LoL Reduction

Roland Zeng

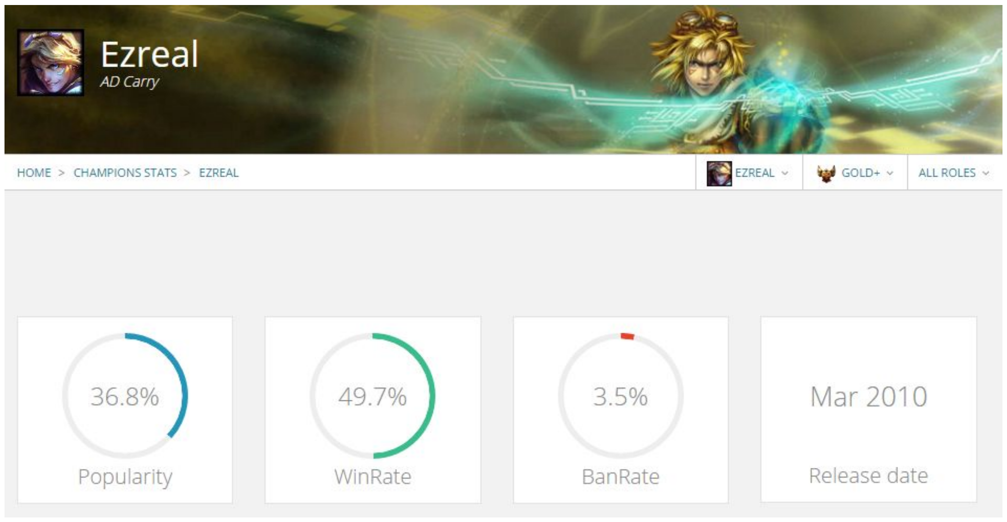
CS170 Prof. Parker
Winter 2016

Introduction

The online game League of Legends is a MOBA (Multiplayer Online Battle Arena), where two teams, each consisting of five players, attempt to destroy the other's base. Each of the ten players selects a character, known as a "champion," out of a list of 129 available champions. The players attempt to utilize quick reflexes, strategies, and teamwork to coordinate and beat their opponents. After each game, a list of statistics is displayed, such as how much damage each champion dealt out, how much gold each player earned, etc. These factors all play an integral role in determining which team won the game.

Riot Games, the creators of League of Legends, owns a publicly available API where all of this match data is stored. There are millions of match data available. For example, the third-party site "www.leagueofgraphs.com," which pulls daily from the API, has analyzed 24,450,300 games as of February 16th. This means there is a LOT of available data, and from this, a lot of cool patterns can be observed.

Generally, a champion has a unique set of abilities that can be used to gain an upper hand over opposing champions. What determines how strong a champion is? One of the biggest factors is the champion's win-rate, a ratio of the number of victories a champion has participated and the total number of games the champion was played in (Figure 1). Ideally, every champion's win-rate should hover around 50%, reflecting a fair and balanced game. Every patch (patches occur every few weeks), weak champions have their skills strengthened, or "buffed," and strong champions have their skills weakened, or "nerfed."



36.8% Popularity

49.7% WinRate

3.5% BanRate

Mar 2010 Release date

Figure 1 : Popularity, Win Rate, and Ban Rate for the champion "Ezreal." A win-rate of 49.7% is good, as it means he's fairly balanced. Picture from: http://www.leagueofgraphs.com/champions/stats/ezreal

In many cases, champions are solely buffed or nerfed based on their win-rates. After all, a champion that is winning most of the time it’s being played has to be very strong. However, I did not think it was right to consider champion strength based solely on win-rate.

Throughout the course of a match, there are several factors that determine the strength of a champion. For example, champions become more powerful by buying items with gold. Gold is acquired by killing enemy champions, enemy minions (tiny AI monsters), and jungle creeps (neutral AI monsters). Hence, a champion’s gold gain ought to be correlated with its strength. Because champions gain large amounts of gold by killing other champions, perhaps the average kills a champion obtains during a game is also a deciding factor. Players have the chance to ban out champions (prevent them from being picked) every match; perhaps champions that are often banned are considered strong.

I wanted to create a holistic approach to determining champion strength, based on various post-match statistics (gold gain, kills, ban rate, etc). Once I had accumulated the data, I could attempt to look for statistical outliers, which could mean champions that were too weak or too strong. Then, I could compare these outliers with previous lists of patch notes to see if these champions were recently buffed or nerfed, and analyze the results of game balance.

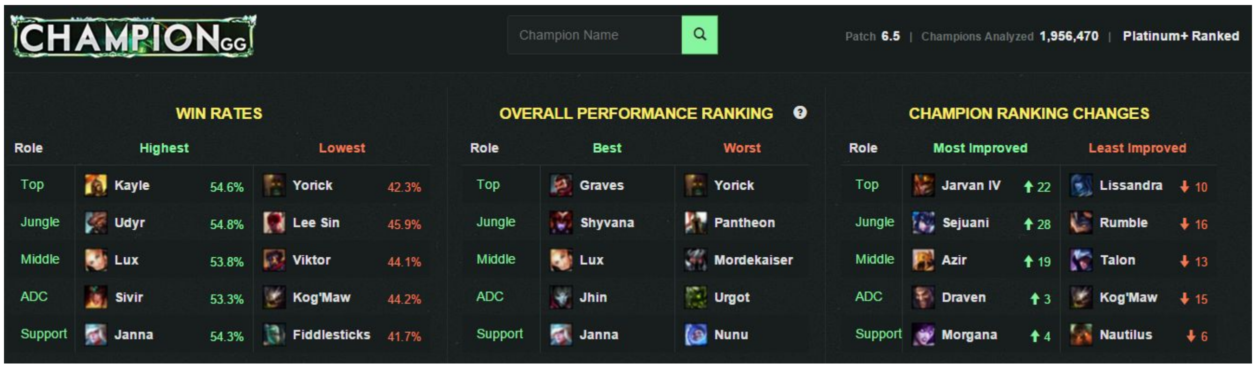
However, I had no way of comparing champions against each other if each champion was represented by a set of metrics. Fortunately, in class, we approached a similar problem of how to compare Hall of Fame baseball players. It was impossible to visually compare baseball players, where each player was described by 22 different metrics. In that homework assignment, we used a random k-D projection to reduce those 22 metrics into an easily visualized x-y graph. Likewise, I can run a similar dimension analysis on the matrix of 129 champions and however many metrics I decide on (champion winrate, popularity, etc.) to attempt to find which champions are outliers. Then, I can perhaps cross this data with patch trends to figure out if champions that are ‘outliers’ of my dimensional analysis were changed in the next patch (patches are meant to balance out champions who are too weak or too strong). This could mean that current champion outliers could predict the pattern of champion changes in the next patch, and lead to more interesting observations and predictions.

Data Acquisition

Part 1 : Finding Data

First, in order to obtain the champion statistics and metrics that I wanted to analyze, I would have to build my own datasets. The most straightforward solution is to use Riot's public API. One API call corresponds to one match, and I would have to make hundreds of thousands of API calls in order to gather an accurate dataset. Riot's API throttles non-registered developers to make only 10 API calls every 10 seconds, so getting a very large dataset is unfeasible and would take too long. Settling for a smaller dataset (100,000s of matches) would lead to inaccurate reporting. As a result, it was way more feasible to search for an existing dataset of matches to use.

Luckily, I found the third-party site ChampionGG (*https://champion.gg*). They had recently released their own API, which allowed developers to call upon the data they had in turn collected from Riot's API. ChampionGG's database contained the results of analyzing 1.9 million matches (Figure 2).



Patch 6.5 | Champions Analyzed 1,956,470 | Platinum+ Ranked

WIN RATES

Role	Highest		Lowest	
Top	Kayle	54.6%	Yorick	42.3%
Jungle	Udyr	54.8%	Lee Sin	45.9%
Middle	Lux	53.8%	Viktor	44.1%
ADC	Sivir	53.3%	Kog'Maw	44.2%
Support	Janna	54.3%	Fiddlesticks	41.7%

OVERALL PERFORMANCE RANKING

Role	Best	Worst
Top	Graves	Yorick
Jungle	Shyvana	Pantheon
Middle	Lux	Mordekaiser
ADC	Jhin	Urgot
Support	Janna	Nunu

CHAMPION RANKING CHANGES

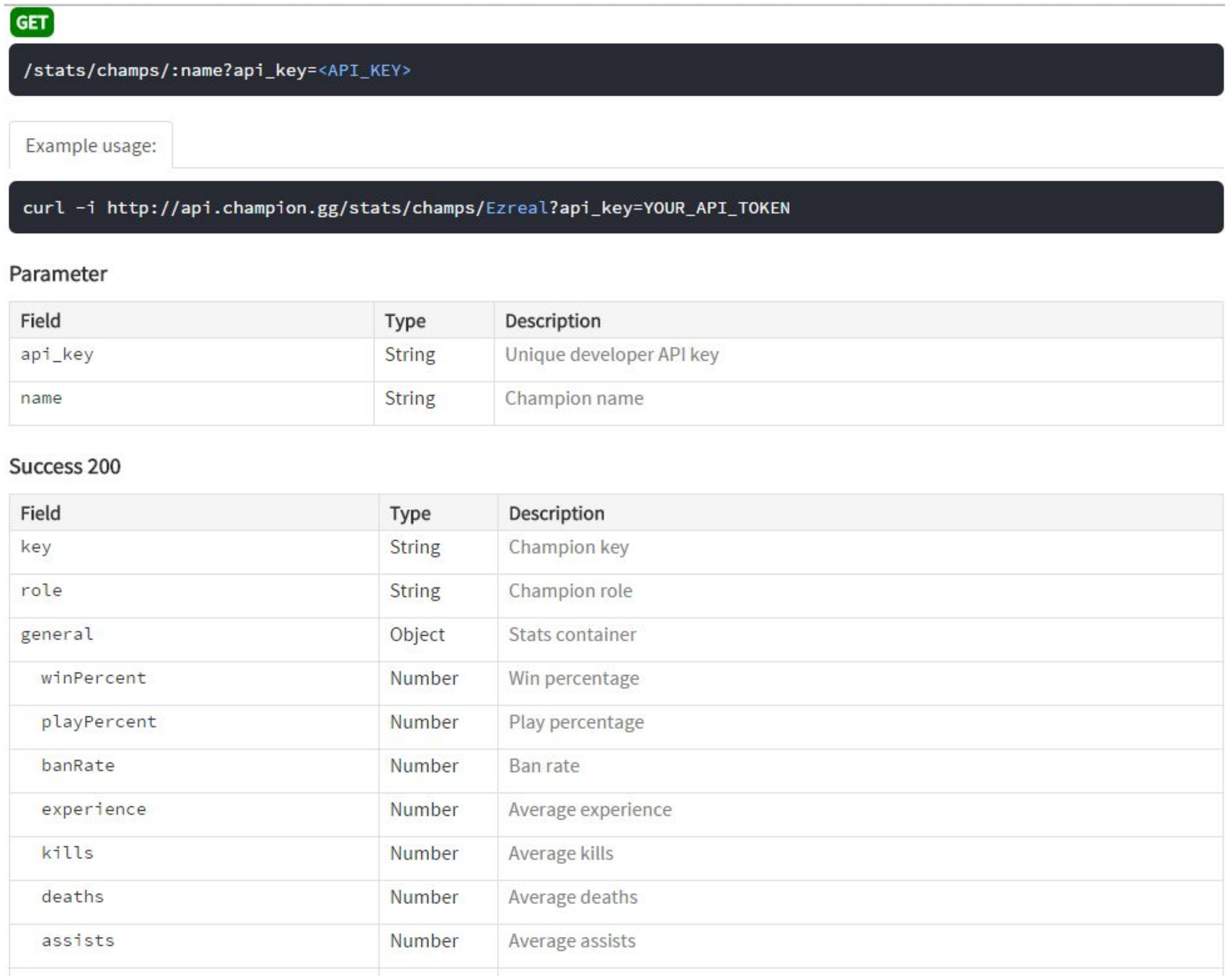
Role	Most Improved		Least Improved	
Top	Jarvan IV	↑ 22	Lissandra	↓ 10
Jungle	Sejuani	↑ 28	Rumble	↓ 16
Middle	Azir	↑ 19	Talon	↓ 13
ADC	Draven	↑ 3	Kog'Maw	↓ 15
Support	Morgana	↑ 4	Nautilus	↓ 6

Figure 2 : Snapshot of ChampionGG home page. Note the upper right hand corner shows that the time of picture being taken, 1,956,470 matches had been analyzed by the site. Picture from: https://champion.gg/

Their API was publicly available, and I signed up for an access key and began playing around with making HTTP calls to their server. Now that I had a reliable database to analyze, I could start choosing metrics to analyze.

Part 2 : Choosing Metrics

As I explored the ChampionGG API documentation, I began to familiarize myself with the type of data I could acquire. I found several data endpoints that I could definitely use. One of them is displayed below (Figure 3).



GET

```
/stats/champs/:name?api_key=<API_KEY>
```

Example usage:

```
curl -i http://api.champion.gg/stats/champs/Ezreal?api_key=YOUR_API_TOKEN
```

Parameter

Field	Type	Description
api_key	String	Unique developer API key
name	String	Champion name

Success 200

Field	Type	Description
key	String	Champion key
role	String	Champion role
general	Object	Stats container
winPercent	Number	Win percentage
playPercent	Number	Play percentage
banRate	Number	Ban rate
experience	Number	Average experience
kills	Number	Average kills
deaths	Number	Average deaths
assists	Number	Average assists

Figure 3 : The endpoint /stats/champs/:name returns a list of basic metrics for a given champion, including win rate, ban rate, KDA, and much more.

There are currently 129 champions in the game. However, champions can be played in different “roles.” For example, the champion “Graves,” a big shotgun-wielding cowboy, is commonly played either as a Marksman or a Jungler. The statistics for Marksman Graves will be different from Jungler Graves, so we want to separate their metrics by treating Marksman Graves and Jungler Graves as unique entries. Luckily, ChampionGG’s API accounts for this by creating separate entries for champions of different roles. By treating champions of different roles as unique entries, we increase the number of champions analyzed from 129 to 193. Now we just have to choose metrics based on the data endpoints available to us from the API.

I identified **16** available champion metrics that I wanted to analyze and eventually reduce. They are briefly described below:

Play Rate : The number of times a champion was picked in a match divided by the total number of matches. This is a describer of how popular a champion is. Perhaps strong champions are picked more by players?

Win Rate : The number of times a champion belonged to the winning team divided by the total number of matches. This is generally the most common descriptor for champion strength.

Ban Rate : How often a champion is banned. Before a Ranked match (all game data analyzed by the API are ranked matches), players from both sides ban a total of six champions, preventing either side from being able to play them for that match. Teams will often ban champions that they do not want to deal with, or consider to be too strong.

Placement : This is a ranking system for champions implemented by ChampionGG, based on their own metric analysis. It is different from our intended method as their placement rankings are determined by weighting individual champion statistics, whereas my intention is to reduce all the champion statistics via k-D projection.

Gold Gain : Champions gain gold by killing other champions, minions, or monsters. Hence, the more gold a champion has, the more kills it accrued during a game. Therefore, champion strength is definitely tied with how much gold it gained during the course of a match.

Kills / Deaths / Assists : Champions that are strong are more likely to have a higher K/D/A than weak champions. For obvious reasons, we are including all three metrics in our analysis.

Jungle Kills (Team and Enemy) : Champions in the “Jungler” role gain gold for killing neutral monsters on the map’s jungle. The more they kill, the more gold they earn. This is tied to the gold gain metric above.

Minions Killed : Minions are tiny AI units belonging to both teams that continually traverse the map. Killing minions from the enemy team rewards the player with gold, once again relating minions killed to gold gain.

Largest Spree : Consecutive kills earned by a single champion become killing sprees. The higher a champion’s killing spree, the more kills they most likely got, and hence more gold.

Healing Done : Certain champions have the ability to heal themselves or allies, thus delaying death. The more healing a champion does, the higher the likelihood that they are contributing to a team’s overall success.

Damage Taken / Dealt : Champions attempt to defeat other champions via abilities and spells. The more damage a champion dealt, the more usefulness it contributed towards defeating other champions. Likewise, the more damage a champion received, the more it may have served to absorb enemy attacks, thus being effective “tanks.

Experience : Champions gain experience by defeating other units. This allows them to “level up,” gaining more abilities and stats.

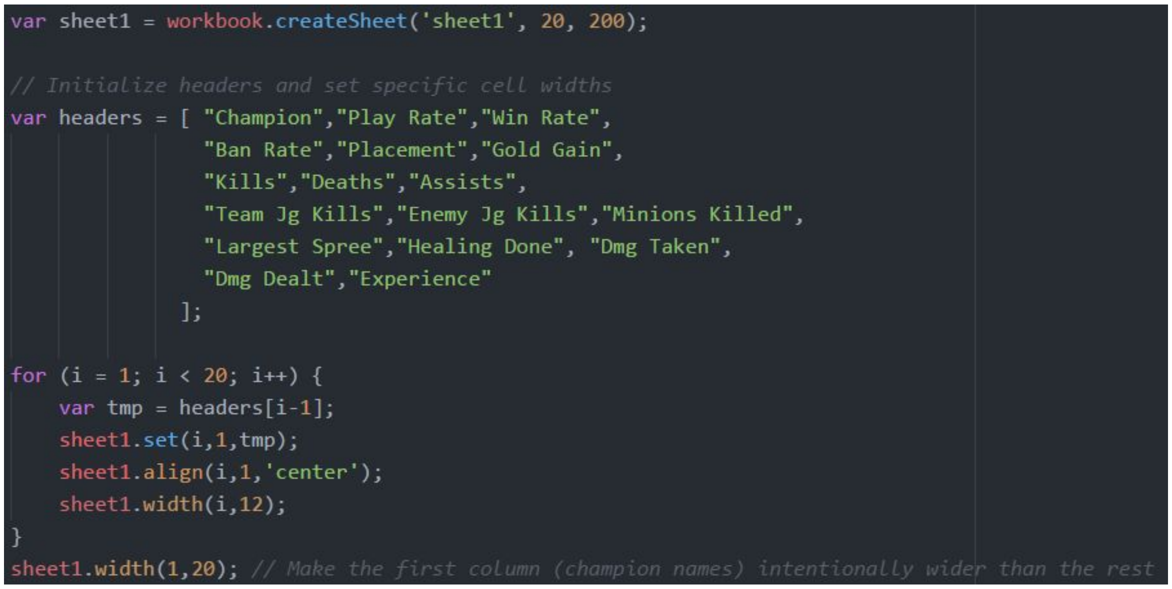
Now that I defined the metrics I would use to describe each champion, I could focus on gathering the data and creating my spreadsheet.

Part 3 : Gathering Data Via Rest Calls

I decided to create my spreadsheet using javascript via Node JS. My full implementation, which will be explained below, can be found here: https://git.io/vau0b

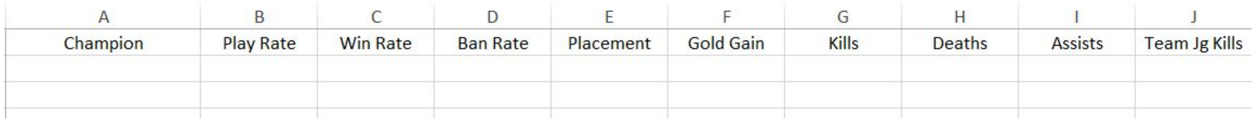
My goal was to use Node JS to parse the API data, store it as a matrix, and return the matrix as an excel spreadsheet. To make HTTP request calls, I used the javascript “request” library. To write data to a spreadsheet, I used the “msexcel-builder” library.

First, I wanted to create my spreadsheet. Using the metrics I defined above, I created the first row of the spreadsheet using the following code:



```
var sheet1 = workbook.createSheet('sheet1', 20, 200);

// Initialize headers and set specific cell widths
var headers = [ "Champion","Play Rate","Win Rate",
                "Ban Rate","Placement","Gold Gain",
                "Kills","Deaths","Assists",
                "Team Jg Kills","Enemy Jg Kills","Minions Killed",
                "Largest Spree","Healing Done", "Dmg Taken",
                "Dmg Dealt","Experience"
              ];

for (i = 1; i < 20; i++) {
    var tmp = headers[i-1];
    sheet1.set(i,1,tmp);
    sheet1.align(i,1,'center');
    sheet1.width(i,12);
}
sheet1.width(1,20); // Make the first column (champion names) intentionally wider than the rest
```

This established the base of my spreadsheet that I could then populate.



A	B	C	D	E	F	G	H	I	J
Champion	Play Rate	Win Rate	Ban Rate	Placement	Gold Gain	Kills	Deaths	Assists	Team Jg Kills

ChampionGG’s API is accessed via HTTP GET calls. For example, calling the URL:

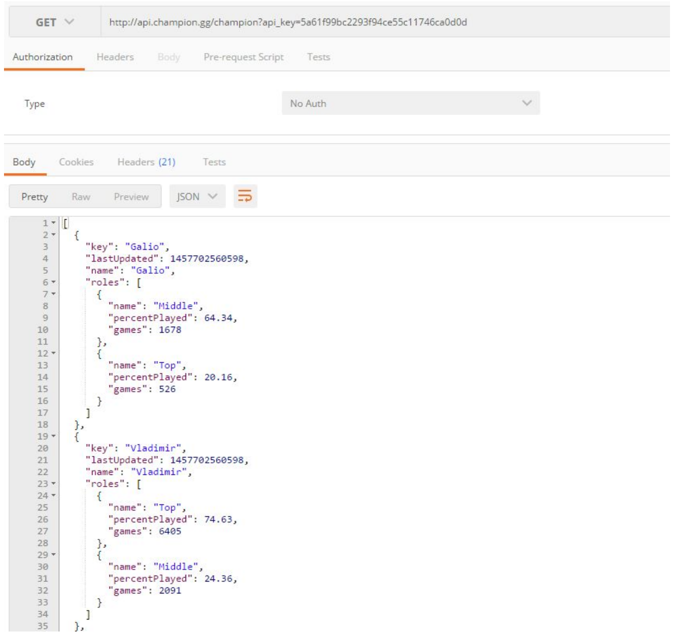
```
http://api.champion.gg/stats/champs/ezreal?api_key=5a61f99bc2293f94ce55c11746ca0d0d
```

Returns the following JSON formatted string:

```
{
  "key": "Ezreal",
  "role": "ADC",
  "title": "Ezreal",
  "general": {
    "overallPositionChange": 1,
    "overallPosition": 4,
    "goldEarned": 12844,
    "neutralMinionsKilledEnemyJungle": 2.38,
    "neutralMinionsKilledTeamJungle": 6.87,
    "minionsKilled": 192.7,
    "largestKillingSpree": 3.56,
    "totalHeal": 2978,
    "totalDamageTaken": 17851,
    "totalDamageDealtToChampions": 25775,
    "assists": 8.18,
    "deaths": 4.98,
    "kills": 6.96,
    "experience": 34.43,
    "banRate": 2.5,
    "playPercent": 28.75,
    "winPercent": 50.63
  }
}
```

Note that these metrics correspond to the ones I chose to analyze above. All that's left was to make a REST GET call to each of the 129 champions and store all of the data as individual spreadsheet cells.

First, I parsed the endpoint "/champion," which returns the names of all champions. A portion of the JSON return is shown below. Note that the return is an array of objects, where each object contains a "key" variable (the champion name). There will be a key for each champion name, as shown in Figure 4.



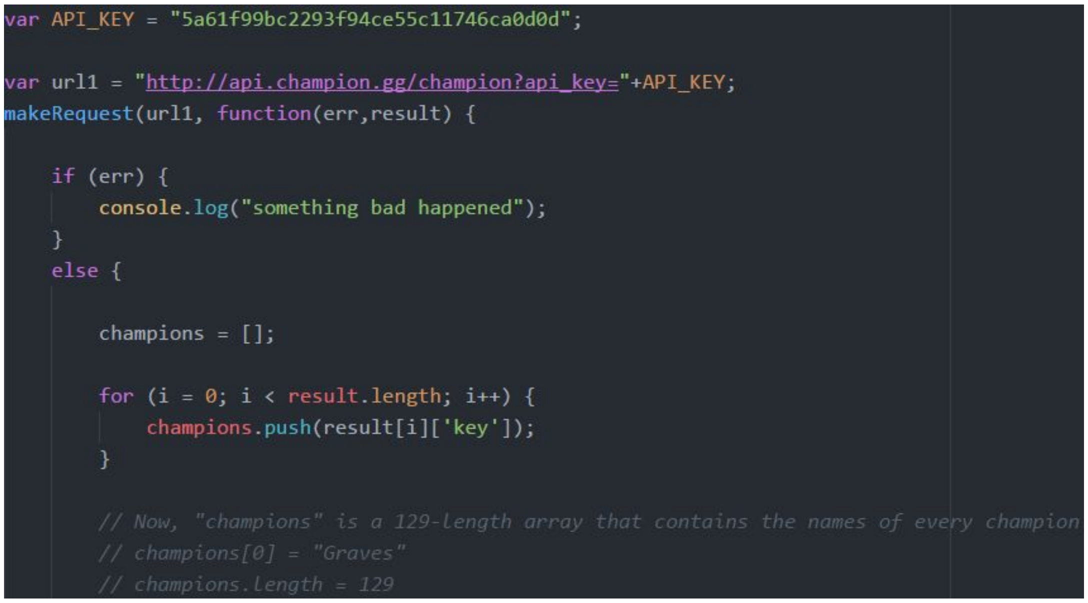
```
http://api.champion.gg/champion?api_key=5a61f99bc2293f94ce55c11746ca0d0d
```

Type: No Auth

```
[
  {
    "key": "Galio",
    "lastUpdated": 1457702560598,
    "name": "Galio",
    "roles": [
      {
        "name": "Middle",
        "percentPlayed": 64.34,
        "games": 1678
      },
      {
        "name": "Top",
        "percentPlayed": 20.16,
        "games": 526
      }
    ]
  },
  {
    "key": "Vladimir",
    "lastUpdated": 1457702560598,
    "name": "Vladimir",
    "roles": [
      {
        "name": "Top",
        "percentPlayed": 74.63,
        "games": 6405
      },
      {
        "name": "Middle",
        "percentPlayed": 24.36,
        "games": 2091
      }
    ]
  },
```

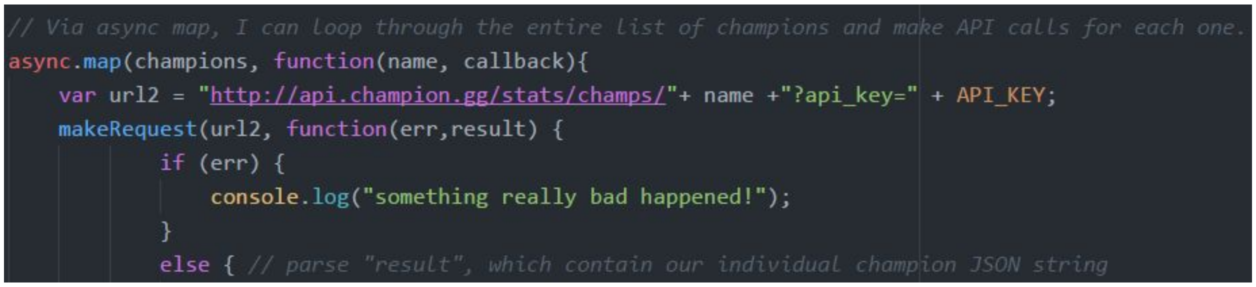
Figure 4: Portion of JSON response. Note that the two champions visible are "Galio" and "Vladimir."

Using the javascript “request” library, which let me make HTTP GET requests, I parsed the /champion endpoint and stored the champion names in an array:



```
var API_KEY = "5a61f99bc2293f94ce55c11746ca0d0d";

var url1 = "http://api.champion.gg/champion?api_key="+API_KEY;
makeRequest(url1, function(err,result) {

    if (err) {
        console.log("something bad happened");
    }
    else {

        champions = [];

        for (i = 0; i < result.length; i++) {
            champions.push(result[i]['key']);
        }

        // Now, "champions" is a 129-length array that contains the names of every champion
        // champions[0] = "Graves"
        // champions.length = 129
```

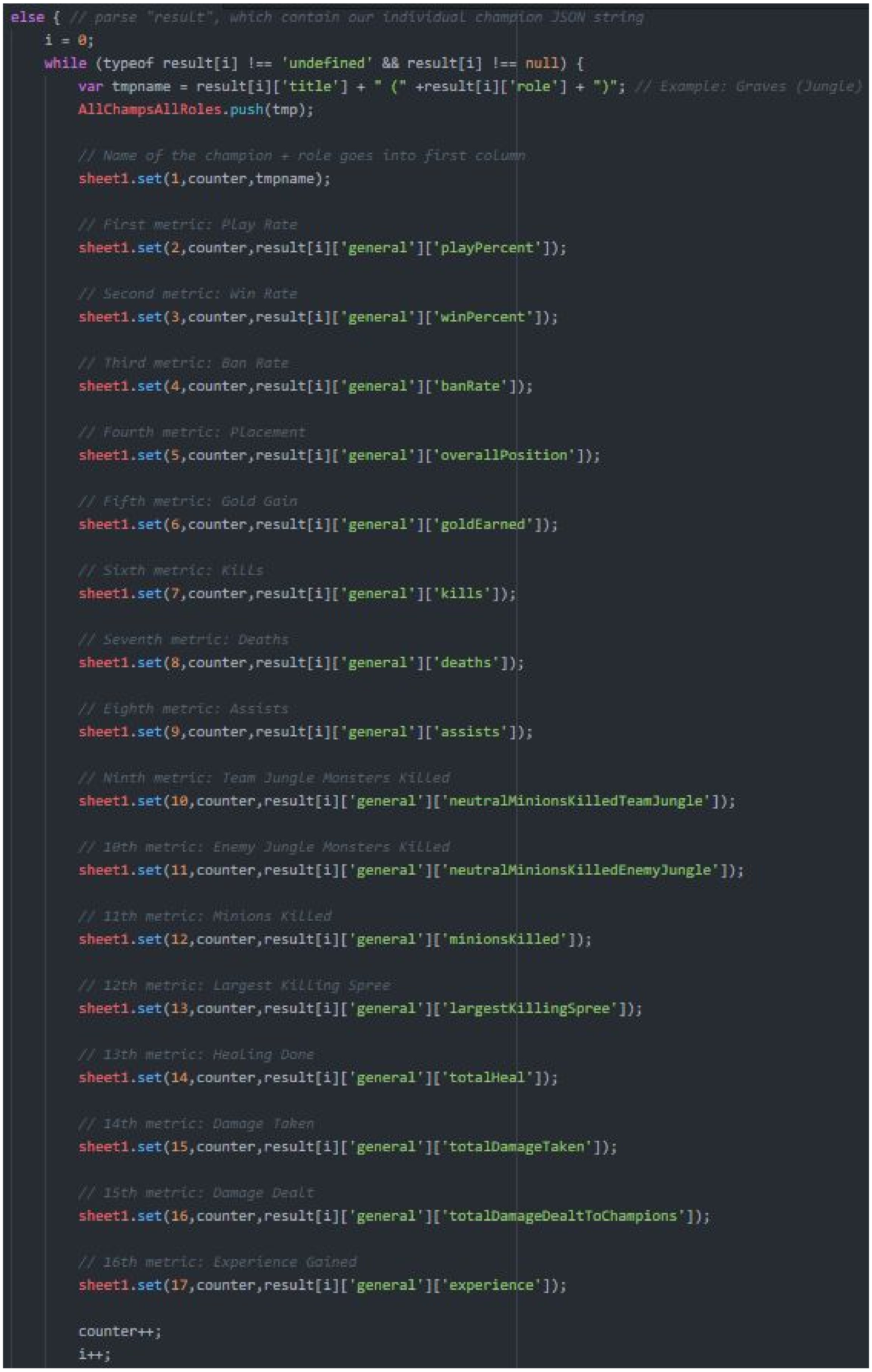
Next, I using the javascript “async” library to create a mapping where each champion name is used to create a new HTTP request. This is explained better in Figure 5:



```
// Via async map, I can loop through the entire list of champions and make API calls for each one.
async.map(champions, function(name, callback){
    var url2 = "http://api.champion.gg/stats/champs/"+ name +"?api_key=" + API_KEY;
    makeRequest(url2, function(err,result) {
            if (err) {
                console.log("something really bad happened!");
            }
            else { // parse "result", which contain our individual champion JSON string
```

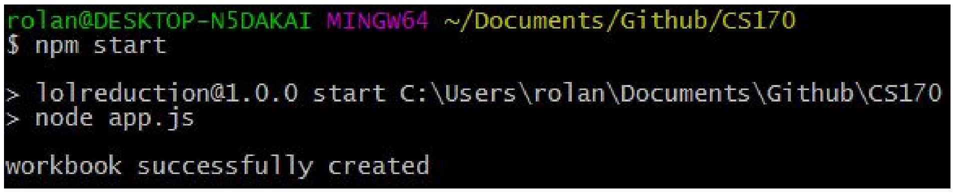
Figure 5: Using “node async” to loop through the “champions” array. In this case, the variable “name” is the iterator used to loop through “champions,” which I defined earlier as the array of 129 champion names.

For each request, I found out how many roles the champion belonged to (e.g. Graves Jungle, Graves Top) and created new data entries for each one. Then, I parsed the result string for each champion to extract the metrics I wanted, and stored these metrics in my excel spreadsheet body. After every champion JSON result had been parsed, I saved the spreadsheet entries and finished script execution. All this code is displayed on the next page.

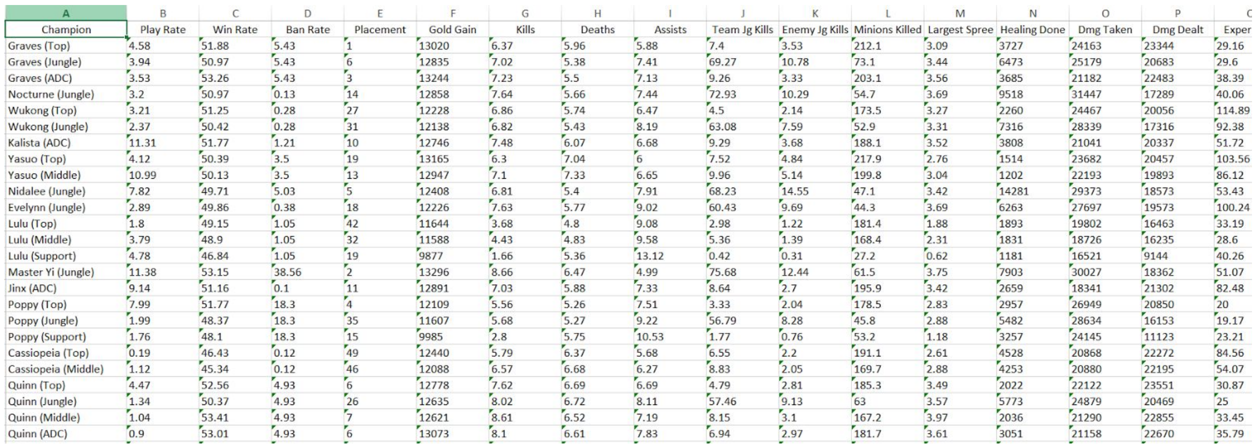


```
else { // parse "result", which contain our individual champion JSON string
    i = 0;
    while (typeof result[i] !== 'undefined' && result[i] !== null) {
        var tmpname = result[i]['title'] + " (" +result[i]['role'] + ")"; // Example: Graves (Jungle)
        AllChampsAllRoles.push(tmp);

        // Name of the champion + role goes into first column
        sheet1.set(1,counter,tmpname);

        // First metric: Play Rate
        sheet1.set(2,counter,result[i]['general']['playPercent']);

        // Second metric: Win Rate
        sheet1.set(3,counter,result[i]['general']['winPercent']);

        // Third metric: Ban Rate
        sheet1.set(4,counter,result[i]['general']['banRate']);

        // Fourth metric: Placement
        sheet1.set(5,counter,result[i]['general']['overallPosition']);

        // Fifth metric: Gold Gain
        sheet1.set(6,counter,result[i]['general']['goldEarned']);

        // Sixth metric: Kills
        sheet1.set(7,counter,result[i]['general']['kills']);

        // Seventh metric: Deaths
        sheet1.set(8,counter,result[i]['general']['deaths']);

        // Eighth metric: Assists
        sheet1.set(9,counter,result[i]['general']['assists']);

        // Ninth metric: Team Jungle Monsters Killed
        sheet1.set(10,counter,result[i]['general']['neutralMinionsKilledTeamJungle']);

        // 10th metric: Enemy Jungle Monsters Killed
        sheet1.set(11,counter,result[i]['general']['neutralMinionsKilledEnemyJungle']);

        // 11th metric: Minions Killed
        sheet1.set(12,counter,result[i]['general']['minionsKilled']);

        // 12th metric: Largest Killing Spree
        sheet1.set(13,counter,result[i]['general']['largestKillingSpree']);

        // 13th metric: Healing Done
        sheet1.set(14,counter,result[i]['general']['totalHeal']);

        // 14th metric: Damage Taken
        sheet1.set(15,counter,result[i]['general']['totalDamageTaken']);

        // 15th metric: Damage Dealt
        sheet1.set(16,counter,result[i]['general']['totalDamageDealtToChampions']);

        // 16th metric: Experience Gained
        sheet1.set(17,counter,result[i]['general']['experience']);

        counter++;
        i++;
```

Finally, I executed my code and opened the generated excel spreadsheet.



```
rolan@DESKTOP-N5DAKAI MINGW64 ~/Documents/Github/CS170
$ npm start

> lolreduction@1.0.0 start C:\Users\rolan\Documents\Github\CS170
> node app.js

workbook successfully created
```



Champion	Play Rate	Win Rate	Ban Rate	Placement	Gold Gain	Kills	Deaths	Assists	Team Jg Kills	Enemy Jg Kills	Minions Killed	Largest Spree	Healing Done	Dmg Taken	Dmg Dealt	Exper
Graves (Top)	4.58	51.88	5.43	1	13020	6.37	5.96	5.88	7.4	3.53	212.1	3.09	3727	24163	23344	29.16
Graves (Jungle)	3.94	50.97	5.43	6	12835	7.02	5.38	7.41	69.27	10.78	73.1	3.44	6473	25179	20683	29.6
Graves (ADC)	3.53	53.26	5.43	3	13244	7.23	5.5	7.13	9.26	3.33	203.1	3.56	3685	21182	22483	38.39
Nocturne (Jungle)	3.2	50.97	0.13	14	12858	7.64	5.66	7.44	72.93	10.29	54.7	3.69	9518	31447	17289	40.06
Wukong (Top)	3.21	51.25	0.28	27	12228	6.86	5.74	6.47	4.5	2.14	173.5	3.27	2260	24467	20056	114.89
Wukong (Jungle)	2.37	50.42	0.28	31	12138	6.82	5.43	8.19	63.08	7.59	52.9	3.31	7316	28339	17316	92.38
Kalista (ADC)	11.31	51.77	1.21	10	12746	7.48	6.07	6.68	9.29	3.68	188.1	3.52	3808	21041	20337	51.72
Yasuo (Top)	4.12	50.39	3.5	19	13165	6.3	7.04	6	7.52	4.84	217.9	2.76	1514	23682	20457	103.56
Yasuo (Middle)	10.99	50.13	3.5	13	12947	7.1	7.33	6.65	9.96	5.14	199.8	3.04	1202	22193	19893	86.12
Nidalee (Jungle)	7.82	49.71	5.03	5	12408	6.81	5.4	7.91	68.23	14.55	47.1	3.42	14281	29373	18573	53.43
Evelynn (Jungle)	2.89	49.86	0.38	18	12226	7.63	5.77	9.02	60.43	9.69	44.3	3.69	6263	27697	19573	100.24
Lulu (Top)	1.8	49.15	1.05	42	11644	3.68	4.8	9.08	2.98	1.22	181.4	1.88	1893	19802	16463	33.19
Lulu (Middle)	3.79	48.9	1.05	32	11588	4.43	4.83	9.58	5.36	1.39	168.4	2.31	1831	18726	16235	28.6
Lulu (Support)	4.78	46.84	1.05	19	9877	1.66	5.36	13.12	0.42	0.31	27.2	0.62	1181	16521	9144	40.26
Master Yi (Jungle)	11.38	53.15	38.56	2	13296	8.66	6.47	4.99	75.68	12.44	61.5	3.75	7903	30027	18362	51.07
Jinx (ADC)	9.14	51.16	0.1	11	12891	7.03	5.88	7.33	8.64	2.7	195.9	3.42	2659	18341	21302	82.48
Poppy (Top)	7.99	51.77	18.3	4	12109	5.56	5.26	7.51	3.33	2.04	178.5	2.83	2957	26949	20850	20
Poppy (Jungle)	1.99	48.37	18.3	35	11607	5.68	5.27	9.22	56.79	8.28	45.8	2.88	5482	28634	16153	19.17
Poppy (Support)	1.76	48.1	18.3	15	9985	2.8	5.75	10.53	1.77	0.76	53.2	1.18	3257	24145	11123	23.21
Cassiopeia (Top)	0.19	46.43	0.12	49	12440	5.79	6.37	5.68	6.55	2.2	191.1	2.61	4528	20868	22272	84.56
Cassiopeia (Middle)	1.12	45.34	0.12	46	12088	6.57	6.68	6.27	8.83	2.05	169.7	2.88	4253	20880	22195	54.07
Quinn (Top)	4.47	52.56	4.93	6	12778	7.62	6.69	6.69	4.79	2.81	185.3	3.49	2022	22122	23551	30.87
Quinn (Jungle)	1.34	50.37	4.93	26	12635	8.02	6.72	8.11	57.46	9.13	63	3.57	5773	24879	20469	25
Quinn (Middle)	1.04	53.41	4.93	7	12621	8.61	6.52	7.19	8.15	3.1	167.2	3.97	2036	21290	22855	33.45
Quinn (ADC)	0.9	53.01	4.93	6	13073	8.1	6.61	7.83	6.94	2.97	181.7	3.61	3051	21158	22670	35.79

Figure 6 : data.xlsx, the 193 x 16 table generated by making API calls to ChampionGG

This table contains a total of 193 columns: 129 champions, with some champions fulfilling multiple roles. For example, the champion “Quinn,” seen at the bottom of Figure 6, is played in four different roles (top, jungle, middle, adc), and contains different datasets for each role. Each champion row contains the 16 metrics that I picked earlier. This results in a 193 x 16 matrix that I could analyze using reduction. Now that I have my data, I can start analysis via Matlab.

Data Analysis

Part 0 : Goals

Our goal is to analyze champion strength by comparing the 16 different metrics of each champion. We can describe each champion by variables such as gold gain, average kills per game, and more. However, many of these properties are related - for example, gold gain is linked with kills earned and minions killed - and as a result, comparing individual metrics between champions is rather redundant. So, we want to find a way to collectively summarize our 16 metrics. We can do this in two ways: k-D projection and principal component analysis.

Part 1A : Dimensional Reduction via k-D Projection

Dimensional reduction is the statistical process of transforming data in a high-dimensional space into a space of fewer dimensions. For example, it is impossible to visually represent 16 different metrics on a single graph. We faced this same issue in class when we attempted to compare iris flowers, which were described by multiple stem and petal measurements. To solve this, we use multiply the iris matrix (dimensions n x p) by a random matrix (dimensions p x 2) to effectively reduce our data to a two-dimensional easily plottable subspace.

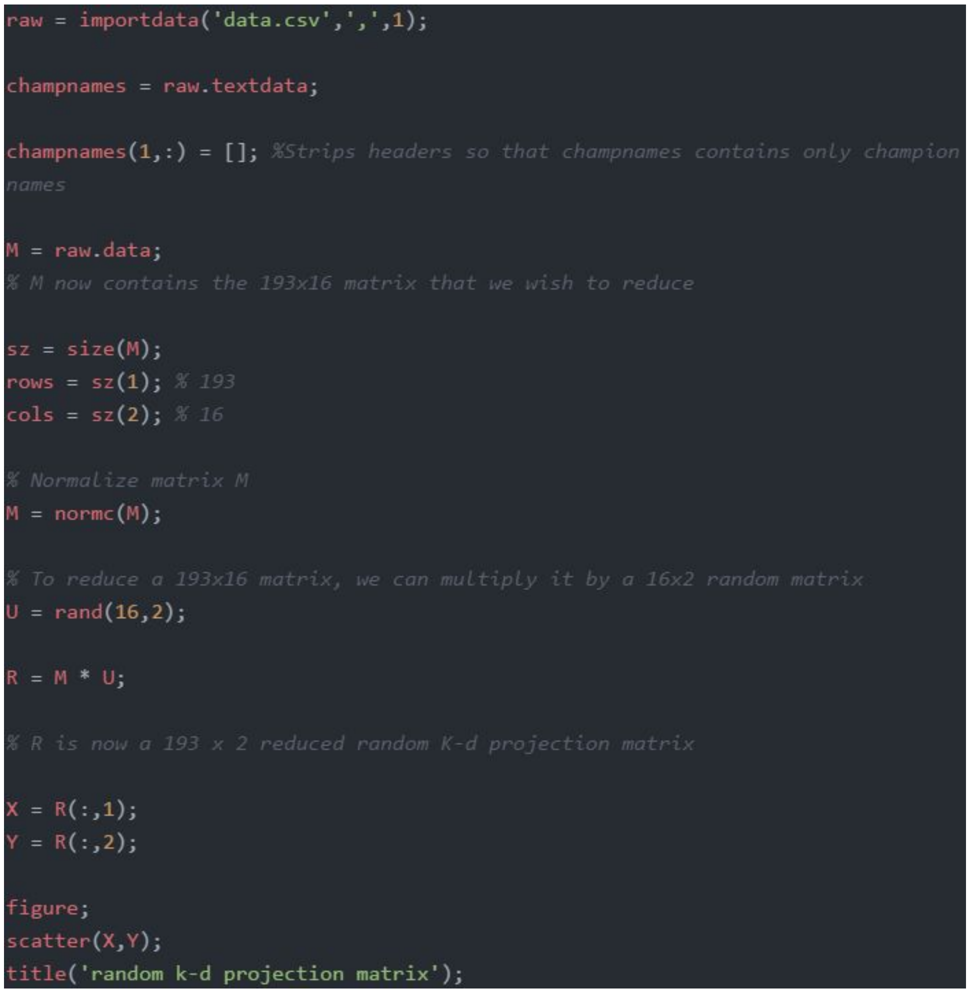
The rationale behind k-D projection is explained in the Johnson-Lindenstrauss lemma, which states that if points in a vector space are of sufficiently high dimension, then they may be projected into a suitable lower-dimensional space in a way which approximately preserves the distances between the points*. Although we lose a lot of data, the k-D projection should still represent an accurate relative comparison of our dataset.

Part 1B : k-D Projection Implementation

Full implementation can be found here: https://git.io/vau0D

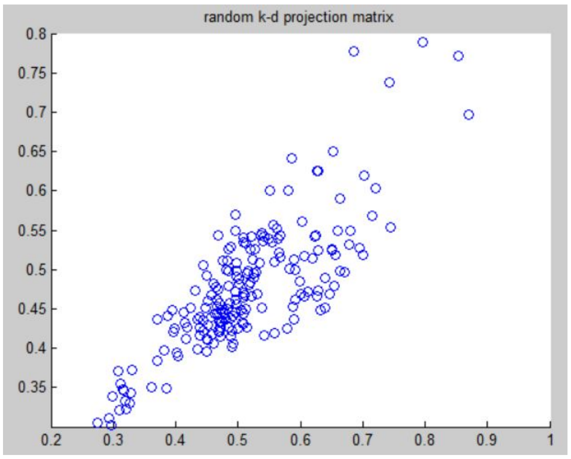
First, I converted the Excel spreadsheet file created during data acquisition into a CSV file. This let me separate the headers from the main body. I parsed the names of all the champions from the headers and placed them into an array called “champnames.” Next, I normalized the data matrix so that no column could singlehandedly determine the result. The data matrix, M, contains dimensions 193 x 16, so to create my kD projection, I would have to multiply M by a (16x2) random matrix. This implementation is shown on the next page.

** https://en.wikipedia.org/wiki/Johnson%E2%80%93Lindenstrauss_lemma*



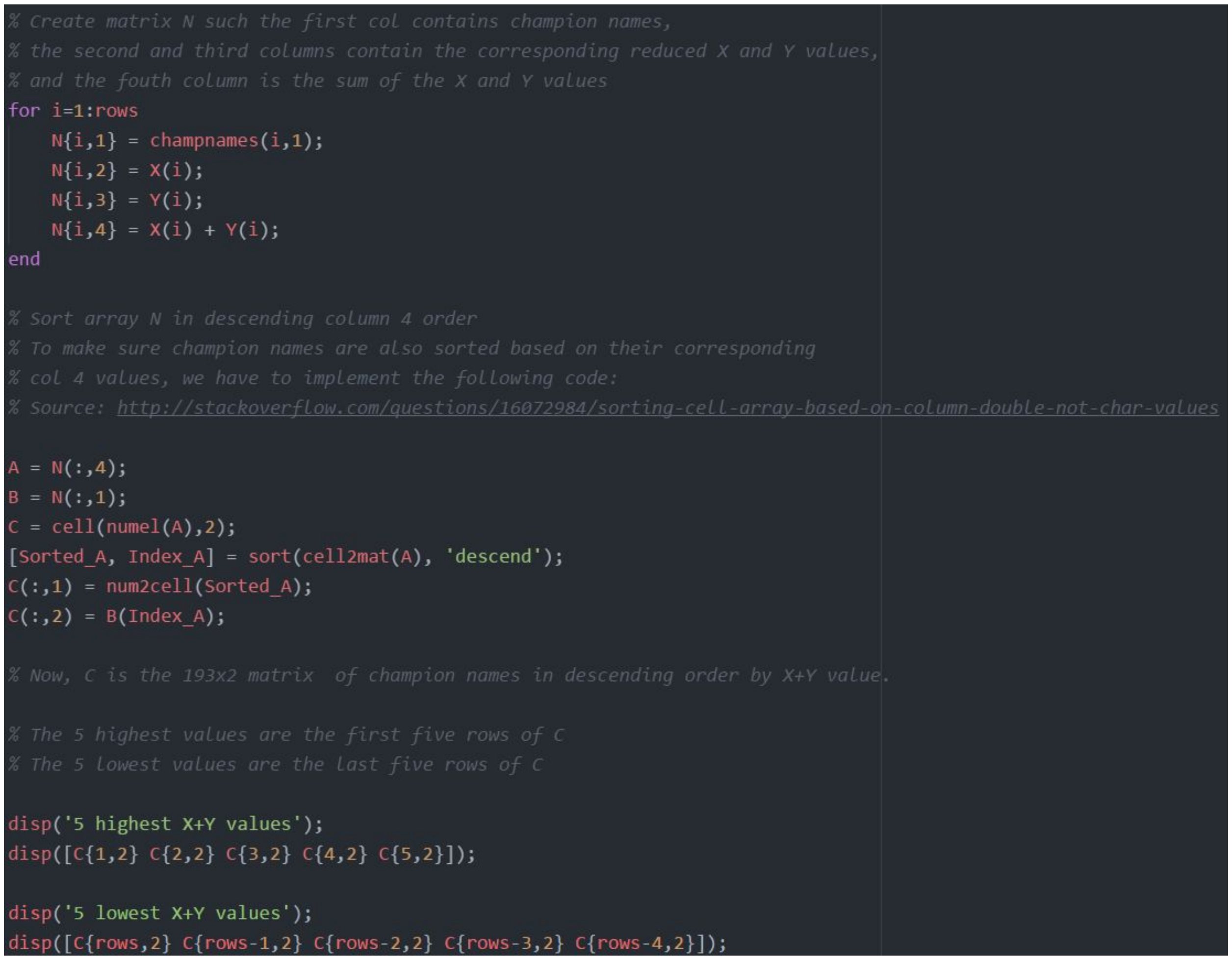
```
raw = importdata('data.csv',',',1);

champnames = raw.textdata;

champnames(1,:) = []; %Strips headers so that champnames contains only champion
names

M = raw.data;
% M now contains the 193x16 matrix that we wish to reduce

sz = size(M);
rows = sz(1); % 193
cols = sz(2); % 16

% Normalize matrix M
M = normc(M);

% To reduce a 193x16 matrix, we can multiply it by a 16x2 random matrix
U = rand(16,2);

R = M * U;

% R is now a 193 x 2 reduced random K-d projection matrix

X = R(:,1);
Y = R(:,2);

figure;
scatter(X,Y);
title('random k-d projection matrix');
```

This code yields a different plot each time the matlab script is executed, since a new random matrix U is being generated each time. The following is one of the observed outputs.



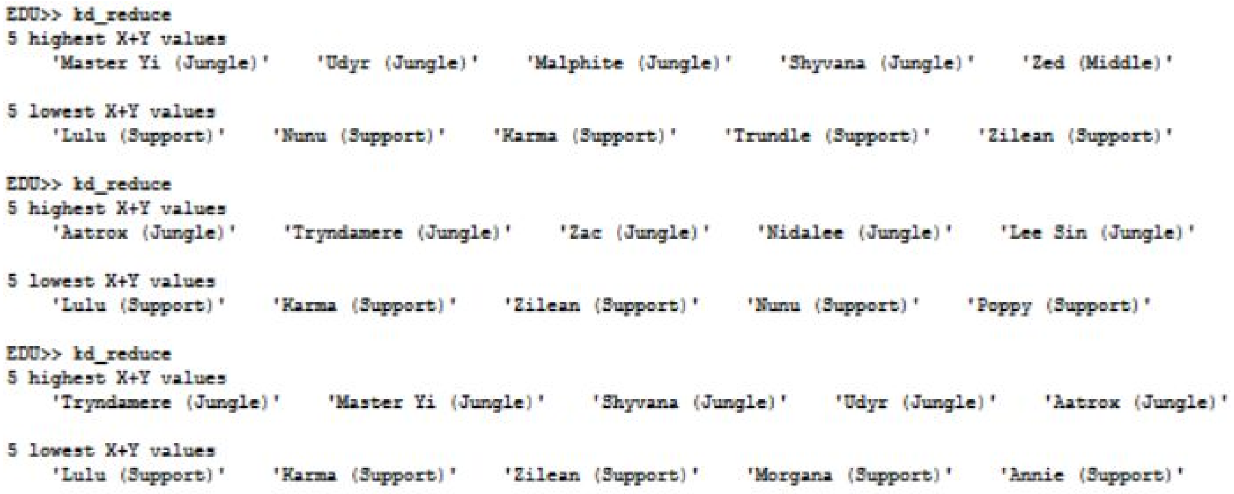
There are clearly seen outliers - These are what can be considered strong champions based on a summary of the metrics I chose.

Next, I attempted to find the champions associated with those outliers. My code finds the top 5 (X+Y) values and the lowest 5 (X+Y) values, and finds the champion names belonging to those values:



```
% Create matrix N such the first col contains champion names,
% the second and third columns contain the corresponding reduced X and Y values,
% and the fouth column is the sum of the X and Y values
for i=1:rows
    N{i,1} = champnames(i,1);
    N{i,2} = X(i);
    N{i,3} = Y(i);
    N{i,4} = X(i) + Y(i);
end

% Sort array N in descending column 4 order
% To make sure champion names are also sorted based on their corresponding
% col 4 values, we have to implement the following code:
% Source: http://stackoverflow.com/questions/16072984/sorting-cell-array-based-on-column-double-not-char-values

A = N(:,4);
B = N(:,1);
C = cell(numel(A),2);
[Sorted_A, Index_A] = sort(cell2mat(A), 'descend');
C(:,1) = num2cell(Sorted_A);
C(:,2) = B(Index_A);

% Now, C is the 193x2 matrix  of champion names in descending order by X+Y value.

% The 5 highest values are the first five rows of C
% The 5 lowest values are the last five rows of C

disp('5 highest X+Y values');
disp([C{1,2} C{2,2} C{3,2} C{4,2} C{5,2}]);

disp('5 lowest X+Y values');
disp([C{rows,2} C{rows-1,2} C{rows-2,2} C{rows-3,2} C{rows-4,2}]);
```

An example of three different executions of this new script yields the following:



```
EDU>> kd_reduce
5 highest X+Y values
    'Master Yi (Jungle)'    'Udyr (Jungle)'    'Malphite (Jungle)'    'Shyvana (Jungle)'    'Zed (Middle)'

5 lowest X+Y values
    'Lulu (Support)'    'Nunu (Support)'    'Karma (Support)'    'Trundle (Support)'    'Zilean (Support)'

EDU>> kd_reduce
5 highest X+Y values
    'Aatrox (Jungle)'    'Tryndamere (Jungle)'    'Zac (Jungle)'    'Nidalee (Jungle)'    'Lee Sin (Jungle)'

5 lowest X+Y values
    'Lulu (Support)'    'Karma (Support)'    'Zilean (Support)'    'Nunu (Support)'    'Poppy (Support)'

EDU>> kd_reduce
5 highest X+Y values
    'Tryndamere (Jungle)'    'Master Yi (Jungle)'    'Shyvana (Jungle)'    'Udyr (Jungle)'    'Aatrox (Jungle)'

5 lowest X+Y values
    'Lulu (Support)'    'Karma (Support)'    'Zilean (Support)'    'Morgana (Support)'    'Annie (Support)'
```

It is seen that champions that are junglers are consistently ranked among the strongest champions. Furthermore, these champions are all “carry” type champions, meaning they deal heavy amounts of damage and generally get many kills per game. If the k-d projection was meant to compress all the champion metrics, then it makes sense that the strongest champions are those generally associated with high gold gain and high damage.

However, looking at the “5 lowest X+Y values” shows the shortcomings of using champion metrics to decide champion strength. All of the champions that are ranked lowest belong to the “support” role. Support champions, which are generally tanks or healers, exist to protect their allies by healing them or absorbing enemy damage. They generally let their team carries take the gold and kills, so it is unsurprising that their metrics are much lower than that of other champions. However, many players are aware that supports play an extremely crucial role on any team, and their strength and usefulness cannot be measured solely by metrics such as gold gain and kills.

This leads to the next question: What metrics affect champion strength the most? k-D random project does not answer this question, as all the metrics are blended together. Instead, we can use another method of dimension reduction: principal component analysis.

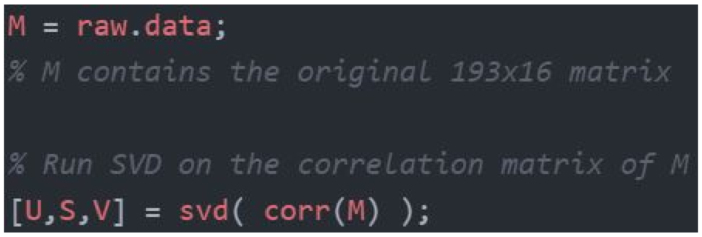
Part 2A: Dimensional Reduction via Principal Component Analysis

Dimensional reduction can also be achieved via principal component analysis (PCA). PCA is a much more expensive, but more accurate method of summarizing data by analyzing the singular value decomposition of a correlation matrix. An additional feature of PCA is the ability to see which metrics were the most important by analyzing the principal eigenvectors. This will be explained further below.

Part 2B: PCA Implementation

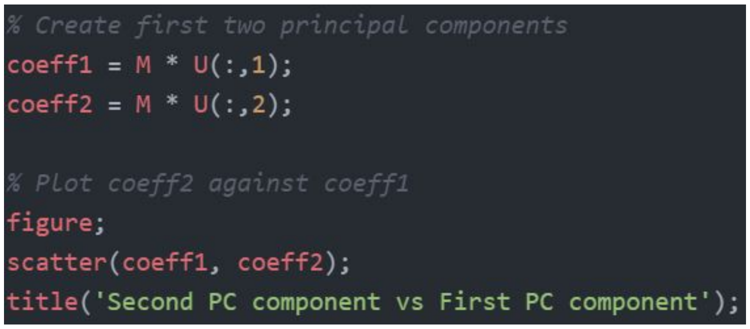
Full implementation can be found here: https://git.io/vau09

Similar to in my k-D implementation, I created M, the 193 x 16 matrix of champions and their metrics. Then, I ran a singular value decomposition (SVD) on the correlation matrix of M, as shown below:



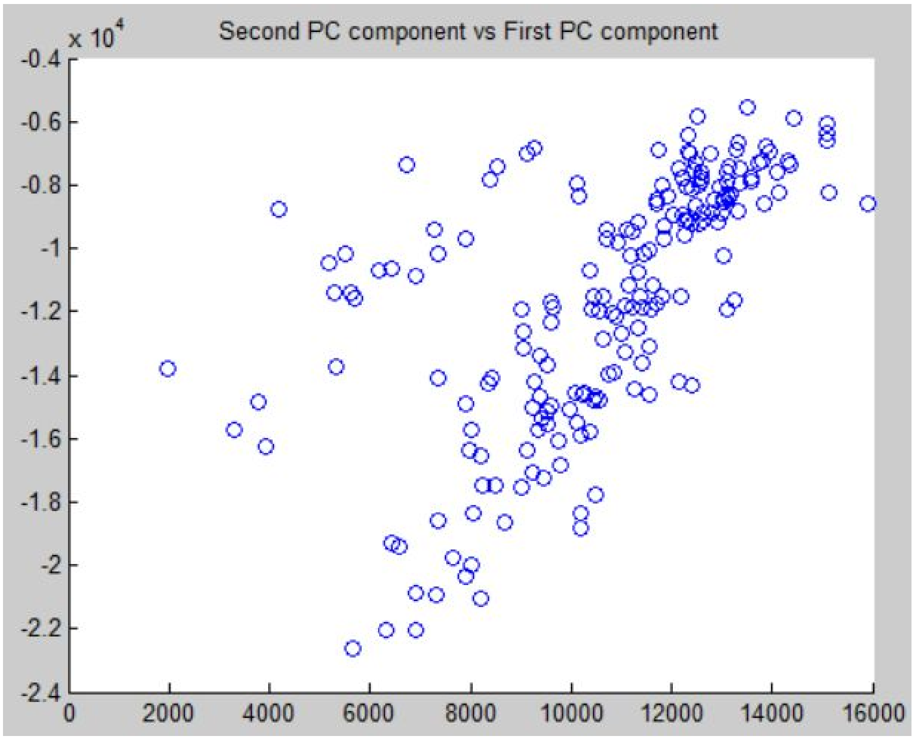
```
M = raw.data;
% M contains the original 193x16 matrix

% Run SVD on the correlation matrix of M
[U,S,V] = svd( corr(M) );
```

In PCA, our dimensional reduction comes from finding the coefficient matrices associated with the first and second columns of the decomposed “U” matrix.



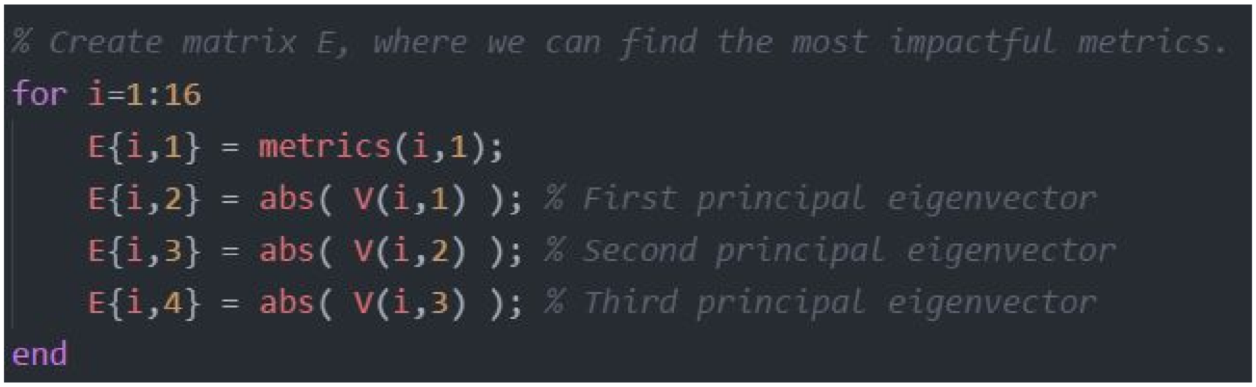
```
% Create first two principal components
coeff1 = M * U(:,1);
coeff2 = M * U(:,2);

% Plot coeff2 against coeff1
figure;
scatter(coeff1, coeff2);
title('Second PC component vs First PC component');
```

Plotting the two coefficients against each other gives the following result:



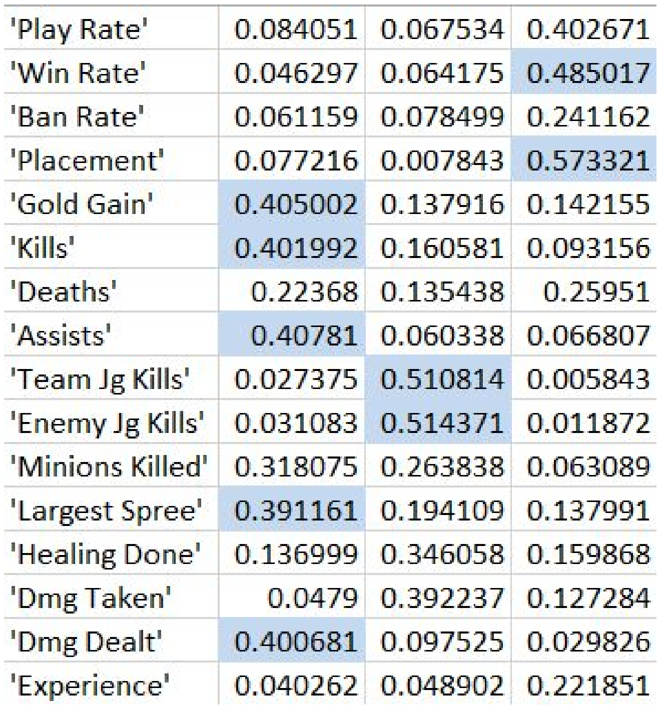
The principal components plot shows that there appear to be two distinct groups of champions. My theory is that the group above the (x=y) diagonal line must be the support champions that I previously identified as weak due to their much lower gold gain and kills.

Next, I analyze the decomposed “V” matrix, where the first three columns can be used to determine which metrics are the most important.



```
% Create matrix E, where we can find the most impactful metrics.
for i=1:16
    E{i,1} = metrics(i,1);
    E{i,2} = abs( V(i,1) ); % First principal eigenvector
    E{i,3} = abs( V(i,2) ); % Second principal eigenvector
    E{i,4} = abs( V(i,3) ); % Third principal eigenvector
end
```

Running the script creates the matrix “E,” which is copied onto a table. The most impactful values have been manually highlighted below.



'Play Rate'	0.084051	0.067534	0.402671
'Win Rate'	0.046297	0.064175	0.485017
'Ban Rate'	0.061159	0.078499	0.241162
'Placement'	0.077216	0.007843	0.573321
'Gold Gain'	0.405002	0.137916	0.142155
'Kills'	0.401992	0.160581	0.093156
'Deaths'	0.22368	0.135438	0.25951
'Assists'	0.40781	0.060338	0.066807
'Team Jg Kills'	0.027375	0.510814	0.005843
'Enemy Jg Kills'	0.031083	0.514371	0.011872
'Minions Killed'	0.318075	0.263838	0.063089
'Largest Spree'	0.391161	0.194109	0.137991
'Healing Done'	0.136999	0.346058	0.159868
'Dmg Taken'	0.0479	0.392237	0.127284
'Dmg Dealt'	0.400681	0.097525	0.029826
'Experience'	0.040262	0.048902	0.221851

This table represents the three eigenvectors of the first three principal components. For each eigenvector, I found the largest eigenvalues and highlighted them. Large eigenvalues correspond to high variance. High variance means that the metric played a large role in deciding champion strength. For example, in the first principal components, the main deciders are gold gain, kills, assists, largest killing spree, and damage dealt. This is very surprising, as I thought champion win rate would be the primary decider of champion strength, but this is not shown until the third principal component.

Conclusion

Distribution of Effort

I spent an equal amount of time working on data acquisition and data analysis. In class, we receive datasets and attempt to analyze them with a hypothesis in mind. However, working on this project, I not only had to create my own data, I also had to attempt to make sense of it, with no clear expectations or end results in mind.

Constructing the dataset was no easy task. Although I had access to ChampionGG's API, I had to find a way to execute store the results of 129 separate API calls. This was made difficult by Node JS's asynchronous nature and the difficulty of working with callbacks. I solved this by using the "async" library, which let me map a different API call to each champion, write each data entry to the same spreadsheet, and then save the spreadsheet once all execution had finished.

Once I had my data, analysis went much smoother. k-D projection and PCA were both techniques we had learned in class, and it was only a matter of applying code learned in lecture, to my own dataset. I ran into trouble implementing PCA when my singular value decomposition of the covariance matrix looked extremely shapeless, but this was solved when I analyzed the correlation matrix instead.

Wish List

I was satisfied with the results obtained. Both dimension reduction methods of k-D projection and PCA yielded ways to determine champion strength given a set of metrics. However, there are definitely ways to improve my results and accuracy.

First off, I should have created multiple datasets with matches separated by patch number. Patches, which buff or nerf champions deemed too weak or too strong, play an important role in determining champion strength.

To make the above happen, I would have to pull my own data from Riot's API instead of using ChampionGG's API. This allows me greater flexibility in choosing which kind of data I want to pull. However, this would require some sort of local database to store collected matches, which could go up to the millions if I wish to have an accurate dataset to choose from.

Finally, in my principal component analysis, I noticed two distinct groups of champions distributed across plot of the first two principal components. I guessed that one of the groups consisted of support champions. I should have added an additional column to my dataset, that ordered champions by role, so that I could specifically mark support champions on my scatterplot. This was not possible in my current dataset because I combined the role of the champion with the champion name - such as “Udyr (Jungle).” Due to time constraints, I did not go back and modify my dataset, and would definitely analyze this area first in the future.

Part 1 Conclusion : k-D Random Projection

By multiplying my normalized data matrix by a random matrix to create a reduced plottable projection, I was able to create a holistic comparison of each champion based on 16 unique metrics. Because my reduction essentially compresses these 16 dimensions into two, the champions I found to be the strongest were “carry” type junglers, such as Master Yi and Udyr, whose in-game abilities let them dish out huge amounts of damage and acquire many kills. The weakest champions were support-type champions such as Lulu and Karma, whose abilities are more for the sake of keeping their teammates alive, rather than trying to kill the enemy. However, this doesn’t mean they are weak champions, as my k-D projection seems to suggest. To determine which of my metrics were truly responsible for deciding champion strength, I analyzed my data using PCA.

Part 2 Conclusion : Principal Component Analysis

I analyzed the first three principal components of my data. In the first principal component, it appeared that the biggest deciders of champion strength were gold gain, kills, assists, kill sprees, and damage dealt. This could be why supports (who lack most of those above traits), scored weakly in the k-D projection. The second principal component showed that jungler monsters killed played a big role in deciding champion strength. The explanation could be why so many junglers were rated highly. Finally, the third principal component showed that win rate was an important deciding factor of champion strength. This makes intuitive sense: champions with high win-rates are generally considered strong, and low win-rates imply weak champions.

Sources and Citations

The entire repository of code used for both data acquisition and data analysis can be found in the following repository: https://github.com/rzeng95/LoLReduction

Node JS packages used for data acquisition:

MS-Excel-Builder : Used to write data to excel spreadsheet
Source: https://github.com/chuanyi/msexcel-builder

Request : Used to make HTTP calls to ChampionGG API
Source: https://github.com/request/request

Async : Used mapping function to iterate over all champion HTTP calls
Source: https://github.com/caolan/async

API used for data acquisition:

ChampionGG: http://api.champion.gg