Riot_Games_API

March 9, 2024

[343]:
```
import requests
import re
from collections import Counter
from collections import defaultdict
```

Remember! Do not share any of your API keys with anyone.

[312]:
```
API_KEY = 'RGAPI-404ec425-3d3b-43f1-abb9-c13c74878967'
```

We will first acquire summoner puuid

[314]:
```
def get_summoner_data(summoner_name):
    url = f'https://eun1.api.riotgames.com/tft/summoner/v1/summoners/by-name/
 ↪{summoner_name}'

    headers = {'X-Riot-Token': API_KEY}

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        return response.json()
    else:
        print('Error:', response.status_code)
        return None

summoner_data = get_summoner_data('Ariel%20Ibagaza%207')
summoner_data
```

[314]:
```
{'id': 'qohtcYP9g75thCs2eWQcywGHry6NbwH7FwzAoBT2Qkh1fX8',
 'accountId': 'yjcPqMaOSPuEs8PVOJ5L14_bZ6wCcSWmdYmdIpNp4xQOiA',
 'puuid': 'mn2BbUNLWkpzuHpSMRRbO2gC_3KbwxqXtLljkwl-I6SCLkYV5icjPl_oMm_rI9OQlTd-
LjNGk6ZVlQ',
 'name': 'Ariel Ibagaza 7',
 'profileIconId': 1665,
 'revisionDate': 1709841857000,
 'summonerLevel': 121}
```

[315]:
```
puuid = summoner_data['puuid']
```

Then the puuid will be used to return a list of matches

[377]:
```
def get_match_ids(puuid):
    url = f'https://europe.api.riotgames.com/tft/match/v1/matches/by-puuid/
 ↪{puuid}/ids?start=0&count=20'
    headers = {'X-Riot-Token': API_KEY}

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        return response.json()
    else:
        print('Error:', response.status_code)
        return None

get_match_ids(puuid)[:2]
```

[377]:
```
['EUN1_3562401510', 'EUN1_3562059367']
```

[232]:
```
match_ids = get_match_ids(puuid)
match_id = get_match_ids(puuid)[0]
```

We will use my last game as an example here and save it into match_id to get the specific data from that match

[386]:
```
def get_match_data(matchid):
    url = f'https://europe.api.riotgames.com/tft/match/v1/matches/{matchid}'

    headers = {
        'X-Riot-Token': API_KEY
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        return response.json()
    else:
        print('Error:', response.status_code)
        print('Response Content:', response.text)
        return None

get_match_data(match_id) ; #Hide large output for pdf format
```

Now we will explore through the json file and the https://developer.riotgames.com/apis#tft-match-v1/GET_getMatch website, to acquire the data that we need for this project

[220]:
```
get_match_data(match_id)['info'].keys()
```

[220]:
```
dict_keys(['endOfGameResult', 'gameCreation', 'gameId', 'game_datetime',
'game_length', 'game_version', 'mapId', 'participants', 'queueId', 'queue_id',
'tft_game_type', 'tft_set_core_name', 'tft_set_number'])
```

[387]:
```
get_match_data(match_id)['info'] ; #Hiding output here too, output is too long␣
↪for pdf format
```

The problem here is that there are 7 participants and we can't manually search for the required username everytime

[388]:
```
participant = next((p for p in get_match_data(match_id)['info']['participants']␣
↪if p['puuid'] == summoner_data['puuid']), None)
participant ;
```

[353]:
```
traits_head = participant['traits'][:2]
traits_head
```

[353]:
```
[{'name': 'Set10_8Bit',
  'num_units': 6,
  'style': 3,
  'tier_current': 3,
  'tier_total': 3},
 {'name': 'Set10_CrowdDive',
  'num_units': 1,
  'style': 0,
  'tier_current': 0,
  'tier_total': 3}]
```

As we can see there are traits with tier_curret = 0, meaning that they are not active, for example for 8 Bit and Crowd Diver

[391]:
```
print(participant['traits'][0]['name'],",", participant['traits'][1]['name'])
```

```
Set10_8Bit , Set10_CrowdDive
```

[363]:
```
print("Is 8 Bit active?", participant['traits'][0]['tier_current'] > 1,
      "\nIs Crowd Diver active ?", participant['traits'][1]['tier_current'] > 1)
```

```
Is 8 Bit active? True
Is Crowd Diver active ? False
```

[223]:
```
participant['augments']
```

[223]:
```
['TFT10_Augment_SticksAndStones',
 'TFT9_Augment_YouHaveMySword',
 'TFT9_Augment_HedgeFundPlusPlus']
```

[224]:
```
new_augments = [' '.join(re.findall('[A-Z][^A-Z]*', augment.split('_', 2)[-1]))␣
↪for augment in participant['augments']]
new_augments
```

[224]:
```
['Sticks And Stones', 'You Have My Sword', 'Hedge Fund Plus Plus']
```

[225]:
```
participant['placement']
```

[225]:
```
1
```

[226]:
```
units = []

for unit in participant['units']:
    character_id = unit['character_id']
    underscore_index = character_id.find('_')
    if underscore_index != -1:
        character_id = character_id[underscore_index + 1:]
    units.append(character_id)
```

[226]:
```
['Corki',
 'KSante',
 'Garen',
 'Mordekaiser',
 'Ekko',
 'Riven',
 'Caitlyn',
 'Blitzcrank',
 'Qiyana',
 'Kayn',
 'Sona']
```

[364]:
```
tiers = []

for unit in participant['units']:
    tier = unit.get('tier', None)
    if tier is not None:
        tiers.append(tier)

print(tiers)
```

```
[1, 2, 1, 2, 2, 2, 3, 1, 1, 2, 2]
```

[393]:
```
participant['units'][5]['itemNames']
```

[393]:
```
['TFT_Item_UnstableConcoction',
 'TFT_Item_SteraksGage',
 'TFT_Item_MadredsBloodrazor']
```

[375]:
```
items = []

for unit in participant['units']:
    if 'itemNames' in unit:
```

```
        item_names_list = unit['itemNames']
        stripped_item_names = []
        for item in item_names_list:
            stripped_item_name = item.split('_', 2)[-1]
            formatted_item_name = ' '.join(re.findall(r'\d+|[a-zA-Z][a-z]*',␣
 ↪stripped_item_name))
            stripped_item_names.append(formatted_item_name)
        items.append(stripped_item_names)

print(items)
```

```
[[], [], [], [], ['Warmogs Armor'], ['Unstable Concoction', 'Steraks Gage',
'Madreds Bloodrazor'], ['Infinity Edge', 'Infinity Edge', 'Spear Of Shojin'],
[], ['8 bit Emblem', 'Statikk Shiv'], [], []]
```

After hours of coding we have the completed function chaining to get as the last game using any player's username NOTE: this is only for Europe region, we could pass an aditional argument to the functions if we want to specify the region we want. (There are more things that can be done here, for example to get data for more games if we add a variable number of games and change the code a little bit)

[321]:
```
BASE_URL = 'https://europe.api.riotgames.com'

def automate_process(summoner_name):
    summoner_data = get_summoner_data(summoner_name)
    if summoner_data:
        puuid = summoner_data['puuid']
        match_ids = get_match_ids(puuid)
        if match_ids:
            match_id = match_ids[0]
            match_data = get_match_data(match_id)
            if match_data:
                participant = next((p for p in␣
 ↪match_data['info']['participants'] if p['puuid'] == puuid), None)
                if participant:
                    new_augments = [' '.join(re.findall('[A-Z][^A-Z]*', augment.
 ↪split('_', 2)[-1])) for augment in participant['augments']]
                    placement = participant['placement']
                    units = [unit['character_id'].split('_')[-1] for unit in␣
 ↪participant['units']]
                    tiers = [unit.get('tier', None) for unit in␣
 ↪participant['units']]
                    items = []
                    for unit in participant['units'][:10]:
                        if 'itemNames' in unit:
                            item_names_list = unit['itemNames']
                            stripped_item_names = []
                            for item in item_names_list:
```

```
                                stripped_item_name = item.split('_', 2)[-1]
                                formatted_item_name = ' '.join(re.
↪findall(r'\d+|[a-zA-Z][a-z]*', stripped_item_name))
                                stripped_item_names.append(formatted_item_name)
                            items.append(stripped_item_names)
                    # Extract active traits and their tiers
                    active_traits = []
                    trait_tiers = []
                    for trait in participant['traits']:
                        if trait.get('tier_current', 0) > 0:
                            trait_name = trait['name']
                            stripped_trait_name = trait_name.split('_')[-1]
                            active_traits.append(stripped_trait_name)
                            trait_tiers.append(trait['tier_current'])

                    return {
                        'new_augments': new_augments,
                        'placement': placement,
                        'units': units,
                        'tiers': tiers,
                        'items': items,
                        'active_traits': active_traits,
                        'trait_tiers': trait_tiers
                    }
                #Some debugging code
                else:
                    print("Participant data not found.")
                    return None
            else:
                print("Match data not found.")
                return None
        else:
            print("Match IDs not found.")
            return None
    else:
        print("Summoner data not found.")
        return None

def format_output(result, summoner_name, match_id):
    if result:
        summoner_name_formatted = summoner_name.replace('%20', ' ')
        new_augments = ', '.join(result['new_augments'])
        placement = result['placement']
        units = result['units']
        tiers = result['tiers']
        items = result['items']
        active_traits = result['active_traits']
```

```
        trait_tiers = result['trait_tiers']

        num_items = sum(len(items) for items in result['items'])

        formatted_placement = "{0}{1}".format(placement, "tsnrhtdd"[((placement␣
↪// 10 % 10 != 1) * (placement % 10 < 4) * placement % 10)::4])

        formatted_output = f"{summoner_name_formatted} in game with match_id␣
↪{match_id} placed {formatted_placement} using {new_augments} augments.\n"

        sorted_active_traits = [trait for _, trait in sorted(zip(trait_tiers,␣
↪active_traits), reverse=True)]

        formatted_output += "Active Traits: " + ', '.join(f"{trait} (Tier␣
↪{tier})" for trait, tier in zip(sorted_active_traits, sorted(trait_tiers,␣
↪reverse=True))) + "\n"

        for i, (unit, tier) in enumerate(zip(units, tiers)):
            champion = unit
            item_list = items[i] if i < len(items) else []
            formatted_output += f"Champion {champion} (Tier {tier}) equipped␣
↪with items: {', '.join(item_list)}\n"

        return formatted_output
    else:
        return "No data available."


summoner_name = input("Enter Summoner Name: ")
result = automate_process(summoner_name)
match_ids = get_match_ids(get_summoner_data(summoner_name)['puuid'])
match_id = match_ids[0] if match_ids else None

formatted_output = format_output(result, summoner_name, match_id)
print(formatted_output)
```

```
Enter Summoner Name: Ariel%20Ibagaza%207
Ariel Ibagaza 7 in game with match_id EUN1_3562401510 placed 1st using Sticks
And Stones, You Have My Sword, Hedge Fund Plus Plus augments.
Active Traits: Sentinel (Tier 3), 8Bit (Tier 3), TwoSides (Tier 1), TrueDamage
(Tier 1), Quickshot (Tier 1), DJ (Tier 1)
Champion Corki (Tier 1) equipped with items:
Champion KSante (Tier 2) equipped with items:
Champion Garen (Tier 1) equipped with items:
Champion Mordekaiser (Tier 2) equipped with items:
Champion Ekko (Tier 2) equipped with items: Warmogs Armor
Champion Riven (Tier 2) equipped with items: Unstable Concoction, Steraks Gage,
```

```
Madreds Bloodrazor
Champion Caitlyn (Tier 3) equipped with items: Infinity Edge, Infinity Edge,
Spear Of Shojin
Champion Blitzcrank (Tier 1) equipped with items:
Champion Qiyana (Tier 1) equipped with items: 8 bit Emblem, Statikk Shiv
Champion Kayn (Tier 2) equipped with items:
Champion Sona (Tier 2) equipped with items:
```

Now we will modify the get_match_ids functions to include how many matches we want to be retrieved from the RIOT GAMES API and return the average placement for a player in X games

[323]:
```python
def get_match_ids(puuid, count):
    url = f'{BASE_URL}/tft/match/v1/matches/by-puuid/{puuid}/ids?
 ↪start=0&count={count}'
    headers = {'X-Riot-Token': API_KEY}
    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        return response.json()
    else:
        print('Error:', response.status_code)
        return None

def average_placement(summoner_name, num_games):
    total_placement = 0
    games_counted = 0

    summoner_data = get_summoner_data(summoner_name)
    if summoner_data:
        puuid = summoner_data['puuid']
        match_ids = get_match_ids(puuid, num_games)
        if match_ids:
            for match_id in match_ids[:num_games]:
                match_data = get_match_data(match_id)
                if match_data:
                    participant = next((p for p in␣
 ↪match_data['info']['participants'] if p['puuid'] == puuid), None)
                    if participant:
                        placement = participant['placement']
                        total_placement += placement
                        games_counted += 1

    if games_counted > 0:
        return total_placement / games_counted
    else:
        return None
```

```
summoner_name = input("Enter Summoner Name: ").replace(" ", "%20")
num_games = int(input("Enter number of games to calculate average placement: "))
average_place = average_placement(summoner_name, num_games)
if average_place:
    print(f"Average placement over {num_games} games for {summoner_name.
 ↪replace('%20', ' ')} is {average_place}.")
else:
    print("No data available for the specified summoner and number of games.")
```

```
Enter Summoner Name: Ariel%20Ibagaza%207
Enter number of games to calculate average placement: 50
Average placement over 50 games for Ariel Ibagaza 7 is 4.56.
```

We will make another function to return the most played units in X games

[329]:
```
def get_most_played_units(summoner_name, num_games):
    units_counter = Counter()

    summoner_data = get_summoner_data(summoner_name)
    if summoner_data:
        puuid = summoner_data['puuid']
        match_ids = get_match_ids(puuid, num_games)
        if match_ids:
            for match_id in match_ids[:num_games]:
                match_data = get_match_data(match_id)
                if match_data:
                    participant = next((p for p in␣
 ↪match_data['info']['participants'] if p['puuid'] == puuid), None)
                    if participant:
                        units = [unit['character_id'].split('_')[-1] for unit␣
 ↪in participant['units']]
                        units_counter.update(units)

    return units_counter.most_common(10)

summoner_name = input("Enter Summoner Name: ")
num_games = int(input("Enter number of games to analyze: "))
most_played_units = get_most_played_units(summoner_name, num_games)
print(f"Top 10 most played units in {num_games} games:")
for unit, count in most_played_units:
    print(f"{unit}: {count} times")
```

```
Enter Summoner Name: Ariel%20Ibagaza%207
Enter number of games to analyze: 25
Top 10 most played units in 25 games:
Mordekaiser: 15 times
Riven: 9 times
Sett: 9 times
```

```
Kayle: 9 times
Ekko: 8 times
Garen: 7 times
Thresh: 7 times
Caitlyn: 6 times
Yone: 6 times
Viego: 6 times
```

Finally, we will create a function that returns the top 10 highest win rate champions for a player, the problem here is that some units were played 1 time with 1 win, averaging an 100% winrate, this is problematic, so we will only count units with >= 4 games

[342]:
```
def get_champion_win_rates(summoner_name, num_games):
    champion_stats = defaultdict(lambda: {'wins': 0, 'total': 0})

    summoner_data = get_summoner_data(summoner_name)
    if summoner_data:
        puuid = summoner_data['puuid']
        match_ids = get_match_ids(puuid, num_games)
        if match_ids:
            for match_id in match_ids[:num_games]:
                match_data = get_match_data(match_id)
                if match_data:
                    participant = next((p for p in␣
↪match_data['info']['participants'] if p['puuid'] == puuid), None)
                    if participant:
                        for unit in participant['units']:
                            champion_name = unit['character_id'].split('_')[-1]
                            placement = participant['placement']
                            champion_stats[champion_name]['total'] += 1
                            if placement == 1:
                                champion_stats[champion_name]['wins'] += 1

    # Win rates for each champion
    champion_win_rates = {}
    for champion_name, stats in champion_stats.items():
        wins = stats['wins']
        total = stats['total']
        if total >=4:  # Check if the champion has been played in at least 4␣
↪games
            win_rate = wins / total
        else:
            win_rate = 0
        champion_win_rates[champion_name] = win_rate

    # Top-performing champions based on win rates
    return sorted(champion_win_rates.items(), key=lambda x: x[1],␣
↪reverse=True)[:10]
```

```
summoner_name = input("Enter Summoner Name: ").replace(" ", "%20")
num_games = int(input("Enter number of games to analyze: "))
top_champions = get_champion_win_rates(summoner_name, num_games)
print(f"Top 10 champions with the highest win rates for {summoner_name}:")
for champion_name, win_rate in top_champions:
    print(f"{champion_name}: {win_rate:.2%}")
```

```
Enter Summoner Name: Ariel%20Ibagaza%207
Enter number of games to analyze: 30
Top 10 champions with the highest win rates for Ariel%20Ibagaza%207:
Caitlyn: 83.33%
Kennen: 57.14%
Qiyana: 50.00%
Senna: 50.00%
Garen: 44.44%
Thresh: 42.86%
Riven: 40.00%
Amumu: 40.00%
Mordekaiser: 38.89%
Blitzcrank: 37.50%
```