Predicting a Legend

Name(s): Palina Volskaya & Hieu Ngyuen

Website Link: https://jaerinx.github.io/League_of_Legends_performance_analysis/

In [1]:

```
# Import core libraries (pandas, numpy, pathlib)
# and configure Plotly as the default plotting backend.
import pandas as pd
import numpy as np
from pathlib import Path
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pd.options.plotting.backend = 'plotly'
# Define a template for consistent plotly renders
pio.templates["wide_responsive"] = go.layout.Template(
    layout=dict(
        autosize=True,
        width=None,
        height=None,
        margin=dict(l=40, r=20, t=60, b=40)
    )
)

# Make it the default for all new figures
pio.templates.default = "wide_responsive"
```

Step 1: Introduction

In [2]:

```
# Load the raw League of Legends 2022 dataset from disk and display the
# resulting dataframe.
league = pd.read_csv('data/LoL2022data.csv',low_memory=False)
league.head()
```

Out[2]:

	gameid	datacompleteness	url	league	year	split	playoffs	da
0	ESPORTSTMNT01_2690210	complete	NaN	LCKC	2022	Spring	0	202 01- 07:44:
1	ESPORTSTMNT01_2690210	complete	NaN	LCKC	2022	Spring	0	202 01- 07:44:
2	ESPORTSTMNT01_2690210	complete	NaN	LCKC	2022	Spring	0	202 01- 07:44:
3	ESPORTSTMNT01_2690210	complete	NaN	LCKC	2022	Spring	0	202 01- 07:44:
4	ESPORTSTMNT01_2690210	complete	NaN	LCKC	2022	Spring	0	202 01- 07:44:

5 rows × 164 columns

Step 2: Data Cleaning and Exploratory Data Analysis

In [3]:

```
# Split the full dataset into team-level and player-level subsets, then preview
# the team records.
teams = league[league['position']=='team']
players = league[league['position']!='team']
teams.head()
```

Out[3]:

	gameid	datacompleteness	url	league	ye
10	ESPORTSTMNT01_2690210	complete	NaN	LCKC	20
11	ESPORTSTMNT01_2690210	complete	NaN	LCKC	20
22	ESPORTSTMNT01_2690219	complete	NaN	LCKC	20
23	ESPORTSTMNT01_2690219	complete	NaN	LCKC	20
34	8401-8401_game_1	partial	https://lpl.qq.com/es/stats.shtml?bmid=8401	LPL	20

5 rows × 164 columns

In [4]:

```
# Restrict the team dataframe to the key columns used later in the report
cols_to_keep = ['gameid','teamname','goldspent','teamid', 'position', 'result',
                'firstdragon', 'league','kills','deaths','gamelength']
cols_to_keep += [i for i in teams.columns if 'at25' in i]
teams = teams[cols_to_keep]
len(teams['gameid'].unique())
```

Out[4]:

```
12549
```

In [5]:

```
# Aggregate the number of games played in each league and visualize this
# distribution as a bar chart.
league_counts = teams['league'].value_counts() / 2
league_counts = league_counts.reset_index()
league_counts.columns = ['league', 'games_played']

fig = px.bar(
    league_counts,
    x='league',
    y='games_played',
    title="Number of games played by league",
    labels={'league': 'League', 'games_played': 'Number of games'}
)
fig.show()
fig.write_html("assets/univariate_analysis.html",include_plotlyjs='cdn')
```

In [6]:

```
# Filter to teams with more than 30 games, compute their average gold spent and
# win rate, and visualize the relationship with a scatter plot.
match_counts = teams.groupby('teamid')['gameid'].transform('count')
group_teams = (teams[match_counts>30].groupby('teamid')[['goldspent','result']]
               .mean())
fig = px.scatter(
    group_teams,
    x='goldspent',
    y='result',
    title="Win-rate of a team against<br> their total gold spent for teams with\
 more than 30 games",
    labels={
        'goldspent': 'Gold spent',
        'result': 'win-rate'
    }
)

fig.update_traces(marker=dict(size=4, color='green'))
fig.show()
fig.write_html("assets/bivariate_analysis_winsvsgold.html",include_plotlyjs='cdn')
```

In [7]:

```
# Compute team-level win rates with and without first dragon and compare their
# distributions using overlapping histograms.
yes_first_dragon = teams[teams['firstdragon']==1]
no_first_dragon = teams[teams['firstdragon']==0]
wins_yes = yes_first_dragon.groupby('teamid')['result'].mean()
wins_no = no_first_dragon.groupby('teamid')['result'].mean()

fig = go.Figure()

fig.add_trace(go.Histogram(
    x=wins_yes,
    name="With first dragon",
    marker=dict(color="blue"),
    opacity=0.75
))

fig.add_trace(go.Histogram(
    x=wins_no,
    name="Without first dragon",
    marker=dict(color="red"),
    opacity=0.7
))

fig.update_layout(
    title="Conditional distribution <br>plot of team win rate with or without \
first dragon",
    xaxis_title="Win-rate",
    yaxis_title="Frequency",
    barmode="overlay",
)

fig.show()
fig.write_html("assets/bivariate_analysis_wins_with_without_firstdragon.html",inclu
```

In [8]:

```
# Construct a pivot table summarizing average gold spent, kills per death
# and win percentage by first-dragon outcome.
copy = teams.copy()
result_arr = copy['kills']/copy['deaths']
final_arr = np.where(np.isinf(result_arr), 0, result_arr)
final_arr = np.nan_to_num(final_arr, nan=0, posinf=0, neginf=0)
```

```
copy['kills per death'] = final_arr
pivot_overall = pd.pivot_table(
    copy,
    values=['result','kills per death','goldspent'],
    index='firstdragon',
    aggfunc='mean'
)

pivot_overall = pivot_overall.rename_axis('firstdragon')
pivot_overall = (pivot_overall
                 .rename(columns={"result":"average win percentage",
                                  "kills per death":"average kills per death"
                                  ,"goldspent":"average gold spent"}))
pivot_overall['average win percentage'] = ((pivot_overall
                                           ['average win percentage']*100)
                                           .round(2))
pivot_overall['average kills per death'] = ((pivot_overall
                                            ['average kills per death'])
                                            .round(2))
pivot_overall['average gold spent'] = ((pivot_overall['average gold spent'])
                                       .round(2))
pivot_overall.index = pivot_overall.index.map(
    {1: 'Got first dragon', 0: 'Did NOT get first dragon'}
)
pivot_overall
```

Out[8]:

	average gold spent	average kills per death	average win percentage
firstdragon			
Did NOT get first dragon	52414.56	1.43	42.16
Got first dragon	53313.00	1.94	57.84

In [9]:

```
# Create a team-level pivot of win percentage with versus without first dragon
# and plot a sample of teams as a grouped bar chart.
pivot_team = pd.pivot_table(
    copy,
    values='result',
    index='teamname',            # one row per team
    columns=['firstdragon'],
    aggfunc='mean'
)

# Convert to percentages & rename columns
pivot_team = (pivot_team)
pivot_team = pivot_team.rename(
    columns={
        0: 'no_first_dragon_win_pct',
        1: 'first_dragon_win_pct'
    }
)
sample = pivot_team.sample(20)
trace1 = go.Bar(name='With First Dragon',
                x=sample.index,
                y=sample['first_dragon_win_pct'])
trace2 = go.Bar(name='Without First Dragon',
                x=sample.index,
                y=sample['no_first_dragon_win_pct'])
fig = go.Figure(data=[trace1, trace2])
fig.update_layout(title="Comparison of win rates for teams with or without \
first dragon", xaxis_title='Team',yaxis_title="win rate")
fig.show()
fig.write_html("assets/interesting_agg_sample_dragon.html"
               ,include_plotlyjs='cdn')
sample
```

Out[9]:

firstdragon	no_first_dragon_win_pct	first_dragon_win_pct
teamname		
Team Secret	0.522727	0.625000
Area of Effect Randoms	0.307692	0.500000
Invulnerables Esports Lotus	0.400000	0.000000
ECORP	0.714286	0.800000
Only Panitas	0.500000	0.333333
Team Fracture	0.000000	0.666667
Anorthosis Famagusta Esports	0.545455	0.676471
Gravitas	0.000000	0.187500
Goose Gaming	0.000000	0.000000
Nongshim RedForce Academy	0.558824	0.709677
devils.one	0.166667	0.400000
Aquatik Esports	0.125000	0.500000
Verity Esports	1.000000	0.000000
Leviatan	0.710526	0.827586
Immortals Academy	0.307692	0.582090
Panathinaikos AC eSports	0.333333	NaN
Hooked Esports	0.166667	0.333333
Eternal Kraken	0.000000	0.750000
One Breath Gaming	1.000000	1.000000
York University	0.166667	0.500000

Step 3: Assessment of Missingness

After analyzing the dataset aggregated by teams, we found the `pick1` column to be NMAR due to its inability to be predicted from other columns, and the missingness potentially occuring due to a variable not included in the dataset. Unlike many of the columns we analyzed, the missingness of `pick1` did not depend on league, as many leagues didn't report certain variables as a whole. Instead, 31 out of the 55 leagues had missingness for `pick1` specifically. The missingness also did not depend on `teamid`, as 381 out of 593 teams had a missing value in `pick1`. Instead, the column's missingness depends on the value itself - the champion chosen. Missingness in this column can occur due to the champion being a newly added character to the game, resulting in a null value being recorded. Additional data that would make this column MAR would be data about whether the champion is a newly added character at the time of the game.

In [10]:

```python
# Run a permutation test to examine whether missing gold-at-25 values
# are associated with game length, and visualize the null distribution
# with the observed mean highlighted.
observed_statistic = teams[teams['goldat25'].isna()]['gamelength'].mean()
observed_means = []
teams_copy = teams.copy()
for i in range(1000):
    teams_copy['permuted_lengths'] = np.random.permutation(teams['gamelength'])
    observed_means.append(teams_copy
                          [teams_copy['goldat25'].isna()]
                          ['permuted_lengths'].mean())
p_value = np.mean(observed_statistic >= observed_means)
gamelength_plot = px.histogram(observed_means)
gamelength_plot.add_vline(x=observed_statistic, line_dash="dash",
                          line_color="red",
                          line_width=2,
                          opacity=1)
gamelength_plot.update_layout(showlegend=False,title="Permutation \
test results for goldat25 and gamelength",
    xaxis_title="game lengths",yaxis_title="frequency")
print(f'p_value: {p_value:.3g}')
gamelength_plot.show()
gamelength_plot.write_html("assets/missingness_gamelength.html",include_plotlyjs='c
```

```
p_value: 0
```

Making use of the mean `gamelength` where `goldat25` is missing and permuting the `goldat25` column, our findings showed us a p value of `0`. This lead us to reject our null hypothesis. Based on the test, it is likely that the missingness of `goldat25` was associated with gamelength; gamelength values seem significantly shorter where `goldat25` is missing. This makes sense logically as games that finish early will be more likely to have unrecorded values past their finish time.

In [11]:

```python
# Run a permutation test to examine whether missing gold-at-25
# values are associated with getting first dragon, and visualize
# the null distribution with the observed mean highlighted.
observed_statistic = teams[teams['goldat25'].isna()]['firstdragon'].mean()
observed_means = []
```

```
permuted_firstdragons = teams.copy()
for i in range(1000):
    permuted_firstdragons['permuted_dragons'] = np.random.permutation(
        teams['firstdragon'])
    observed_means.append(permuted_firstdragons
                          [permuted_firstdragons['goldat25'].isna()]
                          ['permuted_dragons'].mean())
p_value = np.mean(observed_statistic >= observed_means).mean()
fig2 = px.histogram(observed_means)
fig2.add_vline(x=observed_statistic, line_dash="dash", line_color="red",
              line_width=2, opacity=1)
fig2.update_layout(showlegend=False, title="Permutation test results for\
goldat25 and firstdragon", xaxis_title="ratio of firstdragons",
    yaxis_title="frequency")
print(f'p_value: {p_value:.3g}')
fig2.show()
fig2.write_html("assets/missingness_firstdragon.html",include_plotlyjs='cdn')
```

```
p_value: 0.501
```

On the other hand, conducting the same missingness experiment returned an insignificant p-value: `0.442` , meaning that we fail to reject the null hypothesis and the missingness of goldat25 is not influenced by the firstdragon column.

Step 4: Hypothesis Testing

To further our exploration into the impact of first dragons on the chance of a team winning the game, we decided to test out the hypothesis suggested by figure 2.4.

- ***Null Hypothesis***: The win rate between teams who secure the first dragon is equal to the win rate of teams that do not secure the first dragon.
- **Alternate Hypothesis**: The win rate between teams who secure the first dragon is significantly higher than the win rate of teams that do not secure the first dragon.
- **Test statistic**: Difference in win rate between games with and without securing the first dragon. Note: testing at significance level of 0.05.

We ran a permutation test: Permuting over the firstdragon column and calculating the difference in winrate between teams who secure a first dragon and teams who do not. The permutation test resulted in a p-value of 0.0 which is less than 0.05, we reject the null hypothesis. Not only this, basing off the graph, this difference appears to be significantly higher. Based on the data, the win rate of teams who secure the first dragon is significantly higher than teams that do not secure the first dragon.

In [12]:

```
# Perform a permutation test for the difference in win rate between teams
# that secure first dragon and those that do not, and visualize the resulting
# null distribution.
dropped_columns = teams[['gameid','teamid','firstdragon','result']]
N = 1000
# Observed difference in proportions
obs_diff = dropped_columns[dropped_columns
```

```
                           ['firstdragon'] == 1]['result'].mean() - \
           dropped_columns[dropped_columns
                           ['firstdragon'] == 0]['result'].mean()

# Permutation test
n_permutations = 10000
diffs = np.zeros(n_permutations)

for i in range(n_permutations):
    shuffled = np.random.permutation(dropped_columns['result'])
    diffs[i] = shuffled[dropped_columns['firstdragon'] == 1].mean() - \
               shuffled[dropped_columns['firstdragon'] == 0].mean()

# Two-sided p-value
p_value = np.sum(diffs >= obs_diff) / N
fig = px.histogram(diffs,nbins=30)
fig.add_vline(x=obs_diff, line_color='red', line_dash='dash', line_width=2)
fig.add_annotation(
    x=obs_diff,
    y=1,
    text=f'Observed diff = {obs_diff:.3f}',
    yshift=100,
    showarrow=False,
    xshift=10,
    textangle=-90,
)
fig.update_layout(showlegend=False,
                  title="Permutation test for firstdragon against winrate",
                  xaxis_title="Difference in win rate between games with and\
                      without first dragon",
                  yaxis_title="frequency")
fig.show()
print(f"Observed difference: {obs_diff:.3g}")
print(f"Permutation p-value: {p_value:.3g}")

fig.write_html("assets/hypothesis_test.html",include_plotlyjs='cdn')
```

```
Observed difference: 0.157
Permutation p-value: 0
```

Step 5: Framing a Prediction Problem

Our prediction model is a classifier to predict the game result using the data at the 25 minute mark. Since the response variable could have one of two results - winning and losing - our model is a binary classifier. Our response variable of choice aligns with all our previous intentions so far to find the ideal tactic to win the game. To evaluate our model we will use the F1-score, the data reported may vary by league and result in bias, therefore a balance between precision and recall suits our needs.

Step 6: Baseline Model

In [13]:

```
# Build and evaluate a baseline decision-tree classifier that uses 25-minute
# gold/xp metrics to predict game outcome (including precision, recall, and F1).
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
cols = teams.columns
cols = [i for i in cols if '25' in i]
cols.append('result')
X_train, X_test, y_train, y_test = train_test_split(
    teams[['goldat25','xpat25']], teams['result'], test_size=0.2,
    random_state=42
)

pipe = Pipeline([       # preprocessing step
    ("clf", DecisionTreeClassifier(
    criterion="gini",      # or "entropy", "log_loss"
    max_depth=None,        # None = grow until pure / min_samples
    random_state=42        # controls randomness (None = fully random each run)
))
])
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
acc = accuracy_score(y_test, y_pred)
precision = y_pred.astype(int) @ y_test.astype(int) / y_pred.sum()
recall = y_pred.astype(int) @ y_test.astype(int) / y_test.sum()
F_1 = 2*(precision*recall) / (precision+recall)
print(f"Test precision: {precision:.3f}")
print(f'Test recall: {recall:.3f}')
print(f'Test F_1: {F_1:.3f}')
```

```
Test precision: 0.698
Test recall: 0.558
Test F_1: 0.620
```

Step 7: Final Model

In [14]:

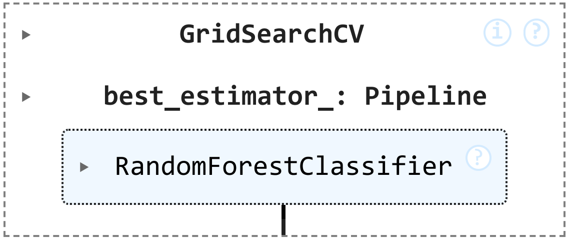
```
# Define a RandomForestClassifier pipeline and hyperparameter grid, preparing
# for cross-validated model tuning using F1 as the scoring metric.
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold




final= teams[cols].dropna(axis=0,how="any")

# Creating Pipeline

pl = Pipeline([
 ('clf', RandomForestClassifier(
 random_state=42,
 n_jobs=-1,
 ))
])
n_features = X_train.shape[1]
kf = KFold(n_splits=5, shuffle=True, random_state=42)
hyperparameters = {
 'clf__max_depth': [None, 5, 10, 20],
 'clf__n_estimators': [100, 200, 300, 500],
 'clf__max_features':list({
        'sqrt',
        'log2',
    }),
 'clf__bootstrap': [True, False],
}
grids = GridSearchCV(
 pl,
 n_jobs=-1,
 param_grid=hyperparameters,
 return_train_score=True,
 cv=kf,
 scoring="f1",
 verbose=2,
)
```

In [15]:

```
# Generate quadratic interaction features for the selected predictors,
# split into train/test sets, and fit the tuned random forest on these
# engineered features.

import itertools
final_no_result = final[[i for i in cols if i !="result"]]
final_result = final['result']

def make_quadratic(df):
    combinations =itertools.combinations(df.columns,r=2)
    return_dict = {}
    for combination in combinations:
        return_dict[f'{combination[0]}_{combination[1]}'] = \
            df[combination[0]]*df[combination[1]]
    return pd.DataFrame(return_dict)

fe_final = make_quadratic(final_no_result)
X_train_fe, X_test_fe, y_train_fe, y_test_fe = train_test_split(
    fe_final,final_result, test_size=0.2, random_state=42
)

grids.fit(X_train_fe, y_train_fe)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
```

Out[15]:



▸ GridSearchCV

▸ best_estimator_: Pipeline

▸ RandomForestClassifier

In [16]:

```
# Evaluate the tuned random-forest model on the test set, reporting best
# hyperparameters, cross-validated F1, and held-out performance.

predictions = grids.predict(X_test_fe)
precision = predictions.astype(int) @ y_test_fe.astype(int) / predictions.sum()
recall = predictions.astype(int) @ y_test_fe.astype(int) / y_test_fe.sum()
print("Best params:", grids.best_params_)
print("Best CV score:", grids.best_score_)
print("Test score:", grids.score(X_test_fe, y_test_fe))
```

```
Best params: {'clf__bootstrap': True, 'clf__max_depth': 5, 'clf__max_features': 'log
2', 'clf__n_estimators': 100}
Best CV score: 0.8393957185372913
Test score: 0.8466716529543755
```

Step 8: Fairness Analysis

In [17]:

```
# Use the final model to compute F1 scores separately for each league and
# visualize these values to support a fairness comparison.

leagues = teams['league'].unique()
dataframe = {}

for league in leagues:
    filtered_no_results = \
        teams[teams['league']==league][[i for i in cols if i !='result']]
    filtered_results = teams[teams['league']==league]['result']
    filtered_final = make_quadratic(filtered_no_results)
    predictions = grids.predict(filtered_final)
    F_1 = grids.score(filtered_final,filtered_results)

    dataframe[league] = {"F_1": F_1}

split_dataframe = pd.DataFrame(dataframe).T
split_dataframe.plot(y='F_1',kind='bar')
fig = px.bar(split_dataframe, barmode='group')
fig.update_layout(title="F-1 scores for each league", yaxis_title="F-1 score",
                  xaxis_title="league",showlegend=False)
fig.show()
fig.write_html("assets/Side-by-side_precision_for_each_league.html")
```

In [18]:

```
# Define a test statistic for the fairness permutation test as the difference
# in F1 score between LCL and all other leagues.

def compute_test_statistic(teams):
    LCL = teams[teams['league']=='LCL'][[i for i in cols if i !='result']]
    Not_LCL = teams[teams['league']!='LCL'][[i for i in cols if i !='result']]
    LCL_results = teams[teams['league']=='LCL']['result']
    Not_LCL_results = teams[teams['league']!='LCL']['result']
    LCL_quad = make_quadratic(LCL)
    Not_LCL_quad = make_quadratic(Not_LCL)
    LCL_f1 = grids.score(LCL_quad,LCL_results)
    Not_LCL_f1 = grids.score(Not_LCL_quad,Not_LCL_results)
    return LCL_f1-Not_LCL_f1
```

```
# Compute Observed statistic and conduct a permutation test across 1000
# iterations
observed_statistic = compute_test_statistic(teams)
N = 1000
f1s = []
for i in range(N):
    copy = teams.copy()
    copy['league'] = np.random.permutation(teams['league'])
    test_statistic = compute_test_statistic(copy)
    f1s.append(test_statistic)
fig = px.histogram(f1s, nbins=30)
fig.add_vline(x=observed_statistic,line_width=2,line_dash="dash",
              line_color="red", opacity=1)
fig.update_layout(showlegend=False,
                  title_text="Distribution of model F-1",
                  title_x=0.5,  # Center the title
                  title_xanchor='center')
fig.show()
fig.write_html('assets/Distribution of Model F-1.html')
```

In [19]:

```
# Estimate and print the permutation p-value for the fairness test comparing LCL
# performance to other leagues.
p_value = (f1s>=observed_statistic).sum() / len(f1s)
print(f'p_value: {p_value:.3g}')
```

```
p_value: 0.001
```