BIG DATA MANAGEMENT PROJECT

Final Project Report

Submitted By	Amber Yeo En (222809U)
Module Group	IT3388-02
Team Name	WASD
Team Number	3
Supervisor's Name	Mr Alvin Tay
Team Members	Amber Yeo En (Leader)
Latest Amendment Date	18/8/2024

Table of Contents

Executive summary

The global gaming industry has **rapidly expanded**, with games like League of Legends, Valorant, and Dota 2 drawing millions of players and viewers worldwide. The rise of e-sports has transformed these games into a **multibillion-dollar market**, featuring professional leagues, major tournaments, and collegiate programs. Events like The International and the League of Legends World Championship **attract large audiences, significant brand investments, and substantial prize pools**.

However, there remains a **gap in understanding** the match dynamics, demographic trends, and community sentiments that shape these games. By analyzing these factors, developers and community managers can **enhance gameplay, improve user experience, and foster long-term player engagement, ensuring the continued success of the e-sports industry**.

Problem Statement

With the rise of e-sports, player retention and satisfaction have become critical challenges. **Newcomers** often struggle with the **steep learning curve**, including character selection and item purchases, which can lead to **negative initial experiences**. This **discourages** them from continuing to play. Additionally, **seasoned players** face **difficulties in keeping up** with the **growing number of game options**, leading to **potential disengagement**.

From this medium article, "There were tons of rules, a game master and several different artifacts to needed tracking. We had spent **over 3 hours** trying to figure out the game and the whole **ordeal was negative**". Suggesting that gaming can get **overwhelming**, leaving a **bad impression** for those who just start out playing the game.

Proposed Solution

To provide a **detailed analysis** of game play, demographic trends, and community feedback, focusing on **3 popular e-sports games**: League of Legends (LOL), Dota 2, and Valorant. Ultimately helping stakeholders improve game enjoyment, informed of current trends, and community involvement**.**

Stakeholders

Primary: E-sports Game Businesses who wants to understand game success and popularity

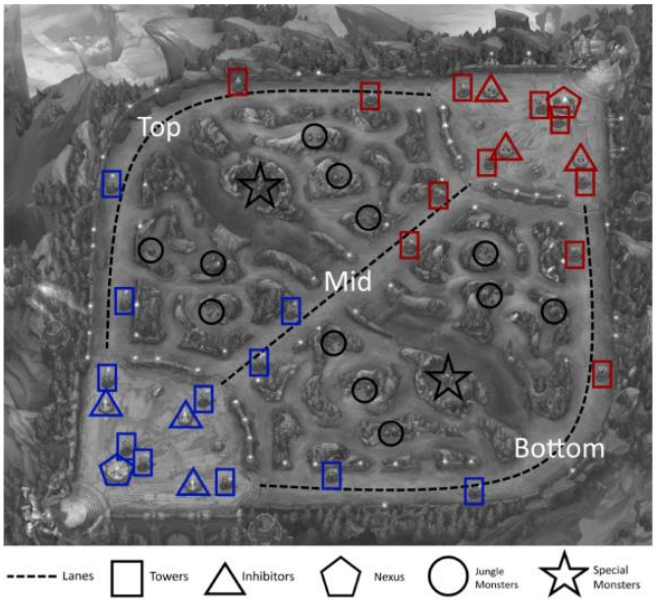
Secondary: Players who want to improve game performance or want to know more about the games.

Hypotheses

Main Hypothesis: The success and popularity of multiplayer games are influenced by match performance, viewership trends, and community sentiment, leading to higher win rates, increased tournament winnings, and overall game popularity.

My sub-hypothesis: Match attributes and statistics in League of Legends affects game outcomes.

Game Play



In League of Legends, there are four lanes: Top (for fighters), Mid (for mages), Bottom (for marksmen supported by a tank or support champion), and Jungle (assassins roaming the map, killing monsters).

The goal is to **destroy the opponent's Nexus**. Players earn gold by killing monsters, opponent's minions, champions, or towers and use it to buy items that strengthen their champions.

Special monsters provide team-wide buffs, offering a temporary advantage.

Jargons

Word	Meaning
Champions	Playable characters in League of Legends
LoL / LOL	League of Legends
Challenger	Highest tier in League of Legends
Items	Are items that can be bought during the game with gold earned to improve champion skills
Perks	Are enhancements that can be added to a champion before a match. There are 3 slots, defence, offense and flex.
puuid	Player universally unique Identifiers with a fixed length of 78 characters
Ranked (matches)	A game mode that allows you to increase your league points by winning. The higher your league points, the higher the tier.
Ban	In ranked game mode, each player can ban a character they don't want their opponent to use. Can be of any role
League Points	Accumulated points from winning ranked matches. If match was lost, points will be deduced
Champion role	Like mage, marksman, jungle, support, tank. Each role has a recommended lane to go

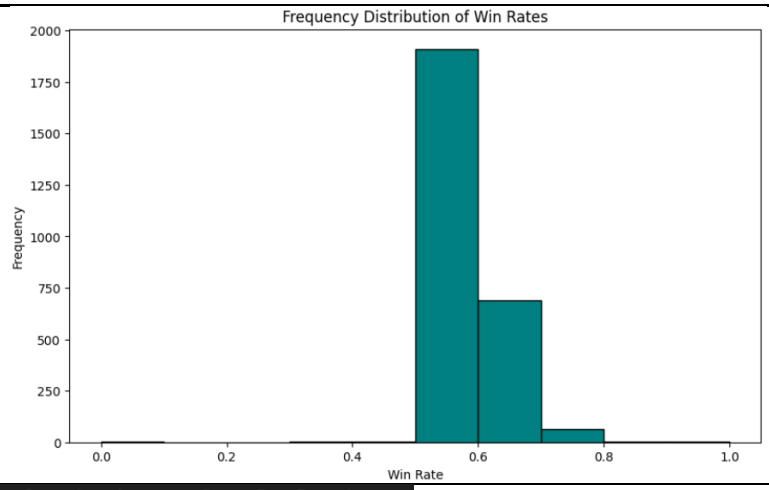
Data Preparation

Extract-Transform-Load Data

Identified user needs from their pain points:

- **Champion selection insights:** Users often struggle with deciding which champion to play as they begin gaming. They want insights on effective champion selection tailored to specific roles, positions, and personal preferences, to make informed decisions on champion selected.
- **Winning strategies:** Users want to know effective strategies that can increase their chances of winning matches. This can be through in-game events like towers taken, dragons killed, champion selected, champion counter picks, perks and items bought.

Final datasets	
No.	Details
1	**Player Data** https://developer.riotgames.com/apis#league-v4/GET_getChallengerLeague https://developer.riotgames.com/apis#summoner-v4/GET_getBySummonerId The first API link returns information of players in Challenger tier (highest tier) like their league points gathered from winning matches, total wins, loss and their summonerids. While the second API link uses the summonerId returned to get their respective puuids to be able to get the match data later. This data provides detailed information about player performance and demographics. This dataset is crucial to retrieve player match data. By integrating this dataset with match data, we can analyze how player attributes, such as rank and win-loss records, correlate with match outcomes. Understanding the profiles of top players and retrieving their match data can offer insights of where top players stand and factors that contribute to higher win rates. Type of dataset (initial): JSON Type of dataset (transformed): CSV Number of rows: 2.6k Columns used: puuid, rank, league points, wins, losses, platform, region, inactive, summonerID Data cleaning and transformation: The first API provides data on Challenger tier players from each platform region with no null rows. After collecting players from all regions, I re-ranked them based on their league points to ensure a consistent and comparable ranking system across different regions. The Challenger league API only provides the player's summonerId, which is not sufficient to retrieve their match data. To obtain comprehensive match information, I needed the player's puuid (Player Unique Identifier). The puuid is required to access detailed match history and performance data for each player Using a second API endpoint, I made an additional API call for each player to retrieve their puuid using the summonerId. This ensured that I could collect and analyze complete match data for Challenger tier players from various regions Data understanding:



The graph showing the frequency of win rates among top players, with an average win rate of 0.5-0.6, indicates that even the best players win just over half of their matches. This highlights the competitive nature of the game at the highest level, where skill disparities are minimal, leading to closely contested matches.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2676 entries, 0 to 2675
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   rank          2676 non-null   int64
 1   summonerId    2676 non-null   object
 2   leaguePoints  2676 non-null   int64
 3   wins          2676 non-null   int64
 4   losses        2676 non-null   int64
 5   veteran       2676 non-null   bool
 6   inactive      2676 non-null   bool
 7   freshBlood    2676 non-null   bool
 8   hotStreak     2676 non-null   bool
 9   platform      2676 non-null   object
 10  region        2676 non-null   object
 11  puuid         2676 non-null   object
dtypes: bool(4), int64(4), object(4)
memory usage: 177.8+ KB
```

2 **Match data**

https://developer.riotgames.com/apis#match-v5/GET_getMatch

It gets a comprehensive match performance breakdown of ranked 5v5 matches played by Challenger tier players, like match outcome and duration, player performance metrics (kills, deaths, assists, damage dealt and taken), team achievements (dragon and tower kills), and player-specific information (champion chosen, item id purchased, perk id used). I got the top 100 matches from 100 top players, with the potential to generate more match data for further analysis.

This data is essential to analyse how various match attributes and player statistics impact win rates. By examining factors like champion effectiveness, item builds, player roles, and team strategies, we can identify patterns and correlations that lead to higher success rates in matches. This helps in understanding the key drivers of performance and developing strategies to improve win probabilities in League of Legends.

	Initial	Transformed
Type of dataset	JSON	CSV
Number of rows	10K	97K
Number of columns	2957	84
Columns	game duration, game outcome, player position, champion chosen, outnumbered kills, quickcleanse, multiKillOneSpell, multiTurretRiftHeraldCount, multikills, poroExplosions,	match_gameDuration, player_win, player_teamPosition, player_champId, player_banChampion, player_goldEarned, player_kills, player_deaths, player_assists,

	multikillsAfterAggressiveFlash, outnumberedKills, outnumbered kills, quickcleanse, outnumberedNexusKill, etc	player_item0, player_item1, player_perk1, team_champion_kills, team_tower_first, etc

Data cleaning and transformation:

Each request from the LoL match data API provides detailed information about a single match, with the returned JSON containing nested data that requires flattening before it can be saved to a dataframe. For example, the "participants" key holds a nested list for each player, containing additional details about their performance. To process this data, we first flattened the JSON to remove nesting, then **split the JSON response so that each participant had their own row**, rather than a single row per match with all participant information in separate columns. Each unique matchID should **have exactly 10 rows** (10 players in a match).

The flattened JSON data initially included over 300 columns. To make the dataset more manageable and focused, we filtered out sparse information such as the number of outnumbered kills and quick cleanse. After creating the dataframe, we removed duplicates (since top players might have played in the same match, ensure that **each matchID has exactly 10 rows or else all rows of that matchID will be removed**), eliminated empty rows (resulting from failed requests), and renamed columns to enhance understandability.

Before flattening:

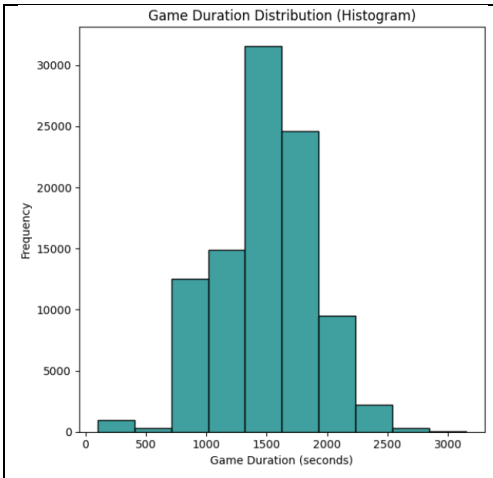

```
'info': {'endOfGameResult': 'GameComplete',
 'gameCreation': 1719088341826,
 'gameDuration': 1261,
 'gameEndTimestamp': 1719089635177,
 'gameId': 5025296953,
 'gameMode': 'CLASSIC',
 'gameName': 'teambuilder-match-5025296953',
 'gameStartTimestamp': 1719088373833,
 'gameType': 'MATCHED_GAME',
 'gameVersion': '14.12.594.4901',
 'mapId': 11,
 'participants': [{'allInPings': 0,
   'assistMePings': 1,
   'assists': 4,
   'baronKills': 0,
   'basicPings': 0,
   'bountyLevel': 0,
   'challenges': {'12AssistStreakCount': 0,
    'InfernalScalePickup': 0,
'perks': {'statPerks': {'defense': 5011, 'flex': 5008, 'offense': 5007},
 'styles': [{'description': 'primaryStyle',
   'selections': [{'perk': 8214, 'var1': 643, 'var2': 0, 'var3': 0},
    {'perk': 8275, 'var1': 8, 'var2': 0, 'var3': 0},
    {'perk': 8210, 'var1': 0, 'var2': 0, 'var3': 0},
    {'perk': 8237, 'var1': 397, 'var2': 0, 'var3': 0}],
   'style': 8200},
  {'description': 'subStyle',
   'selections': [{'perk': 9105, 'var1': 19, 'var2': 40, 'var3': 0},
    {'perk': 8299, 'var1': 725, 'var2': 0, 'var3': 0}],
   'style': 8000}]},
```

After flattening

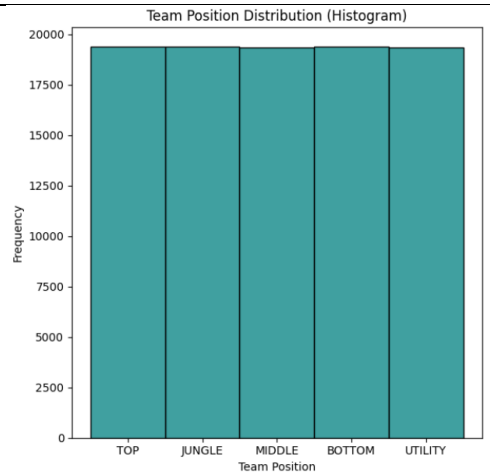
```
'info_participants_1_perks_styles_0_description': 'primaryStyle',
'info_participants_1_perks_styles_0_selections_0_perk': 8128,
'info_participants_1_perks_styles_0_selections_0_var1': 335,
'info_participants_1_perks_styles_0_selections_0_var2': 8,
'info_participants_1_perks_styles_0_selections_0_var3': 0,
'info_participants_1_perks_styles_0_selections_1_perk': 8143,
'info_participants_1_perks_styles_0_selections_1_var1': 198,
'info_participants_1_perks_styles_0_selections_1_var2': 0,
'info_participants_1_perks_styles_0_selections_1_var3': 0,
'info_participants_1_perks_styles_0_selections_2_perk': 8138,
'info_participants_1_perks_styles_0_selections_2_var1': 10,
'info_participants_1_perks_styles_0_selections_2_var2': 0,
'info_participants_1_perks_styles_0_selections_2_var3': 0,
'info_participants_1_perks_styles_0_selections_3_perk': 8135,
'info_participants_1_perks_styles_0_selections_3_var1': 450,
'info_participants_1_perks_styles_0_selections_3_var2': 5,
'info_participants_1_perks_styles_0_selections_3_var3': 0,
'info_participants_1_perks_styles_0_style': 8100,
```

Data understanding:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96850 entries, 0 to 96849
Data columns (total 84 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   match_matchId                   96850 non-null  object
 1   match_gameId                    96850 non-null  float64
 2   match_gameStartTimestamp        96850 non-null  float64
 3   match_gameEndTimestamp          96850 non-null  float64
 4   match_gameDuration              96850 non-null  float64
 5   match_gameMode                  96850 non-null  object
 6   match_gameType                  96850 non-null  object
 7   match_mapId                     96850 non-null  float64
 8   match_platformId                96850 non-null  object
 9   player_summonerId               96850 non-null  object
 10  player_puuid                    96850 non-null  object
 11  player_teamId                   96850 non-null  object
 12  player_teamPosition             96850 non-null  object
 13  player_lane                     96850 non-null  object
 14  player_champName                96850 non-null  object
 15  player_champId                  96850 non-null  float64
 16  player_champLevel               96850 non-null  float64
 17  player_champExp                 96850 non-null  float64
 18  player_win                      96850 non-null  object
 19  player_gameEndedInSurrender     96850 non-null  object
...
 82  team_tower_first                96850 non-null  object
 83  team_tower_kills                96850 non-null  float64
dtypes: float64(59), object(25)
memory usage: 62.1+ MB
```



The average game duration is 1400-1900s (24-32 mins) long. This suggests that the games are generally well-paced, without extremely short games that indicate quick wins due to unbalanced teams, or extremely long games that suggests prolonged stalemates.



This graph shows that the teams have a well-rounded composition of positions. This composition, consistently followed across all matches, suggests that balanced team roles are crucial for effective gameplay. It indicates that players understand the importance of filling each role to maximize team synergy, strategy execution, and overall match success.

3 **Champion data**

https://raw.communitydragon.org/latest/plugins/rcp-be-lol-game-data/global/default/v1/champion-summary.json

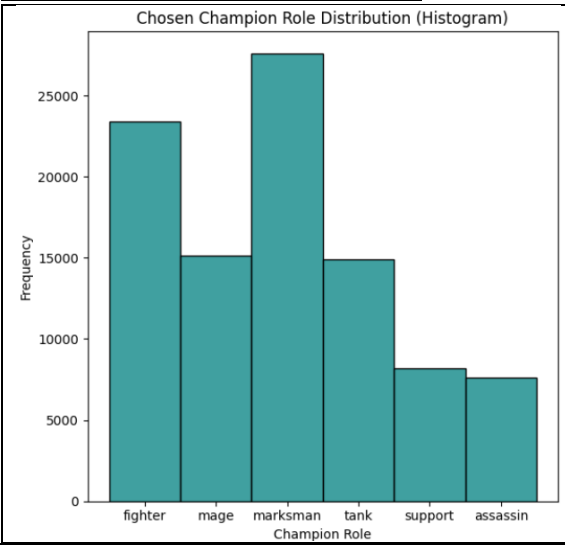
It gives detailed information about champions in League of Legends. The champion dataset includes columns such as id, name, image link, roles, and primary role. This data provides comprehensive details about each champion's identity and function within the game. By integrating this dataset with match data, we can analyze how specific champions and their roles influence match outcomes. For instance, understanding which champions have higher win rates or perform better in particular roles can offer valuable insights into effective champion selection, role effectiveness and team composition.

	Initial	Transformed
Type of dataset	JSON	CSV
Number of rows	168	168
Number of columns	5	5
Columns	Id, name, alias, squarePortraitPath, roles	id, name, image link, roles, primary role

Data transformation: I saved the JSON response as a pandas dataframe. In the dataframe, the 'roles' column contains lists of champion roles since some champions have multiple roles. To facilitate easier retrieval, I created a new column, 'primary role,' which stores the first role from each list. This ensures that each champion's primary role is readily accessible for analysis.

Data understanding:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 168 entries, 0 to 167
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  168 non-null    int64
 1   name                168 non-null    object
 2   squarePortraitPath  168 non-null    object
 3   roles               168 non-null    object
 4   primaryRole         167 non-null    object
dtypes: int64(1), object(4)
memory usage: 6.7+ KB
```



The most popular champion role chosen was marksman and fighter. This suggests that players are inclined towards roles that are pivotal in dealing damage. Since the team positions are balanced and each champion have a recommended position to fill, this means that players prefer roles that allow them to make significant contributions in terms of damage output, often being the primary sources of kills and team fight dominance. Marksmen and fighters are crucial in both early skirmishes and late-game team fights.

4 **Perks data**

https://raw.communitydragon.org/latest/plugins/rcp-be-lol-game-data/global/default/v1/perks.json

It gives me detailed information on perks used on champion in League of Legends. This perks dataset includes columns like id, name, tooltip and attribute. This data provides detailed descriptions of each perk's functionality and its recommended use attributes. By integrating this

perks dataset with match data, we can analyze how different perks affect player performance and match outcomes.

Understanding the impact of various perks on gameplay strategies and win rates can offer valuable insights into the effectiveness of different perk choices. This relevance is crucial to the subhypothesis that match attributes and statistics, including perk selections, significantly affect win rates in League of Legends. By examining which perks top players frequently choose and their corresponding match outcomes, we can better understand the strategies that lead to higher win rates.

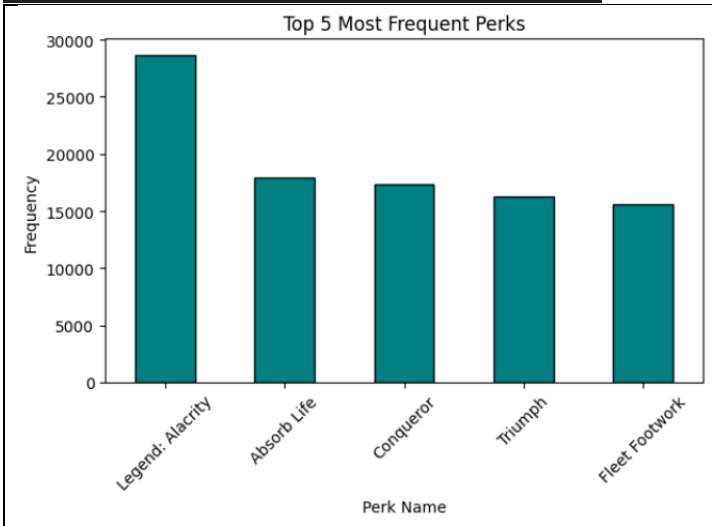
	Initial	Transformed
Type of dataset	JSON	CSV
Number of rows	99	99
Number of columns	10	4
Columns	Id, name, majorChangePatchVersion, tooltip, shortDesc, longDesc, recommendationDescriptor, iconPath , endOfGameStatDescs, recommendationDescriptorAttributes	id, name, tooltip, attributes

Data cleaning and transformation:

I saved the JSON response as a pandas dataframe. The initial dataframe had 10 columns, but some columns contained redundant information, such as 'tooltip', 'shortdesc', and 'longdesc', all of which provided similar descriptions of each perk. I retained only the 'tooltip' column for its concise descriptions. Additionally, I removed other unnecessary columns, ultimately leaving four key columns that are essential for my analysis.

Data understanding:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 4 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   id                                99 non-null     int64
 1   name                              99 non-null     object
 2   tooltip                           99 non-null     object
 3   recommendationDescriptorAttributes  99 non-null     object
dtypes: int64(1), object(3)
memory usage: 3.2+ KB
```



The top 5 perks used are Legend: Alacrity, absorb life, conquer, triumph and fleet footwork. Legend: Alacrity perk was used almost twice as much compared to the 2nd most used perk. This suggests that players prioritize perks that enhance their sustained damage output and survivability. Legend: Alacrity's dominance in usage indicates a strategic emphasis on improving attack speed, crucial for maximizing damage potential and securing kills during gameplay. Absorb life follows closely, highlighting players' interest in sustaining health in combat scenarios

5 **Item data**

https://raw.communitydragon.org/latest/plugins/rcp-be-lol-game-data/global/default/v1/items.json

It gives detailed information about various in-game items available to players in League of Legends. Each row in the dataset represents a unique item like its name, attributes, description and cost price. This is used to supplement item data information to match data.

This data is relevant to my sub-hypothesis as analysing item usage and costs can reveal insights into effective in-game strategies and item builds that contribute to higher win rates. By examining which items are commonly purchased in winning matches, we can determine the impact of item selection on match outcomes. This information helps in understanding the strategic importance of items in gameplay and their influence on player performance and overall match success.

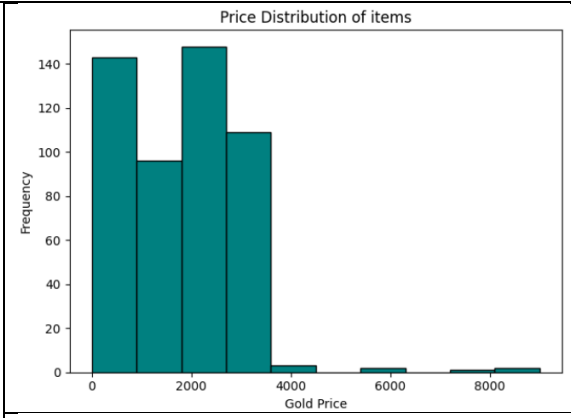
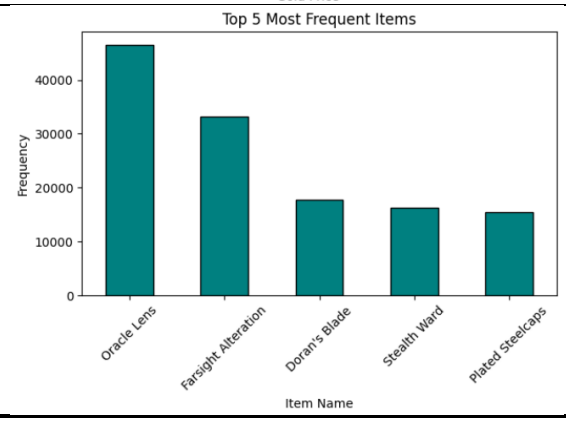
	Initial	Transformed
Type of dataset	JSON	CSV
Number of rows	504	504
Number of columns	18	5
Columns	Id, name, description, active, instore, from, to, categories, maxStacks, requiredChampion, requiredAlly, specialRecipe , requiredBuffCurrencyName, requiredBuffCurrencyCost, isEnchantment, price, priceTotal, iconPath	id, name, description, categories, priceTotal

Data cleaning and transformation:

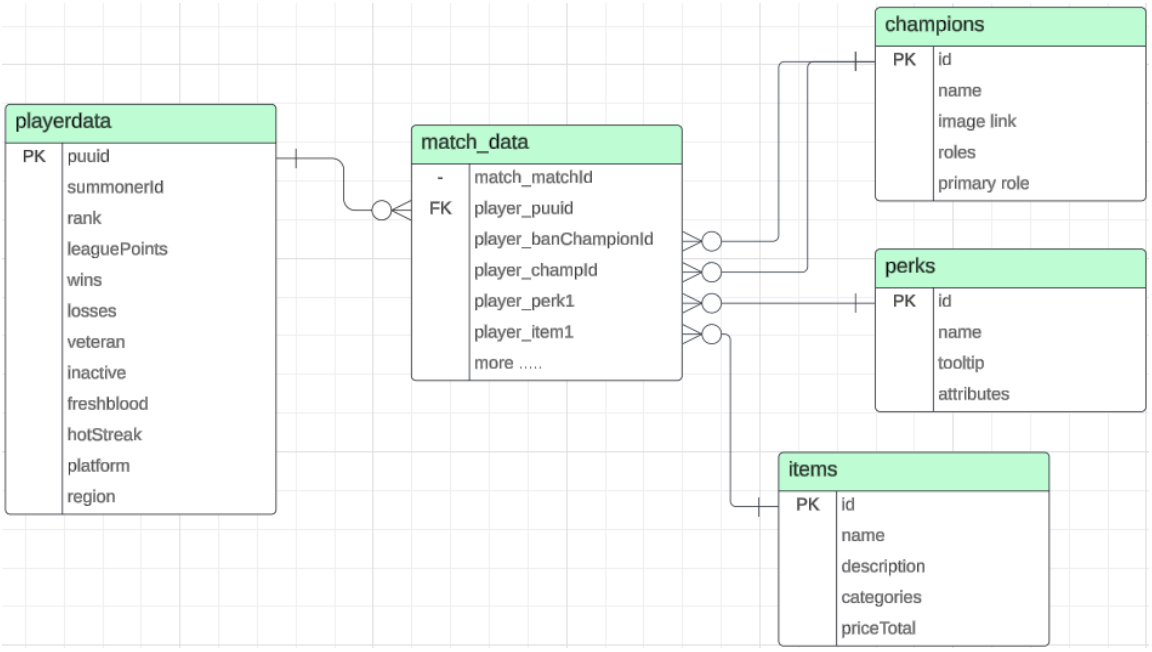
I saved the JSON response as a pandas dataframe. Initially, the dataframe had 18 columns, but many were either similar (e.g., price and priceTotal), sparsely populated (e.g., maxStacks, requiredChampion, requiredAlly, specialRecipe, requiredBuffCurrencyName, requiredBuffCurrencyCost, isEnchantment), or irrelevant to my analysis (e.g., from, to, active, instore). To streamline the data and focus on relevant information, I removed these columns and retained the five essential columns for integration with the match data

Data understanding:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           504 non-null    int64
 1   name         504 non-null    object
 2   description  504 non-null    object
 3   categories   504 non-null    object
 4   priceTotal   504 non-null    int64
dtypes: int64(2), object(3)
memory usage: 19.8+ KB
```

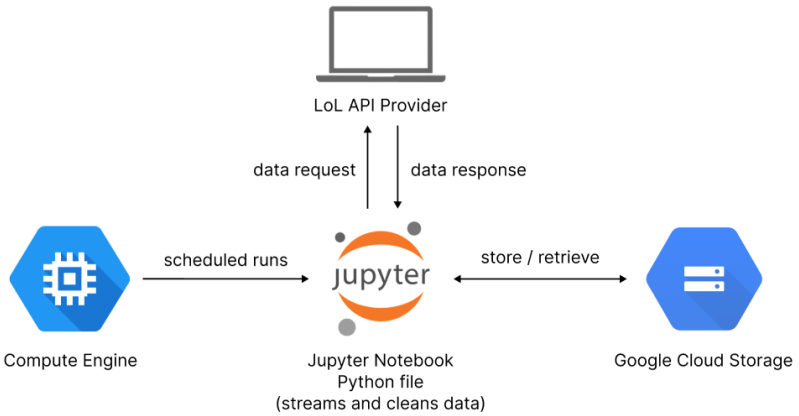
		There is high frequency of items costing below 1000 gold and between 1900 and 3000 gold. Items costing below 1000 gold typically encompasses basic and intermediate items, which players frequently purchase to build towards more powerful late-game items which costs between 1900-3000 gold. This suggests that
		The top 5 most frequently bought items are oracle lens, farsight alteration, doran's blade, stealth ward and plated steelcaps. This suggests players prioritize utility and early-game advantages over raw offensive or defensive stats.

Data Catalogue



Each row in match data corresponds to one row/player in the player data table, while there are multiple rows of match data for each player. For each match/matchid, there are 10 rows of player match performance, 5 from the blue team and 5 from the red team. Additional champions, perks and items information can be found using the common id column in their respective tables.

Data Streaming



I consolidated data processing codes into one jupyter notebook that streams and clean new match data **from the League of Legends API**, **into our shared Google Cloud storage bucket** when executed. Using Google Compute Engine, I set up **VM**, with an attached tmux session so the file could be executed in crontab even if the ssh is closed. The jupyter notebook file is then scheduled to **run daily at 1am, taking about 1hr to finish streaming in new data**.

Data Streaming Video: https://youtu.be/GMTSwFsc93M

This gets up to 10,000 new rows of data each day adding to the previously saved match data.

I retrieve the **top 10 matches from each player** (in the player list), ensuring to update my dataset only with the new matches that don't already exist. As retrieving data takes a while, I am **assuming the most people play is 10 matches in a day**. This process guarantees that my dataset remains current without duplicating previously recorded matches and **quick** (1hr to run finish), **maintaining its accuracy and relevance for ongoing analysis**

Currently: 180,000 rows, 108 columns

Data Modelling

Game Outcome Prediction

	Description	Value
0	Session id	123
1	Target	player_win
2	Target type	Binary
3	Original data shape	(131776, 69)
4	Transformed data shape	(127163, 55)
5	Transformed train set shape	(87630, 55)
6	Transformed test set shape	(39533, 55)
7	Ignore features	15
8	Numeric features	43
9	Rows with missing values	0.0%
10	Preprocess	True
11	Imputation type	simple
12	Numeric imputation	mean
13	Categorical imputation	mode
14	Remove outliers	True
15	Outliers threshold	0.050000
16	Normalize	True
17	Normalize method	zscore
18	Fold Generator	StratifiedKFold
19	Fold Number	10
20	CPU Jobs	-1
21	Use GPU	False
22	Log Experiment	False
23	Experiment Name	clf-default-name
24	USI	cc1f

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.9709	0.9967	0.9748	0.9672	0.9710	0.9417	0.9418	36.0180
lightgbm	Light Gradient Boosting Machine	0.9683	0.9964	0.9723	0.9647	0.9685	0.9366	0.9366	6.7090
gbc	Gradient Boosting Classifier	0.9627	0.9946	0.9647	0.9610	0.9628	0.9255	0.9255	26.1260
rf	Random Forest Classifier	0.9605	0.9936	0.9697	0.9524	0.9609	0.9211	0.9213	9.9630
lr	Logistic Regression	0.9565	0.9918	0.9594	0.9540	0.9567	0.9131	0.9131	1.0400
et	Extra Trees Classifier	0.9559	0.9927	0.9663	0.9466	0.9564	0.9117	0.9119	5.7330
ada	Ada Boost Classifier	0.9547	0.9923	0.9516	0.9577	0.9546	0.9094	0.9095	5.5950
svm	SVM - Linear Kernel	0.9544	0.9904	0.9560	0.9531	0.9545	0.9088	0.9089	0.8540
ridge	Ridge Classifier	0.9527	0.9905	0.9517	0.9537	0.9527	0.9054	0.9055	0.7900
lda	Linear Discriminant Analysis	0.9527	0.9905	0.9517	0.9537	0.9527	0.9054	0.9054	1.2130
dt	Decision Tree Classifier	0.9438	0.9438	0.9421	0.9454	0.9437	0.8876	0.8876	1.6370
knn	K Neighbors Classifier	0.9308	0.9727	0.9332	0.9289	0.9311	0.8617	0.8617	6.2370
nb	Naive Bayes	0.8437	0.9109	0.8557	0.8360	0.8457	0.6875	0.6877	0.7140
qda	Quadratic Discriminant Analysis	0.5638	0.6246	0.4183	0.6674	0.4466	0.1277	0.1581	1.4240
dummy	Dummy Classifier	0.4996	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0690

My first model predicts game outcomes, using PyCaret classification. my **target is game outcome**, and **features are match statistics** like how many opponents did they kill, deaths, assists, damage done**, excluding categorical features** like champion used, items, perks to reduce dimensionality, player lane and positions as all matches have a balanced number so it does not affect win rates, as well as removing outliers and normalizing data.

After comparing models, I chose LightGBM (2nd highest accuracy, time taken: 6s) over CatBoost (highest accuracy, time taken: 36s) due to **its balance of speed and accuracy**.

	Best Hyper-parameters returned	Mean Accuracy
Default	min_child_samples=20, min_split_gain=0.0, n_estimators=100, num_leaves=31, random_state=123, reg_alpha=0.0, reg_lambda=0.0, learning_rate=0.1	0.9683
Tuned 1 (10 iterations)	min_child_samples=66, min_split_gain=0.4, n_estimators=90, num_leaves=90, random_state=123, reg_alpha=0.0005, reg_lambda=0.1, learning_rate=0.1, bagging_fraction=0.6, bagging_freq=6, feature_fraction=0.5	0.9685
Tuned 2 (50 iterations)	min_child_samples=26, min_split_gain=0.6, n_estimators=160, num_leaves=200, random_state=123, reg_alpha=0.2, reg_lambda=0.005, learning_rate=0.05, bagging_fraction=0.9, bagging_freq=0, feature_fraction=0.6,	0.9707

Then I **adjusted the tuning iterations** of the tune_model method in pycaret which returned me different hyper-parameters. The best is at 50 iterations with parameter values like higher bagging and feature fraction that gave the **best mean accuracy of 0.9707**.

Finalised model is **saved as a pkl file** to be used later.

Best Team Compositions

The next model I did was to get the best team champion combinations. Using association rule mining with FP-Growth, I identified the most common team champion **combinations from winning teams** and kept combinations with win rates **above 55%.** This is runned **daily to get the most updated champion combinations**.

	Hyper-parameters	Rules Found
Default	fp-growth: min support = 0.5 (at least 8236 matches) association rules: metric="confidence", min_threshold=0.5	0
Tuned 1	fp-growth: min support = 0.0012 (at least 20 matches) association rules: metric="confidence", min_threshold=0.05	2813
Tuned 2	fp-growth: min support = 0.0012 (at least 20 matches) association rules: metric="support", min_threshold=0.001 (at least 33 matches)	6368

Using the default support (0.5) ensure that the combination occurs in **at least half of the dataset**. However, **as my dataset is big with lots of possible permutations, no rules were found**. I consulted stakeholders who are avid league of legends players, where we determined that the combinations given **using the support metrics where not as accurate as when confidence metrics was used**, as the data was already filtered to only have winning team compositions, so I **should prioritize the combinations based on their predictive power through the confidence metrics**

Champion combinations and win rates are then **saved into a CSV file.**

Best Champion Counters

For counter-pick analysis, I used association rule mining with FP-Growth. I transformed my data into a **2-dimensional array for each unique match**, with (team red combination, team blue combination), **then flattened and one hot encoded the names**. This finds rules that shows the **common counter pick(s) given the opponent picks of all matches**. Win rates is calculated to find which common counters give the best win rates **over 55%.** This is runned **once a week** as compute time is very long.

	Hyper-parameters	Rules Found
Tuned 1	fp-growth: min support = 0.01 (at least 164 matches) association rules: metric="support", min_threshold=0.01 (at least 159 matches)	202
Tuned 2	fp-growth: min support = 0.005 (at least 82 matches) association rules: metric="support", min_threshold=0.001 (at least 80 matches)	894
Tuned 3	fp-growth: min support = 0.001 (at least 16 matches) association rules: metric="support", min_threshold=0.001 (at least 16 matches)	19398
Tuned 4	fp-growth: min support = 0.003 (at least 49 matches) association rules: metric="support", min_threshold=0.001 (at least 48 matches)	2494

For identifying counters, my stakeholder recommended that I find **combinations that happen more frequently over my whole dataset to give a better picture**. Which is why I selected the support metrics to **filter out itemsets that are too infrequent to be considered useful**. Then after iterations of the results with different support threshold, we determined that setting it to at least 49 total matches was able to show the **best observations on counter picks, being not too high or low**.

Champion counter combinations and win rates are then **saved into a CSV file**.

Databricks Dashboards

Data Statistics

This dashboard is for users to know a bit more about league of legends statistics.

Distribution of Player Region (Pie Chart):

We can see that most players in the Challenger League is in Americas, South East Asia, and Europe. Asia has lower number of players as it only includes Korea and Japan. This distribution indicates a strong competitive presence and higher player base in these regions, likely due to larger populations of active players and well-established gaming communities.

Frequency Distribution of Challenger Tier Players Win Rates (Bar Chart):

The graph showing the frequency of win rates among top players, with an average win rate of 0.53-0.6, indicates that even the best players win just over half of their matches. This highlights the competitive nature of the game at the highest level, where skill disparities are minimal, leading to closely contested matches. But also, that it is still **possible to achieve higher win rates** which I will try to identify using their match data.

Game Duration Distribution (Histogram):

The histogram shows that most games tend to last between 20 and 40 minutes, with a peak around the 30-minute mark, indicates a well-paced gameplay. This timeframe suggests that games are typically neither too brief nor excessively extended. The absence of very short games implies that matches are generally competitive and not concluded rapidly due to imbalances or overwhelming

early advantages. Conversely, the lack of unusually long games indicates that matches are resolved before reaching prolonged stalemates or drawn-out engagements. This balanced duration reflects a healthy game environment where teams can effectively execute strategies and adapt to in-game dynamics within a reasonable timeframe, promoting a fair and engaging experience for players.

Top 5 Most Frequent Perks (Bar Chart):

The top 5 perks used are Legend: Alacrity, absorb life, conquer, triumph and fleet footwork. Legend: Alacrity perk was used almost twice as much compared to the 2nd most used perk. This suggests that players prioritize perks that enhance their sustained damage output and survivability. Legend: Alacrity's dominance in usage indicates a strategic emphasis on improving attack speed, crucial for maximizing damage potential and securing kills during gameplay. Absorb life follows closely, highlighting players' interest in sustaining health in combat scenarios

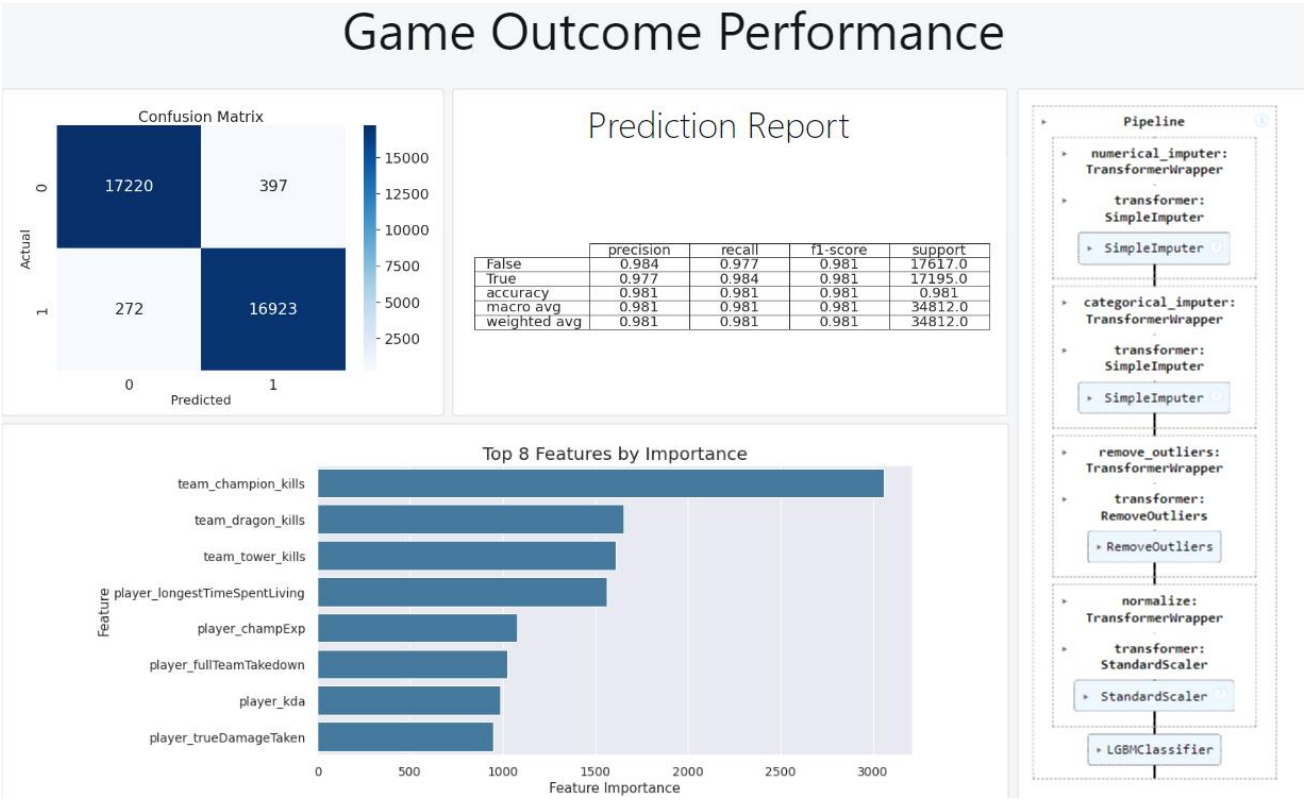
Top 5 Most Frequent Items (Bar Chart):

The top 5 most frequently bought items are oracle lens, farsight alteration, doran's blade, stealth ward and plated steelcaps. Oracle Lens and Farsight Alteration suggest a priority on vision control and map awareness, crucial for gaining information and avoiding ambushes. Doran's Blade and Plated Steelcaps indicate an emphasis on immediate combat effectiveness and survivability, providing a balance between offense and defence. This suggests players prioritize utility and early-game advantages over raw offensive or defensive stats.

Chosen Champion Role Distribution (Histogram):

The most popular champion role chosen was **marksman and fighter**. This suggests that players are inclined towards roles that are pivotal in dealing damage. Since the team positions are balanced and each champion have a recommended position to fill, this means that players prefer roles that allow them to make significant contributions in terms of damage output, often being the primary sources of kills and team fight dominance. Marksmen and fighters are crucial in both early skirmishes and late-game team fights.

Game outcome Performance



Since my data is streamed in, the **number of test data used for evaluation changes**. This allows users to **validate my game outcome prediction model**.

Currently, you can see my model yields **high accuracy of 0.981** and if u look at my **confusion metrics you can see it performed very well, with some wrong predictions**.

The **top features that affected my game outcomes** include total champion kills, tower and dragon kills, true damage taken, champion experience, kills deaths assists. This insight shows that **strategic gameplay** involving these key actions can **directly impact success in the game**, guiding players on where to **prioritize their efforts during matches**.

On the right is my **pycaret model pipeline**, where it handles my data like removing some features, outliers and data normalisation, before passing it to my chosen model, Light Gradient Boosting Machine.

Champion Selection

Champion Selection

Best Team Composition

	Ally Chosen pick	Best Next Ally Pick	win_rate	Count
1	('Bard',)	('DrMundo',)	86.956522	23
2	('Ryze',)	('Ashe',)	86.363636	22
3	('Belveth',)	('Poppy',)	84	25
4	('Alistar', 'Lillia')	('Kaisa',)	77.142857	35
5	('Ezreal', 'Kennen')	('Tristana',)	76	25

Top Played Champions by Role

	Role	Champion	Usage Rate %
1	assassin	Nidalee	16.37
2	fighter	LeeSin	15.6
3	mage	Brand	12.64
4	marksman	Kaisa	35.19
5	support	Rakan	10.66
6	tank	Leona	21.34

Top Champion Counters

	Opponent Chosen pick	Ally Pick	win_rate	count
1	('Kaisa', 'Zeri')	('Corki',)	76.923077	51
2	('Kalista',)	('Ezreal',)	71.666667	60
3	('Kaisa',)	('Ezreal', 'Rell')	71.428571	69
4	('Azir',)	('Viego',)	71.428571	48
5	('Camille',)	('Jax',)	69.230769	91

Top Banned Champions

	Role	Champion	Ban Rate %
1	marksman	Draven	38.69
2	assassin	LeBlanc	35.09
3	support	Pyke	35
4	mage	Karthus	33.23
5	assassin	Nidalee	32.67

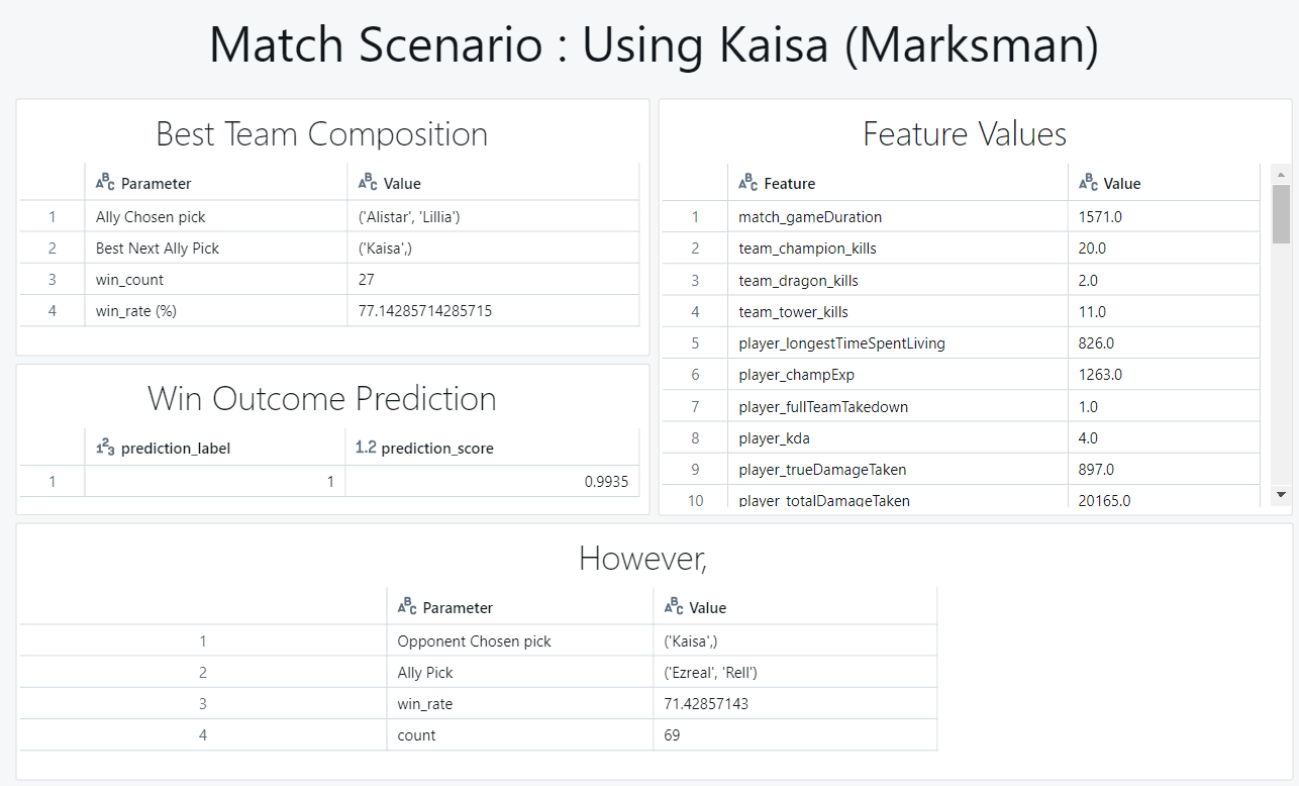
Exploring match attributes reveals a wide array of champion choices. This dashboard highlights **recommended champions, optimal team compositions, and effective counter picks**.

For instance, if your **teammate** selects Bard, **pairing** with DrMundo can lead to a high win rate. Conversely, if the **opponent** picks Kaisa and Zeri, **countering** with Corki is advisable.

Stakeholders can easily identify the most popular and banned champions, **streamlining their decision-making process and enhancing gameplay strategy, especially for those new to the game**.

This dashboard allows all players to **reference** and **help in deciding the optimal champion** for each possible scenario.

Match Scenario



Match Scenario : Using Kaisa (Marksman)

Best Team Composition

	Parameter	Value
1	Ally Chosen pick	('Alistar', 'Lillia')
2	Best Next Ally Pick	('Kaisa',)
3	win_count	27
4	win_rate (%)	77.14285714285715

Win Outcome Prediction

	prediction_label	prediction_score
1	1	0.9935

Feature Values

	Feature	Value
1	match_gameDuration	1571.0
2	team_champion_kills	20.0
3	team_dragon_kills	2.0
4	team_tower_kills	11.0
5	player_longestTimeSpentLiving	826.0
6	player_champExp	1263.0
7	player_fullTeamTakedown	1.0
8	player_kda	4.0
9	player_trueDamageTaken	897.0
10	player_totalDamageTaken	20165.0

However,

	Parameter	Value
1	Opponent Chosen pick	('Kaisa',)
2	Ally Pick	('Ezreal', 'Rell')
3	win_rate	71.42857143
4	count	69

Now using a match scenario to see how to utilise my discoveries. Given that your **teammates** have already chosen Alistar and Lillia, you should pick Kaisa to for a **good team combination** found through my association rule mining. Then I adjusted the **feature values on the right** and passed it to my model, which it **predicted with 99% surety that our team will win**.

However, if your **opponent** chooses champions Ezreal and Rell, which does well against your champion chosen, Kaisa, the **likelihood of your team winning will be lowered**.

Recommendations

From my models, to **increase your win probability**, you should:

Optimise champion chosen based on **ally pick** (choose champion based on best team champion synergy) or **opponent pick** (choose champion based on best opponent champion counters).

You should also **take note** of your champion experience, Kills-Deaths-Assists, total champion kills, damage taken, tower and dragon kills, and stay alive longer!

Problems encountered

Problem	Problem Details	Solution
Streaming	Setting up google cloud vm to run automatically. Jupyter notebook could run manually but not using crontab.	Changing permissions of the notebooks and jupyter library as the user executing in crontab did not have the necessary permissions.
Streaming	The testing Jupyter notebook files was able to run properly, but the actual Jupyter notebook to stream the data didn't finish executing.	VM storage capacity used initially was not enough and had to be upgraded to a more expensive tier.
Streaming	Can't specifically request data from a particular date onwards.	I retrieve the top 10 matches from each player, ensuring to update my dataset only with the new matches that don't already exist. As retrieving data takes a while, I am assuming the most people play 10 matches in a day. This process guarantees that my dataset remains current without duplicating previously recorded matches and quick (1hr to run finish), maintaining its accuracy and relevance for ongoing analysis
Data Modelling	Can't manually add csv files into databricks.	Retrieve from google cloud storage bucket to be used.
Time	Limited data	Tried to retrieve as much match information on first run (about 100,000) rows so that I had a substantial amount of data to work with before the continuous streaming of new data.

Conclusion

Overall, I found that the match attributes and statistics do affect the game outcome. I hope that my models and discovery will help our stakeholders improve game enjoyment.

GitHub Repo: https://github.com/amberyeoe/BDMP-League-of-Legends