Draft Diff: A League of Legends Draft Win Predictor

Erica Jean, Gabriella Rosal-Calit, Lance Tam, Reagan Schmidt

What is League of Legends?

League of Legends is a popular online game played by millions. In this game, two teams with five players each compete to destroy the other's base on a map split into lanes and a jungle area.

Each player controls a unique character known as a 'champion'. Champions have distinct abilities and play styles. There are currently 166 champions.

The map is divided into three lanes – top, middle, and bottom – and a neutral 'jungle' area with non-player characters and objectives.

Teams win by strategically navigating the map, defeating enemy champions, and destroying the enemy base.

What is The Goal?

- Our project aims to predict which team will win based on their selection of champions before the game starts, a process known as 'drafting'.

- The draft is crucial as it determines the composition of each team, influencing their strategy and chances of victory.

- We will analyze historical match data to identify patterns and trends that can predict match outcomes based on draft choices.

Who Are The Stakeholders?

This project is of interest to:

- Players
- Coaches
- Game Analysts
- Riot Balance Team

What Are The Benefits?

Players can see whether their team has a good chance of winning based on their champion picks

Coaches/analysts can get insight into what team compositions are good into other team compositions without having to play many games

Riot's, the creator of League of Legends, balance team can statistically determine if a champion is too powerful based on their impact on the prediction from our model

What Is Our Action Plan?

1. Scrape the data (we encountered an obstacle that the layout made it difficult to scrape so we used an API to pull data from it)
2. Clean the data
3. Use the data to create features that are statistically significant to a game’s outcome
4. Experiment with different models
5. Find the best parameters for each model
6. Compare model accuracy to pick the best model

How Was The Data Collected?

- Queried Riot API directly to get list of Challenger, GM, Master players (NA)
- Javascript Fetch API to query u.gg for match data from each player
- Match data:
  - List of players in the match
  - IDs of champions played
  - ID of position played for each player
  - Game duration, result of match, match ID
- Filtered matches by match ID to ensure uniqueness
- Used mapping from Riot API to match champion ID to champion name

JS Fetch Code

```
promiseArr.push(

    new Promise(async (resolve, reject) => {


        const uri = "https://u.gg/lol/profile/na1/" + currName + "/champion-stats"
        const encode = encodeURI(uri)
        try {
            let currFetch = await fetch("https://u.gg/api",
                {
                    method: "POST",
                    headers: {
                        "Content-Type": "application/json",
                        "Origin": "https://u.gg",
                        //"Referer": "https://u.gg/lol/profile/na1/" + currName+ "/champion-stats"
                        "Referer": encode
                    },
                    body: '{"operationName":"getPlayerStats","variables":{"regionId":"na1","summonerName":"' + currName + '","queueType":[420],"role":7,"seaso
                }
            )

            let parsedResults = await currFetch.json()

            // console.log(JSON.stringify(parsedResults))


            if (currFetch.status !== 200) {
                console.log("ERROR WRONG STATUSSSSS")
            }

            let champData = parsedResults.data.fetchPlayerStatistics[0].basicChampionPerformances
            //console.log(champData)

            if (champData != undefined) {


                for (let i = 0; i < champData.length; i++) {


                    let currChamp = {
                        summoner: currName,
                        championPlayed: champData[i].championId,
                        champAssists: champData[i].assists,
                        champKills: champData[i].kills,
                        champDeaths: champData[i].deaths,
                        totalPlayed: champData[i].totalMatches,
                        totalWon: champData[i].wins
```

```
    for (let i = 0; i < matches.length; i++) {

        let currMatch = {
            summoner: currName,
            championPlayed: "",
            champsPlayed: [],
            didWin: "",
            gameDuration: "",
            otherSummoners: [],
            matchID: ""
        }

        currMatch.didWin = matches[i].win
        currMatch.championPlayed = matches[i].championId
        currMatch.gameDuration = matches[i].matchDuration
        currMatch.matchID = matches[i].matchId

        for (let n = 0; n < matches[i].teamA.length; n++) {
            currMatch.champsPlayed.push("A," + matches[i].teamA[n].championId + "," + matches[i].teamA[n].role)
            currMatch.otherSummoners.push("A," + matches[i].teamA[n].summonerName)
        }

        for (let n = 0; n < matches[i].teamB.length; n++) {
            currMatch.champsPlayed.push("B," + matches[i].teamB[n].championId + "," + matches[i].teamB[n].role)
            currMatch.otherSummoners.push("B," + matches[i].teamB[n].summonerName)
        }

        fs.appendFile("./data.json", JSON.stringify(currMatch) + ",", function(err) {
            if (err) {
                console.log(err);
            }
        });
        /*
        fs.appendFile("./solution.csv", JSON.stringify(currMatch) + "\n", function (err) {
            if (err) {
                return console.log(err);
            }
        });
        */
    }
}
```

What Features Did We Include?

- Ideally, player skill would not be included as a factor in determining the games.
  - Limit to high-elo Challenger games only.
  - Did not count player mastery or win rate
- Did not include any in-game performance metrics such as gold at 10 minutes, items purchased, first bloods, objectives taken, kills, deaths, assists, etc.
- Since this is a Draft PICK analyzer, we wanted to keep it limited to the champion’s played.


```
matchSummaries: [ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, … ]
  0: Object { kills: 2, primaryStyle: 8000, subStyle: 8400, … }
    kills: 2
    primaryStyle: 8000
    subStyle: 8400
    visionScore: 13
    teamA: [ {…}, {…}, {…}, {…}, {…} ]
    items: [ 1055, 3031, 6672, 3006, 3044, 1028, 3363 ]
    win: false
    deaths: 11
    championId: 777
    runes: [ 8008, 9111, 9104, 8299, 8444, 8451 ]
    level: 15
    psHardCarry: 72
    summonerSpells: [ 4, 12 ]
    matchCreationTime: 1701832668239
    killParticipation: 32
    gold: 10654
    cs: 207
    version: "13_23"
    augments: [ 0, 0, 0, 0 ]
    psTeamPlay: 70
    regionId: "na1"
    matchDuration: 1666
    teamB: [ {…}, {…}, {…}, {…}, {…} ]
    maximumKillStreak: 1
    damage: 14582
    jungleCs: 0
    matchId: 4850867437
    role: 4
    summonerName: "Pobelter"
    lpInfo: Object { __typename: "LpInfo", lp: -23, placement: -1, … }
    queueType: "ranked_solo_5x5"
    __typename: "MatchSummary"
    assists: 4
```

First Iteration: Champions on Each Team

- Simply uses which champions are on team A or B to determine who wins

	Match ID	Team A Won?	A Top	A Jungle	A Mid	A Bot	A Support	B Top	B Jungle	B Mid	B Bot	B Support
0	4848223288	1	Jax	Nunu	Syndra	Vayne	Shaco	Akali	Zac	Qiyana	Twitch	Rakan
1	4848131748	1	Yone	Poppy	TwistedFate	Draven	Milio	Chogath	MonkeyKing	Sylas	Kalista	Renata
2	4848087528	1	Jax	Viego	Orianna	Kalista	Sona	Shen	Udyr	Neeko	Jinx	Thresh
3	4847265470	0	Azir	Kayn	Vladimir	Jhin	Alistar	Garen	Nocturne	Zed	Kaisa	Rakan
4	4848043757	1	Rumble	Kindred	Zoe	Caitlyn	Senna	Tryndamere	Khazix	Vladimir	Sivir	Alistar
...	...	...	...	...	...	...	...	...	...	...	...	...
324818	4814345455	1	Aatrox	Nocturne	Sylas	Heimerdinger	Senna	Garen	Maokai	Veigar	Twitch	Thresh
324820	4813002074	0	Renekton	Warwick	Ziggs	Twitch	Alistar	KSante	Briar	Xerath	Tristana	Seraphine
324822	4819650090	0	Jax	Graves	Nasus	Heimerdinger	Bard	Chogath	Trundle	Irelia	Vayne	Milio

Splitting Our Dataset

We split our dataset using the train_test_split function, using a test size of 30% and a training size of 70%.

Then we split the training set into 80% train and 20% validation.

Train and Validation set were used to evaluate models on all DataFrames

Once we picked the best DataFrame - we use cross validation to optimize hyperparameters for each model

Best Model was picked on test accuracy

Model Evaluation

- Decision Tree:

```
clf = tree.DecisionTreeClassifier(random_state=42).fit(X_train, y_train)
clf.score(X_val, y_val)
```

```
0.5080604674513304
```

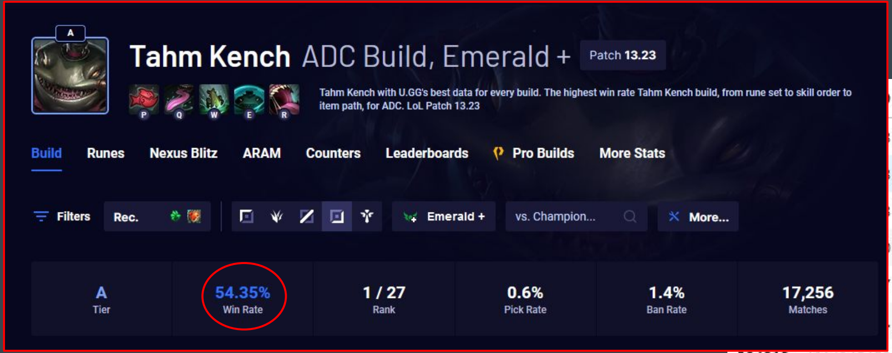
- Neural Network:

```
mlp = MLPClassifier(random_state=42).fit(X_train, y_train)
mlp.score(X_val, y_val)
```

```
0.5106733772995488
```

Second Iteration: Champion Win Rates on Each Team

- Same as the first DataFrame, but replace champion names with that champion's overall win rate
- Originally wanted to pull from U.GG, but issues came up and decided to calculate based on that champion's win rate in our 128,045 unique games dataset.



		Team A Won?	A Top	A Jungle	A Mid	A Bot	A Support	B Top	B Jungle	B Mid	B Bot	B Support
		1	0.521313	0.503173	0.503341	0.516784	0.472906	0.497863	0.505829	0.502770	0.522730	0.506754
		1	0.479153	0.483957	0.505706	0.519829	0.494923	0.502588	0.466134	0.520461	0.510774	0.506617
		1	0.521313	0.485958	0.493604	0.510774	0.512271	0.497370	0.508568	0.506992	0.507348	0.514714
		0	0.549865	0.503162	0.506785	0.494525	0.509598	0.490364	0.486515	0.503803	0.480917	0.506754
		1	0.493161	0.528064	0.514953	0.483096	0.517835	0.523136	0.495181	0.506785	0.482210	0.509598
		...	...	...	...	...	...	...	...	...	...	...
324818	4814345455	1	0.487486	0.486515	0.520461	0.472028	0.517835	0.490364	0.477379	0.464943	0.522730	0.514714
324820	4813002074	0	0.489415	0.470032	0.485156	0.522730	0.509598	0.510396	0.489342	0.471320	0.521180	0.478411
324822	4819650090	0	0.521313	0.517958	0.503480	0.472028	0.521975	0.502588	0.484444	0.496939	0.516784	0.494923
324823	4812981843	1	0.490364	0.505829	0.488907	0.527675	0.483742	0.518345	0.517958	0.505618	0.521180	0.483875
324824	4814686156	1	0.501658	0.484444	0.485156	0.503527	0.492683	0.523136	0.523916	0.500270	0.521180	0.491969

Model Evaluation

- K-Nearest Neighbors:

```
knn = KNeighborsClassifier().fit(X_train, y_train)
knn.score(X_val, y_val)
```

```
0.5158699168851453
```

- Random Forest:

```
# Trying Random Forest Ensemble
clf = RandomForestClassifier(max_depth = 5, random_state=42).fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
0.5656384224912143
```

- Neural Network:

```
knn = KNeighborsClassifier().fit(X_train, y_train)
knn.score(X_val, y_val)
```

```
0.5158699168851453
```

Third Iteration: Including the Bot Lane (ADC + Support) Duo Win Rate for Each Team

- Same as second iteration, but with two more features
- The “Bot Lane Duo Win Rate” takes every combination of (ADC, Support) that showed up in our dataset and calculated its win rate within our dataset
- Accuracy came out to 57.5% on our MLP model


```
duo_win_rate_dict['MissFortune Blitzcrank']
```

```
0.5358851674641149
```

A Duo	B Duo
0.551724	0.554307
0.536680	0.498225
0.596154	0.550744
0.510638	0.468321
0.472785	0.524064
...	...
0.433333	0.526749
0.493007	0.538462
0.250000	0.521595

Model Evaluation

- Random Forest

```
clf = RandomForestClassifier(max_depth = 5, random_state=42).fit(X_train, y_train)
clf.score(X_val, y_val)
```

```
0.5734367155686952
```

- K-Nearest Neighbors

```
knn = KNeighborsClassifier().fit(X_train, y_train)
knn.score(X_val, y_val)
```

```
0.536620739666425
```

Fourth Iteration: Including Team A’s Champion Win Rate vs. Their Opposing Laner

- Adding onto the 3rd Iteration, now we include how the champions on each team fare against their opposing laner.
- (Top lane vs. top lane), (Mid vs. Mid), etc.
- As one can see, in top lane, Riven has a 53.97% win rate against Aatrox and Aatrox has a (100-53.97) = 46.02% win rate against Riven.

```
opp_win_rate_dict['Riven Top,Aatrox Top']

0.5397301349325337

opp_win_rate_dict['Aatrox Top,Riven Top']

0.46026986506746626
```

A Top MU	A Jungle MU	A Mid MU	A Bot MU	A Support MU
0.492611	0.408602	0.512658	0.502732	0.512821
0.493151	0.476190	0.436224	0.531835	0.552239
0.563536	0.544000	0.500000	0.462908	0.484507
0.400000	0.500000	0.468750	0.511128	0.498871
0.443038	0.535117	0.580952	0.513889	0.508224
...	...	...	...	...
0.465549	0.459184	0.500000	0.500000	0.484453
0.520522	0.512195	0.519231	0.512456	0.555556
0.507937	0.484848	0.428571	0.666667	0.537838
0.435294	0.503226	0.416667	0.495356	0.522901
0.563758	0.444444	0.608696	0.455285	0.480769

Model Evaluation

- Neural Network

```
mlp = MLPClassifier(random_state=42).fit(X_train, y_train)
mlp.score(X_val, y_val)
```

```
0.649746192893401
```

- Random Forest

```
clf = RandomForestClassifier(max_depth = 5, random_state=42).fit(X_train, y_train)
clf.score(X_val, y_val)
```

```
0.6341830758074413
```

- K-Nearest Neighbors

```
knn = KNeighborsClassifier().fit(X_train, y_train)
knn.score(X_val, y_val)
```

```
0.6002119707703464
```

Further Model Evaluation - Cross Validation

Neural Network:

```
mlp = MLPClassifier()
mlp_cv = RandomizedSearchCV(mlp, param_distributions=param_dist, n_iter=10, cv=5, scoring='accuracy')
mlp_cv.fit(X_train, y_train)
```

```
mlp_test_accuracy = mlp_cv.score(X_test, y_test)

0.6493958903993728
```

Random Forest:

```
rf = RandomForestClassifier()
rf_cv = RandomizedSearchCV(rf, param_distributions=param_dist, n_iter=10, cv=5, scoring='accuracy')
rf_cv.fit(X_train, y_train)
```

```
rf_test_accuracy = rf_cv.score(X_test, y_test)

0.6424786165117398
```

KNN:

```
knn = KNeighborsClassifier()
knn_cv = RandomizedSearchCV(knn, param_distributions=param_dist, n_iter=10, cv=5, scoring='accuracy')
knn_cv.fit(X_train, y_train)
```

```
knn_test_accuracy = knn_cv.score(X_test, y_test)

0.6173756914073423
```

Parameters Chosen for the Model

Best DataFrame:

- DataFrame with the win rates of champions, Bot/Support Duo, and Opposing Laner

	Match ID	Team A Won?	A Top	A Jungle	A Mid	A Bot	A Support	B Top	B Jungle	B Mid	B Bot	B Support	A Duo	B Duo	A Top MU	A Jungle MU	A Mid MU	A Bot MU	A Support MU
0	4848223288	1	0.521313	0.503173	0.503341	0.516784	0.472906	0.497863	0.505829	0.502770	0.522730	0.506754	0.551724	0.554307	0.492611	0.408602	0.512658	0.502732	0.512821
1	4848131748	1	0.479153	0.483957	0.505706	0.519829	0.494923	0.502588	0.466134	0.520461	0.510774	0.506617	0.536680	0.498225	0.493151	0.476190	0.436224	0.531835	0.552239
2	4848087528	1	0.521313	0.485958	0.493604	0.510774	0.512271	0.497370	0.508568	0.506992	0.507348	0.514714	0.596154	0.550744	0.563536	0.544000	0.500000	0.462908	0.484507
3	4847265470	0	0.549865	0.503162	0.506785	0.494525	0.509598	0.490364	0.486515	0.503803	0.480917	0.506754	0.510638	0.468321	0.400000	0.500000	0.468750	0.511128	0.498871
4	4848043757	1	0.493161	0.528064	0.514953	0.483096	0.517835	0.523136	0.495181	0.506785	0.482210	0.509598	0.472785	0.524064	0.443038	0.535117	0.580952	0.513889	0.508224
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
128040	4814345455	1	0.487486	0.486515	0.520461	0.472028	0.517835	0.490364	0.477379	0.464943	0.522730	0.514714	0.433333	0.526749	0.465549	0.459184	0.500000	0.500000	0.484453
128041	4813002074	0	0.489415	0.470032	0.485156	0.522730	0.509598	0.510396	0.489342	0.471320	0.521180	0.478411	0.493007	0.538462	0.520522	0.512195	0.519231	0.512456	0.555556
128042	4819650090	0	0.521313	0.517958	0.503480	0.472028	0.521975	0.502588	0.484444	0.496939	0.516784	0.494923	0.250000	0.521595	0.507937	0.484848	0.428571	0.666667	0.537838
128043	4812981843	1	0.490364	0.505829	0.488907	0.527675	0.483742	0.518345	0.517958	0.505618	0.521180	0.483875	0.495050	0.494253	0.435294	0.503226	0.416667	0.495356	0.522901
128044	4814686156	1	0.501658	0.484444	0.485156	0.503527	0.492683	0.523136	0.523916	0.500270	0.521180	0.491969	0.428571	0.491379	0.563758	0.444444	0.608696	0.455285	0.480769

128045 rows × 19 columns

Best Model & Parameters:

- Neural Network (MLPClassifier)

```
{'activation': 'relu',
 'alpha': 0.1,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'max_iter': 312,
 'random_state': 42,
 'solver': 'lbfgs'}
```

Neural Network - 0.6493958903993728

Random Forest - 0.6424786165117398

K-Nearest Neighbors - 0.6173756914073423

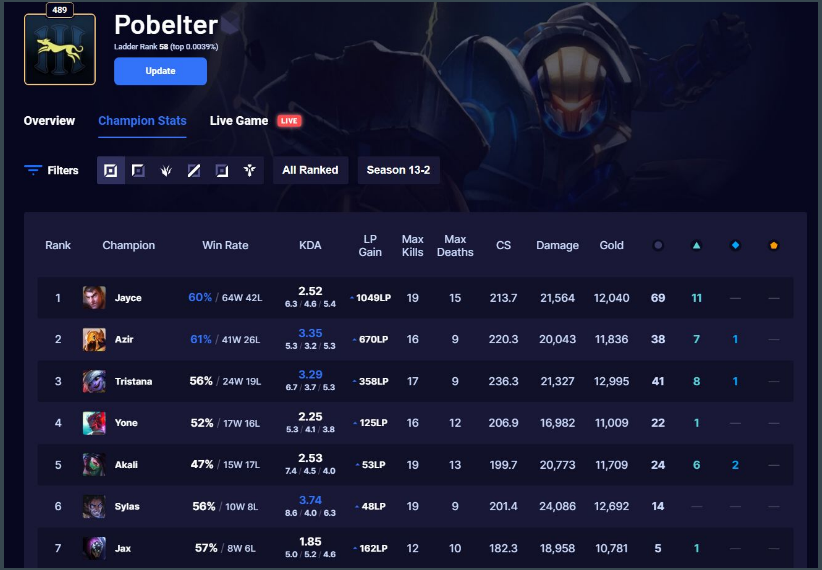
Addendum: What if We Include Player Skill in Our Classifier?

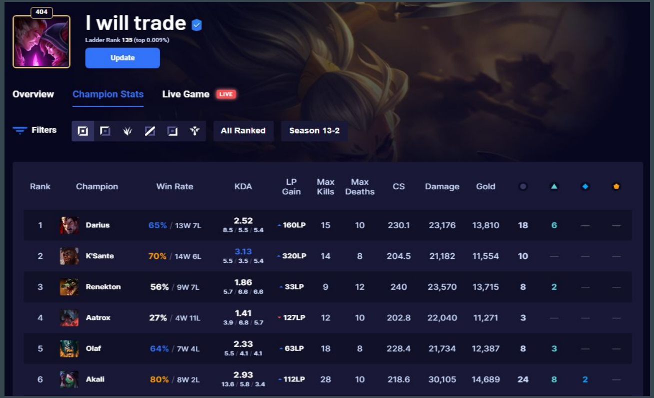
- Wanted to see what would happen if the individual player's win rate on their respective champion was included as a feature.
- Until this point, only overall champion win rate was included. Individual player skill had no influence on our model.

B Jungle	B Mid	...	A Top Player Winrate	A Jungle Player Winrate	A Mid Player Winrate	A Bot Player Winrate	A Support Player Winrate	B Top Player Winrate	B Jungle Player Winrate	B Mid Player Winrate	B Bot Player Winrate	B Support Player Winrate
0.505829	0.502770	...	0.602410	0.525000	0.500000	0.535714	0.615385	0.400000	0.548287	0.546763	0.496711	0.560000
0.466134	0.520461	...	0.545455	1.000000	0.750000	0.516204	0.500000	0.307692	0.250000	0.555556	0.569620	0.375000
0.508568	0.506992	...	0.602410	0.583333	0.700000	0.569620	0.555195	0.550000	0.662162	0.500000	0.583333	0.666667
0.486515	0.503803	...	0.593878	0.636364	0.543796	0.666667	0.510204	0.530201	0.548387	0.600897	1.000000	0.477064
0.495181	0.506785	...	0.750000	0.584016	0.714286	0.476923	0.529412	0.375000	0.541667	0.543796	0.571429	0.391304
...	...	...	...	...	...	...	...	...	...	...	...	...
0.477379	0.464943	...	0.542857	0.409836	0.625000	0.619048	0.468750	0.333333	0.500000	0.568627	0.606061	0.657895
0.489342	0.471320	...	0.166667	0.483696	0.000000	0.500000	0.000000	0.523077	0.550000	1.000000	0.475000	0.750000
0.484444	0.496939	...	0.200000	0.602151	0.523810	0.619048	0.500000	1.000000	0.400000	0.666667	0.454545	0.520000
0.517958	0.505618	...	0.542969	0.500000	0.577287	0.500000	0.375000	0.530466	0.576923	0.000000	0.263158	0.250000
0.523916	0.500270	...	0.520000	0.500000	0.591837	0.666667	0.601852	0.384615	0.666667	0.500000	0.263158	0.636364

Player Champion Stat Data

- Javascript Fetch API to query u.gg for player champion stats
- Champion Stat Data
  - ID of Champion
  - Matches won
  - Matches lost
  - All-time kills, deaths, assists





Quick MLP Classifier

- Accuracy shot up to a whopping 89.4%!
- It seems player skill is a very important predictor of the draft. Although this is to be expected, one would hope it'd be a different story!

```
# Trying MLP
X = winrate_df.drop(['Match ID', 'Team A Won?'], axis=1)
y = winrate_df['Team A Won?']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2, random_state=42)

mlp = MLPClassifier(random_state=42).fit(X_train, y_train)
mlp.score(X_test, y_test)
```

```
0.8939825842477254
```

What Are Our Conclusions?

- Based on initial models not including player win rate, the champion draft only resulted in about 65% accuracy - implies game is relatively balanced.
- Champion meta does improve certain composition's ability to win, but more important is the player piloting it.
- Including player win rates, accuracy jumped to around 89% - implies that player skill with their specific champion plays a significant role in overall win rate
- This is not unexpected, but we thought that the champions themselves might play a more significant role
- Our model can be used to predict draft win probability with reasonable accuracy, but could be expanded even further

What Are Our Next Steps?

- Model can be expanded to other regions (Korea, EU, etc)
- Include data from other ranked tiers, could focus on specific ranks
- Models for other game-modes (Flex, ARAM, etc)
- Include variables for rune setups, which might affect a champion's power
- Include variables for item build paths
- Dragon buffs/soul + Rift Herald interactions