Capstone 1 Milestone Report

Problem Statement:

League of Legends is one of the most popular online games right now. It has risen to the top as an icon in the e-sports industry. League of Legends is an online multi-player game where 10 players fight in 5v5 matches. What I want to solve is which tweakable gameplay elements have the most effect on match outcomes. Can these tweakable elements improve game play interactions? League of Legends if a free to play game, and profit comes from cosmetic purchases for game characters. These cosmetic purchase do not give players any advantages like battle stats nor do they alter game mechanics in any way. Improvements in game play come from character design, item choices, and map interaction design. My hope for this project is to provide both game developers and players more insight into some of the more over statistics and their effect on gameplay. This project also hopes to discover which variables are most influential in match outcomes. For instance, rather than individual character picks, how does composition based on roles (tanks, fighters, etc.) after match times. It may also be useful to predict win rates based on a specific combination of champions versus others.

Data Set:

Step 1 Acquire Data Source

Capstone 1 project relies on data taking from competitive ranked matches from the game League of Legends, developed by Riot Games. This was done direct through Riot Games API which can be found at https://developer.riotgames.com/ and creating a free account.

The first thing I had to do was have an account with Riot Games, which I already had from playing the game. The next step was to acquire an API key and study the API reference documentation to get exactly what I was looking for. I also noted the rate limit of 20 requests every 1 second and 100 requests every 2 minutes. As, a side note I applied for a production API which would increase my rate limit in the thousands. As of writing this document, I have not obtained a production key. The Riot Games API contains many API methods which return data in JSON. The methods I used were SUMMONER-V4 and MATCH-V4 which I will describe below.

My goal was to obtain at least 100,000 matches worth of data and store the data as CSV files to be easily turned into Pandas Data Frames. I eventually obtained 146,704 matches. Each row of the Data Frame would be a single match and columns would each represent statistics and key information of the game such as stats of each individual player and game id. To get matches of individual games would require getting an encrypted account id based on in game player names and the individual match id every game.

The initial process to get at this data goes as follows. First, I needed to use the API method SUMMONER-V4 to get an encrypted account. I wrote a function called get_acct_id() which would take 3 arguments, the players in game name aka summoner

name, region where the game is player, in this case ‘na1’, and the API key itself. I used the Python request library to return the JSON in every case. In every method I would check the response code. Code 200 meant a successful request. If the response code returned was 429 that meant I was over the limit and would have Python sleep for 2 minutes before attempting any more calls to the API. A successful call to the API return the encryted player ID

Next, I needed the match ids for 146,704 games. I searched well known professional players and popular streamers all who played the game at high and competitive level. All the players are considered to know the game inside and out and would lead to the most promising match data as many people tend of lower ranks tend to imitate to players in terms of game play. The API MATCH-V4 method was used to return a list of all ranked games played in the 2018 season for each summoner name using the function get_match_history().

Next, I used the MATCH-V4 method again in a function named get_match_stats() to return the data of a match in JSON.

The main function I created to get 140,000+ match ids is called get_many_matches(). This function could take a list of in-game player names and either return a list of match ids as a new pickled file or add on to an existing pickled file. Due to the API rate limit it would currently take over 33 hour to write all the data into CSV files. I decided to split the ids into 10 files of approximately 10000 match ids in each file. This was partly due to my fear of encountering bugs and being able to monitor printed debugging outputs. I am glad I did because I caught many bugs and silent fails which led to be refining my code multiple times over and add new exception handling.

The final step was to create a function called create_master_data() that would write the both clean/wrangle the data and write the data into a CSV file. In this case multiple files because the easiest was for me to write to a csv file was to first format the data into a Data Frame. The other way would have been to write many lines of csv writing and string formatting code. Because I was converting the

Step 2 Data Wrangling and Cleaning

The way I cleaned and wrangled the data was first converting the JSON to a data frame. I saw the size in memory the data was going to use and decided to make 10 separate CSV files. As mentioned above create_master_data() would clean and wrangle the data. Many helper functions were created to achieve this. But first, I looked at the out put of the JSON. What I noticed was the champions, or the characters that could chosen to play were returned as id numbers, as were the in game items, and spells. My first step was to find a mapping of the id numbers. I found this data in JSON format by asking around on Riot Games API Discord channel found here. I was directed to other APIs which I could extract the mapping of ids to items, champs, and

spells. I created a dictionary in Python for all those ids and wrote functions which would replace those id numbers with the in games. All the cleaning functions are add_player_name(), add_champ_pick(), add_sumpell_pick(), add_stats(), add_team(), and add_team_stats(). Another function create_row() would use all those cleaning functions and create a dictionary which would later be used to create rows for the function create_master_data().

The API never returned missing values which was great. However, the game is updated on almost a weekly basis meaning items are removed and changed. I ran into the case that id numbers for items and champions were missing due to different match ids occurring in different patches or updates to the game. In the case that the ids could not be match, I allowed the id numbers to stay in place rather than remove them or create null values. I originally ran into KeyError exceptions during intital runs and did more defensive programming to my functions.

Each player had close to 100 in game stats each. Further string manipulation was needed as the stats for each player were values in key:value pairs. Therefore, I added a ‘p’ + ‘number’ before any stat name to indicate which stat belonged to each player. For example, p3 would indicate player 3. Other names for certain columns were cleaned in this fashion.

In the context of the game, there are certain outliers for stats like kills. But this is due to the nature of the game. A player can ‘snowball’ if getting first kills very early in the game. This creates a lead which other players can not keep up with. This happens from time to time. It is more common in low rank matches, but does happen in the competitive scene as well. These outliers do not need to removed at all. They are often lights of a players skill in high ranked matches. In a 5v5 game with competent players, come backs can come at any instance and even an extremely high kill to death ratio is sometimes meaningless without looking at other stats like amount of ‘vision’ is constantly being looked at on the map which can be calculated using how many vision wards are placed.

Overall I believe the data set is very reliable as I picked players in the highest ranked matches at the level of Master, Challenger, Grandmaster, and diamond. Below those tiers are platinum and gold rank being the mid level players, and the lowest being silver and bronze ranked. Of course there will be differences in the data if matches were gathered form different tiers, but I was interested in the level of play of professionals and those who stream the game online for hours at a time for a living. These are the matches that are watched and analyzed by the rest of the player base and it is usually the top ranked matches which determine the meta play of the game at any given time.

Exploratory Data Analysis and Additional Data Cleaning:

When initially starting the EDA, I concatenated all the csv files of match data into one data frame using pandas. After, looking over the data more, I found there were columns that existed to hold data from old game modes that could be removed. There were still some columns that help 0 values because they represented that a player did not choose to pick either a perk(special bonuses that affected champion stats) or had no item in an item slot when the match ended.

One thing I wanted to explore was whether the number of a certain role in a match had an effect on game time. I used found a json file online which mapped champion roles to the champion name. I then created a data frame containing champions and champ roles for each match. I then added a column of the match length in mins. What I decided to do next was to remove games less than 15 minutes. 15 minutes was the earliest a team can surrender unless there was a disconnect which gives an even earlier surrender option. 3931 games were removed for a total of 142,773 games left to be used for analysis.

To get a an idea of what match times looked like I made a histogram of the match times. The average match time for the matches was about 27.47 minutes. The longest game time was 72.98 minutes.

Here is the break down of champion roles played in those 142,773 games:

supports: 325173, 12.81%
marksmen: 343258, 13.53%
mage: 504429, 19.88%
tank: 326048, 12.85%
assassin: 419309, 16.52%
fighter: 619549, 24.41%

The marksmen and support roles as picks closely mirrored each other in distribution an d frequency. This makes sense because a marksmen is most often paired together with a support.

I also wanted to see what the composition of roles in game looked like for games lasting over 40 minutes.

supports: 11882, 12.95%
marksmen: 12582, 13.71%
mage: 19385, 21.12%
tank: 11321, 12.33%
assassin: 15025, 16.37%
fighter: 21588, 23.52%

There is not much difference. There appears to be more mages played in longer games but nothing really stands out.

Next I explored the top win rates of champions as well as the worst win rates. The top 25 champs had win rates between 51.68% and 56.54%. The top worst champs in terms of win rates were between 43.95% and 47.99%. This led me to look at factors about why the top win rate champion Talon had such a high win rate.

Talon's primary role is an Assassin, and secondary role is a Fighter. He was played 9313 games out of the 146,704 games from the date, which is a play rate of 6.35%. I created functions that would just return stats for a certain champ played and used this for analyzing Talon. I plotted Talon's damage to map objectives versus damage to other champions. The plots were split into two plots one where Talon got 'First blood' and the other where he did not. Killing champions gives a player more gold to buy items but typically awards this bonus to the player getting the gold. Players helping assist the kill get a smaller reward. Getting map objectives grant gold and other bonuses but help out the team more as the win condition is to enter the enemy's base and destroy the final structure, the nexus. Splitting the plot into getting first blood and not seemed relevant because as an assassin, Talon's skill set allows first single target killing of other players. Getting a first blood awards bonus gold which allows a player to 'snowball' ahead of the rest. There is a clear difference in more overall damage to map objectives and champions when Talon gets first blood. From the data, Talon get first blood 22% of the time. Talon player win more games when focusing down map objectives. This makes sense as League is a team based game. Next I looked at items of Talon players in games won. Duskblade was a clear winner in top item buy. It is an assassin item.

Another statistic I looked at was wards killed versus wards placed. Wards grant vision on the map and killing wards denies enemy vision. Talon has a unique ability which allows him to jump over map terrain, making him mobile and hard to catch. It also gives Talon player the advantage of being in the enemy side of the map with a higher level of safety. Talon has a higher win rate when denying more vision. As an assassin, this gives him more opportunity to catch opponents off guard. But duskblade also has the ability to reveal enemy wards, which may also lead to his high ward kills.

Inferential Statistics EDA

I wanted to use hypothesis tests to see if there was an difference in average game time and the number of roles picked in a match. I wrote functions to help automate the process. I use bootstrapping in one example, but used the frequentist approach for the rest of the hypothesis testing. The null hypothesis in each case was that average game time was the same in overall matches and games with X number of roles in a game. I did this with every role. For example, tested games with 1 to 9 assassins and got p-values for them. I used a significance level of 0.01. What I found was that although there was statistical significance for many of the p-values returned, there was little practical value. The difference in average game times varied by barely more than a minute.

Although there are results that are statistically significant, none of the results have much practical significance. The game times barely vary more than 1 minute based on champion role composition. This may be the result of player knowledge at the very

highest level of the game. They know what champions counter and are good against other champions. For those unfamiliar with the game, there is a draft phase before the start of matches. Each team can banned 5 champions for a total of up to 10 champions banned. players also pick champions in turns, allowing counter picking and formation of strategic team compositions. These findings could also suggest that Riot Games does a very good job at balancing champion power in overall game play. Riot Games also release constant patches to adjust power levels if certain champions are out of control.

For future analysis, it may be more interesting to see the same test done in lower level of play with more casual players. At the highest level of play, you can exact more normalized results.