TECHNICAL DOCUMENT

Ahmed Kamal El Melegy 201700295

Ahmed Eid Abdullah 201701264

CONTENT

- Data Collection
- Data Analysis
  - Challenges faced & how they were solved
  - Optimizations

Data Collection

This section explains how the data were collected, and any detailed steps were performed within this phase. The tools used are

- RiotWatcher
- Riot Portal API End Points
- League of Graphs/RankDistribuation

RiotWatcher

RiotWatcher is wrapper in top of Riot Games API for league of legends .RiotWatcher environment is python. This documentation functions is to understand the functionality of each property.

Riot Portal API End Points

The official riot portal endpoint is used also to know the all type of data that different endpoints provide. Also it used to generate different api_keys and know the rate limit of requests.

League of Graphs/RankDistribuation

The Rank distribution of "EUNE" region is used to collect summoners names and to get insight about the distribution of summoners over different Ranks and Tier. Also used for our knowledge domain, because we did not know about LOL game before that .

The estimated distribution is

```
 For reference the distribuation of eune region https://www.leagueofgraphs
.com/rankings/rank-distribution/eune

# [Diamond 1.7%     Platinum 12%    Gold 28%    Silver 35%    Bronze 19%
  Iron 2.2%]
# for 20 summoner [1D 2P 5G 7S 4B 1I]
# use this endpoint to get summoner Id, name [https://developer.riotgames.
com/apis#league-v4/GET_getLeagueEntries]
```

20 summoners name was selected based on the previous distribution. For Tier selection, an iterative approach is used over all 5 Tiers to estimate normal distribution of the dataset and to not be condensed on only on rank or tier, which is useful for getting insights from the result after the analysis.

Matches ID collected recursively and the total number of collected matches ID is 71990 matchID, the data then saved at My Drive.

This process is in Big_data_collect.ipynb notebook. Due to rate limit (100 requests in 2 minutes) as stated in the official site, we did not could make the analysis in over all the collected matchesID.

Data Analysis

Initialize Spark Session

First step is initializing spark session and with app name 'lol'

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("lol").getOrCreate()
sc = spark.sparkContext
#matches_data = spark.read.json("drive/MyDrive/one_big", multiLine=False)
```

Then spark is used to read the data from the Google drive as json file, this is efficient than reading a big file as ordinary json file.

```
matches_data = spark.read.json("drive/MyDrive/one_big", multiLine=False)
matchs_num=matches_data.count()
matchs_num
```

Champion win, pick, and ban rates – wpb matrix

```
def wpb_matrix(match):
  champions_rates = []
  for part in match["info"]["participants"]:
    if(part["teamId"] == 100): team_number = 0
    else: team_number = 1
    if match["info"]["teams"][team_number]["win"]:
      stat="Won"
    else:
      stat="Lose"
    champions_rates.append((part["championId"], stat))

  for team in match["info"]["teams"]:
    for ban in team["bans"]:
      if(ban["championId"] != None):
        champions_rates.append((ban["championId"],"Banned"))

  return champions_rates
```

wpb_matrix is function to calculate the win ,pick and ban rates

- ✓ JSON Viewer Pro (Google Chrome Extension is used for graphing a json object for better understating of nested objects in the dataset)

Win-Rate calculation

The function takes a match object and start to access participants and teams in the match , to know which team is won and which lose , “100” means this teams is lose so all its champions is also lose,”200” means this teams is won so all its champions is also won . The format is (ChampionId, WonorLose).

Ban rates calculation

The format is all banned champions is accessed then the format is (ChampionId, Banned)

Win-Ban-Pick calculation - Wpb rates

```
def calculate_rates(static,no_matches):
  banned = static.count("Banned")
  won = static.count("Won")
  lose = static.count("Lose")
  total_played_games = won+lose
  if(total_played_games):
    return [banned/no_matches, total_played_games/no_matches, won/total_pl
ayed_games, lose/total_played_games]
  else:
    return [banned/no_matches, "not_complete", "not_complete"]

def show(name,val,x_name,y_name,tit):
    fig = plt.figure(figsize = (10, 5))

    # creating the bar plot
    plt.bar(name, val, color ='maroon',
            width = 0.4)

    plt.xlabel(x_name)
    plt.ylabel(y_name)
    plt.title(tit)
    plt.show()
```

Calculate_rates take the output from wpb_matrix function and start counting how many a champion is banned, won, lose.

So for each champion

Banned rate =banned times/total matches

Picked rate =total champion played games/total matches

Wining rate=win times /total matches

Note: Consider first time player is playing the game

show function is used to make graphs over all analyses , it simple function take two lists ,names in x-axis, values in y-axis, x-axis label, y-axis label, title of graph

RDD file

```
wpb_champion = matches_data.rdd.flatMap(wpb_matrix)
by_champion = wpb_champion.groupByKey().mapValues(list)
wpb_rates = by_champion.map(lambda x:(x[0], calculate_rates(x[1],matchs_nu
m))).filter(lambda x: x[1][2] is not 'not_complete')
champion_rates = wpb_rates.collect()
```

Using rdd.flatMap to apply the function over all instances of the dataset then groupbykey and sum for each key , pass the second element to calculate function to return the wpb rates

x[0] is the ChampionId, then collect and take only top five champions rates after sorting . I used a dictionary to map from ChampionId to ChampionName instead of accessing ChampionName from the json file, because it is more efficient

the format is (ChampionId,[ban rate ,pick rate ,win rate ,lose rate])

Graphs

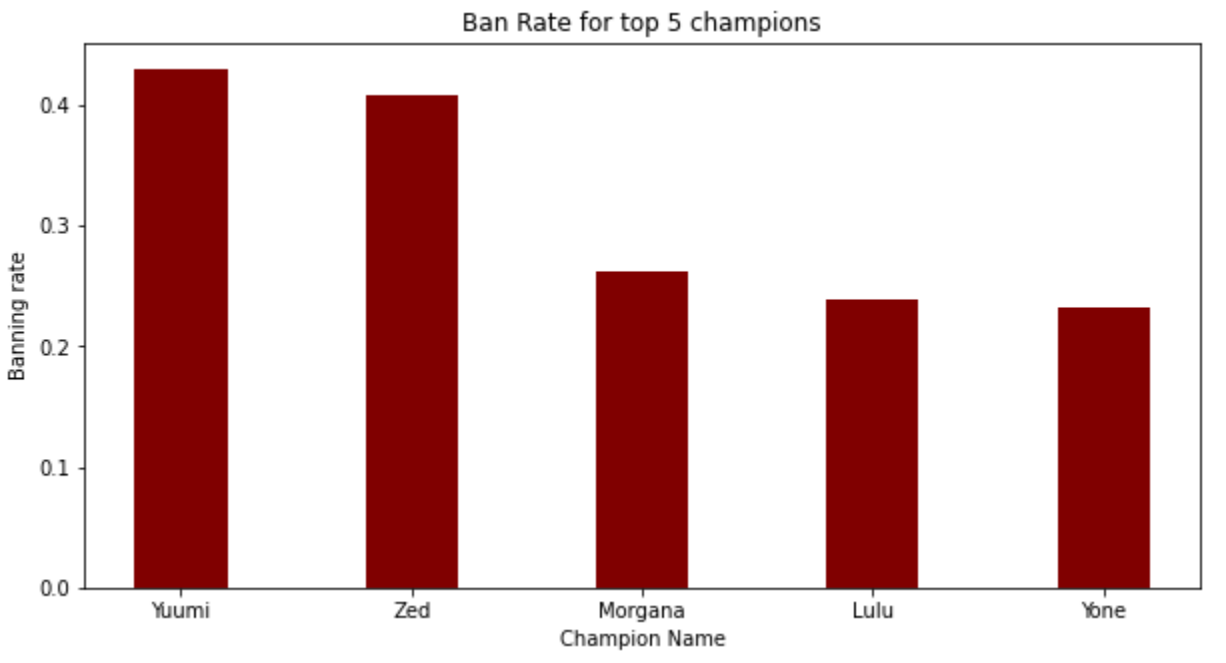
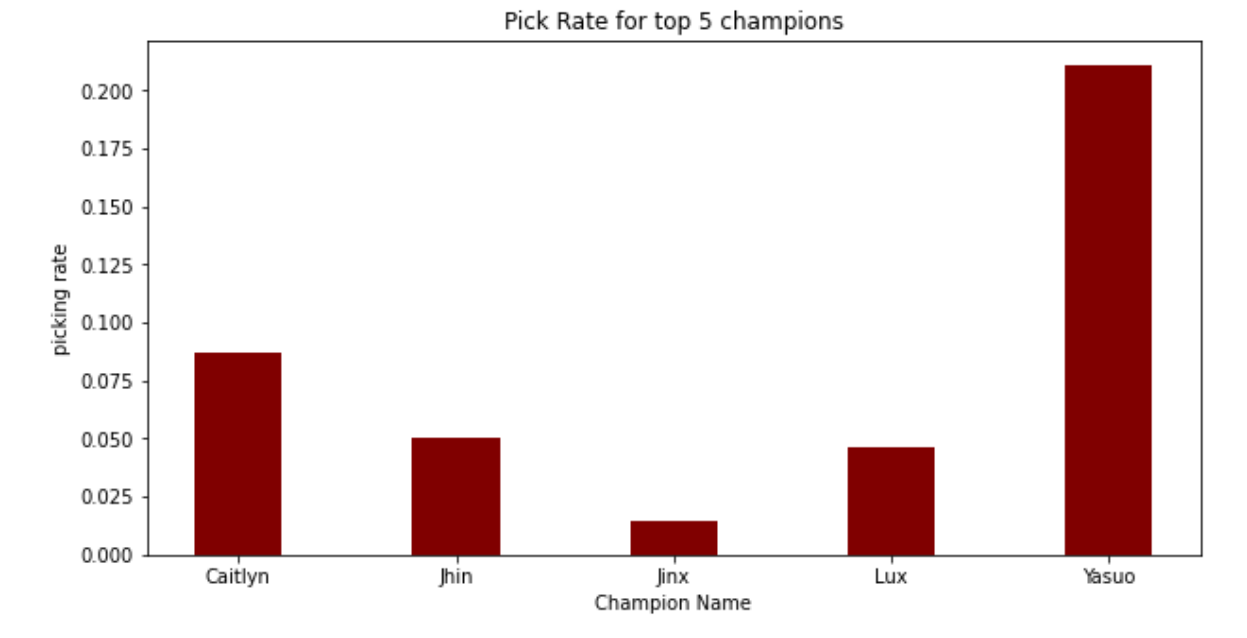
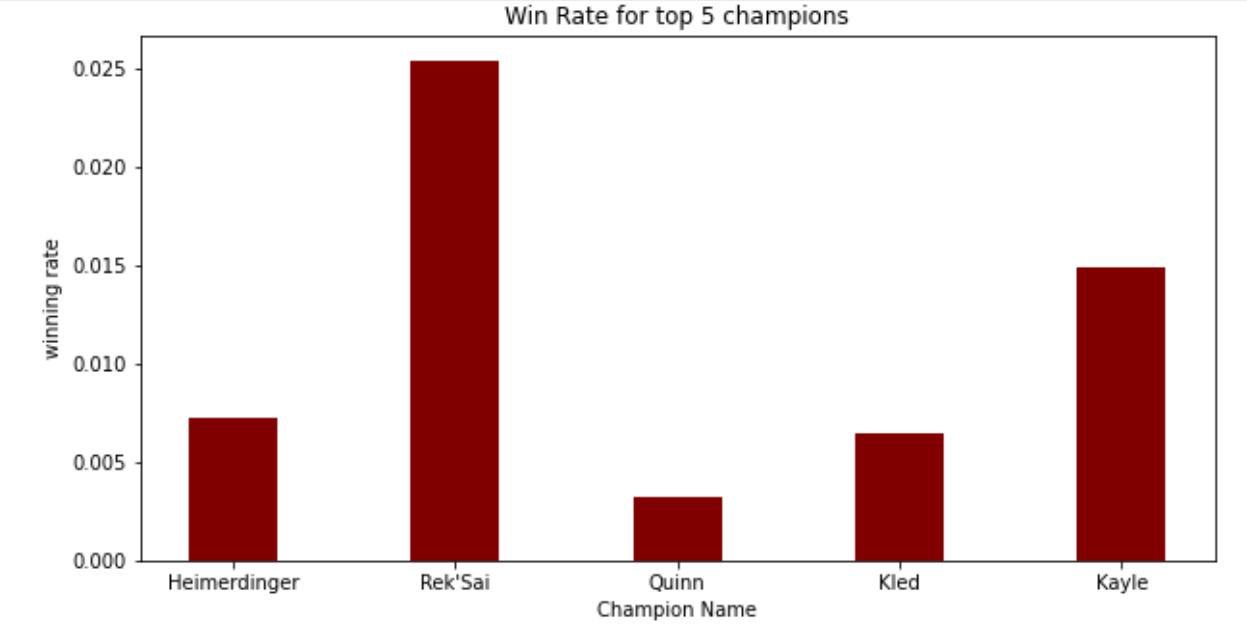


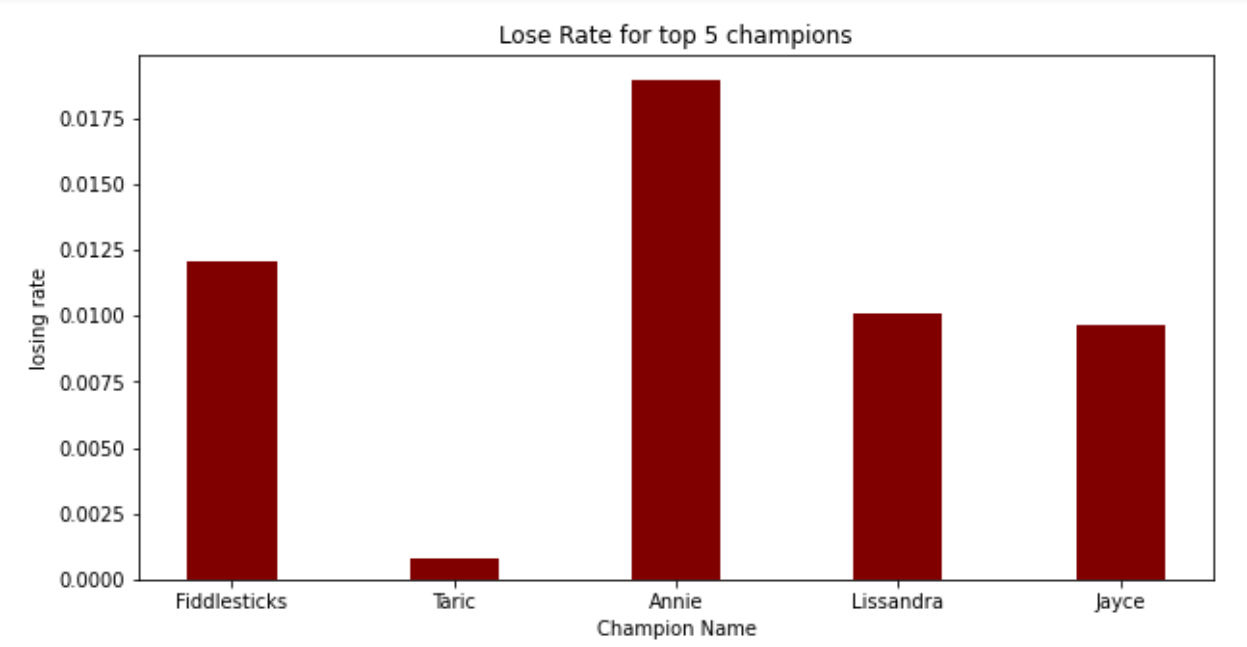
Figure (1) illustrate top 5 champions ban rate



Figure(2) illustrate top 5 champions pick rate



Figure(3)illustrate top 5 champions win rate



Figure(4) illustrate top 5 champions lose rate

Item Win, Pick Rates Computation

Because in the dataset we have not items names, we have only itemid. So we created dictionary from json file contains just item id as key and item name as value

```
import json
#!cp items.json /content/drive/MyDrive
items_id={}
with open('/content/drive/MyDrive/items.json', 'r') as data_file:
    json_data = data_file.read()

data = json.loads(json_data)

for it in data:
  i={it['id']:it['name']}
  items_id.update(i)
```

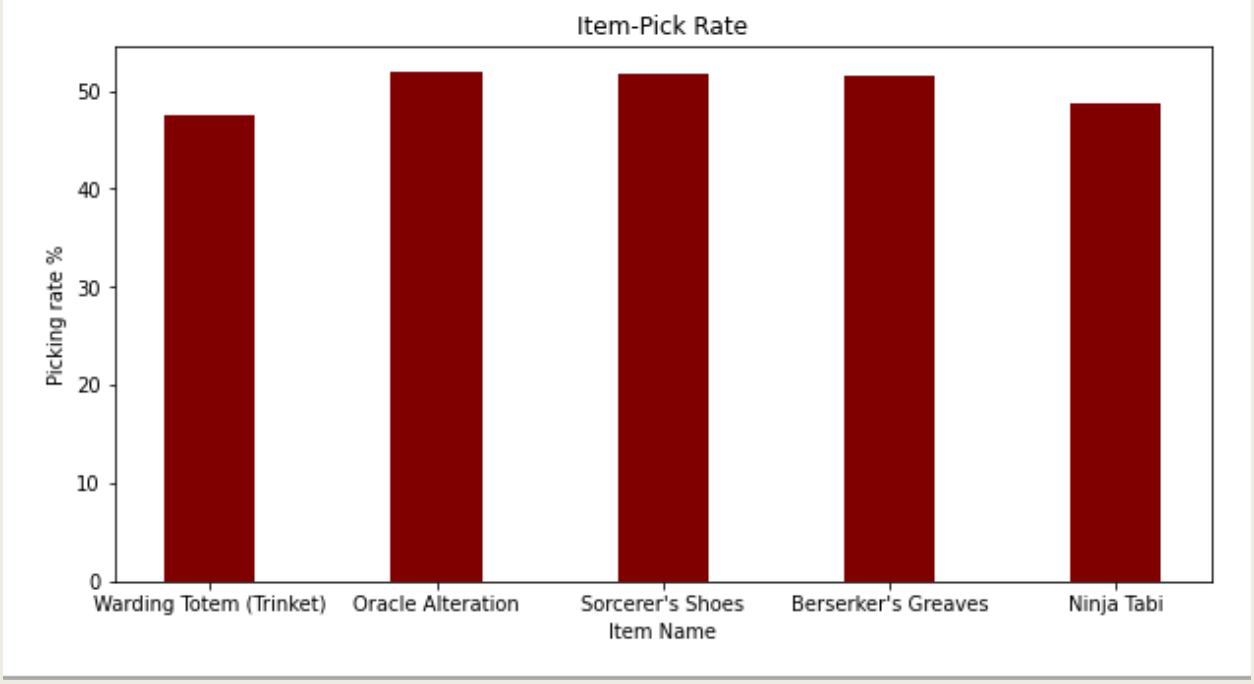
The items files is updated to the last items names and ids , we get it from lol community in Reddit .

```
def item(match):
  '''
  (items(0-6),winorlose)
  '''
  item_win=[]
  for part in match["info"]["participants"]:
    for item_num in range(7):
      item_win.append((part["item"+str(item_num)],part["win"]))
  return item_win

def calculate_item_rates(static,matchs_num):
  win_count = static.count(True)
  lose_count = static.count(False)
  total_played_games = win_count+lose_count
  return [win_count/total_played_games, len(static)/matchs_num]
```

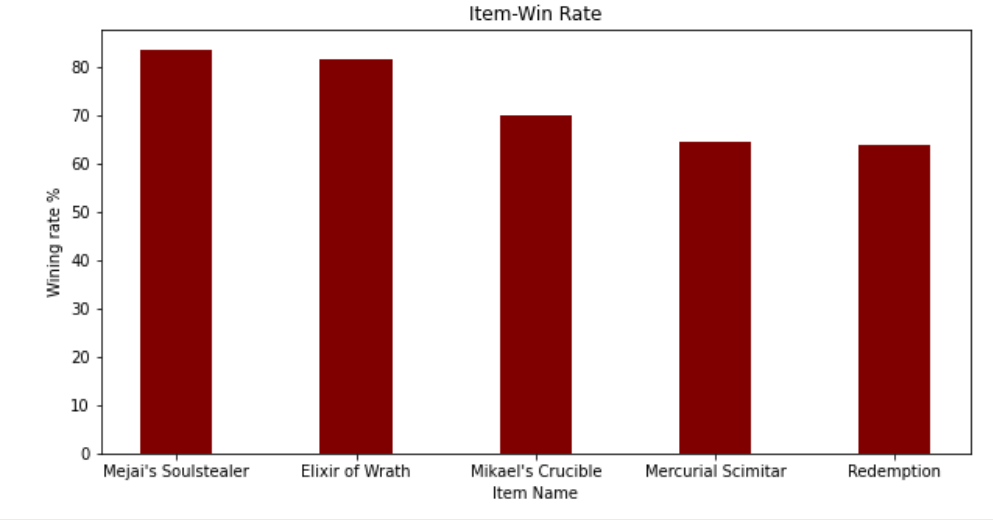
item function is list every item for each champion and if he lose or won in this match(every champion has 6 items in the match)

calculate_item_rates function count every won and lose , then win rate =win count /total played matches

Graphs



Figure(5) illustrate top 5 Item-Pick Rate



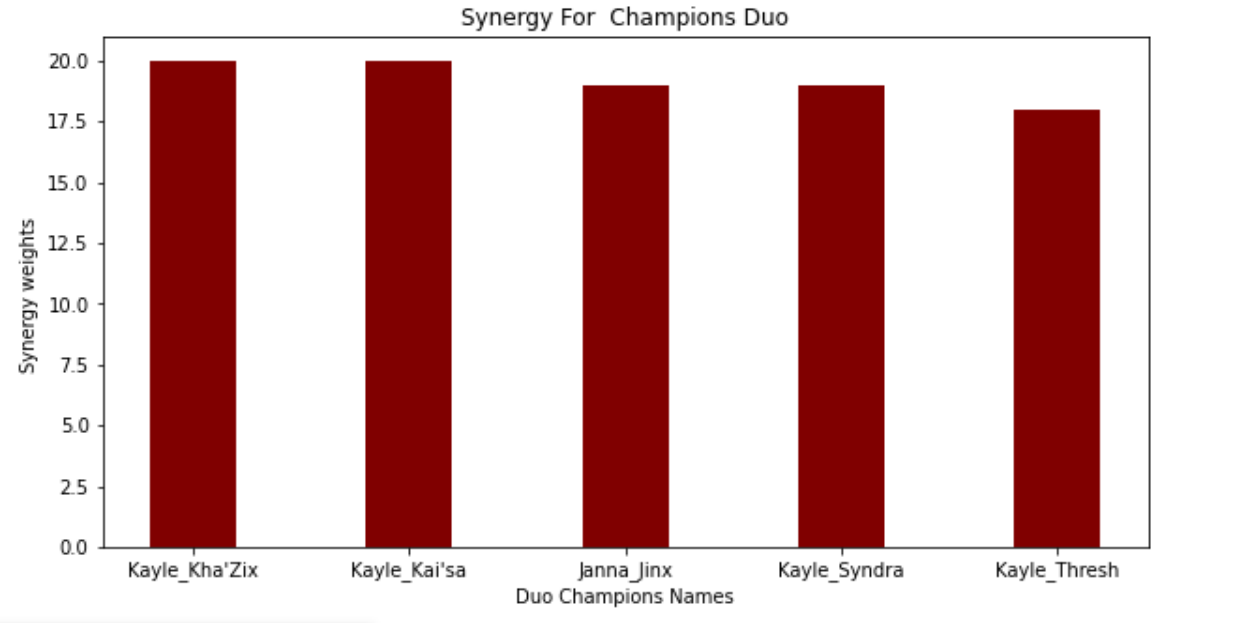
Figure(6) illustrate top 5 Item-Win Rate

Champion Synergies or duos

```
def champion_duos(row):
  duos = []
  team1 = []
  team2 = []
  is_team1_win = False
  is_team2_win = False
  for p in row['info']["participants"]:
    if(p["teamId"] == 100):
      team_number = 0
    else:
      team_number = 1
    if(team_number == 0):
      team1.append(p["championId"])
      is_team1_win = p["win"]
    else:
      team2.append(p["championId"])
      is_team2_win = p["win"]
  team1 = sorted(team1)
  team2 = sorted(team2)
  for c in list(itertools.combinations(team1, 2)):
    if is_team1_win:
      duos.append((c, 1))
    else:
      duos.append((c, 0))

  for c in list(itertools.combinations(team2, 2)):
    if is_team2_win:
      duos.append((c, 1))
    else:
      duos.append((c, 0))
  return duos[0:5]
```

champions_duos function takes a match and start look for each participants if his teams Id is 100 means he lose if his team Id is 200 means he won, then for each combination between all champions in one game add weight of 1 if they the team who won , else add 0 weight . Also for team two (if these two champion won the game with together add weight of 1 , if they lose together add 0)

Graphs



Figure(7) illustrate estimated Synergy weights won game for champions duos for top 5 champions

Item Synergies (item with champion, item with class)

With Champions .First I read json file contains all champions with its classes , because the class of the champion was not stated in the match object then we make a dictionary contains every champion with its possible classes . We get the file from community lol related.

```
with open('/content/drive/MyDrive/champion_class.json') as f:
  data=json.load(f)
tags_id={}

for it in data:
  i={it['championId']:it['tags']}
  tags_id.update(i)
```

```
def item_syn(match):
  item_win=[]
  for part in match["info"]["participants"]:
    for item_num in range(7):
      item_win.append((part["item"+str(item_num)],(part["win"],part["champ
ionId"],tags_id.get(part["championName"]))))
  return item_win
```

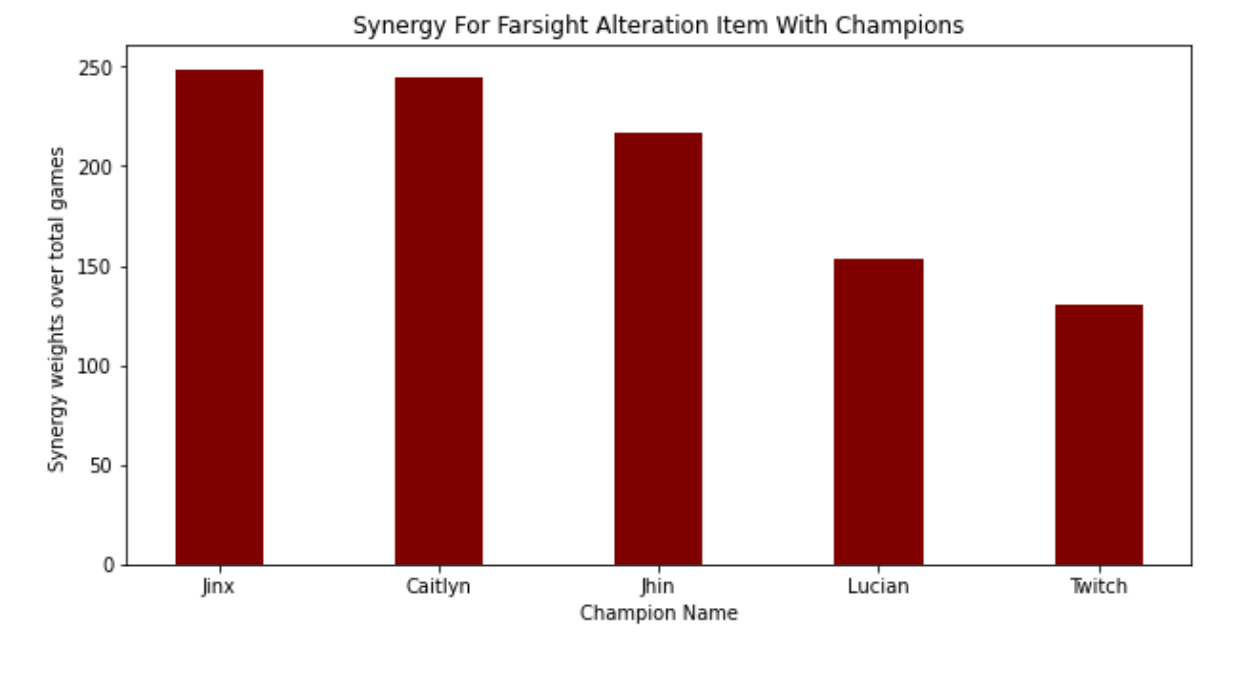
item_syn function takes the match and for each participant record all 6 items, wonorlose ,ChampionId, ChampionName.

```
def champion_item_synergies(static):
  champion_item_syn_rate= {} # championId : item syngery rate
  win_total=static[1][0].count(True)
  lose_total=static[1][0].count(False)
  total_games=win_total+lose_total
  for stats in static[1]:
    championId=stats[1]
    ishewon=stats[0]
    if(championId in champion_item_syn_rate): #if this champion played a m
atch already +0.5
      champion_item_syn_rate[championId] += 0.5
    else:
      champion_item_syn_rate[championId] = 0.5   # if this is the first ma
tch for this player 0.5
    if(ishewon == True):
            champion_item_syn_rate[championId] += 1   # if ths player won
the match 1

  item_synergies = []
  for championId, item_syn_rate in sorted(champion_item_syn_rate.items(),
key=lambda x:x[1], reverse=True):
    item_synergies.append((championId, (item_syn_rate/total_games)))
  return  item_synergies[0:5]
```

champion_item_synergies function add weight of 0.5 if champion owed before this items , set 0.5 if he never owed this item , add 1 if he owed this item and won the match .Format (ChampionId, item synergy weights / total games)

Here I divided it by total games to get as rate, it should by only weight, but the I realized it will be the same percentage for all champions.

Graph



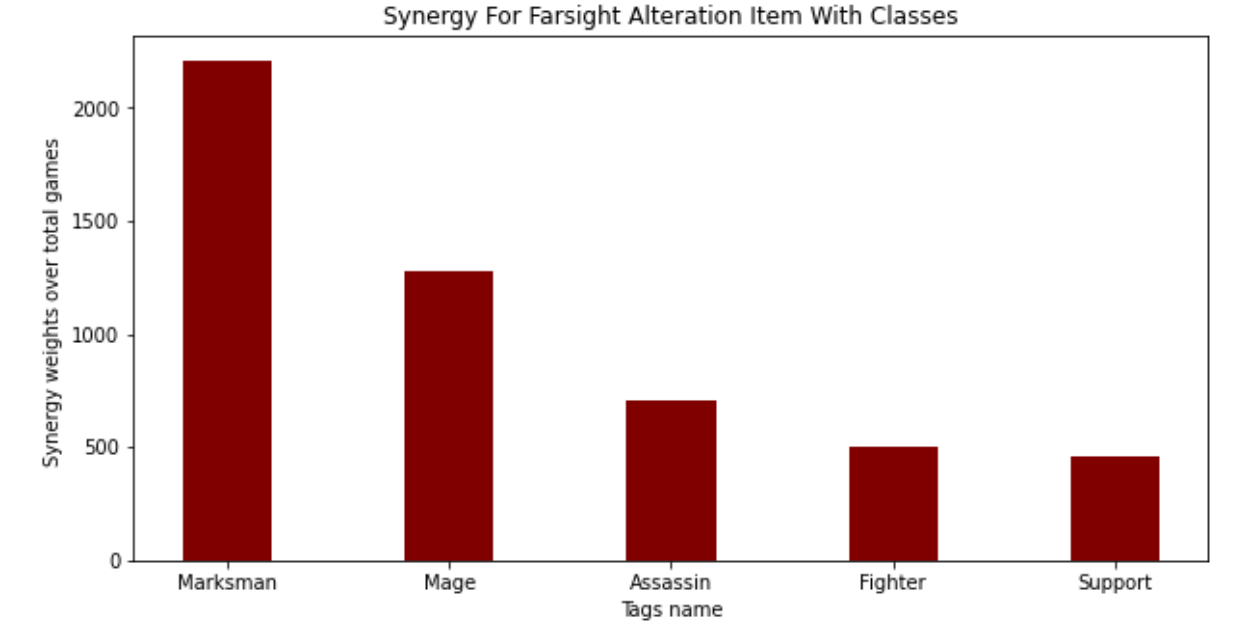
Figure(8)illustrate Synergy of the passed Farsight Alteration item with champions for top 5 champions

Here the graph is different because we pass a specific item to plot it with top 5 champions.

with class

Same steps as previous but instead of item synergy with champion it will with class .we already get the classes as stated above.

Graph



Figure(9) illustrate Synergy for the passed Farsight Alteration items with classes for the top 5 classes .

Item Suggestion

Before we start at this process we loaded a data dragon file contains all info about items such as price etc.

```
with open('/content/items_info') as f:
  data_dragon=json.load(f)
```

Then

```
def item_sugg(match):
  '''
  (items(0-6),winorlose)
  '''
  item_win=[]
  for part in match["info"]["participants"]:
    for item_num in range(7):
      try :
        id=str(part["item"+str(item_num)])
        if  (data_dragon['data'][id]['gold']['total'] >= 1000):
```

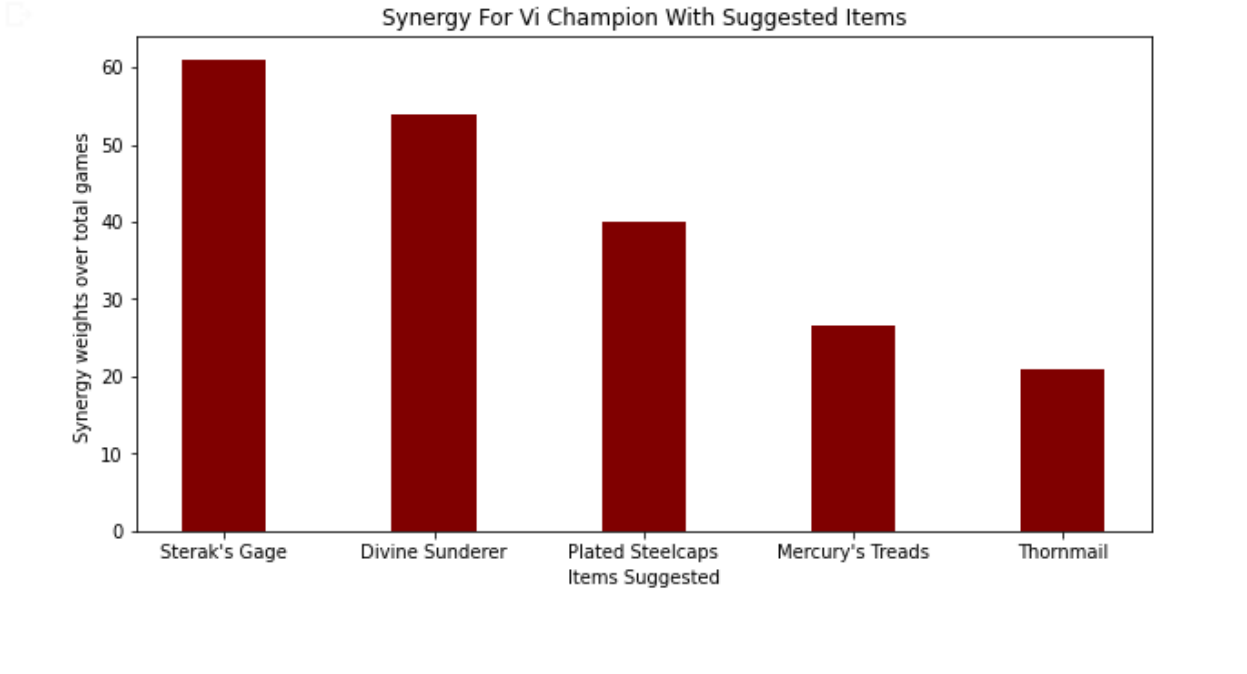
```
          item_win.append((part['championId'],(data_dragon['data'][id]['na
me'],part["win"])))
      except:
        pass
  return item_win

def champion_item_suggest(row):
  item_dict= {}
  for val in row[1]:
    if(val[0] in item_dict):
      item_dict[val[0]] += 0.5
    else:
      item_dict[val[0]] = 0.5

    if(val[0] == True):
            item_dict [val[0]] += 1

  c = []
  for k, r in sorted(item_dict.items(), key=lambda x:x[1], reverse=True):
    c.append((k, r))


  return (row[0], (c[:5]))
```

item_sugg function pass over all participants and for each item for participant record the (champion id, itemName, wonorlose) if and only if this item price is greater than 1000 (this data are gotten from the data dragon json), because we will not offer basic or starter item to champion with high win rate (not logically), so we will offer legendary items to champions wining rate with this item .

champion_item_suggest function will add 1 if the champion with this item has won the game , add 0.5 if the champion with this item lose the game , set to 0.5 if the champion with this item first time to play .

Graph

```
[ ]  item_suggest('Vi')
```



Figure(10) illustrate item suggested for the passed champion Vi for the top 5 items synergy weight

Item_suggest function takes any champion and return the suggested items of the passed champion