Construcción de la base de datos en Python

La base de datos que se ha utilizado para este proyecto proviene de la API pública de Riot Games. Para realizar consultas a través de la API se ha utilizado el framework "Cassiopeia". Cassiopeia extiende la funcionalidad de un wrapper tradicional para facilitar el trabajo de consulta de manera intuitiva organizando las peticiones por clases.

Para obtener acceso a la API de Riot Games se requiere de una cuenta de desarrollador a la que se vincula una clave de desarrollo que dura 24h activa. Existen maneras de ampliar el acceso así como la tasa de consultas permitidas a sus servidores. Para ello habría que registrar el producto. Dado que la finalidad de este proyecto es investigación personal, estamos limitados a una clave personal. También se ha de fijar una región en la que trabajar, se escoge la región "Europa-West". Estos son requisitos básicos de la API para trabajar con ella

In [ ]:

```
cass.set_riot_api_key("clave-API")  # This overrides the value set in your configuration/settings.
cass.set_default_region("EUW")
```

Para acceder a los datos de una partida, se requiere del id de una partida. Esto se puede obtener de varias maneras: en la documentación de la API de Riot Games se pueden encontrar listas de ids para pruebas o se puede acceder al historial de partidas de un jugador.

El objetivo de esta parte del proyecto es conseguir la suficiente cantidad de ids de partidas de una liga específica para analizar los datos posteriormente en R. Para ello, podemos consultar en la API la lista de todos jugadores que pertenecen a la liga objetivo: másters. NO solo eso, también podemos especificar la modalidad de partida de interés, que en nuestro caso son partidas clasificatorias o "ranked".

In [ ]:

```
masters = cass.get_master_league(queue=Queue.ranked_solo_fives, region="EUW")
```

El output es la lista de identificadores de jugadores que pertenecen a esa liga en concreto. A partir de esta lista podemos consultar de cada uno de ellos el historial de partidas. Cada una de esas partidas lleva un id asociado que es el que usaremos para construir la base de datos. El output de la solicitud es un JSON sobre el que se ha de iterar. Para ello se definen dos bucles for que rellenan las listas "jugadores_master" e "historias". En la lista historias se alamacenan los ids de partida

In [ ]:

```
jugadores_master=list()
historias=list()

contador = 1
for master in masters:
    print("{parcial}/{total} - {summoner}".format(parcial=contador, total=len(masters), summoner=ma
ster.summoner.name))
    jugadores_master.append(master.summoner.name)

    summoner      = cass.get_summoner(name = master.summoner.name, region="EUW")
    match_history = summoner.match_history(seasons={Season.season_9},
queues={Queue.ranked_solo_fives})
    #match_history(seasons={Season.season_9}, queues={Queue.ranked_solo_fives})


    for historia in match_history:
        print(historia)
        historias.append(historia.id)

    contador = contador + 1
```

Una vez tenemos la lista de ids, hablaremos del proceso de consulta de una partida. Cassiopeia tiene la función "cass.get_match()" cuyos argumentos son el id de la partida y la región en la que se consulta. El output de la función es un JSON que contiene mucha información de la que necesitamos apartados específicos y estructurados. Para automatizar este proceso se han escrito una serie de funciones anidadas es una función grande que procesará las partidas de una a una y almacenará el resultado en un formato csv listo para exportar a R.

La función grande se ha denominado "procesar_partida". La función requiere de un solo argumento:

La función grande se ha denominado "procesar_partida". La función requiere de un solo argumento:

- id_partida: el id de la partida.

La función almacena en listas los datos del JSON que nos son de interés y luego los almacena respetando la estructura en una lista separada por comas como un csv. El resultado son 42 columnas por cada id de partida.

In [ ]:

```
def procesar_partida(id_partida):
    blue_wins          = list() #Lista principal
    first_baron        = list()
    first_dragon       = list()
    first_blood        = list()
    first_inhibitor    = list()
    first_tower        = list()
    first_rifth        = list()
    tower_kills        = list()
    dragon_kills       = list()
    baron_kills        = list()
    inhibitor_kills    = list()
    rifth_kills        = list()
    picks_lista        = list()
    bans_lista         = list()
    maestrias_lista    = list()
    partida            = cass.get_match(id_partida, region = "EUW")
    equipos            = partida.teams
    participantes      = partida.participants

    # TODO: GetOro(participantes, oro_lista)

    procesar_booleano(partida.blue_team.win                    , blue_wins        )
    procesar_booleano(partida.blue_team.first_baron            , first_baron      )
    procesar_booleano(partida.blue_team.first_dragon           , first_dragon     )
    procesar_booleano(partida.blue_team.first_blood            , first_blood      )
    procesar_booleano(partida.blue_team.first_inhibitor        , first_inhibitor  )
    procesar_booleano(partida.blue_team.first_tower            , first_tower      )
    procesar_booleano(partida.blue_team.first_rift_herald      , first_rifth      )

    procesar_numerico(partida.blue_team.tower_kills            , tower_kills      )
    procesar_numerico(partida.blue_team.dragon_kills           , dragon_kills     )
    procesar_numerico(partida.blue_team.baron_kills            , baron_kills      )
    procesar_numerico(partida.blue_team.inhibitor_kills        , inhibitor_kills  )
    procesar_numerico(partida.blue_team.rift_herald_kills      , rifth_kills      )

    for team in equipos:
        for ban in team.bans:
            if ban is None:
                bans_lista.append('0')
            else:
                bans_lista.append(ban.id)
    for participante in participantes:
        picks_lista.append(participante.champion.id)
        maestria   = maestria_campeon(participante,participante.champion.name)
        maestrias_lista.append(maestria)
    partida_procesada = blue_wins + first_dragon + first_baron + first_blood + first_inhibitor + fi
rst_tower + first_rifth + tower_kills + dragon_kills + baron_kills + inhibitor_kills + rifth_kills
+ picks_lista +bans_lista +maestrias_lista
    return(partida_procesada)
```

La función lleva anidadas tres funciones escritas para automatizar este proceso además de la función propia de Cassiopeia "cass.get_match()". Son las funciones: maestria_campeon, procesar_booleano y procesar_numerico.

La función maestria_campeon es una consulta separada a la API. Cada jugador tiene un nivel de maestria asociado a cada campeon del juego. La función coge el identificador del campeon que ha escogido el jugador y el id del jugador para consultar el nivel de maestria. La función tiene dos argumentos:

- participante: el id del jugador que escoge el campeón.
- champion_name: el nombre del campeon escogido

La función realiza la consulta y devuelve un número: el nivel de maestría.

In [ ]:

```
def maestria_campeon(participante, champion_name):
```

```
    champion_masteries = cass.get_champion_masteries(summoner = participante.summoner.name,
                                                     region = "EUW")
    if champion_masteries[champion_name] is None:
        return 0
    else:
        return champion_masteries[champion_name].points
```

La función procesar_booleano sirve para normalizar los datos del JSON. Así nos aseguramos de que los datos se almacenan codificados en 1 y 0. La función tiene dos argumentos:

- input_bool: el statement booleano.
- list_to_append_value: la lista a la que corresponde el valor asociado.

La función comprueba el valor del booleano y guarda el valor en la lista como 1 o 0.

In [ ]:

```
def procesar_booleano(input_bool, list_to_append_value):
    if input_bool is True:
        list_to_append_value.append('1') #  1 == Victoria del equipo Azul
    else:
        list_to_append_value.append('0')
```

La función procesar_numerico es otra función preventiva. En ocasiones pueden haber valores mal codificados que darían a error de la función procesar_partida, de ahí que en caso de que se diera uno de estos valores se guardarían como 0. Esto se trataría más adelante en el código del análisis en R. La función tiene dos argumentos:

- input numérico: proveniente de varios de los apartados del JSON
- list_to_append_value: la lista a la que se ha de adherir el valor resultante.

In [ ]:

```
def procesar_numerico(input_numerico, list_to_append_value):
    if input_numerico is None:
        list_to_append_value.append('0')
    else:
        list_to_append_value.append(input_numerico)
```

Una vez definidas las funciones que procesaran los JSON que provienen de la API se establece un bucle para consultar uno a uno los ids de las partidas y se guardan en una lista en el orden establecido. Se recupera la lista "historias" donde se almacenan todos los ids de partidas y se itera sobre ellas. El resultado se guarda en la lista partidas_procesadas

In [ ]:

```
partidas_procesadas = list()
contando = 1
for id_partida in historias[0]:
    print("{parcial}/{total}".format(parcial=contando, total=len(historias)))
    try:
        partida_procesada = procesar_partida(int(id_partida))
        partidas_procesadas.append(partida_procesada)
    except Exception:
        pass
    contando = contando + 1
```

Por último se vuelca el resultado en un fichero para exportar los datos a R

In [ ]:

```
with open ('partidas.csv','w') as file:
        for partida in partidas_procesadas:
            numPartidas = len(partida)
            contador = 0
            for elemento_partida in partida:
                if contador < numPartidas-1:
                    file.write("%u," % int(elemento_partida))
                else:
                    file.write("%u" % int(elemento_partida))
```

```
                contador = contador + 1
            file.write("\n")
        file.close()
```