCup  …         NaN

   opp_killsat25  opp_assistsat25 opp_deathsat25
10           NaN              NaN            NaN
11           NaN              NaN            NaN
```

[2 rows x 161 columns]

## 1.2 Data Cleaning and Exploratory Data Analysis

### 1.2.1 Data Cleaning

[149]:
```
# Extract team data and target columns
target_columns = ['result', 'side', 'firstblood', 'firstdragon', 'firstbaron',
                  'firsttower', 'firstmidtower', 'firsttothreetowers',
                  'gamelength', 'golddiffat10', 'golddiffat15', 'golddiffat20',
                  'xpdiffat10', 'xpdiffat15', 'xpdiffat20']
data = (
        data
        .loc[data['position'] == 'team', target_columns]
        .reset_index()
        .drop('index', axis=1)
       )
data.head()
```

[149]:
```
   result  side  firstblood  firstdragon  …  golddiffat20  xpdiffat10  \
0       0  Blue         0.0          NaN  …           NaN         NaN
1       1   Red         1.0          NaN  …           NaN         NaN
2       0  Blue         0.0          NaN  …           NaN         NaN
3       1   Red         1.0          NaN  …           NaN         NaN
4       1  Blue         1.0          NaN  …           NaN         NaN

   xpdiffat15  xpdiffat20
0         NaN         NaN
1         NaN         NaN
2         NaN         NaN
3         NaN         NaN
4         NaN         NaN

[5 rows x 15 columns]
```

[150]:
```
# After filtering and selection, the dataset contains: 19596 rows (2 teams ×␣
↪9798 matches) and 15 columns
data.shape
```

[150]: (19596, 15)

[151]:
```
# Check and modify NaN
# At least 2,822 team records contain incomplete data
data.isna().sum()
```

[151]:
```
result                0
side                  0
```

```
firstblood              0
firstdragon          2782
firstbaron           2782
firsttower           2782
firstmidtower        2784
firsttothreetowers   2782
gamelength              0
golddiffat10         2784
golddiffat15         2786
golddiffat20         2822
xpdiffat10           2784
xpdiffat15           2786
xpdiffat20           2822
dtype: int64
```

[152]:
```
need_drop = ['firsttower', 'firstmidtower', 'firsttothreetowers',
             'golddiffat10', 'golddiffat15', 'golddiffat20',
             'xpdiffat10', 'xpdiffat15', 'xpdiffat20']
data = data.dropna(subset=need_drop)
data.isna().sum()
```

[152]:
```
result                0
side                  0
firstblood            0
firstdragon           0
firstbaron            0
firsttower            0
firstmidtower         0
firsttothreetowers    0
gamelength            0
golddiffat10          0
golddiffat15          0
golddiffat20          0
xpdiffat10            0
xpdiffat15            0
xpdiffat20            0
dtype: int64
```

[153]:
```
# After deleting NaN data, the dataset contains:16774 rows (2 teams × 8387␣
 ↪matches) and 15 columns
data.shape
```

[153]:
```
(16774, 15)
```

[154]:
```
# Categorize Gamelength
# The gamelength ranges from 1143 to 3482 seconds.
fig = px.histogram(
```

```
    data,
    x='gamelength',
    nbins=150,
    title='Game Count by Game Duration (seconds)',
    marginal='box',
    color_discrete_sequence=['#AB63FA'],
    width=700,
    height=400
)
fig.update_layout(
    xaxis_title='Game Duration (seconds)',
    yaxis_title='Number of Games'
)

fig.show()
```

[155]:

```
# The gamelength column needs to be categorized into time periods to reveal the
# relationship between general time periods (in minutes)
gametime = ['<=25(mins)', '25-30(mins)',
            '30-35(mins)', '35-40(mins)', '>=40(mins)']
def group_time(time):
    if time <= 1499:
        return gametime[0]
    elif 1500 <= time <= 1799:
        return gametime[1]
    elif 1800 <= time <= 2099:
        return gametime[2]
    elif 2100 <= time <= 2399:
        return gametime[3]
    else:
        return gametime[4]

data = (
        data
        .assign(time_label = data['gamelength'].apply(group_time))
        .drop('gamelength', axis=1)
       )
```

[156]:

```
data['time_label'].value_counts().reindex(gametime)
```

[156]:

```
time_label
<=25(mins)     1786
25-30(mins)    5348
30-35(mins)    5522
35-40(mins)    2714
>=40(mins)     1404
Name: count, dtype: int64
```

[157]:
```
counts = data['time_label'].value_counts().reindex(gametime).reset_index()

fig = px.bar(
    counts,
    x='time_label',
    y='count',
    title='Game Count by Game Duration (minutes)',
    color_discrete_sequence=['#AB63FA'],
    width=700,
    height=400
)
fig.update_layout(
    xaxis_title='Game Duration (minutes)',
    yaxis_title='Number of Games'
)
fig.show()
```

[158]:
```
# Recategorize result as win
data = (
        data
        .assign(win = data['result'].apply(lambda x: True if x == 1 else False))
        .drop('result', axis=1)
       )
```

### 1.2.2 Univariate Analysis

[159]:
```
# XP Difference at 10 minutes distribution
# For red side teams, 95% of XP differences range from -2129 to 1903, with a␣
 ↪median of -63.
df = data.loc[data['side'] == 'Red']
fig = px.histogram(
    df,
    x='xpdiffat10',
    nbins=150,
    title='Team Count by XP Difference at 10 minutes',
    marginal='box',
    color_discrete_sequence=['#CE2029'],
    width=700,
    height=400
)
fig.update_layout(
    xaxis_title='XP Difference at 10 minutes',
    yaxis_title='Number of Teams'
)
lower, upper = df['xpdiffat10'].quantile([0.025, 0.975])
fig.add_vline(
    x=lower,
```

```
    line_dash='dash',
    line_color='#CE2029',
    line_width=2,
    annotation_text=f'2.5% ({lower:.0f})',
    annotation_position='top left',
    annotation_font_color='black',
    annotation_bgcolor='white'
)

fig.add_vline(
    x=upper,
    line_dash='dash',
    line_color='#CE2029',
    line_width=2,
    annotation_text=f'97.5% ({upper:.0f})',
    annotation_position='top right',
    annotation_font_color='black',
    annotation_bgcolor='white'
)

fig.show()
```

```
[160]: # For blue side teams, 95% of XP differences range from -1903 to 2129, with a␣
        ↪median of 63.
df = data.loc[data['side'] == 'Blue']
fig = px.histogram(
    df,
    x='xpdiffat10',
    nbins=150,
    title='Team Count by XP Difference at 10 minutes',
    marginal='box',
    color_discrete_sequence=['#4682B4'],
    width=700,
    height=400
)
fig.update_layout(
    xaxis_title='XP Difference at 10 minutes',
    yaxis_title='Number of Teams'
)
lower, upper = df['xpdiffat10'].quantile([0.025, 0.975])
fig.add_vline(
    x=lower,
    line_dash='dash',
    line_color='#4682B4',
    line_width=2,
    annotation_text=f'2.5% ({lower:.0f})',
    annotation_position='top left',
```

```
    annotation_font_color='black',
    annotation_bgcolor='white'
)

fig.add_vline(
    x=upper,
    line_dash='dash',
    line_color='#4682B4',
    line_width=2,
    annotation_text=f'97.5% ({upper:.0f})',
    annotation_position='top right',
    annotation_font_color='black',
    annotation_bgcolor='white'
)

fig.show()
```

### 1.2.3 Bivariate Analysis

[161]:
```
# First Secured Info Exploration
target_columns = ['firstblood', 'firstdragon', 'firstbaron', 'firsttower',
                  'firstmidtower', 'firsttothreetowers']
data[target_columns] = data[target_columns] == 1
```

[162]:
```
# Win Rate for each side and firstblood
fig = px.bar(
    data.groupby(['side', 'firstblood'])['win'].mean().reset_index(),
    x='firstblood',
    y='win',
    color='side',
    barmode='group',
    color_discrete_map={'Blue': 'steelblue', 'Red': 'crimson'},
    title='Win Rate by Side and First Blood',
    width=700,
    height=400
)

fig.update_layout(
    xaxis_title='First Blood',
    yaxis_title='Average Win Rate'
)

fig.show()
```

[163]:
```
df1 = (
    data
    .groupby(['side', 'firsttothreetowers'])
```

```
    ['win']
    .mean()
    .reset_index()
)
df2 = (
    data
    .groupby(['side', 'firstmidtower'])
    ['win']
    .mean()
    .reset_index()
)
df3 = (
    data
    .groupby(['side', 'firsttower'])
    ['win']
    .mean()
    .reset_index()
)
df4 = (
    data
    .groupby(['side', 'firstdragon'])
    ['win']
    .mean()
    .reset_index()
)
df5 = (
    data
    .groupby(['side', 'firstbaron'])
    ['win']
    .mean()
    .reset_index()
)
df6 = (
    data
    .groupby(['side', 'firstblood'])
    ['win']
    .mean()
    .reset_index()
)
```

[164]:
```
df1 = df1.rename(columns={'firsttothreetowers': 'First Info Result'})
df1['First Info Detail'] = 'First to Three Towers'

df2 = df2.rename(columns={'firstmidtower': 'First Info Result'})
df2['First Info Detail'] = 'First Mid Tower'

df3 = df3.rename(columns={'firsttower': 'First Info Result'})
```

```
df3['First Info Detail'] = 'First Tower'

df4 = df4.rename(columns={'firstdragon': 'First Info Result'})
df4['First Info Detail'] = 'First Dragon'

df5 = df5.rename(columns={'firstbaron': 'First Info Result'})
df5['First Info Detail'] = 'First Baron'

df6 = df6.rename(columns={'firstblood': 'First Info Result'})
df6['First Info Detail'] = 'First Blood'
```

[165]:
```
df_all = pd.concat([df1, df2, df3, df4, df5, df6], ignore_index=True)
df_all['Side_First_Info'] = (
    df_all['side'] + ' - ' + df_all['First Info Result']
    .astype(str)
)
```

[166]:
```
color_map = {
    'Blue - False': '#4B8BBE',
    'Blue - True': '#306998',
    'Red - False': '#FF7F7F',
    'Red - True': '#D62728'
}
desired_order = ['Red - False', 'Blue - False', 'Red - True', 'Blue - True']

# Win Rate by Side and First Objective Secured
# Securing First Baron or First to Three Towers shows the strongest correlation␣
↪with winning, for both sides.
fig = df_all.plot(kind='bar',
            x='First Info Detail',
            y='win',
            color='Side_First_Info',
            barmode='group',
            category_orders={
        'First Info Detail': [
            'First Blood', 'First Dragon', 'First Tower',
            'First Mid Tower', 'First to Three Towers', 'First Baron'
        ],
        'Side_First_Info': desired_order
    },            color_discrete_map=color_map,
            title='Win Rate by Side and Tower'
)
fig.update_layout(
    width=1000,
    height=400
)
fig
```

[167]:
```python
# Difference in Gold and XP at 10 Minutes Across Game Lengths
# The spread of XP differences narrows as game length increases.
fig = px.violin(
    data,
    y='xpdiffat10',
    color='time_label',
    box=True,
    category_orders={
        'time_label': ['<=25(mins)', '25-30(mins)', '30-35(mins)',
                       '35-40(mins)', '>=40(mins)']
    },
    title='Distribution of XP Difference at 10 mins',
    orientation='v'
)

fig.update_layout(
    yaxis_title='XP Difference at 10 Minutes',
    width=700,
    height=400
)

fig.show()
```

### 1.2.4 Interesting Aggregates

[168]:
```python
# The quantified differences in win rate, first objective secured rate,
# and gold/XP difference between the two sides
target_columns = ['firstblood', 'firstdragon', 'firstbaron', 'firsttower',
                  'firstmidtower', 'firsttothreetowers', 'golddiffat10',
                  'golddiffat15', 'golddiffat20', 'xpdiffat10',
                  'xpdiffat15', 'xpdiffat20', 'win']
df = data.groupby('side')[target_columns].mean()
df
```

[168]:
```
      firstblood  firstdragon  firstbaron  firsttower  …  xpdiffat10  \
side                                                   …
Blue        0.52         0.38        0.50        0.55  …        66.9
Red         0.48         0.61        0.46        0.45  …       -66.9

      xpdiffat15  xpdiffat20   win
side
Blue       94.46       95.87  0.53
Red       -94.46      -95.87  0.47

[2 rows x 13 columns]
```

```
[169]: # The quantified differences in win rate between Blue and Red sides across␣
         ↪different game durations
       df = data.pivot_table(index='side',
                             columns='time_label',
                             values='win',
                             aggfunc='mean').reindex(columns=gametime)
       df
```

```
[169]: time_label  <=25(mins)  25-30(mins)  30-35(mins)  35-40(mins)  >=40(mins)
       side
       Blue               0.6         0.52         0.52         0.51        0.53
       Red                0.4         0.48         0.48         0.49        0.47
```

### 1.2.5 Imputation

Imputation is not required in this case, as the cleaned dataset contains no missing (NaN) values.

## 1.3 Framing a Prediction Problem

- Question: Whether a team wins or loses a match based on their in-game performance features collected by the 20-minute mark
- Prediction Type: Binary Classification
- Response Variable: win(True=Win, False=Lose), the only variable represents the match outcomes, interpretable
- Evaluation Matrics: confusion matrix, accuracy, ROC curve, AUC score. Unlike accuracy and precision, which depend on a specific classification threshold typically 0.5, ROC AUC evaluates model performance across all possible thresholds. This gives a more complete view of the classifier's ability to separate the two classes. Also, in the dataset, there may be a slight imbalance in match outcomes (blue side winning more often). ROC AUC is robust to class imbalance, whereas accuracy may be misleading in such cases.
- Except for time_label, all the used features are known at the time of prediction(before the game end).

## 1.4 Baseline Model

```
[170]: from sklearn.preprocessing import OneHotEncoder, StandardScaler
       from sklearn.model_selection import train_test_split
       from sklearn.linear_model import LogisticRegression
       from sklearn.pipeline import Pipeline, make_pipeline
       from sklearn.compose import make_column_transformer

       # Use side, first baron, XP difference at 10 minutes as predictors
       X = data[['side', 'firstbaron', 'xpdiffat10']]
       # Use win as responder
       y = data['win']

       # Train:Test = 7:3
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=123)

def baseline_model(X_train, y_train):

    preprocessor = make_column_transformer(
        (OneHotEncoder(drop='first', handle_unknown='ignore'),
         ['side', 'firstbaron']),
        (StandardScaler(), ['xpdiffat10'])
    )

    model = make_pipeline(preprocessor, LogisticRegression())

    model.fit(X_train, y_train)

    return model

base = baseline_model(X_train, y_train)
base
```

[170]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('onehotencoder',
                                                  OneHotEncoder(drop='first',
handle_unknown='ignore'),
                                                  ['side', 'firstbaron']),
                                                 ('standardscaler',
                                                  StandardScaler(),
                                                  ['xpdiffat10'])])),
                ('logisticregression', LogisticRegression())])
```

[171]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
def predict_thresholded(model, X_test, T):
    probs = model.predict_proba(X_test)[:, 1]
    return (probs >= T).astype(int)

def get_confusion_heatmap(model, X_test, y_test, T, title):
    y_pred = predict_thresholded(model, X_test, T)
    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)

    return go.Heatmap(
        z=cm,
        x=['Predicted Negative', 'Predicted Positive'],
        y=['Actual Negative', 'Actual Positive'],
        colorscale='Blues',
        text=[['True Negatives (TN)', 'False Positives (FP)'],
              ['False Negatives (FN)', 'True Positives (TP)']],
```

```
        texttemplate='%{text}<br>%{z}',
        textfont=dict(size=11),
        hovertemplate='Count: %{z}<br>Category: %{text}',
        showscale=False,
        name=title
    ), acc
```

[172]:

```
# The accuracy of the basic model is 82.8%
heatmap, acc = get_confusion_heatmap(base, X_test, y_test, 0.5,
                                     "Basic Logistic Regression")

fig = go.Figure(data=[heatmap])
fig.update_layout(
    title=f"Basic Logistic Regression (Accuracy = {acc:.3f})",
    width=500,
    height=450
)
fig.show()
```

## 1.5 Final Model

### 1.5.1 Logistic Regression

[173]:

```
import numpy as np
from sklearn.preprocessing import FunctionTransformer

def compute_per_min(X):
    return ((X.iloc[:, 0] / 10 + X.iloc[:, 1] / 15 + X.iloc[:, 2] / 20) / 3).
 ↪to_numpy().reshape(-1, 1)

def compute_tower_score(X):
    return X.sum(axis=1).to_numpy().reshape(-1, 1)

def compute_diff_drop(X):
    return (X.iloc[:, 0] - X.iloc[:, 1]).to_numpy().reshape(-1, 1)

X = data[['side', 'firstbaron', 'firsttothreetowers', 'firstmidtower',
          'firsttower', 'firstdragon', 'firstblood', 'xpdiffat10',
          'xpdiffat15', 'xpdiffat20', 'golddiffat10',
          'golddiffat15', 'golddiffat20']]
y = data['win']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=123)

def final_model_1(X_train, y_train):
```

```
xp_per_min_transformer = make_pipeline(
    FunctionTransformer(func=compute_per_min),
    StandardScaler()
)

gold_per_min_transformer = make_pipeline(
    FunctionTransformer(func=compute_per_min),
    StandardScaler()
)

tower_score_transformer = make_pipeline(
    FunctionTransformer(func=compute_tower_score),
    StandardScaler()
)

gold_drop_1015_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

gold_drop_1520_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

xp_drop_1015_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

xp_drop_1520_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

preprocessor = make_column_transformer(
    (OneHotEncoder(drop='first'), ['side', 'firstbaron', 'firstdragon',
                                   'firstblood']),
    (xp_per_min_transformer, ['xpdiffat10', 'xpdiffat15', 'xpdiffat20']),
    (gold_per_min_transformer, ['golddiffat10', 'golddiffat15',
                                'golddiffat20']),
    (tower_score_transformer, ['firsttower', 'firstmidtower',
                               'firsttothreetowers']),
    (gold_drop_1015_transformer, ['golddiffat10', 'golddiffat15']),
    (gold_drop_1520_transformer, ['golddiffat15', 'golddiffat20']),
    (xp_drop_1015_transformer, ['xpdiffat10', 'xpdiffat15']),
    (xp_drop_1520_transformer, ['xpdiffat15', 'xpdiffat20']),
```

```
    )

    model = make_pipeline(preprocessor, LogisticRegression())

    model.fit(X_train, y_train)

    return model

final1 = final_model_1(X_train, y_train)
final1
```

[173]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('onehotencoder',
                                                  OneHotEncoder(drop='first'),
                                                  ['side', 'firstbaron',
                                                   'firstdragon',
                                                   'firstblood']),
                                                 ('pipeline-1',
Pipeline(steps=[('functiontransformer',
FunctionTransformer(func=<function compute_per_min at 0x306244310>)),
('standardscaler',
StandardScaler())]),
                                                  ['xpdiffat10', 'xpdiffat15',
                                                   'x…
FunctionTransformer(func=<function compute_diff_drop at 0x306246710>)),
('standardscaler',
StandardScaler())]),
                                                  ['xpdiffat10', 'xpdiffat15']),
                                                 ('pipeline-7',
Pipeline(steps=[('functiontransformer',
FunctionTransformer(func=<function compute_diff_drop at 0x306246710>)),
('standardscaler',
StandardScaler())]),
                                                  ['xpdiffat15',
                                                   'xpdiffat20'])])),
                ('logisticregression', LogisticRegression())])
```

### 1.5.2 Random Forest

Random Forest's optimal tree depth is 6 based on 10 folds cross validation

[174]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
def final_model_2(X_train, y_train, k=10):

    xp_per_min_transformer = make_pipeline(
        FunctionTransformer(func=compute_per_min),
        StandardScaler()
```

```python
)

gold_per_min_transformer = make_pipeline(
    FunctionTransformer(func=compute_per_min),
    StandardScaler()
)

tower_score_transformer = make_pipeline(
    FunctionTransformer(func=compute_tower_score),
    StandardScaler()
)

gold_drop_1015_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

gold_drop_1520_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

xp_drop_1015_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

xp_drop_1520_transformer = make_pipeline(
    FunctionTransformer(func=compute_diff_drop),
    StandardScaler()
)

preprocessor = make_column_transformer(
    (OneHotEncoder(drop='first'), ['side', 'firstbaron', 'firstdragon',
                                   'firstblood']),
    (xp_per_min_transformer, ['xpdiffat10', 'xpdiffat15', 'xpdiffat20']),
    (gold_per_min_transformer, ['golddiffat10', 'golddiffat15',
                                'golddiffat20']),
    (tower_score_transformer, ['firsttower', 'firstmidtower',
                               'firsttothreetowers']),
    (gold_drop_1015_transformer, ['golddiffat10', 'golddiffat15']),
    (gold_drop_1520_transformer, ['golddiffat15', 'golddiffat20']),
    (xp_drop_1015_transformer, ['xpdiffat10', 'xpdiffat15']),
    (xp_drop_1520_transformer, ['xpdiffat15', 'xpdiffat20']),
)

pipe = make_pipeline(preprocessor, RandomForestClassifier(random_state=123))
```

```
    param_grid = {
        'randomforestclassifier__max_depth': np.arange(1, 11)
    }

    grid = GridSearchCV(pipe, param_grid, cv=k, scoring='roc_auc')

    grid.fit(X_train, y_train)

    return grid

final2 = final_model_2(X_train, y_train, 5)
final2
```

[174]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('columntransformer',
ColumnTransformer(transformers=[('onehotencoder',
OneHotEncoder(drop='first'),
['side',
'firstbaron',
'firstdragon',
'firstblood']),
('pipeline-1',
Pipeline(steps=[('functiontransformer',
          FunctionTransformer(func=<function compute_per_min at 0x306244310>)),
         ('standardscaler',
          StandardScaler())]),
                                                                        […
('pipeline-7',
Pipeline(steps=[('functiontransformer',
          FunctionTransformer(func=<function compute_diff_drop at
0x306246710>)),
         ('standardscaler',
          StandardScaler())]),
['xpdiffat15',
'xpdiffat20'])])),
                                       ('randomforestclassifier',
RandomForestClassifier(random_state=123))]),
             param_grid={'randomforestclassifier__max_depth': array([ 1,  2,  3,
4,  5,  6,  7,  8,  9, 10])},
             scoring='roc_auc')
```

### 1.5.3 Decision Tree

Decision Tree's optimal tree depth is 5 based on 10 folds cross validation

[175]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder,␣
↪FunctionTransformer
from sklearn.model_selection import GridSearchCV

def final_model_3(X_train, y_train):

    xp_per_min_transformer = make_pipeline(
        FunctionTransformer(func=compute_per_min),
        StandardScaler()
    )

    gold_per_min_transformer = make_pipeline(
        FunctionTransformer(func=compute_per_min),
        StandardScaler()
    )

    tower_score_transformer = make_pipeline(
        FunctionTransformer(func=compute_tower_score),
        StandardScaler()
    )

    gold_drop_1015_transformer = make_pipeline(
        FunctionTransformer(func=compute_diff_drop),
        StandardScaler()
    )

    gold_drop_1520_transformer = make_pipeline(
        FunctionTransformer(func=compute_diff_drop),
        StandardScaler()
    )

    xp_drop_1015_transformer = make_pipeline(
        FunctionTransformer(func=compute_diff_drop),
        StandardScaler()
    )

    xp_drop_1520_transformer = make_pipeline(
        FunctionTransformer(func=compute_diff_drop),
        StandardScaler()
    )

    preprocessor = make_column_transformer(
        (OneHotEncoder(drop='first'), ['side', 'firstbaron', 'firstdragon',
                                       'firstblood']),
```

```
        (xp_per_min_transformer, ['xpdiffat10', 'xpdiffat15', 'xpdiffat20']),
        (gold_per_min_transformer, ['golddiffat10', 'golddiffat15',
                                    'golddiffat20']),
        (tower_score_transformer, ['firsttower', 'firstmidtower',
                                   'firsttothreetowers']),
        (gold_drop_1015_transformer, ['golddiffat10', 'golddiffat15']),
        (gold_drop_1520_transformer, ['golddiffat15', 'golddiffat20']),
        (xp_drop_1015_transformer, ['xpdiffat10', 'xpdiffat15']),
        (xp_drop_1520_transformer, ['xpdiffat15', 'xpdiffat20']),
    )

    pipe = make_pipeline(
        preprocessor,
        DecisionTreeClassifier(random_state=123)
    )

    param_grid = {
        'decisiontreeclassifier__max_depth': np.arange(1, 11)
    }

    grid = GridSearchCV(pipe, param_grid, cv=5, scoring='roc_auc')
    grid.fit(X_train, y_train)

    return grid

final3 = final_model_3(X_train, y_train)
final3
```

[175]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('columntransformer',
ColumnTransformer(transformers=[('onehotencoder',
OneHotEncoder(drop='first'),
['side',
'firstbaron',
'firstdragon',
'firstblood']),
('pipeline-1',
Pipeline(steps=[('functiontransformer',
          FunctionTransformer(func=<function compute_per_min at 0x306244310>)),
         ('standardscaler',
          StandardScaler())]),
                                                                         […
('pipeline-7',
Pipeline(steps=[('functiontransformer',
          FunctionTransformer(func=<function compute_diff_drop at
0x306246710>)),
         ('standardscaler',
```

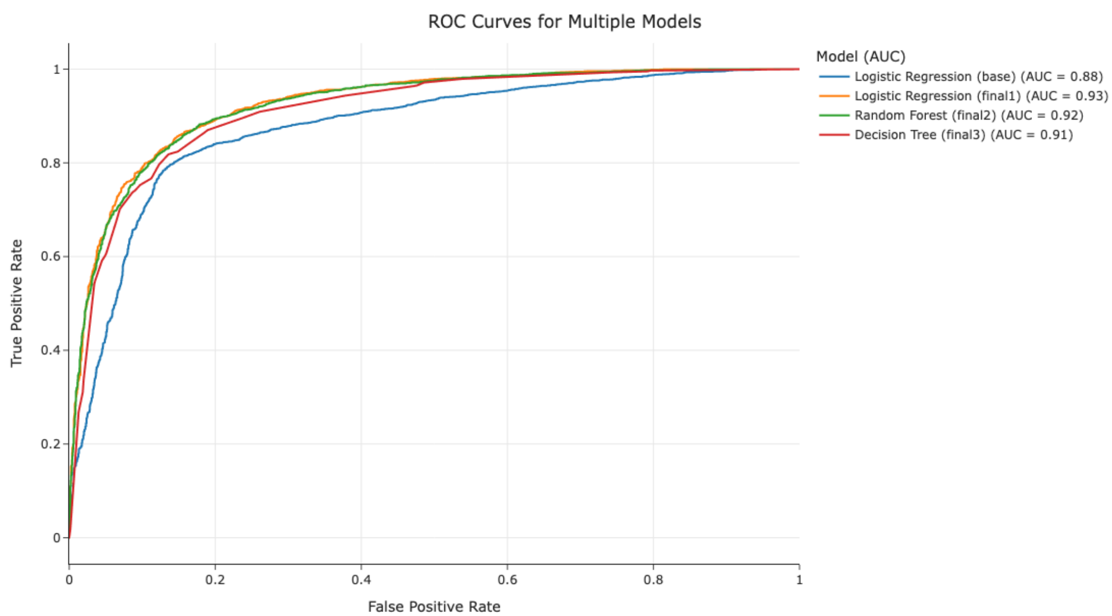
```
          StandardScaler())]),
['xpdiffat15',
'xpdiffat20'])])),
                                       ('decisiontreeclassifier',
DecisionTreeClassifier(random_state=123))]),
             param_grid={'decisiontreeclassifier__max_depth': array([ 1,  2,  3,
4,  5,  6,  7,  8,  9, 10])},
             scoring='roc_auc')
```

### 1.5.4 Models Comparison

- The Logistic Regression model performs 85.08 accuracy and 0.93 AUC.
- The Random Forest model performs 84.98 accuracy and 0.92 AUC.
- The Decision Tree model performs 83.75 accuracy and 0.91 AUC.

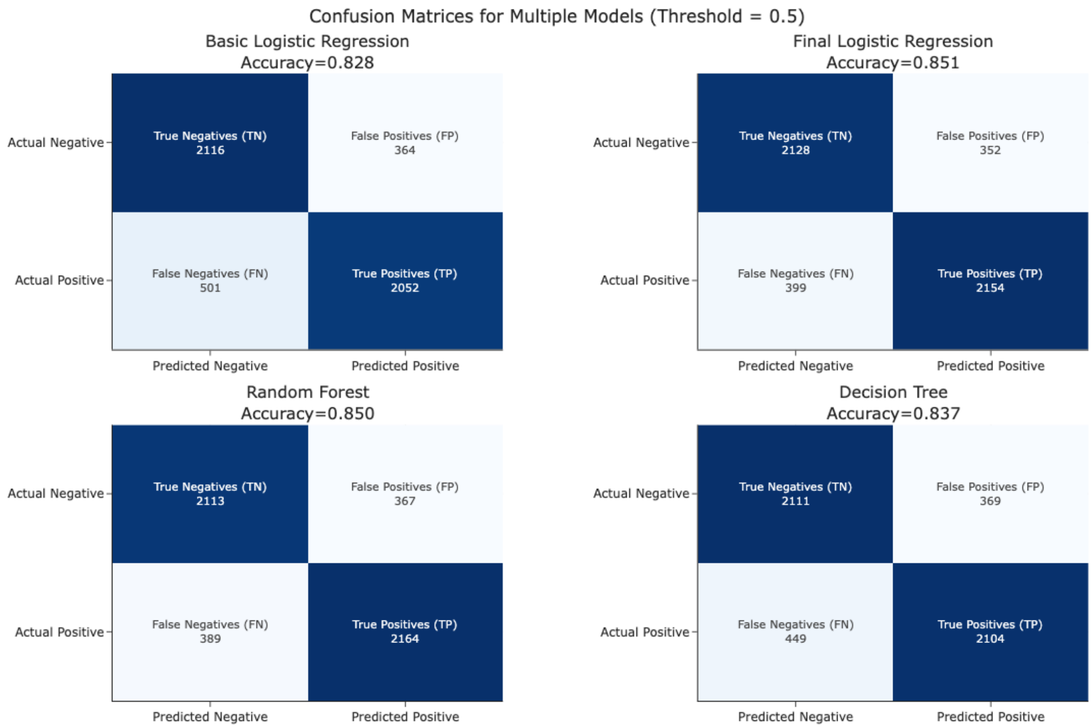
[176]:
```python
# ROC curves
# The final Logistic Regression model has the best performance on AUC score.
from sklearn.metrics import roc_curve, auc
def draw_roc_curves(models, X_test, y_test):
    all_roc_data = []

    for label, model in models.items():
        probs = model.predict_proba(X_test)[:, 1]
        fprs, tprs, thresholds = roc_curve(y_test.to_numpy(), probs)
        roc_auc = auc(fprs, tprs)

        for fpr, tpr in zip(fprs, tprs):
            all_roc_data.append({
                'FPR': fpr,
                'TPR': tpr,
                'Model': f'{label} (AUC = {roc_auc:.2f})'
            })

    df_roc = pd.DataFrame(all_roc_data)

    fig = px.line(
        df_roc,
        x='FPR',
        y='TPR',
        color='Model',
        title='ROC Curves for Multiple Models',
        labels={'FPR': 'False Positive Rate', 'TPR': 'True Positive Rate'},
        width=1000,
        height=600
    )
```

```
    fig.write_html("roc_curves.html")
    fig.update_layout(legend_title='Model (AUC)')
    fig.show()

models = {
    'Logistic Regression (base)': base,
    'Logistic Regression (final1)': final1,
    'Random Forest (final2)': final2,
    'Decision Tree (final3)': final3,
}

draw_roc_curves(models, X_test, y_test)
```




[177]:
```
# Confusion Matrices
# The final Logistic Regression model has the best performance on accuracy.
def show_multiple_confusions(models, X_test, y_test, T=0.5):

    heatmaps = []
    accs = []
    for i, (name, model) in enumerate(models.items()):
        heatmap, acc = get_confusion_heatmap(model, X_test, y_test, T,
                                             title=name)
        heatmaps.append(heatmap)
        accs.append((name, acc))

    fig = make_subplots(
        rows=2, cols=2,
```

```
        subplot_titles=[f"{name}<br>Accuracy={acc:.3f}"
                        for (name, acc) in accs],
        horizontal_spacing=0.2,
        vertical_spacing=0.12
    )

    for i, heatmap in enumerate(heatmaps):
        row = i // 2 + 1
        col = i % 2 + 1
        fig.add_trace(heatmap, row=row, col=col)

    fig.update_layout(
        width=800,
        height=750,
        title_text=f"Confusion Matrices for Multiple Models (Threshold = {T})",
        title_x=0.5,
        margin=dict(t=100)
    )
    fig.write_html("confusion_matrices.html")
    fig.update_yaxes(autorange='reversed')
    fig.show()

models = {
    'Basic Logistic Regression': base,
    'Final Logistic Regression': final1,
    'Random Forest': final2,
    'Decision Tree': final3
}

show_multiple_confusions(models, X_test, y_test, T=0.5)
```




### 1.5.5 Final Model Selection

As a result, the final Logistic Regression model is selected as the final model since it has the highest accuracy and AUC score while it's also simple and easy to interpret.

Compared to the base logistic regression model, the final model demonstrates a notable improvement in predictive performance:

- Accuracy increased from 82.81% to 85.08%
- AUC score improved from 0.88 to 0.93

Overall, 85.08% accuracy is not perfect for prediction model. But 0.93 AUC indicates excellent performance, with the model having a high ability to distinguish between classes. The final model now is more confident and accurate in ranking match outcomes.

[ ]: