Capstone Project 1 Final Report:
A Retrospective Meta-Analysis on how to win League of Legends

Lucas Friedman, Springboard DS Career Track

League of Legends is one of the most widely-known computer games, not only due to its popularity, but also due to the depth, strategy, and competitive nature innate in the game. Released in 2009, the game falls into the genre known as Multiplayer Online Battle Arena, or MOBA for short. While sophisticated in play methods, the goal is simple: Destroy the enemy Nexus. This is the problem I want to solve: What is the best strategy one can have to win a game of League of Legends?

Figure 1. The 2018 LoL Championship had about 100 million viewers
https://www.riftherald.com/2018/12/11/18136237/riot-2018-league-of-legends-world-finals-viewers-prize-pool

There are many considerations that take place at the beginning of a game, usually starting from choosing which champion to play, to which items the player should build. Then there are complex choices and aspects that might not have gotten the attention they deserve: Is getting First Blood (first kill in the game) essential? What about destroying the first tower? Should our team always go for killing the dragon if it's a Cloud Elemental? Should I not even consider playing this certain champion…? These are the type of questions that can be answered if one looks into the data available, and give them the Data Science treatment.

The problem this project aims to solve is how to win a game of League of Legends, which is something every single LoL player wants to accomplish. By looking at the analysis, the player should be able to make confident choices in his gameplay, knowing that he or she has taken all the necessary steps for the best possible chance to win the round.

I will be using match data from Riot Games, the developers of the game, and their web-based API. I will use Python3 and the Requests package to set up the queries and upload the match data onto a JSON file. I hope to have at least 1000 matches worth of data, with each row constituting a match, and many different aspects of the game that took place noted on columns, after appropriate data wrangling.

My approach to solving this problem will involve choosing and focusing on the appropriate features that will end up being noted column-wise. This starts with the parts of the game over which the player has the most control: the selected champion, the selected runes, the items built, etc. Then, I will attempt to consider actions that the individual player does not have as much control over, and would otherwise require more of a team effort. Examples include more optional objectives such as securing Dragon and Baron kills, both of which give significant boosts of strength to the team. My deliverables will include the code, and a report clearly describing the thought process behind the choices behind, and construction of, the code.

Data Wrangling:

In order to acquire and wrangle the data I aim to use for this project, I used the **Requests** package in Python to set up *get* requests with the Riot API (https://developer.riotgames.com/api-methods/). I interacted with several endpoints throughout this process. Through my own Riot account, I requested a match list, containing game IDs, which can then be used to interact with a different endpoint, and thus give me access to detailed information about each game respectively, including the winning and losing players' stats. These endpoints have a rate limit of 100 requests every 2 minutes. Therefore, I set up my requests such that one is devoted to obtaining a match list, and 99 are for obtaining the game info from the match list through their respective game IDs. I repeated this process using a for-loop, as well as applying a *time* command from the **Time** package, in order to delay the iteration from each loop, so as to not exceed the rate limit.

Through this process I obtained a JSON file that contains detailed information on hundreds of games. I extracted the relevant dictionaries, which are nested such that each dictionary constitutes one game. **Figure 2** below shows an example of a section of one game.

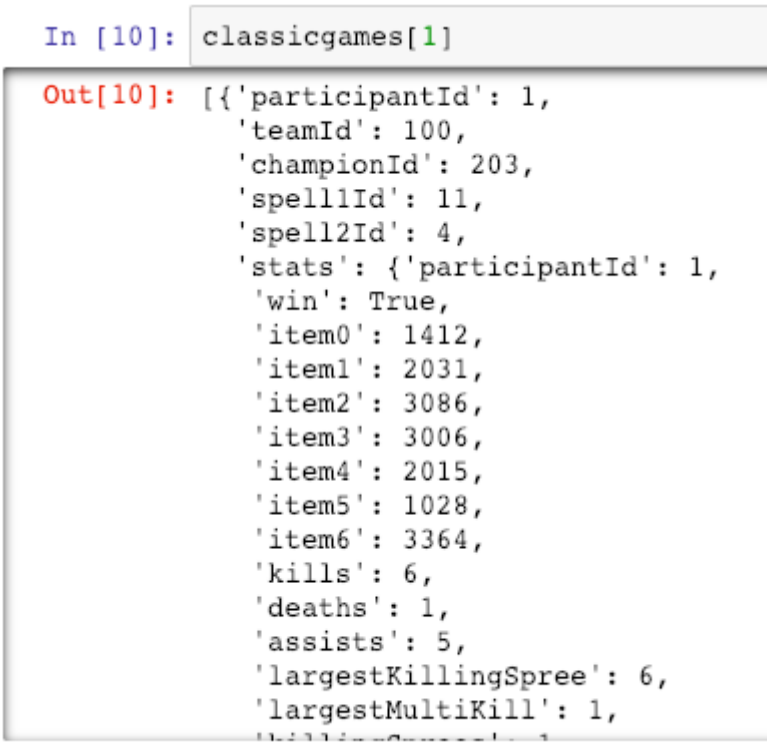

Figure 2. Example of some of one player's stats from one game

I then converted each game into a **Pandas** row using *pd.DataFrame.from_dict(),* as well as the *.apply()* method to apply this to all games. With all rows in Pandas format, I concatenated the rows to obtain a single dataframe

The dataframe at that point included the win/lose status of each player to their own columns (5 win columns, and 5 lose columns). I simplified this by removing those columns and adding prefixes to each column header, specifying to which player the stat is referring as well as whether the player won lost that game. Using this dataframe, I created a new dataframe that consists of aggregated stats from both the winning and losing team to prepare them for statistical analysis.

While there were no missing values or outliers, there were rows that could not be used simply because they corresponded to games played through a different game mode which, for the purposes of this project, were irrelevant and needed to be removed. As we only care about 'CLASSIC' games, filtering out the rest was easily done through conditional statements.

Data Storytelling/Inferential Statistics

The trends that we care about are those that are statistically significant and prevalent among the winning team, and perhaps the losing team. As mentioned before, after obtaining a

comprehensive dataframe with each row and column signifying a game and its features respectively, certain numerical features, such as gold earned, number of kills and deaths, and damage done to champions and turrets, were then aggregated according the winning team and the losing team. Next, I made several box plots to compare these aggregated values among the two teams. For the most part, the values from both teams looked fairly even. The winning team did have slightly higher means in some features, such as gold earned and damage dealt to champions, however, there is a very evident and large difference in one feature: Damage dealt to Turrets.

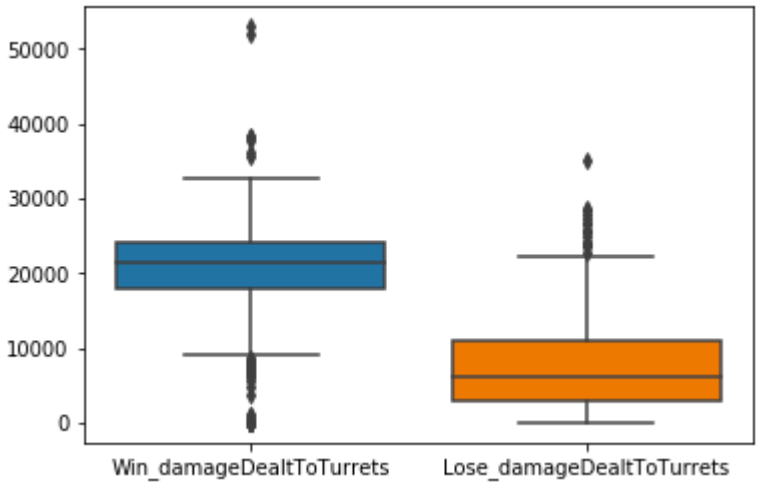

Figure 3. Large difference between Winning and Losing team when comparing this statistic

While somewhat surprising, and most would suspect difference in number of Kills in general to be a primary indicator, the fact that turret damage appears to be a major proponent does actually make a lot of sense. To reiterate, the goal of a game of League of Legends is to destroy the enemy Nexus, or base for clarity. While enemy champions can certainly get in the way of one accomplishing this, another component that is objectively in the way are these enemy turrets. Each team has 11 turrets total, 3 in each of the three lanes, and 2 right near the turret. As a fundamental aspect of the game, to access the nexus, these turrets have to be eliminated.

Figure 4. Summoner's Rift, Where every Ranked LoL game occurs
https://www.researchgate.net/figure/A-Map-of-the-League-of-Legends-game-play-in-the-classic-mode_fig1_319839481

To be more precise, 5 turrets at minimum must be destroyed. Besides a team surrendering, there is no way around this. Getting damage and kills on enemy champions can make it easier for a team to destroy the enemy nexus, but it IS NOT ESSENTIAL, whereas destroying these turrets IS ESSENTIAL. There have been games wrapped up with kill scores as little as 3-5 and as large as 45-50, and sometimes, the team with the lower kill count would still win. Why? Because they destroyed the enemy nexus. While there are many objectives laying around in the games, getting kills, last-hitting minions, getting gold, what's most imperative is how the team translates these features into increasing their advantage and getting closer to the enemy nexus. It would be safe to say that destroying these turrets is one of, if not the most, effective and reliable way to push a team's advantage.

From these insights given by the box plots, I decided it would be worthwhile to test the hypothesis of whether there are certain champions, in particular, champions with 'special abilities' or emphasis on dealing massive damage to turrets.

Exploratory Data Analysis

From my previous data visualization findings, it appears that the winning teams tend to have a much higher mean of Turret Damage compared to the losing team. The reason/intuition behind this is that destroying Turrets is in fact a MANDATORY objective, more so than killing enemy champions, getting a higher creep score, or any other objective in the game. This led me to my next question, does there tend to be a higher win rate among champions who 'specialize' in dealing damage to towers? I approached this question by forming a **two-sample t-test for comparing means**. One sample would include games (rows) that included a champion of interest, and the other sample would not include the champion at all. I would always utilize my whole dataset, meaning that whichever games that did not include my champion of interest would be included in the other sample. I did this first with personally one of my favorite champions, Tristana, who has special abilities to deal more damage to turrets, and then I created a function that would accomplish this with any champion. From here I would generate a list of p-values. Since at this point we are generating multiple null hypotheses and conducting multiple comparison tests, **false discovery rate controlling procedures,** more specifically, **Benjamini-Hochberg procedures** were applied in order to control the amount of Type I errors one may typically see in these multiple comparison tests The p-values underneath our threshold would signify that their respective champions have a statistically significant impact on turret damage.

```
In [49]: # Set threshold at 0.25
         turrets_pval[turrets_pval['Corrected P-value'] < 0.25]
```

Out[49]:

	Champion	P-value	Corrected P-value
1	Ahri	0.006998	0.244437
36	Hecarim	0.003219	0.244437
41	Janna	0.005959	0.244437
70	MasterYi	0.008519	0.244437
120	Udyr	0.010328	0.244437
135	Yorick	0.010273	0.244437

Figure 4. The data tells us these champions have a statistical impact on turret damage

Hecarim, Master Yi, Udyr, and Yorick have their unique strengths, but one thing they all have in common is that they are all strong **split-pushers**. To understand what split-pushers are, it's first important to understand that Summoner's Rift, the name of the 'area' where the game takes place, is fairly large. In general, it takes about a solid 25 seconds to go from one corner of the map to the other. Also, League of Legends uses Fog of War mechanics, meaning unless you or anyone on your team is next to an enemy, enemies cannot be seen. Split-pushing is the act of going to a lane, typically the top or bottom lane (whichever is farthest from where most of the enemy team is located), and dealing damage to turrets on that lane, on your own.

Here is a very common situation in League of Legends: A Fire Dragon is up at the Dragon Pit, which is on the bottom-right corner of the map. Whoever destroys the dragon gets a boost of attack damage for their whole team. All 5 members from Team A are at the pit, and only 4 members from Team B are at the pit. The remaining member from Team B is at the top lane, **split** from the rest of the team, **pushing** and dealing damage to turrets. Team B is obviously at a disadvantage in one sense, because numbers can have a strong influence on how team fights play out. While the 4 members from Team B are pursuing this high-risk high-reward strategy (risk: less members, reward: getting champion kills and a fire dragon). The split-pusher from Team B up top is getting guaranteed turret damage, which ultimately facilitates access to the enemy nexus. Trade-offs play a major theme in League of Legends, and gauging them and deciding whether to take action or not is simply part of the game.

The most frequent qualities of a good split-pusher are: high movement speed, high attack speed, and decent tankiness (They absorb damage well). The four champions previously mentioned indeed have those characteristics.

In addition, I performed statistical tests to determine the amount of correlations there are between all of my selected independent variables, and turret damage, in order to see if perhaps there are other specific features that should be considered a significant influence on turret damage and by consequence, winning the game. I used **Pearson's** correlation test as well as **Spearman's** correlation test, and found that all features I've chosen do observe highly correlate with turret damage. League of Legends is very punishing game in the sense that one small advantage can really dictate the outcome of the game and be used to create more advantages. Getting a champion kill can lead to a gold advantage, which can lead to more kills, then more gold, and then more items which destroys turrets, thus creating more gold and more advantages.

Machine Learning

The next step is to prepare a dataframe for machine learning. This dataframe is merely different in terms of structure and organization, such that features are labeled according to Team A/B, instead of Winning/Losing Team, which facilitated doing certain statistical analysis. Another column was added, which contains the predictor variables for our model. A value of 1 means

Team A won that match, whereas a value of 0 means Team B won that match. While Machine Learning is more popularly known for making predictions, we are also, if not more interested in, determining the strongest features that dictate win conditions. Implementing a **Logistic Regression model** would be one of the most effective ways to accomplish this, since it is easy to extract coefficient values for all features. Just like what was done with the dataframe for statistical analysis, the dataframe for machine learning had its numerical columns aggregated team-wise. In addition, the categorical values were also aggregated team-wise by count. This includes items, runes, and technically champions, but since champions can only have counts of 0s or 1s, they can be thought of as having Pandas' **get_dummies** function applied to the columns pertaining to them. All champions in Ranked matches must be unique.

As seen in **Figure 5** below**,** The logistic regression model, *logreg,* was defined. Then, a **GridSearchCV** object was instantiated to perform a 5-fold cross validation on the model, and to find the optimal hyperparameters for the model. Here, the hyperparameters of interest are *C*, whose potential values are defined in *c_space*, and *penalty*, which could be either *l1 (Lasso),* or *l2 (Ridge)*. These are put into the variable, *param_grid*, which can be in turn used as a parameter for the GridSearchCV object. *X_train* and *y_train* were created using the **train_test_split** function from sklearn, on the ML dataset, and fit into the model. From the output, we see that the accuracy score looks very strong (~98%), and it looks like the model benefited from implementing a **Ridge Regularization**. The low, optimal Logistic Regression Parameter, C, implies that the regularization strength is pretty high.

```
# Begin tuning our first model

c_space = np.logspace(-10, 8, 20) # Covers a WIDE range of possible C values
param_grid = {'C': c_space, 'penalty': ['l1', 'l2']}

# Instantiate the logistic regression classifier: logreg
logreg = LogisticRegression()

# Instantiate the GridSearchCV object: logreg_cv
logreg_cv = GridSearchCV(logreg, param_grid, cv=5)

# Fit it to the training data
logreg_cv.fit(X_train, y_train)

# This will be useful for creating our ROC curves
y_pred_prob = logreg_cv.predict_proba(X_test)[:,1]

# Print the optimal parameters and best score
print("Tuned Logistic Regression Parameter: {}".format(logreg_cv.best_params_))
print("Tuned Logistic Regression Accuracy: {}".format(logreg_cv.best_score_))
```

```
Tuned Logistic Regression Parameter: {'C': 6.951927961775605e-08, 'penalty': 'l2'}
Tuned Logistic Regression Accuracy: 0.9809045226130654
```

Figure 5. Code for the first model and accuracy results.

Since the features vary in numerical range, from 0 to 1, to 10,000 to 100,000, I was curious to see if standardizing all features would improve model accuracy. To test this, I created a machine learning pipeline that implements a standardization step, called **StandardScaler**, into the model. The accuracy ended up being lower than that the original model.

I extracted the top 10 and bottom 10 coefficients from my models, and generated side bar graphs to better visualize the impact that their corresponding features have. Since a victory for Team A and Team B are represented as values 1 and 0 respectively, it makes sense mathematically that the highest values indicate the biggest proponents for Team A's victory, and likewise for the lowest values on Team B's victory.

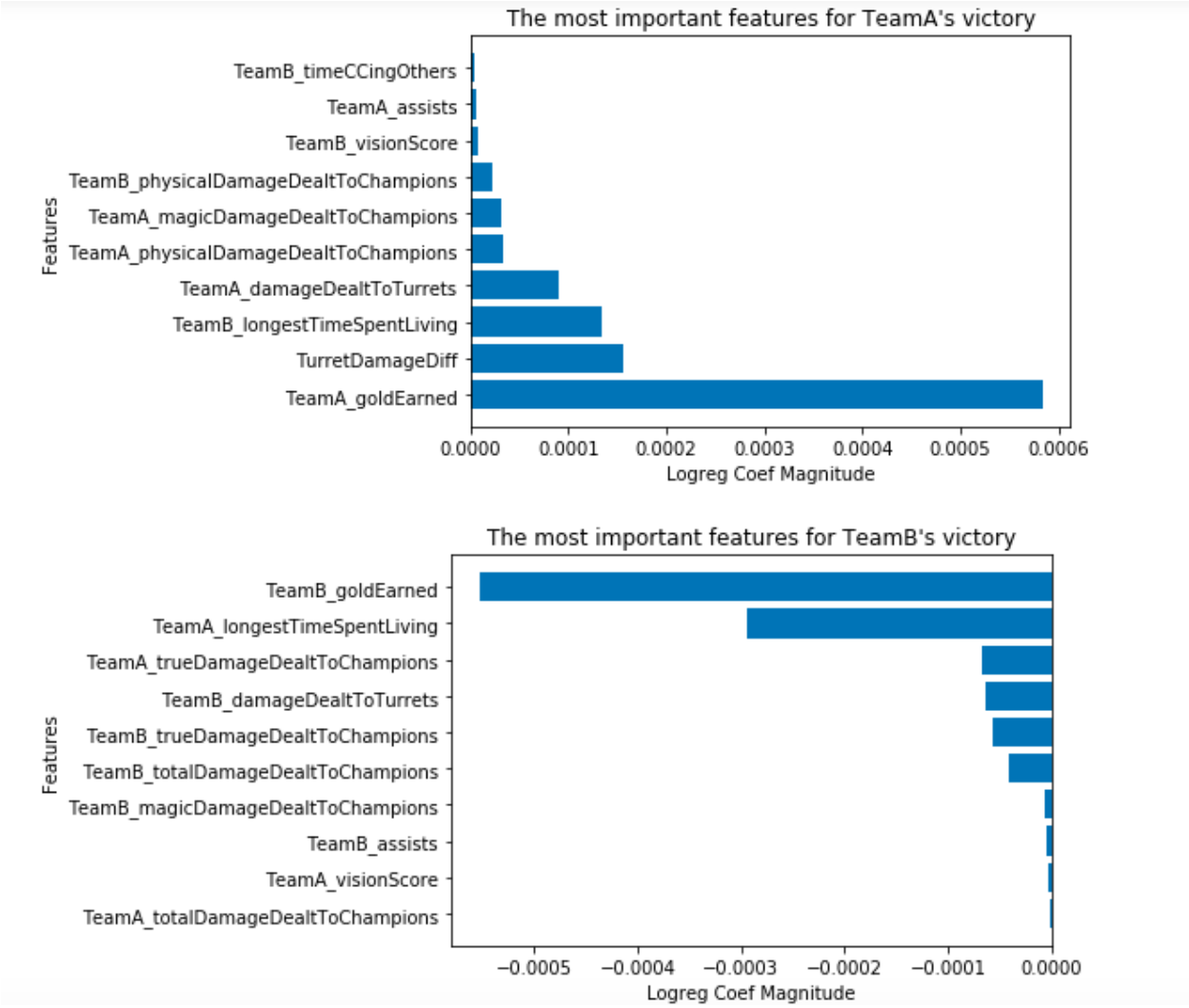

Figure 6. The most important features for Team A/B's victory

Seeing the same feature on both graphs is a rather helpful confirmation that the feature is indeed an important feature to predict victory. For example, from our first model we see that the amount of gold earned was the biggest indicator of both Team A and Team B's victory. This makes sense because simply put, generating gold allows one to buy items, which makes accomplishing all available objectives in League of Legends easier. This was a given from the start, but I was more interested in other underlying details that can perhaps make this gold generation easier.

Another interesting feature that seemed relevant in both teams' victory is the OTHER team's longest time spent living. My personal take in this is that the recently implemented Bounty System is the primary cause for this. Understanding this also requires some background knowledge of the game. As mentioned before, LoL is a punishing game in that getting even a small advantage in the game can translate into dramatically pushing that advantage towards victory. In other words, the chances of making a comeback in the game tend to be rather unlikely for the losing team. In order to increase likelihood of comebacks, Riot Games added a bounty system where players that are particularly far ahead would be WORTH MORE GOLD if destroyed by the other team. The higher a player's time avoiding death is a very strong indicator of how far that player is in the game. The data seems to imply that this bounty system generates so much gold for the the 'losing team' that it has become a top indicator for their victory! Comebacks have not only become more frequent, they dictate victories! Whether this system was intended to have that much influence is certainly up for debate.

TurretDamageDiff made its way as the second most important feature for Team A's victory, since that feature was set to equal Team A's damage dealt to turrets minus Team B's damage dealt to turrets. The higher this value, the more damage Team A did than Team B. This further substantiates what we've concluded previously, that dealing damage to turrets is an important strategy to commit to. Other features common for both team's victory are all forms of damage damage dealt to champions, and, though lesser in impact, supplying vision for the team. Since these values are comparatively lower, it's too speculatory to say committing to these strategies will help a team win.

As seen in **Figure 7**, I created two more side bars to show what the scaled Logistic Regression Model could tell us. Once again, TurretDamageDiff makes its way to the top, for predicting Team A's victory, but the rest of the features presented seem to be rather random (They don't overlap in both Team A and Team B's graphs.

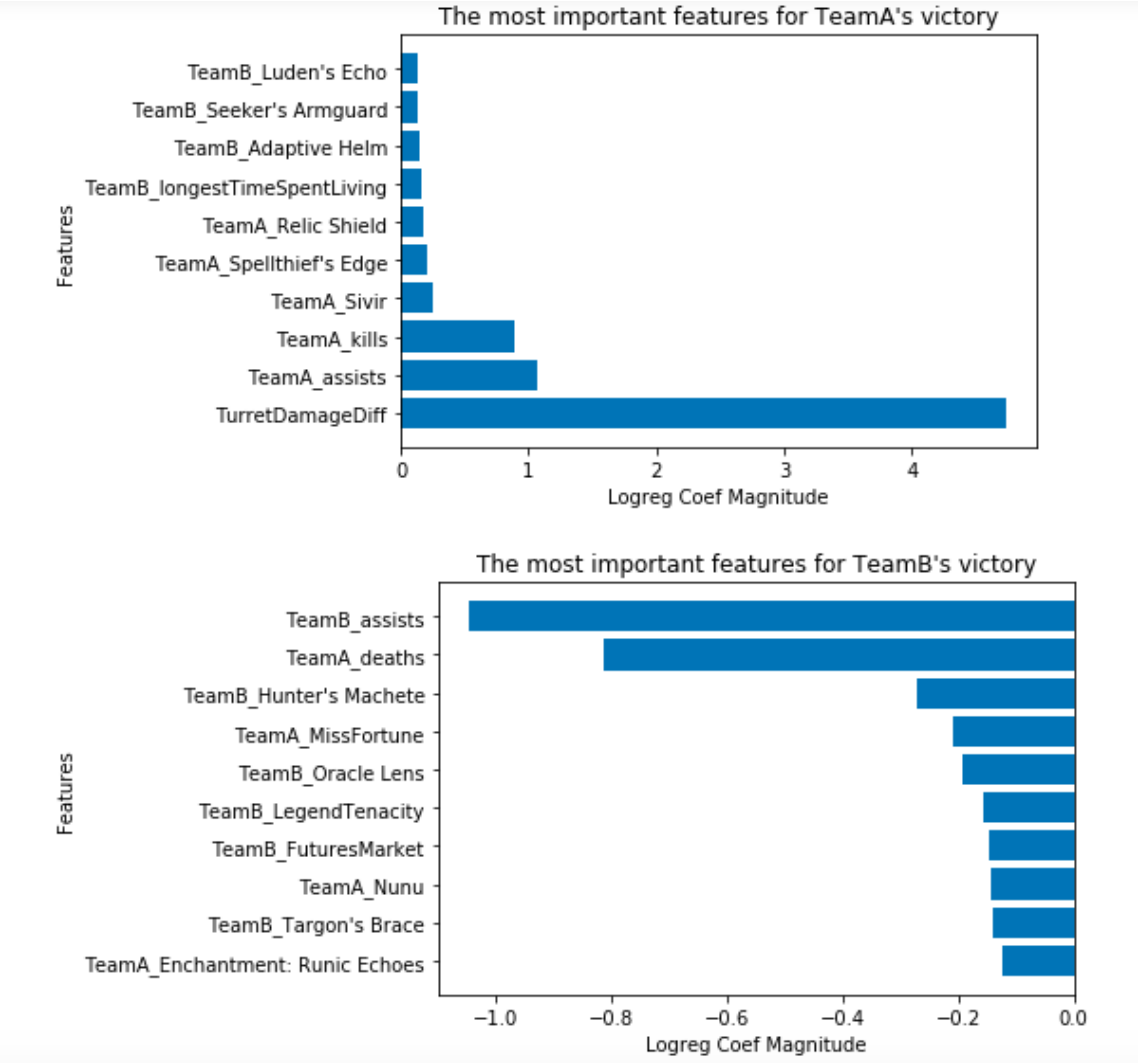

Figure 7. The most important features for Team A/B's victory (Scaled)

In a perfect world, I would create a model that successfully predicts which team would win based only on features that can be determined before the game even begins. Many of the features I used previously are those that happen in game. Having a model that can be used to help decide which players and runes would maximize one's chances of winning would undoubtedly be very appealing to every LoL player. With that in mind, I created yet another yet another dataframe, only containing features that can in fact be controlled before the game begins, including champions and runes, to see how it would fair when fit into a Logistic Regression model.

Unfortunately and as somewhat expected, the accuracy of 54.5% makes this model rather unreliable. Features matter! This lack of performance can be further visualized from a Receiver Operating Characteristic Curve (ROC Curve) for the model in **Figure 8**. ROC curves of the other models were also generated. We can see based on the area under the curve (AUC)

not only the poor performance from the pregame model, and but the rather strong performance from the other models! The higher the AUC, the better the model can distinguish between the predictor variables.

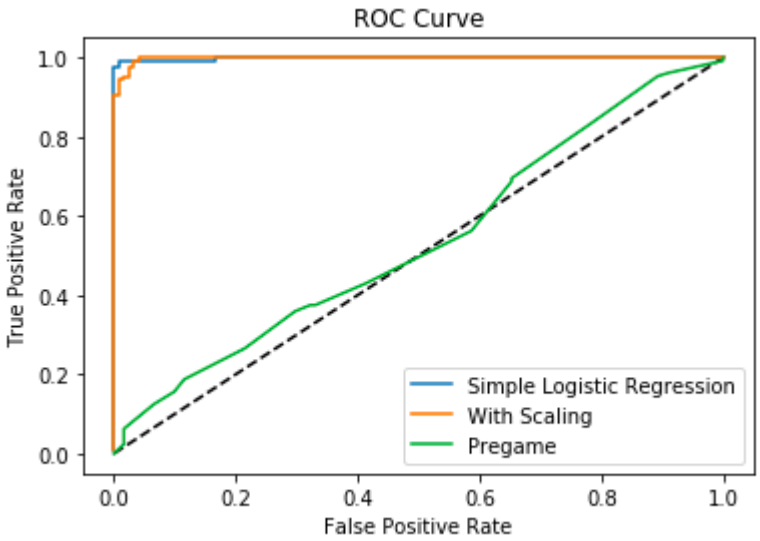

Figure 8. The ROC curves of the three generated models

The data presented to us gives several meaningful conclusions on improving one's chance of winning a game of LoL:

- Maximizing turret damage is a worthy strategy to focus on
- In particular, Hecarim, Master Yi, Udyr, and Yorick are strong contenders since the data implies they significantly impact turret damage difference between teams
- Letting the other team get ahead in order to increase their bounties could potentially be an effective strategy

With this in mind, there is certainly more future work that can be done in order draw more conclusions and insights:

- **Bring in more data.** This is always going to be an effective move. The reason it is so for this case is that the API was interacted with through using my personal account information. This means that all games used in the dataset fell into a fairly specific rank level. The rank levels that exist are Iron, Bronze, Silver, Gold, Platinum, Diamond, Master, Grandmaster, and Challenger. I fall into High Silver/Low Gold Rank, meaning all games used include players from similar ranks. In order to make the model more robust, more data from a wider distribution of ranks should be included.
- **Implement more feature engineering.** We saw from the side bar charts that Turret Damage difference was a significant feature that the model used to make its predictions. This can be done to a greater extent with essentially other features.