Project Analysis

Problem:

In League of Legends when a player finishes a game by default they are provided with some basic data about how their game went such as damage, gold earned and kills so that a player can see how it went. However, the riot API made by the developers of the game can provide you with a host of other data about the game which the game doesn't.

In league of legends there are many playstyles and strengths and weaknesses players can have. It is difficult to spot them just by playing the game, riot games has released an API for 3rd party developers to make their own solutions to game analytics which can provide with a host of other game data. However, 3rd parties often do it in the form of cloud processing many players simultaneously and become limited in how much data they can utilise and how much processing they can do, whereas I plan to do it locally on the user's computer meaning I will not be limited to small amounts of processing capacity for each player.

End users: League of Legends players who use analytics

Interview:

A league of legends player

How does analytics come into play with league of legends?

"Analytics pushes players to better themselves. Players after finishing a game are able to look back on how much they farmed, killed and what they were rated as in comparison to their team and opposition."

What analytics system are you currently using now and what data can it provide you with?

"OP.GG. It provides with the amount of cs, kills, damage, vision score, items purchased and tier average."

What kind of other people use analytics systems like this?

"Everyone uses it"

[referring to the player base of the game]

What problems do you encounter with current 3rd party analytics providers?

"The major problem with the analytical website is the rating. When seeing someone play and how they impact the game isn't mirrored by their score as they don't have the highest rating."

[By rating he is referring to the MVP system that ranks players in the match compared to each other]

What is your proficiency with computers?

"Very proficient."

What are your requirements in an analytics system?

"Easy to use and intuitive"

Summary:

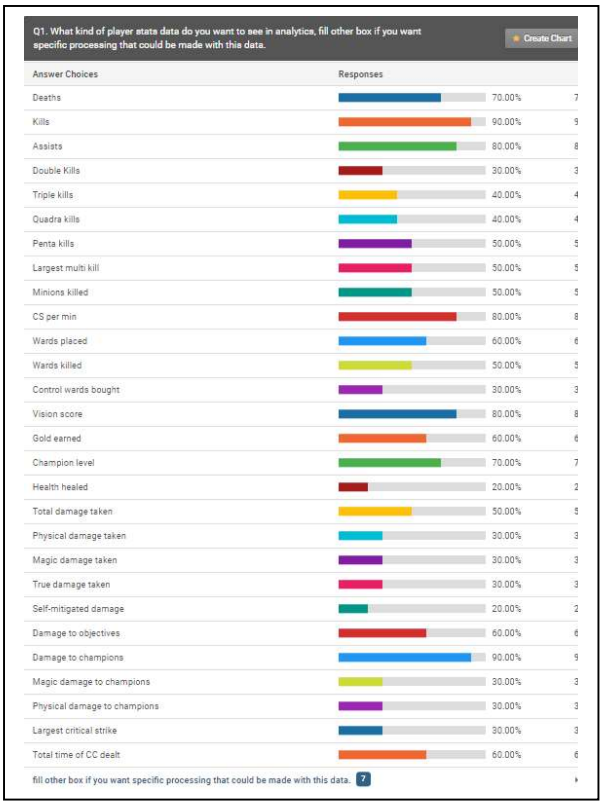
Purpose - Players that I have briefly interviewed use analytics after every single game, they have told me it provides them with knowledge on what their weaknesses are. In addition to this by having their performance quantified as numbers allows them to compare themselves to other players and friends which makes them more competitive.

Problems - Currently they use the game's built in system and 3rd party analytics providers such as OP.GG, they outlined that these systems lack in presenting certain game data such as kill participation %.

Data gathering - How the current analytics systems work, data is collected from the game and is stored in the riot API data base, the game and 3rd parties retrieve their data from there and can present it in any way they want.

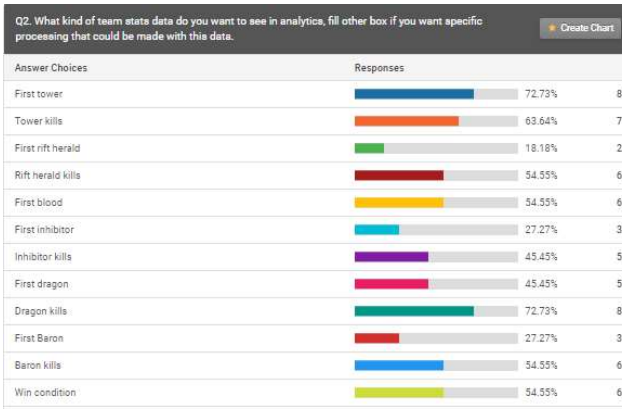
Security - security is not a concern as every player's data is publicly available to everyone and is provided to anyone that requests it. It is only a concern if the processing is done locally then the player will need to generate their own API key which needs to be kept safe by the player. To meet this concern my software will have an option whether it saves the key. If the user has a reason to believe that their system is insecure for the key to be saved on it, there will be an option to disable autosave.

Survey - I also made a survey that I gave out to some league legends players that use the analytics for league of legends and asked them about what kind of data and processing they want to see and what other problems they have with the analytics systems such as OP.GG, so that I can make sure I don't repeat any mistakes. 9 people filled the survey.



Data processing features requested:

-Kill participation %
-AD/AP damage ratio received and given
-Team carry/ MVP
-Software to provide explicit points where to improve
-Ability to check on other players
-Calculate number of neutral objectives taken



Existing system: OP.GG

Currently there are analytics websites like op.gg which utilise the riot API. But when they are analysing and storing thousands of accounts worth of data on the cloud, they become limited in how much data they can compute and store – causing them to miss out on data that is available from the riot API.

Figure 1 is some tables I have constructed to show the game data that is available from the riot API and how much is used by analytics providers such as OP.GG.

The table makes it clear why it can be so difficult to process thousands of accounts of data in the cloud if all of it is utilised.

Player Stats:	OP.GG	Survey
Deaths	Y	7
Kills	Y	9
Assists	Y	8
Double Kills	N	3
Triple kills	N	4
Quadra kills	N	4
Penta kills	N	5
Largest multi kill	Y	5
Minions killed	Y	5
CS per min	Y	8
Wards placed	N	6
Wards killed	N	5
Control wards bought	N	3
Vision score	N	8
Gold earned	Y	6
Champion level	Y	7
Health healed	N	2
Total damage taken	N	5
Physical damage taken	N	3
Magic damage taken	N	3
True damage taken	N	3
Self-mitigated damage	N	2
Damage to objectives	N	6
Damage to champions	Y	9
Magic damage dealt	N	3
Physical damage dealt	N	3
True damage dealt	N	3
Largest critical strike	N	3
Total time CC dealt	N	6

Team stats:	OP.GG	Survey
First tower	N	8
Tower kills	Y	7
First rift herald	N	2
Rift herald kills	N	6
First blood	N	6
First inhibitor	N	3
Inhibitor kills	N	5
First dragon	N	5
Dragon kills	Y	8
First Baron	N	3
Baron kills	Y	6
Win condition	Y	6

Data processing/features requests:	OP.GG
Kill participation %	Y
AD/AP damage ratio	N
Team carry/ MVP	Y
Improvement guidance	N
Check other players	Y
Calculate objectives	N

Game Data:	OP.GG
Game Duration	Y
Map	N
Game Mode	N
Queue Type	Y
Season	N
Banned Champions IDs	N

Player Data:	OP.GG
Name	Y
Champion	Y
Mastery	N
Rank	Y
Role	N

Now that I have these tables showing me what features players want me to include and knowing what OP.GG does and does not have. I now have an idea of what data will be needed from the riot API and can use this to prioritise what is needed in my system.

IPSO	Information
Input	User input: -Player username -Region -Player's game's account API key
Process	API: Request player encrypted ID
Input	API: -Player encrypted ID
Process	Program: -Request player's match list using player encrypted ID

Input	API returns match IDs and references: -gameIDs -roles -seasons -champions [will need to be compared to champion dictionary to convert to string as return is integer] -queues[will need to be compared to queue dictionary to convert to string as return is integer] -lanes -timestamps
Input	User: -Selected match
Process	Program: -Request selected match data from API
Input/Store	API Player Stats (10 columns, 1 for each player): -Deaths -Kills -Assists -Double Kills -Triple kills -Quadra kills -Penta kills -Largest multi kill -Minions killed -CS per min -Wards placed -Wards killed -Control wards bought -Vision score -Gold earned -Champion level -Health healed -Total damage taken -Physical damage taken -Magic damage taken -True damage taken -Self-mitigated damage -Damage to objectives -Damage to champions -Magic damage dealt -Physical damage dealt -True damage dealt -Largest critical strike -Total time CC dealt
Input/Store	Team stats (2 columns, 1 for each team): -First rift herald -Rift herald kills -First blood -First inhibitor -Inhibitor kills -First dragon -Dragon kills -First Baron -Baron kills -Win condition

Input/Store	API Game Data: -Game Duration -Map -Game Mode -Queue Type -Season
Process	Program: -Kill participation %, add kills of each player together to get total team kills, for each player get value assists + kills, put it into a % compared to total team kills -AP/AD/True damage received/given ratios for team and individual players, plot a pie chart for each player with matplotlib -Total neutral team objectives taken, rift herald kills + baron kills + dragon kills -Points to improve algorithm, create default values for things like CS per min for which if the player falls below the program would indicate by highlighting data with colour codes, below target[red/orange], on target[green] -MVP of the game, use points system where each data point is compared to all 10 players if player has highest match value in certain statistic, they are awarded 10 points, 2nd 9 points and so on. Points are added together for each player and at end player with most points will be labelled with MVP.
Output	Program: -All data in a table - players left hand side, columns presenting each data point. Boxes to be colour coded green, orange and red judging whether values meet certain pre-determined points by user or default

Data Item	**Data Type**	**Sample Data**
Player username	String	Stegi56
Region	String	EUW
API key	String	
Player encrypted ID	String	
GameID	String	
Role	String	Support
Season	Integer	8
Champion	Integer	25
Queue	Integer	
Lane	String	Bottom
Time stamp	String	25:37
GameID	String	
Deaths	Integer	2
Kills	Integer	10
Assists	Integer	15
Double kills	Integer	4
Triple kills	Integer	2
Quadra kills	Integer	1
Penta kills	Integer	0
Largest multi kill	Integer	4
Minions killed	Integer	206
CS per min	Float	6.4
Wards placed	Integer	12
Wards killed	Integer	4
Control wards bought	Integer	6
Vision score	Integer	35
Gold earned	Integer	10,375

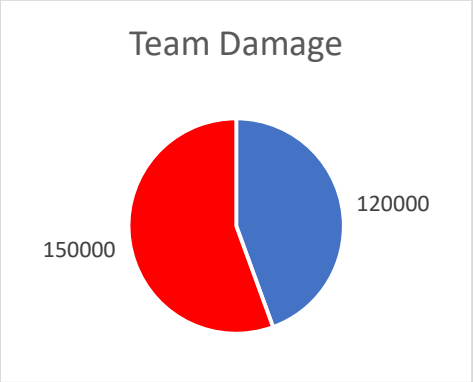
Champion level	Integer	14
Health healed	Integer	2,380
Total damage taken	Integer	6,241
Physical damage taken	Integer	6,000
Magic damage taken	Integer	200
True damage taken	Integer	41
Self-mitigated damage	Integer	2,000
Damage to objectives	Integer	4,000
Damage to champions	Integer	25,000
Magic damage dealt	Integer	20,000
Physical damage dealt	Integer	4,000
True damage dealt	Integer	1,000
Largest critical strike	Integer	600
Total time CC dealt	Integer	38
First herald	Boolean	1
Herald kills	Integer	2
First Blood	Boolean	0
First inhibitor	Boolean	1
Inhibitor kills	Integer	4
First dragon	Boolean	0
Dragon kills	Integer	3
First baron	Boolean	0
Baron kills	Integer	1
Win	Boolean	0
Game duration	String	1258
Map	Integer	
Game mode	String	
Queue Type	Integer	
Banned champions	List	
Name	String	Stegi
Champion	Integer	
Mastery	Integer	7
Rank	Integer	
Role	String	Support

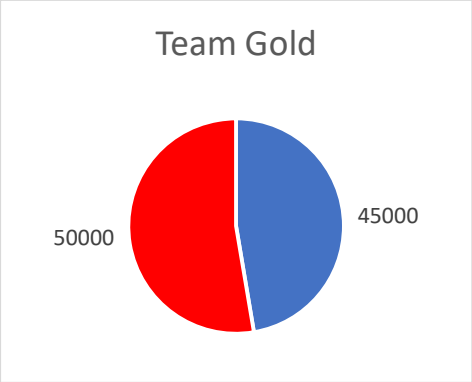
Objectives:

No	Objective	Performance criteria
1	**Establish connection to API and retrieve all data required.**	-UI to get player info -Use info to send request to API -Scrub return data from link and convert from json to py dictionary -Using player ID stored in dictionary send request to API for match history -Store match history in a dictionary -Using player ID request player profile from API -Store profile in a dictionary -Take match ID that user selected and make request to API -Store match data to a dictionary
2	**Error message**	-Display an appropriate error message indication which input the user has incorrectly entered
3	**Autofill feature**	- Save user inputs if the opt in to do so via checkbox -Inputs should be loaded when program is opened again
4	**Output profile**	-Display user rank/s and LP -Display user icon -Display username -Calculate and display winrate
5	**Assign each match to class**	-Class should be initialised with matchID -Initialy class must retrieve , player champion, KDA, win/loss, items, queue type
6	**Match history output**	-Display each match as a button with the match class assigned to it -Each match button must show: • Player champion • KDA • Win/Loss • Items • Queue type
7	**Process data**	-Calculate damage given/taken ratios for individual players and team as a whole and present in a pie chart -Kill participation % take each player's kill/assist scores, add together and compare to total team kills as % -Calculate number of objectives taken, take score for each objective – dragon kills, herald kills, baron kills and add together -Algorithm that suggests where to improve each data point will have an average made for each game to which each player's data points will be compared to, if data point falls 1 standard deviation below average it will be flagged red, 1.5 standard deviations above average it will be flagge green.
8	**Write MVP algorithm**	-Use data given about each player in match to flag a player as the MVP using a points-based system for being best in various statistics that game.
9	**Output match data in tables using tKinter and matplotlib**	-Each player needs to have their own row, rows to be group into two teams and ordered by MVP score within teams. Data points to be in columns so that they can be easily compared to other players top down. -Not everything can be represented intuitively as a numbers so damage ratios will need to be presented in pie charts. True damage – white, physical damage – red, magic damage – blue.

Rough Output:

Team RED						Team BLUE				
Player name + icon	#2	#3	#4	#5	Stat Type	#1	#2	#3	#4	#5
					Kills					
					Assists					
					Kill participation %					
					Damage					
					Etc.					
-First baron -Baron kills -Dragon kills -First blood -etc.					Bullet Points where relevant	-First dragon kill -Dragon kills -etc.				





Your weak points:

Stat	Your score	Target
CS/min	4.2	6
Vision /min	0.8	1.5
KDA	0.3	2

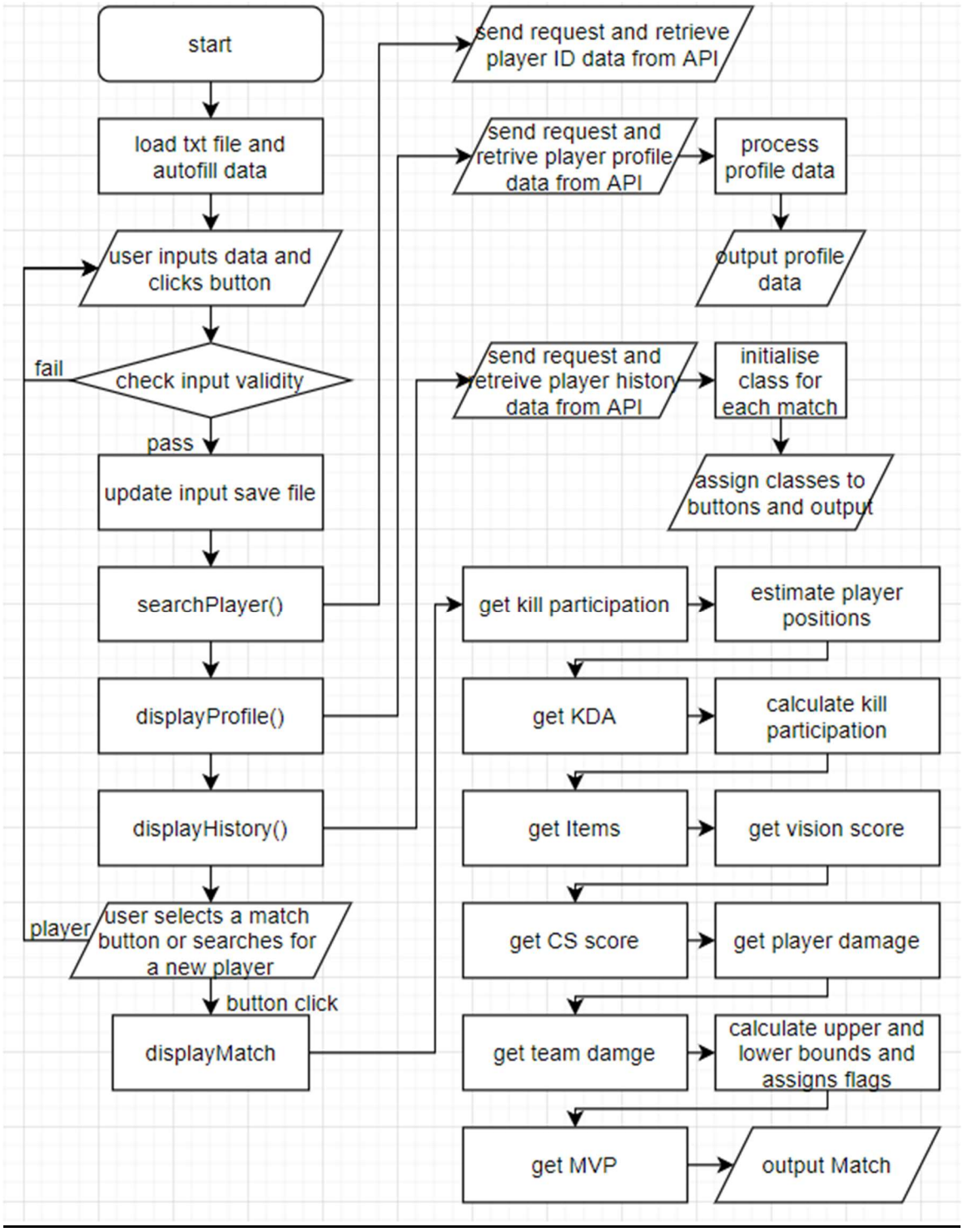
Potential solutions:

I plan to use python to code this as it is an easy language to write in and comes with lots of modules such as tKinter and matplotlib that I will find useful while I am working on the project and will likely make it much faster and easier to complete.

A downside to this is that python is not an ideal language for front end development, despite this I believe that the time lost in front end will be overwhelmed by the time I will save implementing the backend. If I find something later down the line that can work alongside python for front end, I will consider utilising it.

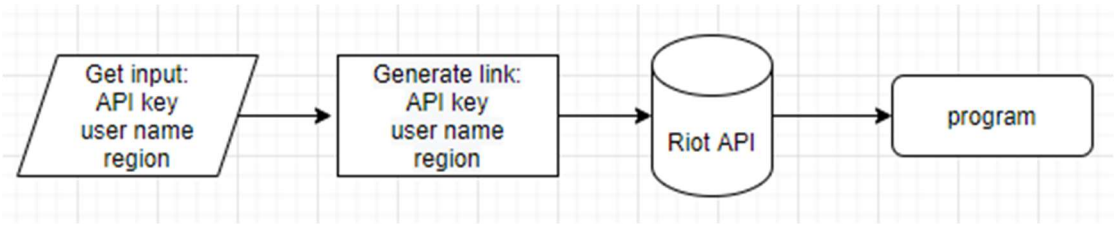
Design

System overview)



Part 1)

-Write a system to get user inputs and to retrieve riot API data.



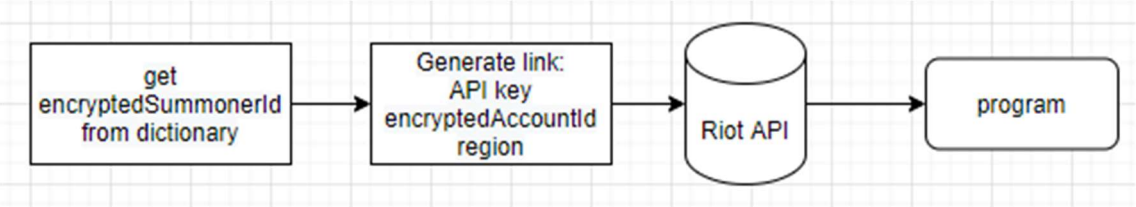
- Get user input
- Generate a link with input data to generate a request from API
  - https://euw1.api.riotgames.com/lol/summoner/v4/summoners/by-name/stegi56?api_key=RGAPI-73365e51-83e7-41cb-a6ba-d234e4a94eab
  - Region, username, API key
- Go to link and fetch json file from it using a requests python library
- Expected data:


```
{
    "id": "O8PIK5QQb7uOGJrj5Bcq3f2z0j65hpWHi-ES8oFCviRzpHVk",
    "accountId": "tvnBltrvistasmK6mbObTPchXSjpRufLvcO72dGfBdpV9AA",
    "puuid": "yvGz5esOATAyzDX-sok742z5oJz-bAcVZtHuFKuc4dL0iIaA7OI2PPmdmo4HakU1OA70Am7fY5w7Cg",
    "name": "stegi56",
    "profileIconId": 2074,
    "revisionDate": 1611103951000,
    "summonerLevel": 144
}
```


- Use .json() to convert json file to a SUMMONER python dictionary

Part 2)

-Write a new request using retrieved data to get more information about the user



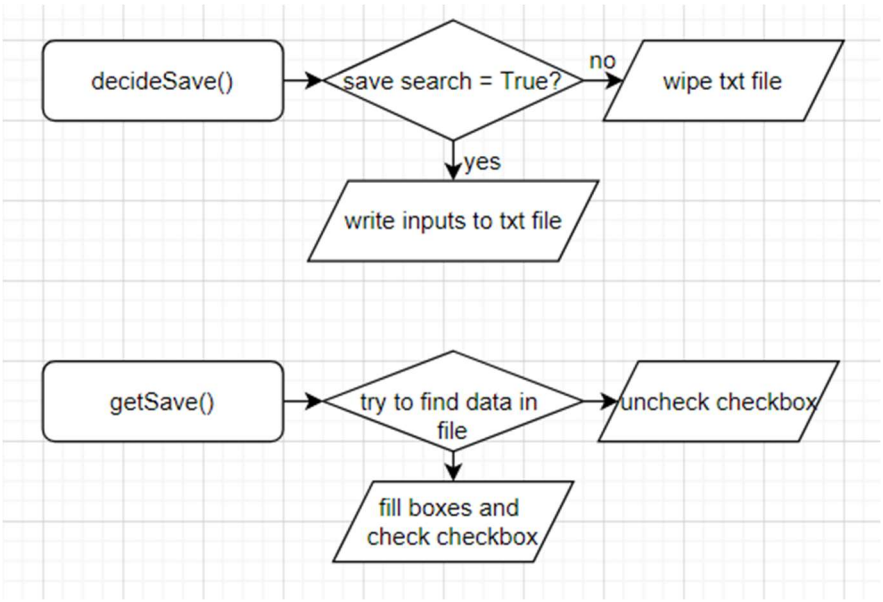
- Get encryptedSummonerId from dictionary
- Generate a link with input data to generate a request from API
  - https://euw1.api.riotgames.com/lol/league/v4/entries/by-summoner/O8PIK5QQb7uOGJrj5Bcq3f2z0j65hpWHi-ES8oFCviRzpHVk?api_key=RGAPI-73365e51-83e7-41cb-a6ba-d234e4a94eab
  - Region, encryptedSummonerId, API key
- Go to link and fetch json file from it using a requests python library
- Expected data:


```
[
    {
        "leagueId": "d61235a1-7db2-4743-818d-12653b6c7f14",
        "queueType": "RANKED_SOLO_5x5",
        "tier": "GOLD",
        "rank": "IV",
        "summonerId": "O8PIK5QQb7uOGJrj5Bcq3f2z0j65hpWHi-ES8oFCviRzpHVk",
        "summonerName": "stegi56",
        "leaguePoints": 0,
        "wins": 12,
        "losses": 13,
        "veteran": false,
        "inactive": false,
        "freshBlood": true,
        "hotStreak": false
    },
    {
        "leagueId": "ddf47c2a-8c54-424f-8089-300a83aec831",
        "queueType": "RANKED_FLEX_SR",
        "tier": "SILVER",
        "rank": "II",
        "summonerId": "O8PIK5QQb7uOGJrj5Bcq3f2z0j65hpWHi-ES8oFCviRzpHVk",
        "summonerName": "stegi56",
        "leaguePoints": 59,
        "wins": 9,
        "losses": 10,
        "veteran": false,
        "inactive": false,
        "freshBlood": false,
        "hotStreak": false
    }
]
```


- Use .json() to convert json file to a PROFILE python dictionary

Part 3)

-Start making a GUI



_____	API key	Rank solo	Rank flex
_____	Username	silver	gold
_____	Region	winrate	winrate
Match 1	Button(info)	Match 3	Button(info)
Match 2	Button(info)	Match 4	Button(info)

- 3 input boxes for API key, username and region
  - If user does not have API key there should be a button that would lead them to a weblink to get the key using their game login.
  - To make the process easier there should be a tick box, when enabled the program will retain the inputs and load them into the input boxes for the next time the program is run
  - Search button that will attempt to run part 1, if that does not work an appropriate error message should be displayed
- Display profile
  - Assets required: Icons and tiers
  - Display username at top
  - Load an icon from asset folder based on icon id in SUMMONER dictionary
  - Load tier icon/s from asset folder based on tier/s in PROFILE dictionary
  - Under tier icon/s print the rank and LP from the PROFILE dictionary
  - Print tier type, wins and losses next to tier icon/s from PROFILE dictionary
  - Calculate win-rate % for each tier using wins and losses from PROFILE dictionary
- Display history

Part 4)

Get matchlist and match data for last 10 matches of player, store each match as an object in a class

- In this part I will be making many requests to the API. The riot API has a limit on how many requests can be made per second/minute so I need to make sure that when I am making calls to get matches after the profile has been created, there will need to be a delay timer.
  This issue could be fixed if I were to host a server myself and would register for a production API key from Riot, which has a much more generous rate limit, but currently that is out of the range of my skillset. This would also limit the functionality of the program as I do not have the capacity to maintain a server 24/7. It is wiser for requests to be executed locally on the user's machine.
  - **RATE LIMITS**

    20 requests every 1 seconds(s)
    100 requests every 2 minutes(s)
  - The starting rate limit of a production API key is much larger than the development key:
    - 500 requests every 10 seconds
    - 30,000 requests every 10 minutes
  -
- Get matchlist
- Generate link
  - https://euw1.api.riotgames.com/lol/match/v4/matchlists/by-account/tvnBltrvistasmK6mbObTPchXSjpRufLvcO72dGfBdpV9AA?api_key=RGAPI-73365e51-83e7-41cb-a6ba-d234e4a94eab
  - Region, accountID, API key
- Go to link and fetch json file from it using a requests python library
- Expected data:

```
    "gameId": 5054754343,
            "champion": 40,
            "queue": 450,
            "season": 13,
            "timestamp": 1611614483209,
            "role": "DUO_SUPPORT",
            "lane": "NONE"
        },
        {
            "platformId": "EUW1",
            "gameId": 5054639289,
            "champion": 34,
            "queue": 400,
            "season": 13,
            "timestamp": 1611612804087,
            "role": "DUO_SUPPORT",
            "lane": "BOTTOM"
        },
```

  -
- Use .json() to convert json file to a PROFILE python dictionary
- Make a Match class
  - All attributes of every match will be stored here data for each match from MATCHLIST and MATCH dictionaries will be merged and stored here
- Get last 10 matches
  - 20 requests per second allowed, 3 requests for player info and match list 10 for individual matches – this will prevent needing to make a buffer
  - Once implemented consider making a "load more" buffered option and caching on sql
- Generate link
  - https://euw1.api.riotgames.com/lol/match/v4/matches/5016181103?api_key= RGAPI-73365e51-83e7-41cb-a6ba-d234e4a94eab

- Region, matchID, API key
- matchID taken from matchlist

- Expected output

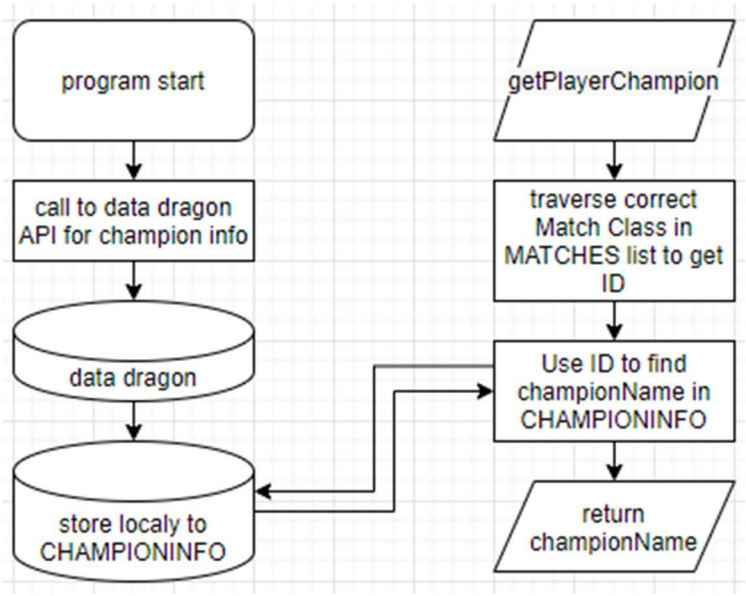

```
{
  "gameId": 5054639289,
  "platformId": "EUW1",
  "gameCreation": 1611612804087,
  "gameDuration": 1438,
  "queueId": 400,
  "mapId": 11,
  "seasonId": 13,
  "gameVersion": "11.2.353.8505",
  "gameMode": "CLASSIC",
  "gameType": "MATCHED_GAME",
  "teams": [
    {
      "teamId": 100,
      "win": "Fail",
      "firstBlood": false,
      "firstTower": false,
      "firstInhibitor": false,
      "firstBaron": false,
      "firstDragon": false,
      "firstRiftHerald": true,
      "towerKills": 3,
      "inhibitorKills": 0,
      "baronKills": 0,
      "dragonKills": 0,
      "vilemawKills": 0,
      "riftHeraldKills": 2,
      "dominionVictoryScore": 0,
      "bans": [
        {
          "championId": 145,
          "pickTurn": 1
        },
        {
          "championId": 63,
          "pickTurn": 2
        },
        {
          "championId": 238,
          "pickTurn": 3
        },
        {
          "championId": 7,
          "pickTurn": 4
        },
        {
          "championId": 122,
          "pickTurn": 5
        }
      ]
    },
    {
      "teamId": 200,
      "win": "Win",
      "firstBlood": true,
      "firstTower": true,
      "firstInhibitor": true,
      "firstBaron": false,
      "firstDragon": true,
      "firstRiftHerald": false,
      "towerKills": 10,
      "inhibitorKills": 2,
      "baronKills": 0,
      "dragonKills": 2,
      "vilemawKills": 0,
      "riftHeraldKills": 0,
      "dominionVictoryScore": 0,
      "bans": [
        {
          "championId": 202,
          "pickTurn": 6
        },
        {
          "championId": 35,
          "pickTurn": 7
        },
        {
          "championId": 420,
          "pickTurn": 8
        },
        {
          "championId": 157,
          "pickTurn": 9
        },
        {
          "championId": 99,
          "pickTurn": 10
        }
      ]
    }
  ],
  "participants": [
    {
      "participantId": 1,
      "teamId": 100,
      "championId": 18,
      "spell1Id": 4,
      "spell2Id": 14,
      "stats": {
        "participantId": 1,
        "win": false,
        "item0": 1055,
        "item1": 6672,
        "item2": 3006,
        "item3": 1018,
        "item4": 2015,
        "item5": 0,
        "item6": 3340,
        "kills": 4,
        "deaths": 12,
        "assists": 6,
        "largestKillingSpree": 2,
        "largestMultiKill": 2,
        "killingSprees": 1,
        "longestTimeSpentLiving": 232,
        "doubleKills": 1,
        "tripleKills": 0,
        "quadraKills": 0,
        "pentaKills": 0,
        "unrealKills": 0,
        "totalDamageDealt": 35942,
        "magicDamageDealt": 7871,
        "physicalDamageDealt": 23520,
        "trueDamageDealt": 4550,
        "largestCriticalStrike": 277,
        "totalDamageDealtToChampions": 8668,
        "magicDamageDealtToChampions": 2016,
        "physicalDamageDealtToChampions": 5469,
        "trueDamageDealtToChampions": 1183,
        "totalHeal": 2420,
        "totalUnitsHealed": 1,
        "damageSelfMitigated": 5895,
        "damageDealtToObjectives": 703,
        "damageDealtToTurrets": 703,
        "visionScore": 5,
        "timeCCingOthers": 5,
        "totalDamageTaken": 18671,
        "magicalDamageTaken": 4365,
        "physicalDamageTaken": 11333,
        "trueDamageTaken": 2972,
        "goldEarned": 7848,
        "goldSpent": 6300,
        "turretKills": 1,
        "inhibitorKills": 0,
        "totalMinionsKilled": 47,
        "neutralMinionsKilled": 0,
        "neutralMinionsKilledTeamJungle": 0,
        "neutralMinionsKilledEnemyJungle": 0,
        "totalTimeCrowdControlDealt": 28,
        "champLevel": 11,
        "visionWardsBoughtInGame": 0,
        "sightWardsBoughtInGame": 0,
        "wardsPlaced": 5,
        "wardsKilled": 0,
        "firstBloodKill": false,
        "firstBloodAssist": false,
        "firstTowerKill": false,
        "firstTowerAssist": false,
        "firstInhibitorKill": false,
        "firstInhibitorAssist": false,
        "combatPlayerScore": 0,
        "objectivePlayerScore": 0,
        "totalPlayerScore": 0,
        "totalScoreRank": 0,
        "playerScore0": 0,
        "playerScore1": 0,
        "playerScore2": 0,
        "playerScore3": 0,
        "playerScore4": 0,
        "playerScore5": 0,
        "playerScore6": 0,
        "playerScore7": 0,
        "playerScore8": 0,
        "playerScore9": 0,
        "perk0": 8005,
        "perk0Var1": 547,
        "perk0Var2": 387,
        "perk0Var3": 160,
        "perk1": 9111,
        "perk1Var1": 291,
        "perk1Var2": 200,
        "perk1Var3": 0,
        "perk2": 9104,
        "perk2Var1": 23,
        "perk2Var2": 30,
        "perk2Var3": 0,
        "perk3": 8014,
        "perk3Var1": 134,
        "perk3Var2": 0,
        "perk3Var3": 0,
        "perk4": 8139,
        "perk4Var1": 820,
        "perk4Var2": 0,
        "perk4Var3": 0,
        "perk5": 8135,
        "perk5Var1": 628,
        "perk5Var2": 5,
        "perk5Var3": 0,
        "perkPrimaryStyle": 8000,
        "perkSubStyle": 8100,
        "statPerk0": 5005,
        "statPerk1": 5008,
        "statPerk2": 5002
      },
      "timeline": {
        "participantId": 1,
        "creepsPerMinDeltas": {
          "10-20": 2,
          "0-10": 2.6
        },
        "xpPerMinDeltas": {
          "10-20": 277.6,
          "0-10": 228.1
        },
        "goldPerMinDeltas": {
          "10-20": 362,
          "0-10": 160.5
        },
        "csDiffPerMinDeltas": {
          "10-20": -2.2499999999999996,
          "0-10": -2.35
        },
        "xpDiffPerMinDeltas": {
          "10-20": -231.05,
          "0-10": -78
        },
        "damageTakenPerMinDeltas": {
          "10-20": 773.1,
          "0-10": 427.8
        },
        "damageTakenDiffPerMinDeltas": {
          "10-20": 102.04999999999995,
          "0-10": 26.549999999999997
        },
        "role": "DUO_CARRY",
        "lane": "BOTTOM"
      }
    },
```

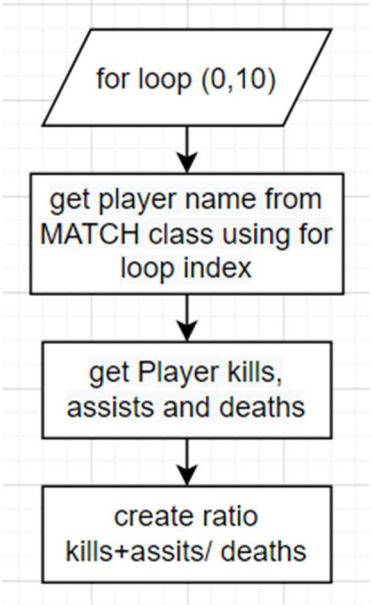

- This is the cut down output of data for 1/10 players in one match

- Use .json() to convert json file to a temporary MATCH python dictionary
- Each cycle I will need to initialise a new match class and store the dictionary for each match

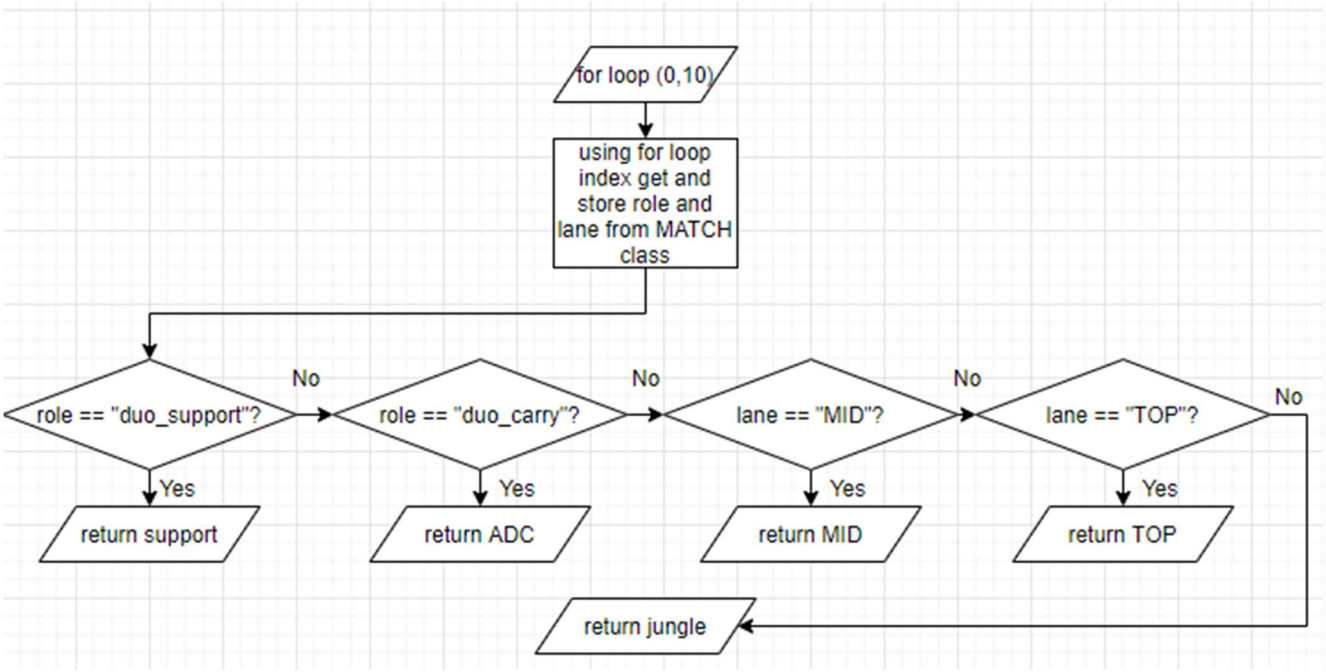
Part 5)

-Match class features, processes and MVP algorithm

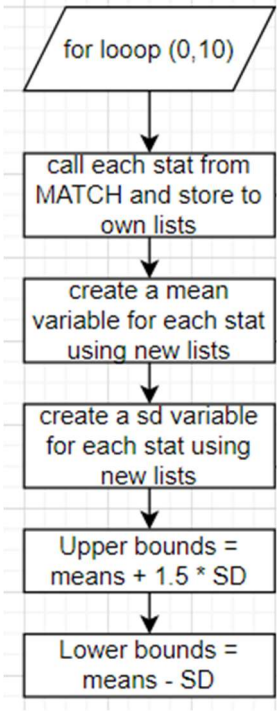
- Make a function in match class to convert championId to championName. This is needed to find the picture of the champion being played in the assets folder.
  - 

  - Throughout writing the code I realised that I will need another database that stores champion data, this will allow me to convert ID's from the riot API to names.
  - Each time the program is started it calls imports the data from another API "Data Dragon" which is another riot games API that stores more data.
  - Once I retrieve the player's champion ID from the MATCH dictionary I can use the CHAMPIONINFO dictionary to convert the ID to the champion name.
- Create ratio for KDA for each player so that it can be compared to others
  - 

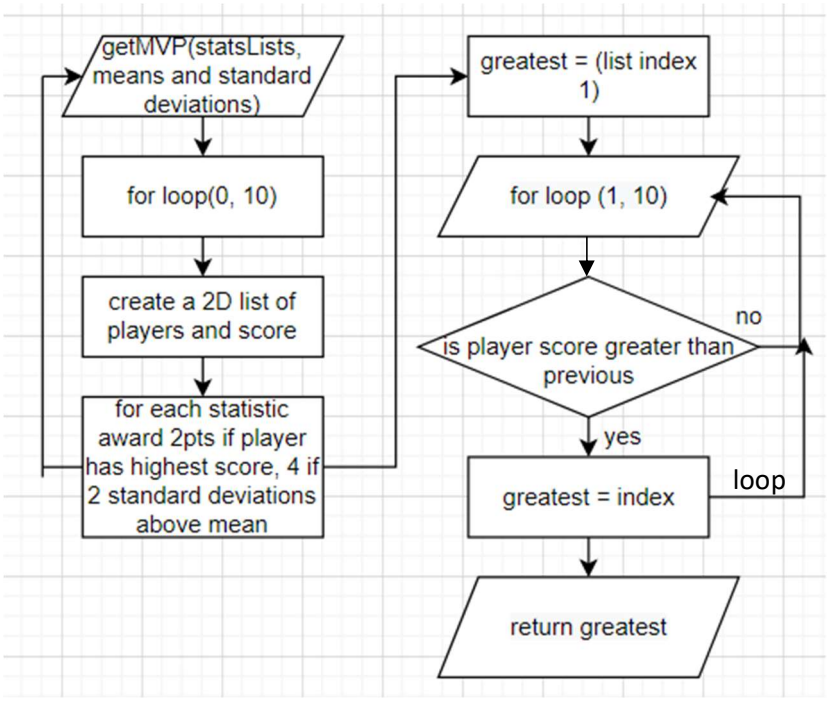
- Create algorithm to determine player position, the riot API only provides two estimates of the player: the role and position in the game. Using this I can create my own estimate of the player's position.



- Complete data processing for each statistic for flagging



  - To flag each data point as red or green they will need to be either 1.5 standard deviations above the mean to be flagged green or 1 below to be flagged red.
  - To do this I will need each data point inside their own lists to calculate the mean and standard deviation using the built in python mean() function and numpy module std()
  - Once I have the mean and standard deviation for each stat, to get the upper bound I can add 1.5 standard deviations to the mean and for the lower I can take 1 away.
  - When it comes to outputting the data the data type will be compared to its bounds and it falls within one an appropriate colour will be applied to flag it
  - The reason I chose not to use pre-set values for flagging data in games is because each game is different and I believe a player should be graded based on the performance in relation to everyone in the game.

- MVP algorithm
  - 

  - This algorithm fetches the lists, averages and bounds made earlier.
  - It creates a new list of players and points
  - Cycles through each player index calculates points and adds to points in list
  - Cycles through all items in list, finds greatest value and returns the player name associated with it.

Part 6)

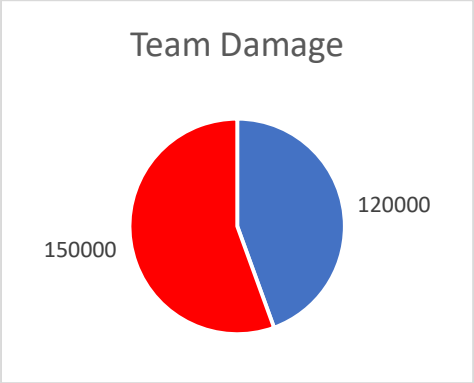
-Display profile and match history

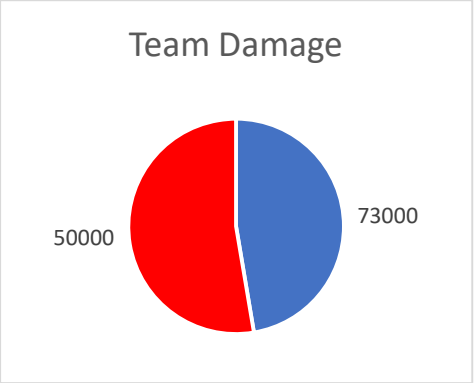
api key resets every 24hrs	[Get key]	profile icon	solo	flex
**********	API input		rank	rank
stegi56	username		lp	lp
euw1	region		win rate	win rate
Icon|KDA|Spells|Items	Icon|KDA|Spells|Items			
Icon|KDA|Spells|Items	Icon|KDA|Spells|Items			
Icon|KDA|Spells|Items	Icon|KDA|Spells|Items			
Icon|KDA|Spells|Items	Icon|KDA|Spells|Items			
Icon|KDA|Spells|Items	Icon|KDA|Spells|Items			

Part 7)

-Match info

Team RED						Team BLUE				
Player name + icon	#2	#3	#4	#5	Stat Type	#1	#2	#3	#4	#5
0	10	6	4	5	KDA					
					KP%					
					Items					
					Vision					
					Damage + pie charts					
					MVP					
-First baron -Baron kills -Dragon kills -First blood -etc.					Bullet Points where relevant	-First dragon kill -Dragon kills -etc.				





Part 8)

-Error messages

- Invalid API key
  - The riot API documentation states that a 403 error will be returned when an invalid key is requested, using this I can make a check for that return and output an API key error, this error will also guide the user to update their key.
- Invalid username
  - Similar to the API key error the API also returns a “Data not found - summoner not found” error if a there is no match for a username. Again I can make a check for this and inform the user to check their username.
- Invalid region
  - The region is the start of the hyperlink that is made to the riot API. If this is incorrect a connection to the API will not be made and a python error will occur. To prevent this, I can use an exception and guide the user to enter a valid region.

Technical Solution

```
#Allows python to open webbrowser
import webbrowser
#Allows to scrape sites for data, required for retreving data from API
import requests

#Imports tkinter, framework for GUI
from tkinter import *
#Tkinter is outdated and does not allow png images, this library allows me to fix that
from PIL import ImageTk, Image

#Allows me to calculate means for analytics
from statistics import mean
#Allows me to do complex mathematical operations such as finding the standard deviation for
alalytics
import numpy

#Allows to draw pie charts to make data easier to read
import matplotlib.pyplot as plt
#Allows to port tables into tkinter to minimise the number of open windows
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

#Allows me to create windows notifications for error messages
import ctypes

#Each Match class will be stored here
MATCHES = []
#Downloads latest database of champions for converting indexes to names
CHAMPIONINFO =
requests.get("http://ddragon.leagueoflegends.com/cdn/11.3.1/data/en_US/champion.json")
CHAMPIONINFO = CHAMPIONINFO.json()
#Downloads latest database of spells for converting indexes to names
SUMMONERSPELL =
requests.get("http://ddragon.leagueoflegends.com/cdn/11.3.1/data/en_US/summoner.json")
SUMMONERSPELL = SUMMONERSPELL.json()

class Match:
    def __init__(self, MATCHLISTinfo, region, APIkey, game):
        self.MATCHLISTinfo = MATCHLISTinfo
        matchId = str(MATCHLISTinfo["gameId"])
        self.MATCH = requests.get("https://" + region + ".api.riotgames.com/lol/match/v4/matches/"
+ matchId + "?api_key=" + APIkey)
        self.MATCH = self.MATCH.json()

        print("Loading: " + str(game + 1) + "0%")

    def getMatch(self):
        #returns the match dictionary
        return self.MATCH

    def getParticipant(self, name):
        #finds the player index based on name
        for player in range(0,10):
            if self.MATCH["participantIdentities"][player]["player"]["summonerName"] == name:
                return int(player)

    def getPlayerChampion(self, participant):
        #finds champion played using player index in MATCH dictionary
        championID = self.MATCH["participants"][participant]["championId"]
        for i in CHAMPIONINFO["data"]:
            if CHAMPIONINFO["data"][i]["key"] == str(championID):
                return CHAMPIONINFO["data"][i]["image"]["full"]

    def getPlayerName(self, participant, type):
        #finds player name using player index in MATCH dictionary,
        #parameter determines whether name is returned in short or full form
        if type == "truncated":
            participantName =
self.MATCH["participantIdentities"][participant]["player"]["summonerName"]
            truncatedName = participantName[0:9]  # Limits characters to name to prevent overlap
            return truncatedName
        if type == "full":
            return self.MATCH["participantIdentities"][participant]["player"]["summonerName"]
```

```
    def getKills(self, participant):
        return int(self.MATCH["participants"][participant]["stats"]["kills"])

    def getAssists(self, participant):
        return int(self.MATCH["participants"][participant]["stats"]["assists"])

    def getDeaths(self, participant):
        return int(self.MATCH["participants"][participant]["stats"]["deaths"])

    def getKDA(self, participant, type):
        #Finds kills, deaths and assists, using participant index
        #List parameter type determines whether a calculation needs to be done for a ratio and
what form it is returned in
        if type == "ratio":
            try:
                KDA =  ((self.getKills(participant) + self.getAssists(participant)) /
self.getDeaths(participant))
                return KDA
            except:
                KDA = self.getKills(participant) + self.getAssists(participant)
                return KDA

        elif type == "string":
            KDA = "KDA: "
            KDA += str(self.getKills(participant)) + " / "
            KDA += str(self.getDeaths(participant)) + " / "
            KDA += str(self.getAssists(participant))
            return KDA

        elif type == "short string":
            KDA = str(self.getKills(participant)) + " / "
            KDA += str(self.getDeaths(participant)) + " / "
            KDA += str(self.getAssists(participant))
            return KDA

    def getKillParticipation(self, participant, teamKills):
        try:
            killsAssits = self.getKills(participant)
            killsAssits += self.getAssists(participant)
            killParticipation = (100 / teamKills) * killsAssits
        except:
            killParticipation = 0
        return killParticipation

    #returns a list of indexes for the spells used by the participant
    def getSummonerSpells(self, participant):
        spells = []
        spells.append(self.MATCH["participants"][participant]["spell1Id"])
        spells.append(self.MATCH["participants"][participant]["spell2Id"])
        for i in SUMMONERSPELL["data"]:
            if SUMMONERSPELL["data"][i]["key"] == str(spells[0]):
                spells[0] = SUMMONERSPELL["data"][i]["image"]["full"]
        for i in SUMMONERSPELL["data"]:
            if SUMMONERSPELL["data"][i]["key"] == str(spells[1]):
                spells[1] = SUMMONERSPELL["data"][i]["image"]["full"]
        return spells

    def getMatchType(self):
        #It is possible to download a database of match indexes and names, however most of them
are too long for the GUI
        #Therefore I make a check for the two most common and substitute my own shorter output
        if self.MATCHLISTinfo["queue"] == 420:
            return "Ranked SOLO/DUO"
        elif self.MATCHLISTinfo["queue"] == 440:
            return "Ranked FLEX"
        else:
            return "Other"

    def getItems(self, participant):
        #returns a list of item indexes using participant index, returns in list form
        items = []
        items.append(self.MATCH["participants"][participant]["stats"]["item0"])
        items.append(self.MATCH["participants"][participant]["stats"]["item1"])
        items.append(self.MATCH["participants"][participant]["stats"]["item2"])
```

```
        items.append(self.MATCH["participants"][participant]["stats"]["item3"])
        items.append(self.MATCH["participants"][participant]["stats"]["item4"])
        items.append(self.MATCH["participants"][participant]["stats"]["item5"])
        items.append(self.MATCH["participants"][participant]["stats"]["item6"])
        return items

    def getWinLossColour(self, participant):
        #checks whether player is on winning or lossing team and returns color based on this
        #needed for match history buttons
        if self.MATCH["participants"][participant]["stats"]["win"] == True:
            return "#007007"
        else:
            return "#770000"

    def getPlayerPosition(self, participant):
        #algorithm to determine what position a player is playing based on estimations of role and
lane made by the API
        if self.MATCH["participants"][participant]["timeline"]["role"] == "DUO_SUPPORT":
            return "positions/support.png"
        elif self.MATCH["participants"][participant]["timeline"]["role"] == "DUO_CARRY":
            return "positions/ADC.png"
        elif self.MATCH["participants"][participant]["timeline"]["lane"] == "MID":
            return "positions/mid.png"
        elif self.MATCH["participants"][participant]["timeline"]["lane"] == "TOP":
            return "positions/top.png"
        else:
            return "positions/jungle.png"

    #Finds player vision score using participant index
    def getVisionScore(self, participant):
        return self.MATCH["participants"][participant]["stats"]["visionScore"]

    #Fetches the amount of CS a player gets at different time frames and rounds digits to 1dp
    def getCSperMin10(self, participant):
        try:
            return
float(format(self.MATCH["participants"][participant]["timeline"]["creepsPerMinDeltas"]["0-10"],
".1f"))
        except:
                return 0.0

    def getCSperMin20(self, participant):
        try:
            return
float(format(round((self.MATCH["participants"][participant]["timeline"]["creepsPerMinDeltas"]["10-
20"]), 2), ".1f"))
        except:
            return 0.0

    def getCSperMin30(self, participant):
        try:
            return
float(format(round((self.MATCH["participants"][participant]["timeline"]["creepsPerMinDeltas"]["20-
30"]), 2), ".1f"))
        except:
            return 0.0

    def getPlayerDamage(self, participant, type):
        #returns true, magic and physical damage for player,
        #type parameter determines whether damage is returned in list form or as a total
        if type == "list":
            damage = []

damage.append(self.MATCH["participants"][participant]["stats"]["trueDamageDealtToChampions"])

damage.append(self.MATCH["participants"][participant]["stats"]["physicalDamageDealtToChampions"])

damage.append(self.MATCH["participants"][participant]["stats"]["magicDamageDealtToChampions"])
            return damage
        if type == "total":
            damage =
self.MATCH["participants"][participant]["stats"]["trueDamageDealtToChampions"]
            damage +=
self.MATCH["participants"][participant]["stats"]["physicalDamageDealtToChampions"]
            damage +=
```

```
self.MATCH["participants"][participant]["stats"]["magicDamageDealtToChampions"]
            return damage

    #Sums up team magic, physical and true damage
    def getTeamDamage(self, team):
        #Goes through each player in team based on which team parameter was passed, stores total
in list and returns
        damage = [0,0,0]
        if team == "A":
            for participant in range(0, 5):
                damage[0] +=
self.MATCH["participants"][participant]["stats"]["trueDamageDealtToChampions"]
                damage[1] +=
self.MATCH["participants"][participant]["stats"]["physicalDamageDealtToChampions"]
                damage[2] +=
self.MATCH["participants"][participant]["stats"]["magicDamageDealtToChampions"]
            return damage
        if team == "B":
            for participant in range(5, 10):
                damage[0] +=
self.MATCH["participants"][participant]["stats"]["trueDamageDealtToChampions"]
                damage[1] +=
self.MATCH["participants"][participant]["stats"]["physicalDamageDealtToChampions"]
                damage[2] +=
self.MATCH["participants"][participant]["stats"]["magicDamageDealtToChampions"]
            return damage

    #Fetches values for objectives for each team
    def getDragons(self, team):
        return self.MATCH["teams"][team]["dragonKills"]
    def getTowers(self, team):
        return self.MATCH["teams"][team]["towerKills"]
    def getRiftHeralds(self, team):
        return self.MATCH["teams"][team]["riftHeraldKills"]
    def getBarons(self, team):
        return self.MATCH["teams"][team]["baronKills"]
    def getFirstBlood(self, team):
        return self.MATCH["teams"][team]["firstBlood"]
    def getFirstTower(self, team):
        return self.MATCH["teams"][team]["firstTower"]
    def getFirstInhibitor(self, team):
        return self.MATCH["teams"][team]["firstInhibitor"]
    def getFirstBaron(self, team):
        return self.MATCH["teams"][team]["firstBaron"]

    def getMVP(self, KDA, killParticipation, vision, playerDamage, KDAupper, CSperMin10,
CSperMin20, CSperMin30, killParticipationUpper, visionUpper,
                       playerDamageUpper, CSperMin10Upper, CSperMin20Upper, CSperMin30Upper):
        #MVP algorithm, is passed all data of each player and awards points for each player index
based on how they
        #preformed in each statistic. At end player with highest points is returned to be
displayed as MVP
        players = []
        for participant in range(0,10):
            points = 0
            name = self.getPlayerName(participant, "full")

            # KDA
            if participant == KDA.index(max(KDA)):
                if KDA[participant] > KDAupper:
                    points += 4
                else:
                    points += 2

            #Kill participation %
            if participant == killParticipation.index(max(killParticipation)):
                if killParticipation[participant] > killParticipationUpper:
                    points += 4
                else:
                    points += 2

            #Vision score
            if participant == vision.index(max(vision)):
                if vision[participant] > visionUpper:
                    points += 4
```

```
                else:
                    points += 2

            #CS points are lower as there are 3 similar types
            #CS 10min
            if participant == CSperMin10.index(max(CSperMin10)):
                if CSperMin10[participant] > CSperMin10Upper:
                    points += 2
                else:
                    points += 1

            #CS 20min
            if participant == CSperMin20.index(max(CSperMin20)):
                if CSperMin20[participant] > CSperMin20Upper:
                    points += 2
                else:
                    points += 1

            #CS 30min
            if participant == CSperMin30.index(max(CSperMin30)):
                if CSperMin30[participant] > CSperMin30Upper:
                    points += 2
                else:
                    points += 1

            if participant == playerDamage.index(max(playerDamage)):
                if playerDamage[participant] > playerDamageUpper:
                    points += 4
                else:
                    points += 2

            players.append([name,points])

        topIndex = 0
        #search for player with highest points
        for participant in range(0,10):
            if players[participant][1] > players[topIndex][1]:
                topIndex = participant
        return "MVP :" + players[topIndex][0]


    def displayMatch(self):
        #initialise window
        Stats = Tk()
        Stats.title("Lol Analytics - Your game")
        Stats.geometry("900x670")
        Stats.resizable(False, False)
        Stats.iconbitmap(r"anivia.ico")

        #places a background image
        backgroundStats = PhotoImage(file="background.png", master=Stats)
        backgroundStatsLabel = Label(Stats, image=backgroundStats)
        backgroundStatsLabel.image = backgroundStats
        backgroundStatsLabel.place(x=0, y=0, relwidth=1, relheight=1)

        #Finds total kills by each team to forward as parameter to kill participation function
        teamAkills = 0
        for participant in range(0, 5):
            teamAkills += self.getKills(participant)

        teamBkills = 0
        for participant in range(5, 10):
            teamBkills += self.getKills(participant)

        #Creating lists for various stats, index being the player
        #They will be used for finding means and standard deviations
        KDA = []
        for participant in range(0,10):
            KDA.append(self.getKDA(participant, "ratio"))

        killParticipation = []
        for participant in range(0, 5):
            killParticipation.append(self.getKillParticipation(participant,teamAkills))
        for participant in range(5,10):
            killParticipation.append(self.getKillParticipation(participant,teamBkills))
```

```
        vision = []
        for participant in range(0,10):
            vision.append(self.getVisionScore(participant))

        CSperMin10 = []
        for participant in range(0,10):
            CSperMin10.append(self.getCSperMin10(participant))

        CSperMin20 = []
        for participant in range(0,10):
            CSperMin20.append(self.getCSperMin20(participant))

        CSperMin30 = []
        for participant in range (0,10):
            CSperMin30.append(self.getCSperMin30(participant))

        playerDamage = []
        for participant in range (0,10):
            playerDamage.append(self.getPlayerDamage(participant, "total"))

        #Means
        KDAmean = mean(KDA)
        killParticipationMean = mean(killParticipation)
        visionMean = mean(vision)
        CSperMin10mean = mean(CSperMin10)
        CSperMin20mean = mean(CSperMin20)
        CSperMin30mean = mean(CSperMin30)
        playerDamageMean = mean(playerDamage)

        #Standard deviations
        KDAsd = numpy.std(KDA)
        killParticipationSD = numpy.std(killParticipation)
        visionSD = numpy.std(vision)
        CSperMin10sd = numpy.std(CSperMin10)
        CSperMin20sd = numpy.std(CSperMin20)
        CSperMin30sd = numpy.std(CSperMin30)
        playerDamageSD = numpy.std(playerDamage)

        #If a player performs greater than 1.5 standard deviations above or 1 sd below the mean,
        #it means they performed either exceptionally well or poorly in this statistic, this will
be marked red or green
        #Upper bounds (+1.5sd)
        KDAupper = KDAmean + 1.5 * KDAsd
        killParticipationUpper = killParticipationMean + 1.5 * killParticipationSD
        visionUpper = visionMean + 1.5 * visionSD
        CSperMin10Upper = CSperMin10mean + 1.5 * CSperMin10sd
        CSperMin20Upper = CSperMin20mean + 1.5 * CSperMin20sd
        CSperMin30Upper = CSperMin30mean + 1.5 * CSperMin30sd
        playerDamageUpper = playerDamageMean + 1.5 * playerDamageSD

        #Lower Bounds (-1sd)
        KDAlower = KDAmean - KDAsd
        killParticipationLower = killParticipationMean - killParticipationSD
        visionLower = visionMean - visionSD
        CSperMin10Lower = CSperMin10mean - CSperMin10sd
        CSperMin20Lower = CSperMin20mean - CSperMin20sd
        CSperMin30Lower = CSperMin30mean - CSperMin30sd
        playerDamageLower = playerDamageMean - playerDamageSD


        #Label generation and output
        #Titles of collumns
        statsLabel = Label(Stats, text="Stats", bg="black", fg="white", width=9, font="Helvetica
15 bold")
        KDAtitleLabel = Label(Stats, text="KDA", bg="black", fg="white", font="Helvetica 12 bold")
        killParticipationTitleLabel = Label(Stats, text="Kill participation", bg="black",
fg="white",
                                            font="Helvetica 10 bold")
        spellTitleLabel = Label(Stats, text="Spells", bg="black", fg="white", font="Helvetica 12
bold")
        itemTitleLabel = Label(Stats, text="Items", bg="black", fg="white", font="Helvetica 12
bold")
        visionScoreTitleLabel = Label(Stats, text="Vision score", bg="black", fg="white",
font="Helvetica 12 bold")
```

```
        CSperMinTitleLabel = Label(Stats, text="CS:10/20/30min", bg="black", fg="white",
font="Helvetica 11 bold")
        playerDamageTitleLabel = Label(Stats, text="Player damage\nbreakdown", bg="black",
fg="white", width=11,
                                       font="Helvetica 11 bold")
        objectiveTitleLabel = Label(Stats, text="Overall\nteam\ndamage\nand\nobjectives",
bg="black", fg="white",
                                    font="Helvetica 12 bold")
        #MVP label
        MVPLabel = Label(Stats, text=self.getMVP(KDA, killParticipation, vision, playerDamage,
KDAupper, CSperMin10,
                                                 CSperMin20, CSperMin30, killParticipationUpper,
visionUpper,
                                                 playerDamageUpper, CSperMin10Upper,
CSperMin20Upper, CSperMin30Upper),
                         bg="black", fg="white", font="Helvetica 15 bold")

        for participant in range(0,10):
            # Initialise labels
            #Name
            participantNameLabel = Label(Stats, text=self.getPlayerName(participant, "truncated"),
bg="black",
                                         fg="white", width=9, font="Helvetica 9 bold")

            #Champion Icon
            championIndex = self.getPlayerChampion(participant)
            championLocation = "champion/" + championIndex      #Index converted to path name
            champion = Image.open(championLocation)     #path is opened
            championResize = champion.resize((65, 65), Image.ANTIALIAS)     #Image processed
            championResize = ImageTk.PhotoImage(championResize, master=Stats)   #Tkinter png image
bugfix
            championLabel = Label(Stats, image=championResize, width=75)    #Image given label
            championLabel.image = championResize    #Tkinter has another bug which garbage
collects images
                                                    #made in functions, this keep a reference to
prevent that
            championLabel.configure(bg="black")

            #Position Icon
            positionLocation = self.getPlayerPosition(participant)
            position = Image.open(positionLocation)
            positionResize = position.resize((20,20), Image.ANTIALIAS)
            positionResize = ImageTk.PhotoImage(positionResize, master=Stats)
            positionLabel = Label(Stats, image=positionResize)
            positionLabel.image = positionResize
            positionLabel.configure(bg="black")

            #KDA label
            #Decides red/ green flag
            if KDA[participant] > KDAupper:
                KDAlabel = Label(Stats, text=self.getKDA(participant, "short string"), bg="black",
fg="#008008", font="Helvetica 10 bold")
            elif KDA[participant] < KDAlower:
                KDAlabel = Label(Stats, text=self.getKDA(participant, "short string"), bg="black",
fg="#d00000", font="Helvetica 10 bold")
            else:
                KDAlabel = Label(Stats, text=self.getKDA(participant, "short string"), bg="black",
fg="white", font="Helvetica 10 bold")

            #KillParticipationLabel
            #Decides red/ green flag
            if killParticipation[participant] > killParticipationUpper:
                killParticipationLabel = Label(Stats,
text=str(int(killParticipation[participant])) + "%", bg="black", fg="#008008", font="Helvetica 10
bold")
            elif killParticipation[participant] < killParticipationLower:
                killParticipationLabel = Label(Stats,
text=str(int(killParticipation[participant])) + "%", bg="black", fg="#d00000", font="Helvetica 10
bold")
            else:
                killParticipationLabel = Label(Stats,
text=str(int(killParticipation[participant])) + "%", bg="black", fg="white", font="Helvetica 10
bold")
```

```
            # Summoner spells 1 and 2
            spells = self.getSummonerSpells(participant)
            try:
                spell1 = "SummonerSpells/" + spells[0]
            except:
                spell1 = "SummonerSpells/0.png"  #forwards to placeholder image if no spell
            spell1 = Image.open(spell1)
            spell1Resize = spell1.resize((32, 32), Image.ANTIALIAS)
            spell1Resize = ImageTk.PhotoImage(spell1Resize,master=Stats)
            spell1Label = Label(Stats, image=spell1Resize)
            spell1Label.image = spell1Resize
            spell1Label.configure(bg="black")

            spells = self.getSummonerSpells(participant)
            try:
                spell2 = "SummonerSpells/" + spells[1]
            except:
                spell2 = "SummonerSpells/0.png"  # forwards to placeholder image if no spell
            spell2 = Image.open(spell2)
            spell2Resize = spell2.resize((32, 32), Image.ANTIALIAS)
            spell2Resize = ImageTk.PhotoImage(spell2Resize, master=Stats)
            spell2Label = Label(Stats, image=spell2Resize)
            spell2Label.image = spell2Resize
            spell2Label.configure(bg="black")

            # Item Labels: fetches item indexes, gets images, generates labels and output
            items = self.getItems(participant)
            if participant <=4:
                for itemIndex in range(7):
                    # Make Label
                    item = "item/" + str(items[itemIndex]) + ".png"
                    if item != "item/0.png": #check if there is a item so that place holder is not
used, this will make
                                             #the item frame seem transparent
                        item = Image.open(item)
                        itemResize = item.resize((32, 32), Image.ANTIALIAS)
                        itemResize = ImageTk.PhotoImage(itemResize, master=Stats)
                        itemLabel = Label(Stats, image=itemResize)
                        itemLabel.image = itemResize
                        itemLabel.configure(bg="black")

                        # Output Label
                        if itemIndex < 2:
                            if itemIndex == 0:
                                itemLabel.grid(column=participant, row=5, sticky=W, padx=4)
                            else:
                                itemLabel.grid(column=participant, row=5, sticky=E, padx=4)
                        elif 2 <= itemIndex < 4:
                            if itemIndex == 2:
                                itemLabel.grid(column=participant, row=6, sticky=W, padx=4)
                            else:
                                itemLabel.grid(column=participant, row=6, sticky=E, padx=4)
                        elif 4 <= itemIndex < 6:
                            if itemIndex == 4:
                                itemLabel.grid(column=participant, row=7, sticky=W, padx=4)
                            else:
                                itemLabel.grid(column=participant, row=7, sticky=E, padx=4)
                        elif itemIndex == 6:
                            itemLabel.grid(column=participant, row=8, sticky=W, padx=4)
            else:
                for itemIndex in range(7):
                    # Make Label
                    item = "item/" + str(items[itemIndex]) + ".png"
                    if item != "item/0.png":
                        item = Image.open(item)
                        itemResize = item.resize((32, 32), Image.ANTIALIAS)
                        itemResize = ImageTk.PhotoImage(itemResize, master=Stats)
                        itemLabel = Label(Stats, image=itemResize)
                        itemLabel.image = itemResize
                        itemLabel.configure(bg="black")

                        # Output Label
                        if itemIndex < 2:
                            if itemIndex == 0:
                                itemLabel.grid(column=participant+1, row=5, sticky=W, padx=4)
```

```
                            else:
                                itemLabel.grid(column=participant+1, row=5, sticky=E, padx=4)
                        elif 2 <= itemIndex < 4:
                            if itemIndex == 2:
                                itemLabel.grid(column=participant+1, row=6, sticky=W, padx=4)
                            else:
                                itemLabel.grid(column=participant+1, row=6, sticky=E, padx=4)
                        elif 4 <= itemIndex < 6:
                            if itemIndex == 4:
                                itemLabel.grid(column=participant+1, row=7, sticky=W, padx=4)
                            else:
                                itemLabel.grid(column=participant+1, row=7, sticky=E, padx=4)
                        elif itemIndex == 6:
                            itemLabel.grid(column=participant+1, row=8, sticky=E, padx=4)

            #Vision Label
            #Decides flag
            if vision[participant] > visionUpper:
                visionLabel = Label(Stats, text=vision[participant], bg="black", fg="#008008",
font="Helvetica 12 bold")
            elif vision[participant] < visionLower:
                visionLabel = Label(Stats, text=vision[participant], bg="black", fg="#d00000",
font="Helvetica 12 bold")
            else:
                visionLabel = Label(Stats, text=vision[participant], bg="black", fg="white",
font="Helvetica 12 bold")

            # CS Labels
            #Decides flags for each 10/20/30 min intervals
            if CSperMin10[participant] > CSperMin10Upper:
                CSperMin10Label = Label(Stats, text=str(CSperMin10[participant]) + "/",
bg="black", fg="#008008",
                                        font="Helvetica 10 bold")
            elif CSperMin10[participant] < CSperMin10Lower:
                CSperMin10Label = Label(Stats, text=str(CSperMin10[participant]) + "/",
bg="black", fg="#d00000",
                                        font="Helvetica 9 bold")
            else:
                CSperMin10Label = Label(Stats, text=str(CSperMin10[participant]) + "/",
bg="black", fg="white",
                                        font="Helvetica 9 bold")

            if CSperMin20[participant] > CSperMin20Upper:
                CSperMin20Label = Label(Stats, text=str(CSperMin20[participant]), bg="black",
fg="#008008",
                                        font="Helvetica 10 bold")
            elif CSperMin20[participant] < CSperMin20Lower:
                CSperMin20Label = Label(Stats, text=str(CSperMin20[participant]), bg="black",
fg="#d00000",
                                        font="Helvetica 9 bold")
            else:
                CSperMin20Label = Label(Stats, text=str(CSperMin20[participant]), bg="black",
fg="white",
                                        font="Helvetica 9 bold")

            if CSperMin30[participant] > CSperMin30Upper:
                CSperMin30Label = Label(Stats, text="/" + str(CSperMin30[participant]),
bg="black", fg="#008008",
                                        font="Helvetica 10 bold")
            elif CSperMin30[participant] < CSperMin30Lower:
                CSperMin30Label = Label(Stats, text="/" + str(CSperMin30[participant]),
bg="black", fg="#d00000",
                                        font="Helvetica 9 bold")
            else:
                CSperMin30Label = Label(Stats, text="/" + str(CSperMin30[participant]),
bg="black", fg="white",
                                        font="Helvetica 9 bold")

            #Player damage breakdown pie chart and number
            damagePieChart = plt.figure(figsize=[75, 75], dpi=1)    #pie chart size
            damagePieChart.patch.set_facecolor('xkcd:black')

            slices = self.getPlayerDamage(participant, "list")
            colors = ["white", "#b20000", "#50A6C2"]
            #applies parameters so that piechart has black background, and white,red,blue slices
```

```
            #in a clockwise order because it looks better
            plt.pie(slices, startangle=90, colors=colors, counterclock=False)
            plt.tight_layout()

            # ports matplotlib piechart to tkinter
            damagePieChartLabel = FigureCanvasTkAgg(damagePieChart, master=Stats)

            #Player damage total
            #Decides flag
            if playerDamage[participant] > playerDamageUpper:
                playerDamageLabel = Label(Stats, text="{:,}".format(playerDamage[participant]),
bg="black", fg="#008008",
                                          font="Helvetica 10 bold")
            elif playerDamage[participant] < playerDamageLower:
                playerDamageLabel = Label(Stats, text="{:,}".format(playerDamage[participant]),
bg="black", fg="#d00000",
                                          font="Helvetica 10 bold")
            else:
                playerDamageLabel = Label(Stats, text="{:,}".format(playerDamage[participant]),
bg="black", fg="white",
                                          font="Helvetica 10 bold")

            #Objectives
            teamAdragonLabel = Label(Stats, text="Dragons: " + str(self.getDragons(0)),
bg="black", fg="white",
                                     font="Helvetica 12 bold")
            teamBdragonLabel = Label(Stats, text="Dragons: " + str(self.getDragons(1)),
bg="black", fg="white",
                                     font="Helvetica 12 bold")
            teamAtowerLabel = Label(Stats, text="Towers: " + str(self.getTowers(0)), bg="black",
fg="white",
                                    font="Helvetica 12 bold")
            teamBtowerLabel = Label(Stats, text="Towers: " + str(self.getTtowers(1)), bg="black",
fg="white",
                                    font="Helvetica 12 bold")
            teamAheraldLabel = Label(Stats, text="Heralds: " + str(self.getRiftHeralds(0)),
bg="black", fg="white",
                                     font="Helvetica 12 bold")
            teamBheraldLabel = Label(Stats, text="Heralds: " + str(self.getRiftHeralds(1)),
bg="black", fg="white",
                                     font="Helvetica 12 bold")
            teamAbaronLabel = Label(Stats, text="Barons: " + str(self.getBarons(0)), bg="black",
fg="white",
                                    font="Helvetica 12 bold")
            teamBbaronLabel = Label(Stats, text="Barons: " + str(self.getBarons(1)), bg="black",
fg="white",
                                    font="Helvetica 12 bold")

            if self.getFirstBlood(0) == True:
                teamAfirstBloodLabel = Label(Stats, text="First Blood", bg="black", fg="white",
                                             font="Helvetica 12 bold")
            elif self.getFirstBlood(1) == True:
                teamBfirstBloodLabel = Label(Stats, text="First Blood", bg="black", fg="white",
                                             font="Helvetica 12 bold")

            if self.getFirstTower(0) == True:
                teamAfirstTowerLabel = Label(Stats, text="First Tower", bg="black", fg="white",
                                             font="Helvetica 12 bold")
            elif self.getFirstTower(1) == True:
                teamBfirstTowerLabel = Label(Stats, text="First Tower", bg="black", fg="white",
                                             font="Helvetica 12 bold")

            if self.getFirstInhibitor(0) == True:
                teamAfirstInhibitorLabel = Label(Stats, text="First Inhibitor", bg="black",
fg="white",
                                                 font="Helvetica 12 bold")
            elif self.getFirstInhibitor(1) == True:
                teamBfirstInhibitorLabel = Label(Stats, text="First Inhibitor", bg="black",
fg="white",
                                                 font="Helvetica 12 bold")


            if self.getFirstBaron(0) == True:
                teamAfirstBaronLabel = Label(Stats, text="First Baron", bg="black", fg="white",
                                             font="Helvetica 12 bold")
```

```
            elif self.getFirstBaron(1) == True:
                teamBfirstBaronLabel = Label(Stats, text="First Baron", bg="black", fg="white",
                                             font="Helvetica 12 bold")

            #Overall team damage charts
            if participant == 1:
                teamDamageA = plt.figure(figsize=(200, 200), dpi=1)
                teamDamageA.patch.set_facecolor('xkcd:black')
                slices = self.getTeamDamage("A")
                plt.pie(slices, startangle=90, colors=colors, counterclock=False)
                plt.tight_layout()
                teamDamageALabel = FigureCanvasTkAgg(teamDamageA, master=Stats)

                teamDamageB = plt.figure(figsize=(200, 200), dpi=1)
                teamDamageB.patch.set_facecolor('xkcd:black')
                slices = self.getTeamDamage("B")
                plt.pie(slices, startangle=90, colors=colors, counterclock=False)
                plt.tight_layout()
                teamDamageBLabel = FigureCanvasTkAgg(teamDamageB, master=Stats)

            # display labels LHS
            if participant == 1:
                #output team/final data
                counterA = 0    # this is to make objectives look like they are stacked evenly
                counterB = 0

                teamDamageALabel.get_tk_widget().grid(column=0, row=12, columnspan=3, rowspan=8)
                teamDamageBLabel.get_tk_widget().grid(column=8, row=12, columnspan=3, rowspan=8)

                teamAtowerLabel.grid(column=3, row=12, columnspan=2)
                teamBtowerLabel.grid(column=6, row=12, columnspan=2)

                teamAdragonLabel.grid(column=3, row=13, columnspan=2)
                teamBdragonLabel.grid(column=6, row=13, columnspan=2)

                teamAheraldLabel.grid(column=3, row=14, columnspan=2)
                teamBheraldLabel.grid(column=6, row=14, columnspan=2)

                teamAbaronLabel.grid(column=3, row=15, columnspan=2)
                teamBbaronLabel.grid(column=6, row=15, columnspan=2)

                #Sometimes not all objectives will be taken if a search for them is attempted the
program will crash
                #because there is no such index, so to prevent this I use exceptions
                try:
                    teamAfirstBloodLabel.grid(column=3, row=16, columnspan=2)
                    counterA += 1
                except:
                    None

                try:
                    teamBfirstBloodLabel.grid(column=6, row=16, columnspan=2)
                    counterB += 1
                except:
                    None

                try:
                    teamAfirstTowerLabel.grid(column=3, row=16 + counterA, columnspan=2)
                    counterA += 1
                except:
                    None

                try:
                    teamBfirstTowerLabel.grid(column=6, row=16 + counterB, columnspan=2)
                    counterB += 1
                except:
                    None

                try:
                    teamAfirstInhibitorLabel.grid(column=3, row=16 + counterA, columnspan=2)
                    counterA += 1
                except:
                    None

                try:
```

```
                    teamBfirstInhibitorLabel.grid(column=6, row=16 + counterB, columnspan=2)
                    counterB += 1
                except:
                    None

                try:
                    teamAfirstBaronLabel.grid(column=3, row=16 + counterA, columnspan=2)
                    counterA += 1
                except:
                    None

                try:
                    teamBfirstBaronLabel.grid(column=6, row=16 + counterB, columnspan=2)
                    counterB += 1
                except:
                    None

            #Labels are placed on window
            if participant <= 4:
                #display labels LHS
                championLabel.grid(column=participant, row=0)
                positionLabel.grid(column=participant, row=0, sticky=SE)
                participantNameLabel.grid(column=participant, row=1)
                KDAlabel.grid(column=participant, row=2)
                killParticipationLabel.grid(column=participant, row=3)
                spell1Label.grid(column=participant, row=4, sticky=W, padx=4)
                spell2Label.grid(column=participant, row=4, sticky=E, padx=4)
                visionLabel.grid(column=participant, row=8, sticky=E, padx=9)
                CSperMin10Label.grid(column=participant, row=9, sticky=W, padx=4)
                CSperMin20Label.grid(column=participant, row=9)
                CSperMin30Label.grid(column=participant, row=9,sticky=E, padx=4)
                damagePieChartLabel.get_tk_widget().grid(column=participant, row=10)
                playerDamageLabel.grid(column=participant, row=11)



            # display labels center
            if participant == 5:
                statsLabel.grid(column=participant, row=0)
                spellTitleLabel.grid(column=participant, row=4)
                KDAtitleLabel.grid(column=participant, row=2)
                killParticipationTitleLabel.grid(column=participant, row=3)
                itemTitleLabel.grid(column=participant, row=6)
                visionScoreTitleLabel.grid(column=participant, row=8)
                CSperMinTitleLabel.grid(column=participant, row=9)
                playerDamageTitleLabel.grid(column=participant, row=10, sticky=S)
                objectiveTitleLabel.grid(column=participant, row=11, rowspan=8)
                MVPLabel.grid(column=participant-1, row=20, columnspan=3)


            # display labels RHS
            if participant >= 5:
                championLabel.grid(column=participant + 1, row=0)
                positionLabel.grid(column=participant + 1, row=0, sticky=SW)
                participantNameLabel.grid(column=participant + 1, row=1)
                KDAlabel.grid(column=participant + 1, row=2)
                killParticipationLabel.grid(column=participant + 1, row=3)
                spell1Label.grid(column=participant + 1, row=4, sticky=W, padx=5)
                spell2Label.grid(column=participant + 1, row=4, sticky=E, padx=5)
                visionLabel.grid(column=participant + 1, row=8, sticky=W, padx=10)
                CSperMin10Label.grid(column=participant + 1, row=9, sticky=W, padx=4)
                CSperMin20Label.grid(column=participant + 1, row=9)
                CSperMin30Label.grid(column=participant + 1, row=9, sticky=E, padx=4)
                damagePieChartLabel.get_tk_widget().grid(column=participant + 1, row=10)
                playerDamageLabel.grid(column=participant + 1, row=11)

def openAPIlogin():
    #Guides user to website to get API key if they click the get key button
    webbrowser.open("https://auth.riotgames.com/login#client_id=riot-developer-
portal&redirect_uri=https%3A%2F%2Fd"
                    "eveloper.riotgames.com%2Foauth2-
callback&response_type=code&scope=openid%20email%20summoner")

def searchPlayer():
    #Fetches data in input boxes
```

```
    APIkey = APIinput.get()
    name = idInput.get()
    region = regionInput.get()
    decideSave()

    print("\nFetching your ID...")

    #checks if there is a region error
    try:
        SUMMONER = requests.get("https://" + region +
".api.riotgames.com/lol/summoner/v4/summoners/by-name/" + name + "?api_key=" + APIkey)
    except:
        errorMessage("region")
        return None
    SUMMONER = SUMMONER.json()  # converts json to py dictionary

    #Checks for username error
    try:
        if SUMMONER["status"]["message"] == "Forbidden":
            errorMessage("API key")
            return None
        if SUMMONER["status"]["message"] == "Data not found - summoner not found":
            errorMessage("username")
            return None
    except: None

    #Fetches data from SUMMONER dictionary
    encryptedPlayerID = SUMMONER["id"]
    accountID = SUMMONER["accountId"]
    iconID = str(SUMMONER["profileIconId"])
    level = SUMMONER["summonerLevel"]
    #This corrects the name to have the correct capitalisation
    #as in other API inputs it is case sensetive
    correctName = SUMMONER["name"]

    displayProfile(region, encryptedPlayerID, APIkey, iconID, level, correctName)
    getMatches(region, accountID, APIkey, correctName)
    displayHistory(correctName)

#Profile Output
def displayProfile(region, encryptedPlayerID, APIkey, iconID, level, name):
    print("Generating " + name + "'s profile...")

    #Fetches user's profile data from API
    PROFILE = requests.get("https://" + region + ".api.riotgames.com/lol/league/v4/entries/by-
summoner/"
                           + encryptedPlayerID + "?api_key=" + APIkey)
    PROFILE = PROFILE.json()

    # Clears previous frame if exists, else: createse new frame to save memory and prevent
unwanted overlapping
    try:
        root.Profile.grid_forget()
    except:
        None

    #Test if player has a rank/s and which index they are saved to as it can vary
    solo = False
    flex = False
    try:
        if PROFILE[0]["queueType"] == 'RANKED_SOLO_5x5':
            solo = True
            soloIndex = 0
    except:
        None

    try:
        if PROFILE[1]["queueType"] == 'RANKED_SOLO_5x5':
            solo = True
            soloIndex = 1
    except:
        None
    try:
        if PROFILE[0]["queueType"] == 'RANKED_FLEX_SR':
            flex = True
```

```
            flexIndex = 0
    except:
        None

    try:
        if PROFILE[1]["queueType"] == 'RANKED_FLEX_SR':
            flex = True
            flexIndex = 1
    except:
        None

    #Profile frame
    Profile.grid(row=0, column=6, rowspan=5, columnspan=5)
    Profile.configure(bg="black")

    #Profile Icon
    iconLocation ="profileicon/" + iconID + ".png"
    icon = Image.open(iconLocation)
    iconResize = icon.resize((110, 110), Image.ANTIALIAS)
    iconResize = ImageTk.PhotoImage(iconResize)
    iconLabel = Label(Profile, image=iconResize)
    iconLabel.img = iconResize
    iconLabel.configure(bg="black")

    #Solo Rank Image
    if solo == True:
        soloTier = str(PROFILE[soloIndex]["tier"].lower())
        soloTier = soloTier.capitalize()
        soloTier = "Emblem_" + soloTier + ".png"
        soloTier ="Ranks/" + soloTier
        soloTier = Image.open(soloTier)
        soloTierResize = soloTier.resize((50, 50), Image.ANTIALIAS)
        soloTierResize = ImageTk.PhotoImage(soloTierResize)
        soloTierLabel = Label(Profile, image=soloTierResize)
        soloTierLabel.img = soloTierResize
        soloTierLabel.configure(bg="black")
    else:
        #Uses placeholder
        soloTier = Image.open("Ranks/Emblem_Unranked.png")
        soloTierResize = soloTier.resize((50, 50), Image.ANTIALIAS)
        soloTierResize = ImageTk.PhotoImage(soloTierResize)
        soloTierLabel = Label(Profile, image=soloTierResize)
        soloTierLabel.img = soloTierResize
        soloTierLabel.configure(bg="black")

    #Flex Rank Image
    if flex == True:
        flexTier = str(PROFILE[flexIndex]["tier"].lower())
        flexTier = flexTier.capitalize()
        flexTier = "Emblem_" + flexTier + ".png"
        flexTier ="Ranks/" + flexTier
        flexTier = Image.open(flexTier)
        flexTierResize = flexTier.resize((50, 50), Image.ANTIALIAS)
        flexTierResize = ImageTk.PhotoImage(flexTierResize)
        flexTierLabel = Label(Profile, image=flexTierResize)
        flexTierLabel.img = flexTierResize
        flexTierLabel.configure(bg="black")
    else:
        #Uses placeholder
        flexTier = Image.open("Ranks/Emblem_Unranked.png")
        flexTierResize = flexTier.resize((50, 50), Image.ANTIALIAS)
        flexTierResize = ImageTk.PhotoImage(flexTierResize)
        flexTierLabel = Label(Profile, image=flexTierResize)
        flexTierLabel.img = flexTierResize
        flexTierLabel.configure(bg="black")

    #Other Profile items
    #LP, wins, loses, winrate
    nameLabel = Label(Profile, text=name, width=13)
    nameLabel.configure(bg="black", fg="#dddddd", font="Helvetica 18 bold")
    levelLabel = Label(Profile, text=level)
    levelLabel.configure(bg="black", fg="#dddddd")
    if solo == True:
        soloRankLabel = Label(Profile, text=(PROFILE[soloIndex]["rank"] +" " +
str(PROFILE[soloIndex]["leaguePoints"]) + "LP"))
```

```
        soloRankLabel.configure(bg="black", fg="#dddddd", font="Verdana 8")
        soloWinsLossLabel = Label(Profile,
                                  text=("Wins:" + str(PROFILE[soloIndex]["wins"]) + "  Losses:" +
str(PROFILE[soloIndex]["losses"])))
        soloWinsLossLabel.configure(bg="black", fg="#dddddd")
        #winrate calculation
        soloWinRateLabel = Label(Profile, text=("Win rate: " + str(int((100 /
(PROFILE[soloIndex]["wins"] + PROFILE[soloIndex]["losses"])) * PROFILE[soloIndex]["wins"])) +
"%"))
        soloWinRateLabel.configure(bg="black", fg="#dddddd")
    else:
        soloWinsLossLabel = Label(Profile, text="Wins:N/A  Losses:N/A")
        soloWinsLossLabel.configure(bg="black", fg="#dddddd")
        soloWinRateLabel = Label(Profile, text=("Win rate: N/A"))
        soloWinRateLabel.configure(bg="black", fg="#dddddd")
        soloRankLabel = Label(Profile, text=("[UNRANKED]"))
        soloRankLabel.configure(bg="black", fg="#dddddd", font="Verdana 7")
    if flex == True:
        flexRankLabel = Label(Profile, text=(PROFILE[flexIndex]["rank"] + " " +
str(PROFILE[flexIndex]["leaguePoints"]) + "LP"))
        flexRankLabel.configure(bg="black", fg="#dddddd", font="Verdana 8")
        flexWinsLossLabel = Label(Profile, text=("Wins:" + str(PROFILE[flexIndex]["wins"]) + "
Losses:" + str(PROFILE[flexIndex]["losses"])))
        flexWinsLossLabel.configure(bg="black", fg="#dddddd")
        #winrate calculation
        flexWinRateLabel = Label(Profile, text=("Win rate: " + str(int((100 /
(PROFILE[flexIndex]["wins"] + PROFILE[flexIndex]["losses"])) * PROFILE[flexIndex]["wins"])) +
"%"))
        flexWinRateLabel.configure(bg="black", fg="#dddddd")
    else:
        flexWinsLossLabel = Label(Profile, text="Wins:N/A  Losses:N/A")
        flexWinsLossLabel.configure(bg="black", fg="#dddddd")
        flexWinRateLabel = Label(Profile, text=("Win rate: N/A"))
        flexWinRateLabel.configure(bg="black", fg="#dddddd")
        flexRankLabel = Label(Profile, text=("[UNRANKED]"))
        flexRankLabel.configure(bg="black", fg="#dddddd", font="Verdana 7")

    soloLabel = Label(Profile, text="SOLO/DUO")
    soloLabel.configure(bg="black", fg="#dddddd")

    flexLabel = Label(Profile, text="FLEX")
    flexLabel.configure(bg="black", fg="#dddddd")

    #Profile label locations and placement
    Profile.grid_columnconfigure(6, minsize=30)
    iconLabel.grid(row=0, column=0, rowspan=5)
    soloRankLabel.grid(row=4, column=1)
    soloLabel.grid(row=2, column=2)
    soloTierLabel.grid(row=2, column=1, rowspan=2)
    soloWinsLossLabel.grid(row=3, column=2)
    soloWinRateLabel.grid(row=4, column=2)

    flexRankLabel.grid(row=4, column=4)
    flexWinsLossLabel.grid(row=3, column=5)
    flexWinRateLabel.grid(row=4, column=5)
    flexLabel.grid(row=2,column=5)
    flexTierLabel.grid(row=2, column=4, rowspan=2)

    nameLabel.grid(row=0, column=1, columnspan=2, sticky=W)
    levelLabel.grid(row=4, column=0, sticky=S)

def getMatches(region, accountID, APIkey, name):
    #Fetches match history
    MATCHLIST = requests.get("https://" + region + ".api.riotgames.com/lol/match/v4/"
                                                   "matchlists/by-account/" + accountID +
"?api_key=" + APIkey)
    MATCHLIST = MATCHLIST.json()
    MATCHES.clear()
    print("Getting " + name + "'s matches...")
    #Creates and stores each match class into MATCHES list
    for game in range(10):
        MATCHES.append(Match(MATCHLIST["matches"][game], region, APIkey, game))

def displayHistory(name):
    # Clears previous frame if exists, to save memory and prevent unwanted overlapping
```

```
    try:
        root.History.gird_forget()
    except:
        None

    History.grid(row=5, column=0,rowspan=16, columnspan=9, sticky=S)
    History.configure(bg="black")
    #History Labels
    historyLabel = Label(History, text=("Your match history:"))
    historyLabel.configure(bg="black", fg="white", font=("Helvetica 18 bold", 14))
    counter = 0

    for match in MATCHES:
        counter += 1

        participant = match.getParticipant(name)
        champion = match.getPlayerChampion(participant)

        #Champion Label
        championLocation = "champion/" + champion
        championImage = Image.open(championLocation)
        championResize = championImage.resize((45, 45), Image.ANTIALIAS)
        championResize = ImageTk.PhotoImage(championResize)
        championLabel = Label(History, image=championResize)
        championLabel.image = championResize
        championLabel.configure(bg="black")

        #Match Button
        colourKey = match.getWinLossColour(participant)

        KDA = match.getKDA(participant, "string")
        matchType = match.getMatchType()

        #recursion cannot be used  (another tKinter bug) as buttons cannot store indexes or
variables locally, only calls to other functions or strings can be made/stored
        #lambda used to prevent the command from being run while it is being initialised in the
button

        matchButton0 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[0].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton1 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[1].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton2 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[2].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton3 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[3].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton4 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[4].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton5 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[5].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton6 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[6].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton7 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[7].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton8 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[8].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")
        matchButton9 = Button(History, text=KDA + "\n" + matchType,command=lambda:
MATCHES[9].displayMatch(), width=16, height=2, bg=colourKey, fg="#dddddd", font="Helvetica 9
bold")

        #Summoner spells
        spells = match.getSummonerSpells(participant)
        spellLocation = "SummonerSpells/" + spells[0]
        spellImage = Image.open(spellLocation)
        spell1Resize = spellImage.resize((19, 19), Image.ANTIALIAS)
        spell1Resize = ImageTk.PhotoImage(spell1Resize)
```

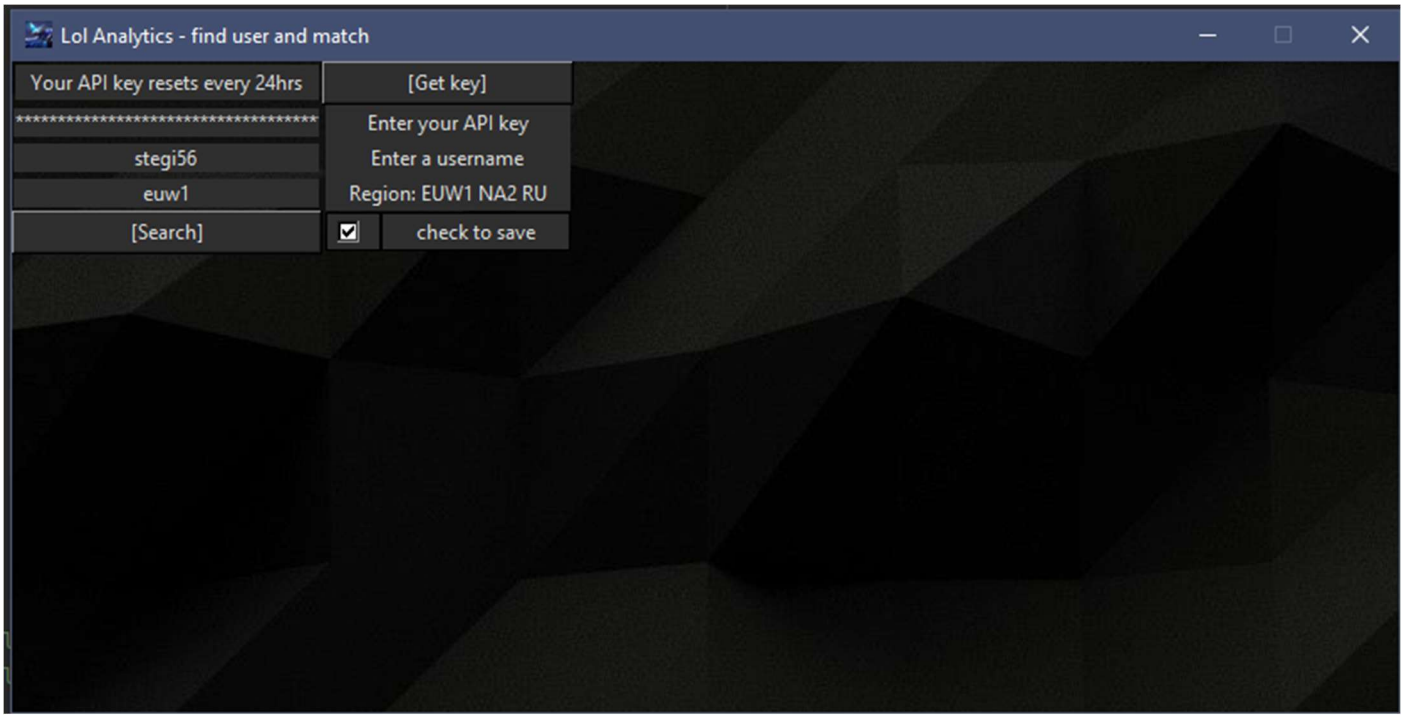
```
        spell1Label = Label(History, image=spell1Resize)
        spell1Label.image = spell1Resize
        spell1Label.configure(bg="black")

        spellLocation = "SummonerSpells/" + spells[1]
        spellImage = Image.open(spellLocation)
        spell2Resize = spellImage.resize((19, 19), Image.ANTIALIAS)
        spell2Resize = ImageTk.PhotoImage(spell2Resize)
        spell2Label = Label(History, image=spell2Resize)
        spell2Label.image = spell2Resize
        spell2Label.configure(bg="black")

        # Item Labels
        items = match.getItems(participant)
        for itemIndex in range(7):
            # Make Label
            itemLocation = "item/" + str(items[itemIndex]) + ".png"
            itemImage = Image.open(itemLocation)
            itemResize = itemImage.resize((27, 27), Image.ANTIALIAS)
            itemResize = ImageTk.PhotoImage(itemResize)
            itemLabel = Label(History, image=itemResize)
            itemLabel.image = itemResize
            itemLabel.configure(bg="black")

            # Output Label
            if counter <= 5:
                itemLabel.grid(row=(counter*2), column=8+itemIndex*2, columnspan=2, rowspan=2)
            else:
                itemLabel.grid(row=(counter*2)-10, column=(itemIndex*2)+33, columnspan=2,
rowspan=2)

        #History Button Locations
        #Problem of buttons not remembering indexes prevents use of recursion here as well
        if counter <= 5:
            championLabel.grid(row=counter*2, column=0, rowspan=2)
            if counter == 1:
                matchButton0.grid(row=counter*2, column=2, columnspan=5, rowspan=2)
            if counter == 2:
                matchButton1.grid(row=counter*2, column=2, columnspan=5, rowspan=2)
            if counter == 3:
                matchButton2.grid(row=counter*2, column=2, columnspan=5, rowspan=2)
            if counter == 4:
                matchButton3.grid(row=counter*2, column=2, columnspan=5, rowspan=2)
            if counter == 5:
                matchButton4.grid(row=counter*2, column=2, columnspan=5, rowspan=2)
            spell1Label.grid(row=counter*2, column=7, sticky="S")
            spell2Label.grid(row=(counter*2)+1, column=7, sticky="N")
        else:
            championLabel.grid(row=(counter*2)-10, column=24, rowspan=2)
            if counter == 6:
                matchButton5.grid(row=(counter*2)-10, column=26, columnspan=5, rowspan=2)
            if counter == 7:
                matchButton6.grid(row=(counter*2)-10, column=26, columnspan=5, rowspan=2)
            if counter == 8:
                matchButton7.grid(row=(counter*2)-10, column=26, columnspan=5, rowspan=2)
            if counter == 9:
                matchButton8.grid(row=(counter*2)-10, column=26, columnspan=5, rowspan=2)
            if counter == 10:
                matchButton9.grid(row=(counter*2)-10, column=26, columnspan=5, rowspan=2)
            spell1Label.grid(row=(counter*2)-10, column=31, sticky="S")
            spell2Label.grid(row=(counter*2)-9, column=31, sticky="N")

    #History Title Location
    historyLabel.grid(row=0, column=0, columnspan=5)


def getSave():
    #fetches save file and returns a list of content (APIkey, username and region)
    save = []
    with open("saveSearch.txt") as file:
        for line in file:
            save.append(line.replace("\n", ""))
    return save

def decideSave():
```

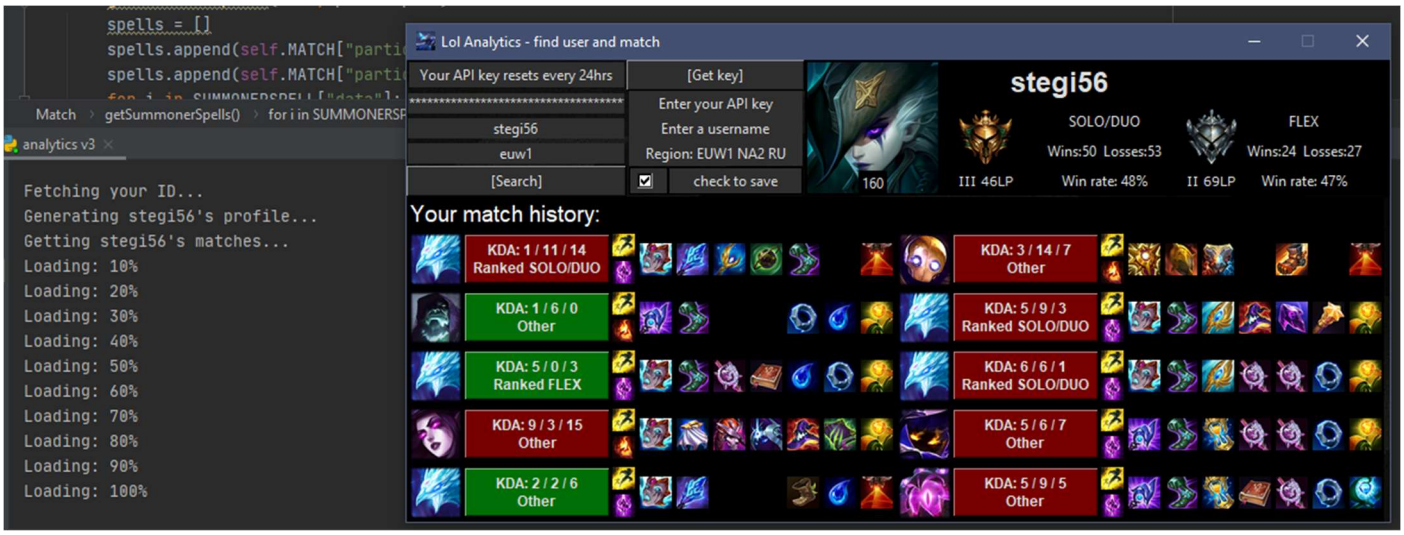
```
    #Every time a search is made the checkbox is checked on whether the program should save or
clear the save
    if boxStatus.get() == True:
       saveSearch()
    else:
        clearSave()

def saveSearch():
    #To save the search the program fetches the input boxes in list form and stores each item on
its own line
    clearSave()
    save = [APIinput.get(),idInput.get(),regionInput.get()]
    with open('saveSearch.txt', 'w') as file:
        for item in save:
            file.write("%s\n" % item)

def clearSave():
    #wipe the txt file
    file = open("saveSearch.txt", "w")
    file.truncate(0)

def autoFill():
    #Attempt to fetch data from txt file to autofill input boxes
    try:
        save = getSave()
        APIinput.insert(0, str(save[0]))
        idInput.insert(0, str(save[1]))
        regionInput.insert(0, str(save[2]))
        saveInputs.select()
    except:
        saveInputs.deselect()

#calls for windows to display notifications for user errors
def errorMessage(type):
    if type == "API key":
        ctypes.windll.user32.MessageBoxW(0, "Enter a valid API key.\nClick on the [GET KEY] button
for a new one.", "ERROR", 0)
    elif type == "username":
        ctypes.windll.user32.MessageBoxW(0, "Enter a valid username.", "ERROR", 0)
    elif type == "region":
        ctypes.windll.user32.MessageBoxW(0, "Enter a valid region.", "ERROR", 0)

#initialises the GUI, prevents adjustment of box size, sets icon to one in folder and sets title
root = Tk()
root.title("Lol Analytics - find user and match")
root.geometry("845x388")
root.resizable(False, False)
root.iconbitmap(r"anivia.ico")

#Background
background = PhotoImage(file="background.png")
Label(root, image = background).place(relwidth=1, relheight=1)

#Frames
#Makes frames for each group of items in GUI to make them easier to manipulate
Profile = Frame()
History = Frame()
SaveFrame = Frame(bg="black")

#active items (buttons and checkbox)
getKey = Button(root, text="[Get key]", command=openAPIlogin, width=20, bg="#2f2f2f",
fg="#dddddd")
getInputs = Button(root, text="[Search]", command=searchPlayer, width=25, bg="#2f2f2f",
fg="#dddddd")
boxStatus = BooleanVar()
saveInputs = Checkbutton(SaveFrame, bg="#2f2f2f", onvalue=True, offvalue=False,
variable=boxStatus, width=1, bd=0)

#labels
SaveFrame.grid(row=4, column=3, columnspan=2)
APIlabel1 = Label(root, width=25, text="Your API key resets every 24hrs", bg="#2f2f2f",
fg="#dddddd")
APIlabel2 = Label(root, text="Enter your API key", width=20, bg="#2f2f2f", fg="#dddddd")
idLabel = Label(root, text="Enter a username", width=20, bg="#2f2f2f", fg="#dddddd")
regionLabel = Label(root, text="Region: EUW1 NA2 RU", width=20, bg="#2f2f2f", fg="#dddddd")
```

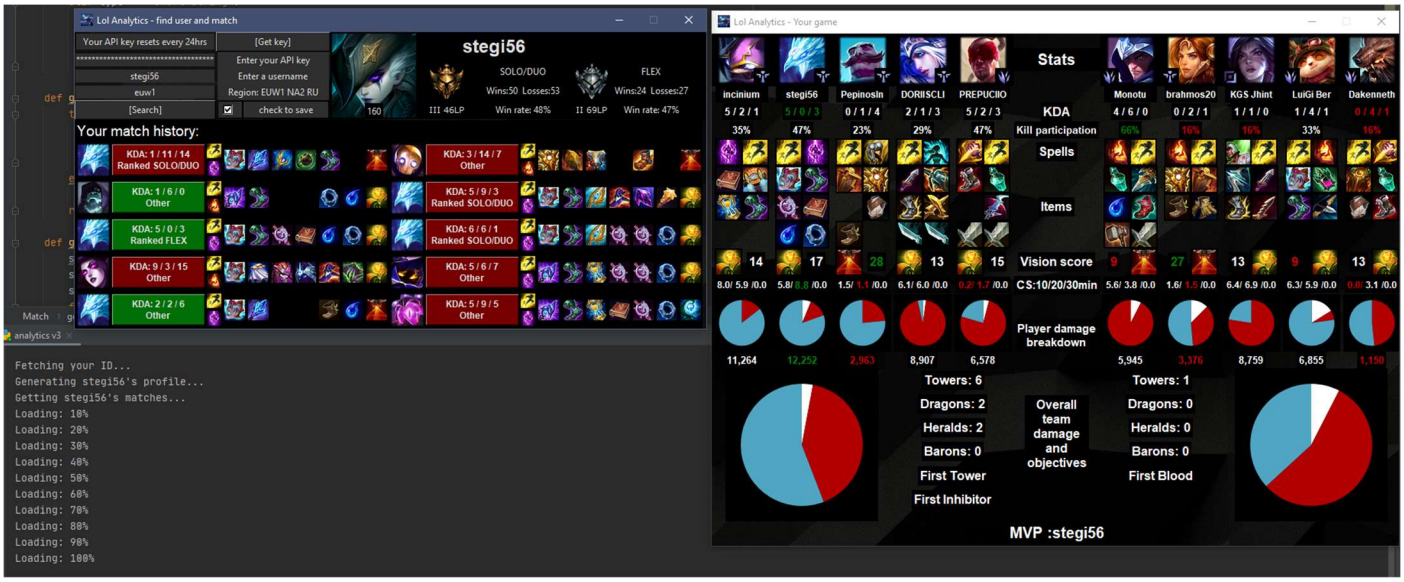
```
checkLabel = Label(SaveFrame, text="check to save", width=15, bg="#2f2f2f", fg="#dddddd")

#inputs
APIinput = Entry(root, show="*", width=30, justify="center", bg="#2f2f2f", fg="#dddddd", bd=0)
idInput = Entry(root, width=30, justify="center", bg="#2f2f2f", fg="#dddddd", bd=0)
regionInput = Entry(root, width=30, justify="center", bg="#2f2f2f", fg="#dddddd", bd=0)

#positions
APIlabel1.grid(row=0, column=0,columnspan=3)
APIlabel2.grid(row=1, column=3, columnspan=2)
idLabel.grid(row=2, column=3, columnspan=2)
regionLabel.grid(row=3, column=3, columnspan=2)
checkLabel.grid(row=4,column=1,padx=1, pady=1)

APIinput.grid(row=1, column=0, columnspan=3)
idInput.grid(row=2, column=0, columnspan=3)
regionInput.grid(row=3, column=0, columnspan=3)

getKey.grid(row=0, column=3, columnspan=2)
getInputs.grid(row=4, column=0, columnspan=3)
saveInputs.grid(row=4,column=0,padx=1, pady=1)

#attempts to autofill input boxes
autoFill()
mainloop()
```

Program running:

Video demo: https://youtu.be/3EK6u667jH8





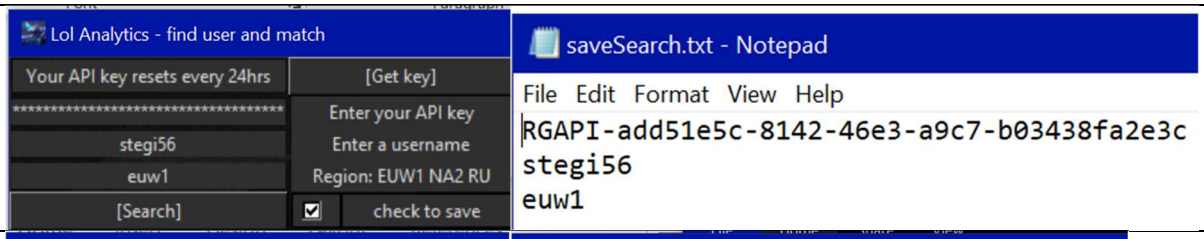
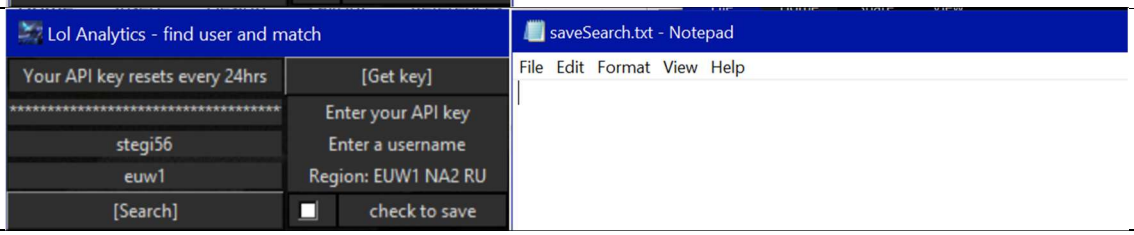


Testing

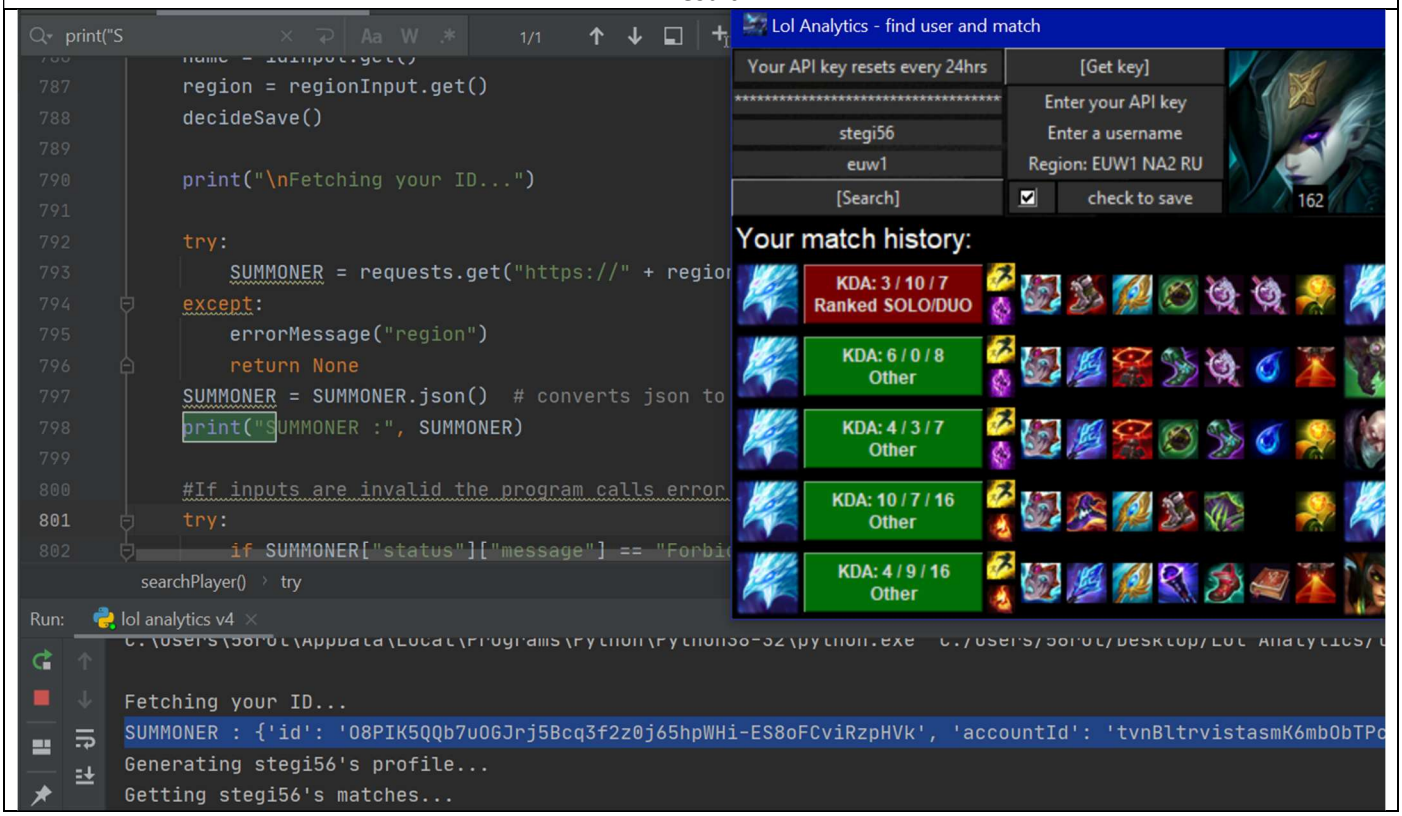
Number in square brackets outlines objective being tested e.g: "[2 – autofill]"

Objectives:

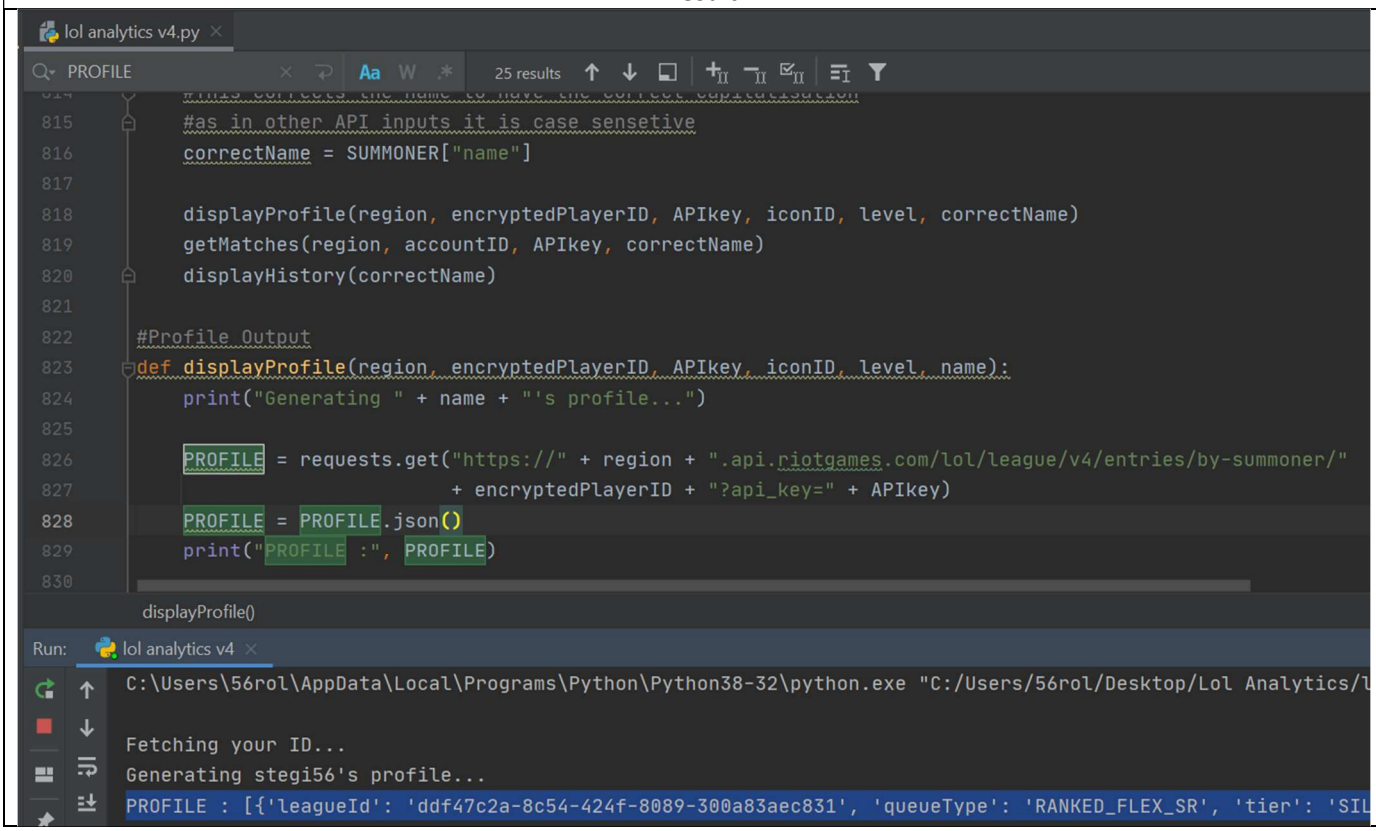
No	Objective	Performance criteria
1	**Establish connection to API and retrieve all data required.**	-UI to get player info -Use info to send request to API -Scrub return data from link and convert from json to py dictionary -Using player ID stored in dictionary send request to API for match history -Store match history in a dictionary -Using player ID request player profile from API -Store profile in a dictionary -Take match ID that user selected and make request to API -Store match data to a dictionary
2	**Error message**	-Display an appropriate error message indication which input the user has incorrectly entered
3	**Autofill feature**	- Save user inputs if the opt in to do so via checkbox -Inputs should be loaded when program is opened again
4	**Output profile**	-Display user rank/s and LP -Display user icon -Display username -Calculate and display winrate
5	**Assign each match to class**	-Class should be initialised with matchID -Initialy class must retrieve , player champion, KDA, win/loss, items, queue type
6	**Match history output**	-Display each match as a button with the match class assigned to it -Each match button must show: • Player champion • KDA • Win/Loss • Items • Queue type
7	**Process data**	-Calculate damage given/taken ratios for individual players and team as a whole and present in a pie chart -Kill participation % take each player's kill/assist scores, add together and compare to total team kills as % -Calculate number of objectives taken, take score for each objective – dragon kills, herald kills, baron kills and add together -Algorithm that suggests where to improve each data point will have an average made for each game to which each player's data points will be compared to, if data point falls 1 standard deviation below average it will be flagged red, 1.5 standard deviations above average it will be flagge green.
8	**Write MVP algorithm**	-Use all data given about each player in match to flag a player as the MVP using a points-based system for being best in various statistics that game.
9	**Output match data in tables using tKinter and matplotlib**	-Each player needs to have their own row, rows to be group into two teams and ordered by MVP score within teams. Data points to be in columns so that they can be easily compared to other players top down. -Not everything can be represented intuitively as a numbers so damage ratios will need to be presented in pie charts. True damage – white, physical damage – red, magic damage – blue.

Test 1 - Store/update/delete input boxes to a txt file [3- autofill]	
Context	This is needed to make the user experience easier, if the software is opened again the previous inputs will be saved if the user has selected the check box to do so.
Test	Check txt file with box selected and deselected and whether it is being updated apropriately
Input	Checkbox selected/deselected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	Txt file is updated with input boxes if check box is selected Txt file is wipe if check box is deselected
Results	
Selected	
Deselected	

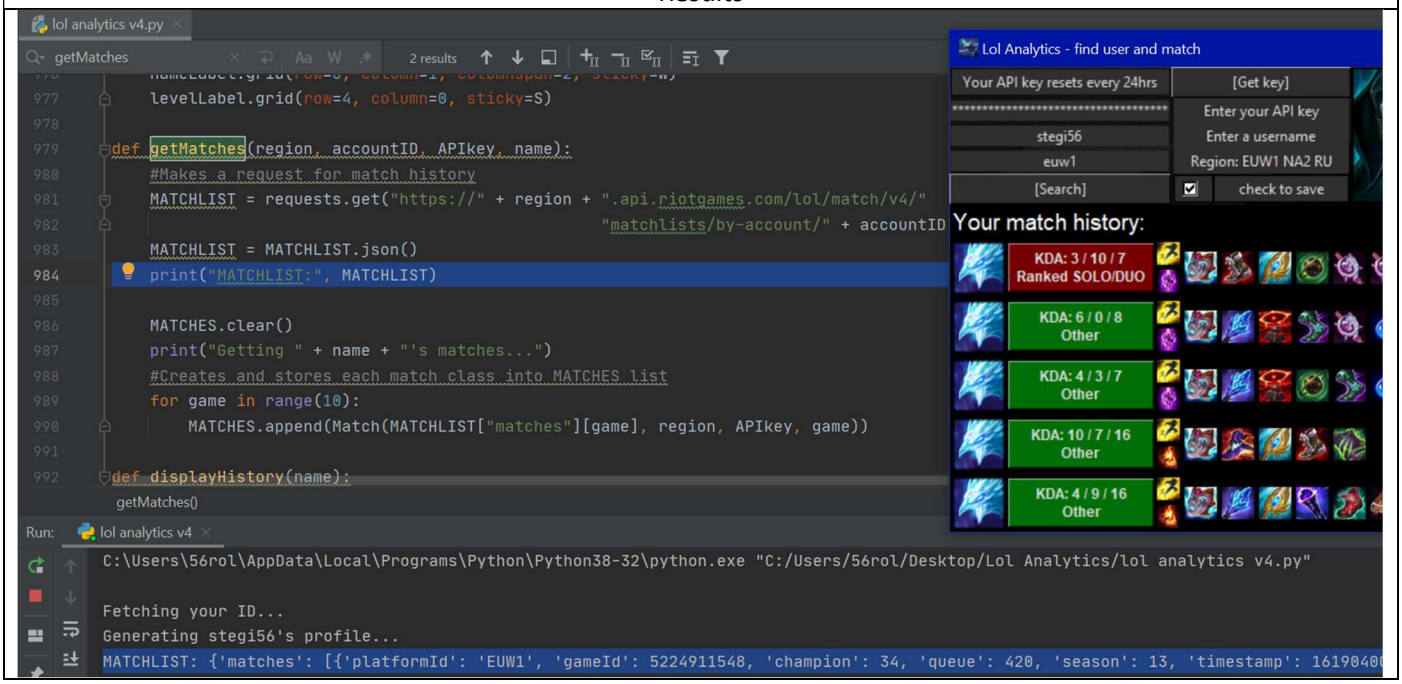
Test 2 – Find user [1 - API]	
Context	This is required in order to make other calls to the riot API to retrieve other data about the user
Test	Check SUMMONER
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	SUMMONER should have required user data returned by the API
Result	



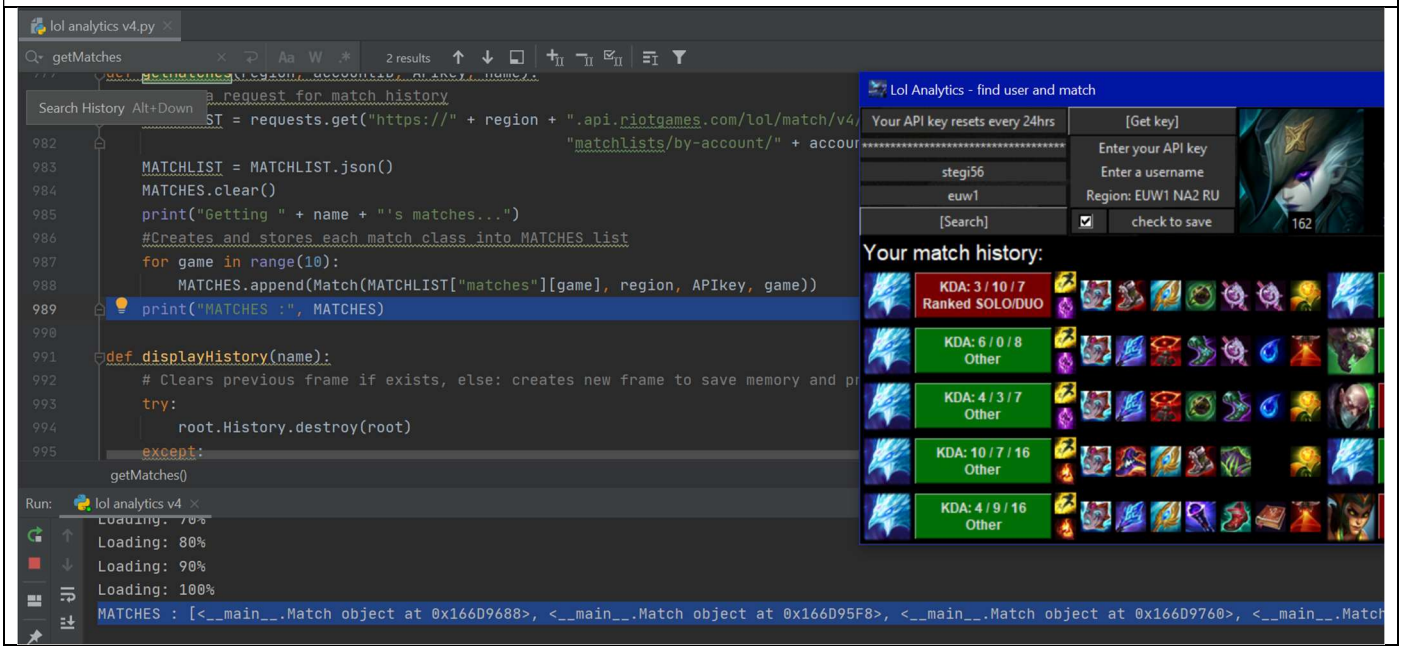
Test 3 - Find user PROFILE [1 – API]	
Context	This is needed to display the user's rank/s and their winrate for each queue type
Test	Check PROFILE
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	PROFILE should have the user rank/s data
Result	



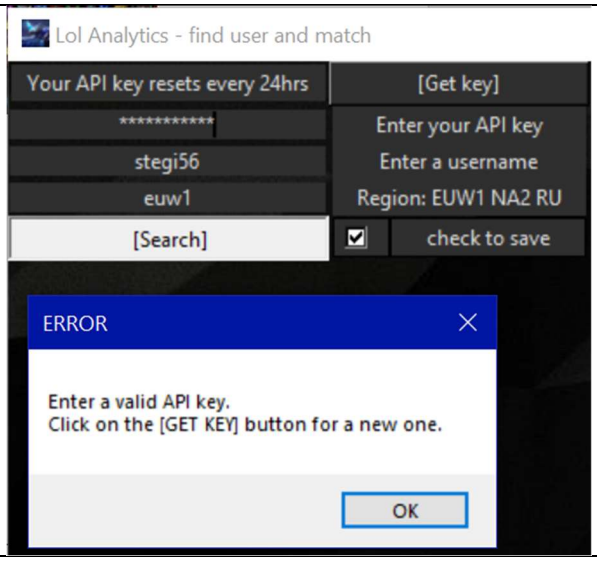
Test 4 - Program finds MATCHLIST history [1 - API]	
Context	This is needed to display the match history and to use to make calls for match information
Test	Check MATCHLIST
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	MATCHLIST should have the MatchIds and type of all previous matches
Results	



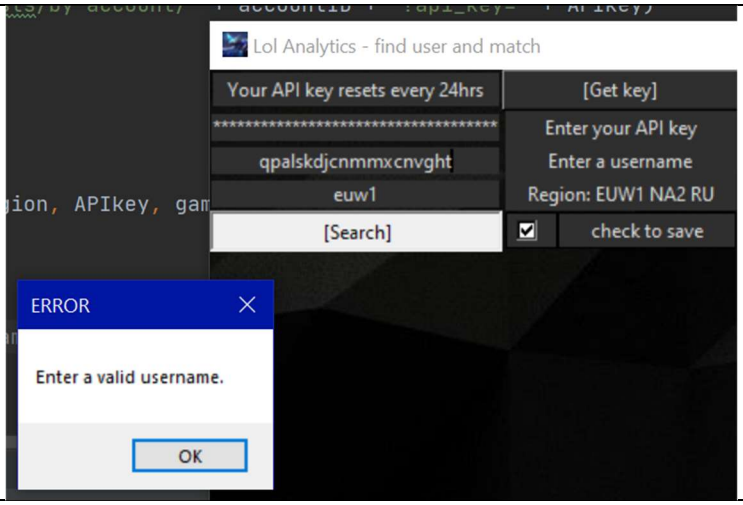
Test 5 - Create a list of MATCHES, where each Match is stored in class form [5 – initialise classes]	
Context	Program will run through last 10 matches and initialise a class for each one and store to MATCHES list
Test	Check MATCHES list
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	Should have a list of MATCH objects
Results	



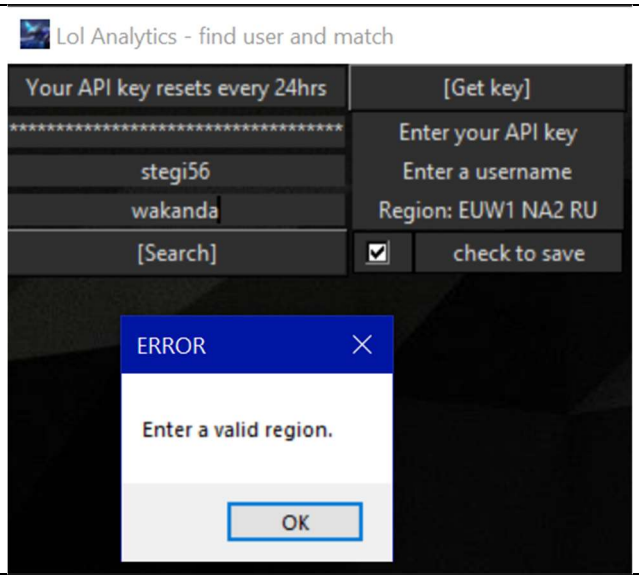
Test 6 - API key error [2 – Error messages]	
Context	When a call is made to the API the API key is checked and it returns an error if it is incorrect, a check for this error can be made an the same error can be forwarded to the user.
Test	Enter an incorrect API key
Input	Checkbox selected API key: notAnAPIkey Username: stegi56 Region: euw1
Expected	An API key error message should appear
Result	



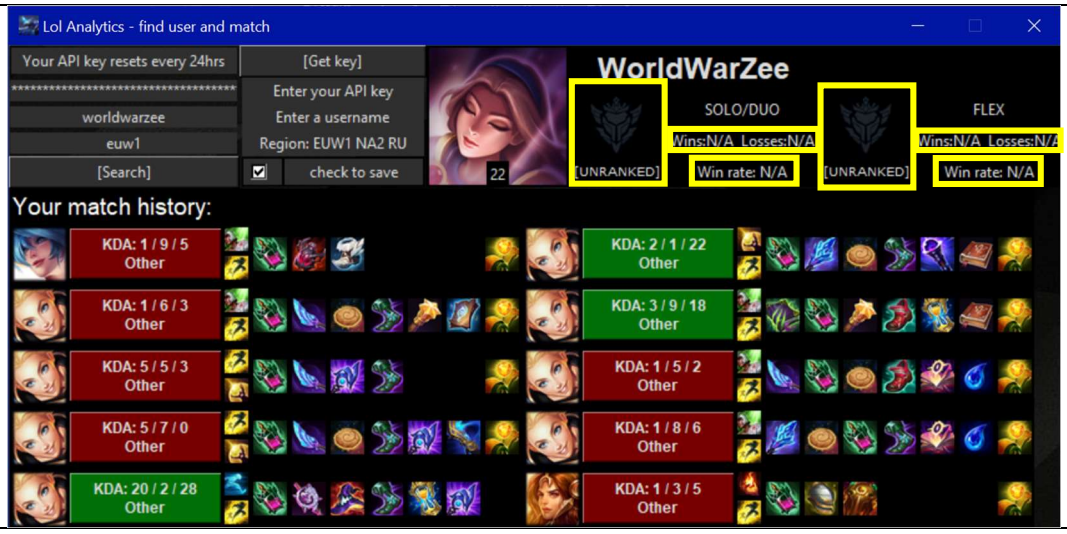
Test 7 - username error [2 – Error messages]	
Context	When the username is subited to the API to retrieve data, if it is not found the API will return a user not found error. A check for this can be made and if detected the error can be forwarded to the user.
Test	Enter a username that is not likely not to be used by a sane person
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: qpalskdjcnmmxcnvght Region: euw1
Expected	A username error should appear
Results	



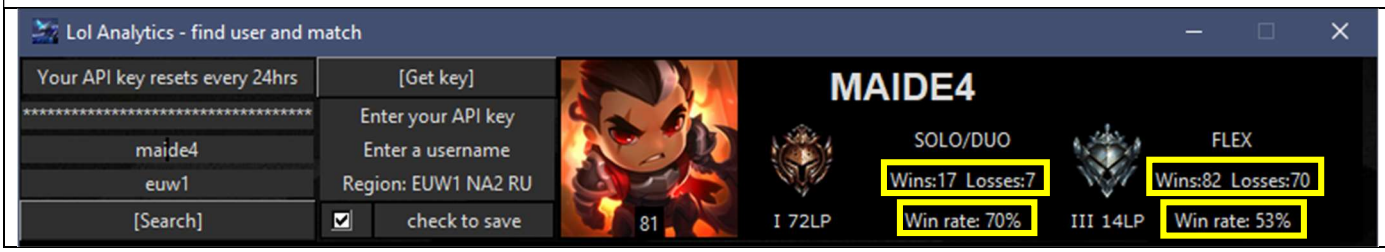
Test 8 - region error [2 – Error messages]	
Context	The region entered by the user determines which server a connection is made to, if this connection fails then it is most likely a region error.
Test	A region that does not exist is entered
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: wakanda
Expected	A region error should appear
Results	

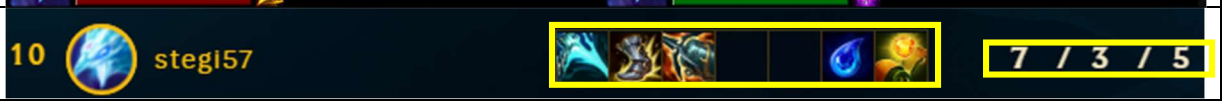


Test 9 - profile rank placeholders [4 – output profile]	
Context	The indexing for the profile ranks can vary in order and can also not exist in the API return if a player does not have a rank. This can cause errors and empty spaces in the UI.
Test	Enter the username of a new player that does not have a rank
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: worldwarzee (new player without ranks) Region: euw1
Expected	Program does not crash A place holder image and text are used for missing data in profile
Results	

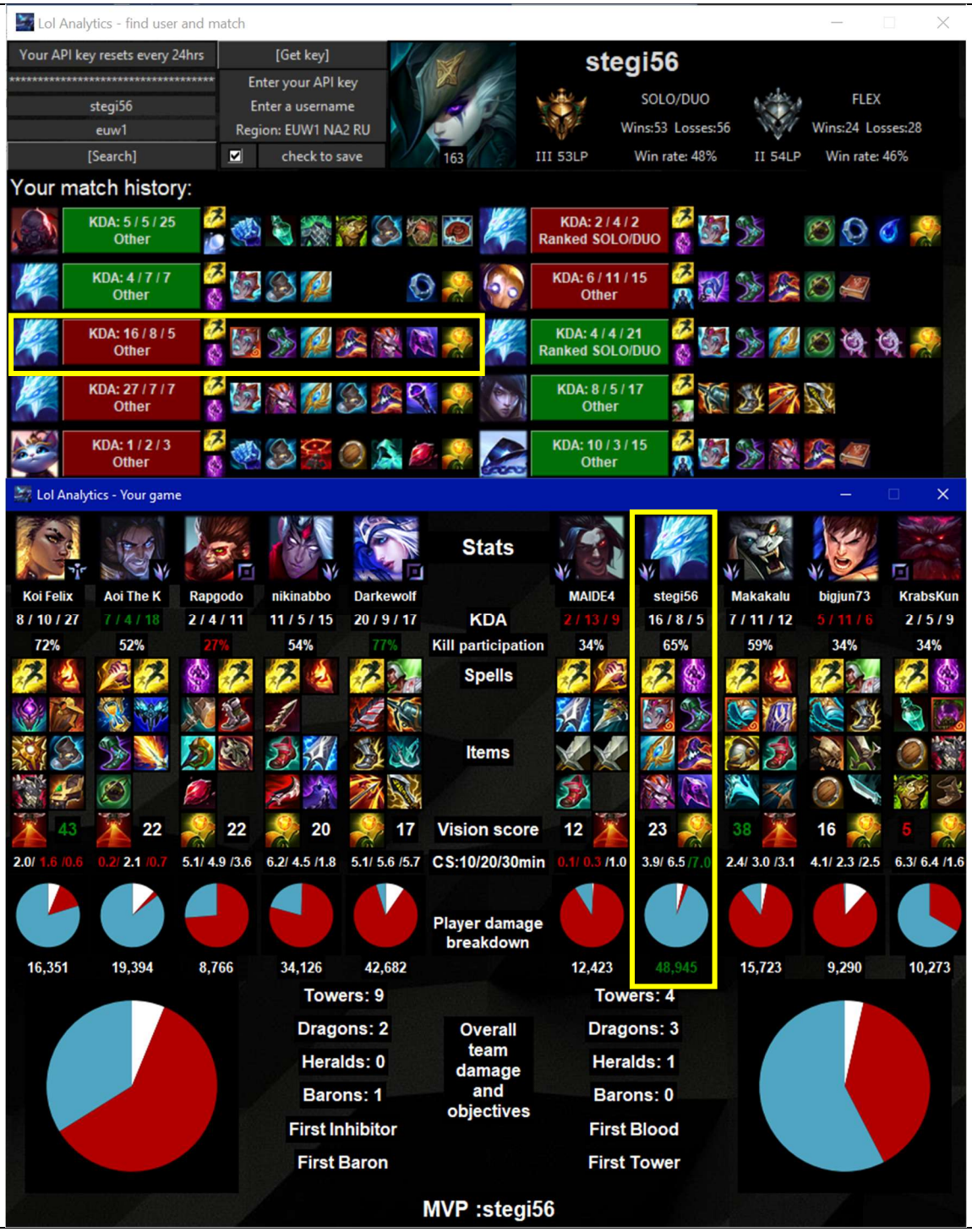


Test 10 - winrate is calculated and truncated [4 – output profile]	
Context	Winrate is not provided by the API and has to be calculated
Test	Observe wins and win rate
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: maide4 Region: euw1
Expected	(100/(17 + 7)) * 17 = 70.8, truncated to 70%
Results	

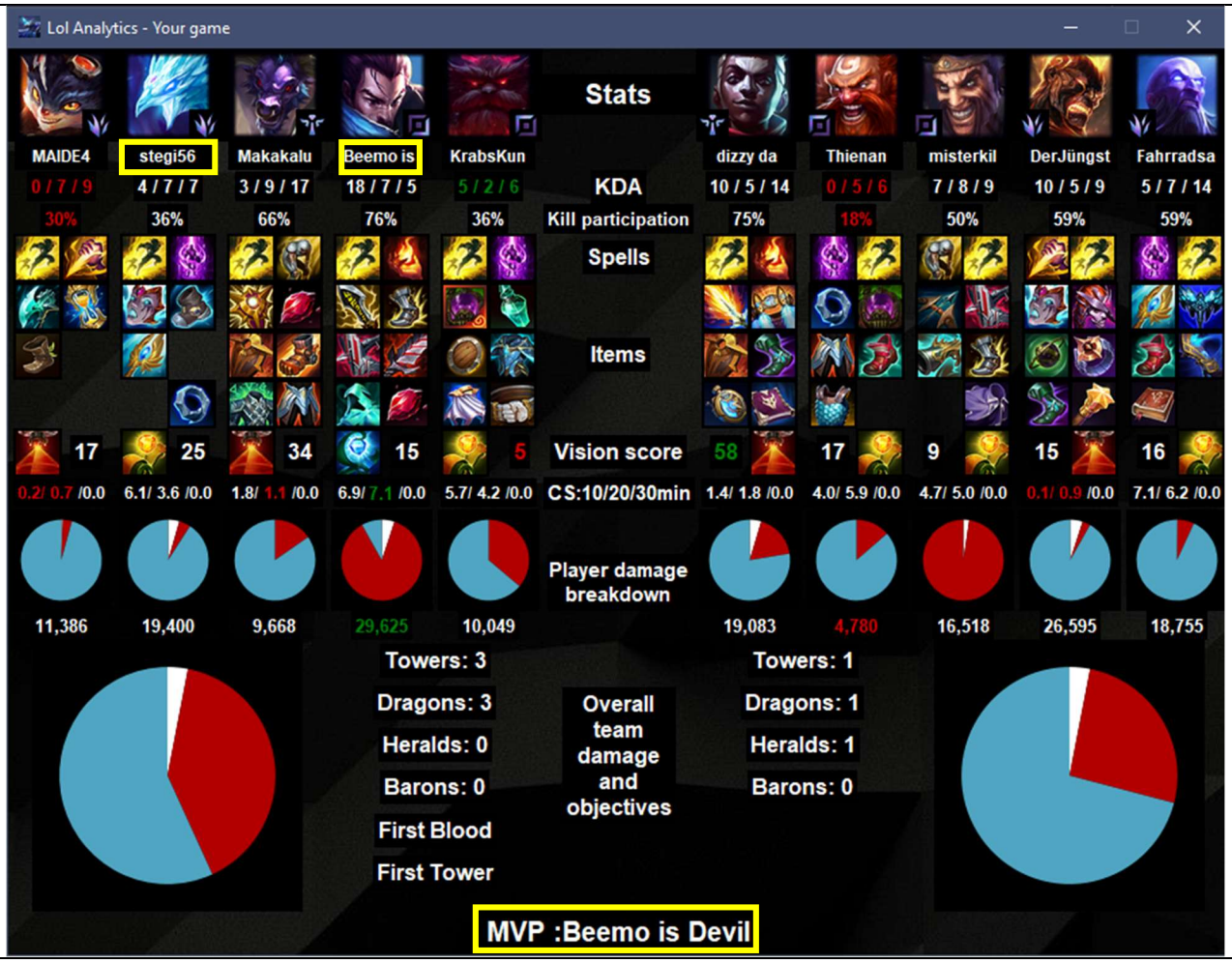


Test 11 - Match history details displayed corectly [6 – match history output]	
Context	The first image has to be the icon of the champion the player has played that match, and images on the right should be the items and spells that the player was using during the match.
Test	Compare match history with my own matches
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	Should be the same stats that I had during match
Results	
Output	
Game	

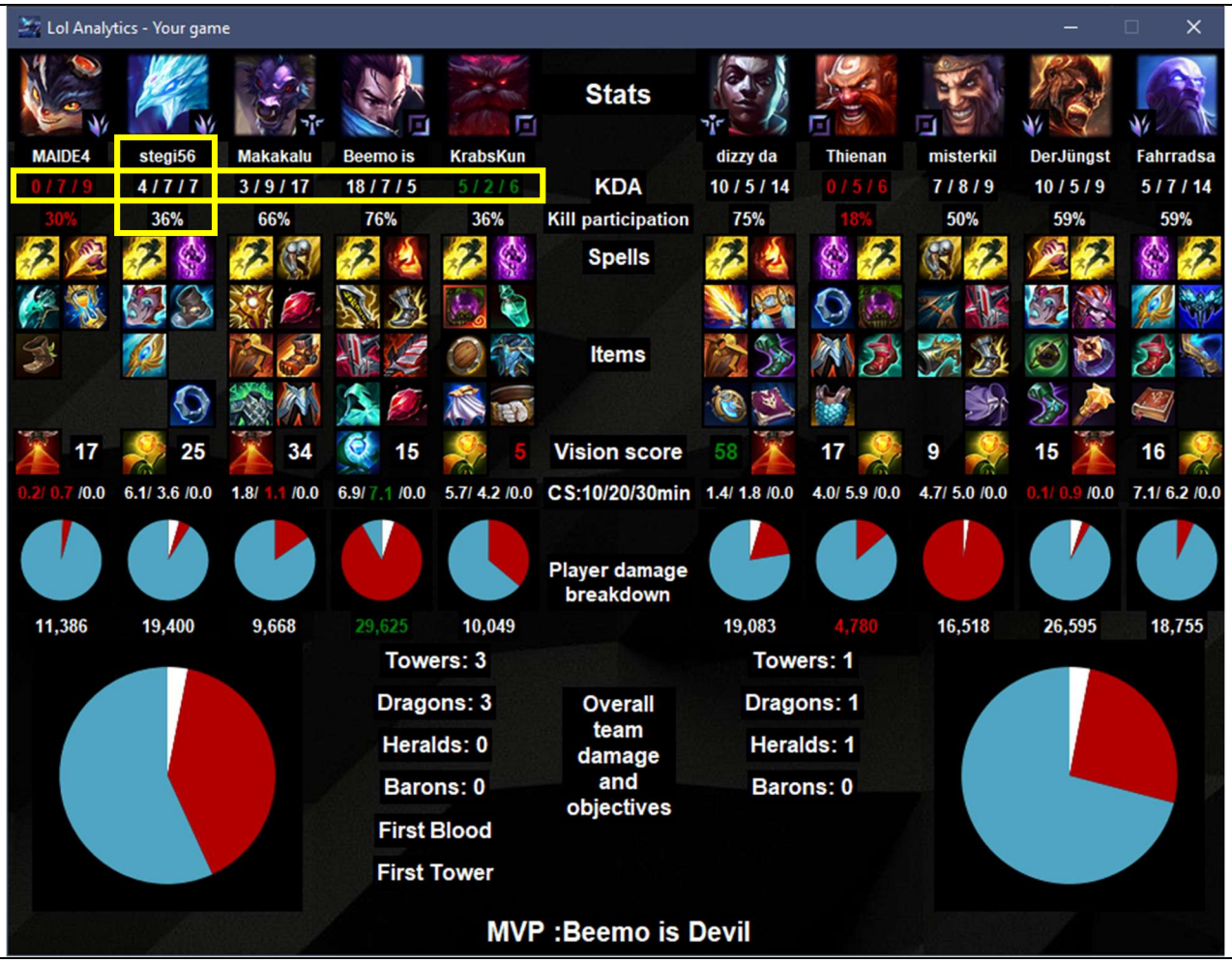
	Test 12 - Match history buttons open correct MATCH class [6 – match history output]
Context	Each button should lead to the match that it is representing
Test	Click a button
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1 16/8/5 button clicked
Expected	Expected, the simplified data displayed on the button should be seen again when a breakdown is generated after it is clicked.
Results	



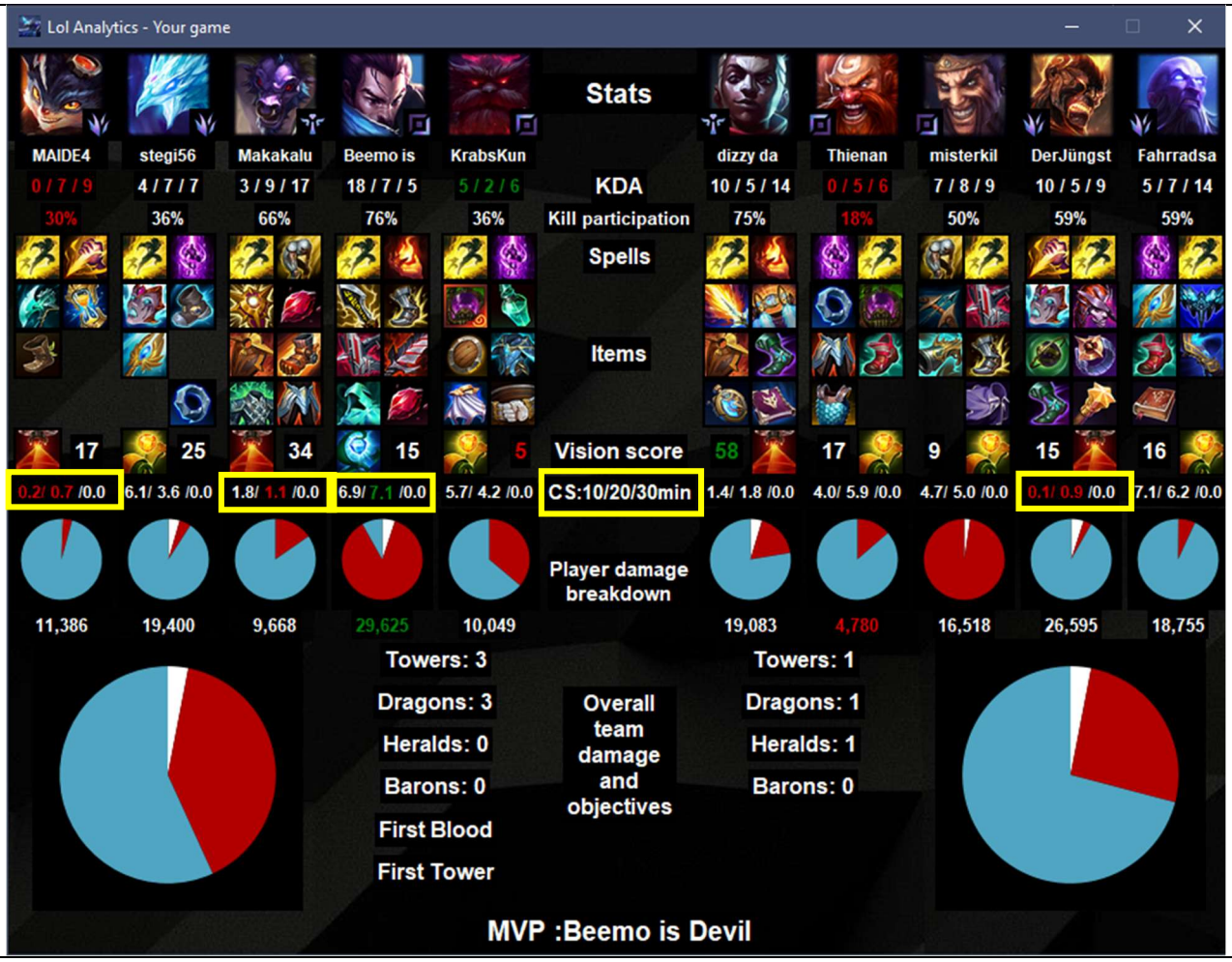
Test 13 - Player name truncation [7,9 match data processing and output]	
Context	Each column can only fit so many characters before they start interfering with other columns, so my program needs to truncate names to 9 characters.
Test	Use a match where a player has a long name that is truncated but is also an MVP
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	“Beemo is Devil” should be truncated in the column and displayed fully on the MVP tab
Results	



Test 14 – Kill participation [7 – match data processing]	
Context	Kill participation can be a good indicator of your positioning and whether you are being useful in the game.
Test	Observe team KDA, and a player's KP%
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	Kill participation = (100/total team kills) * (your kills + assists). In the end if the value is a decimal it should be truncated. (100/(0+4+3+18+5))*(4+7) = 36
Results	

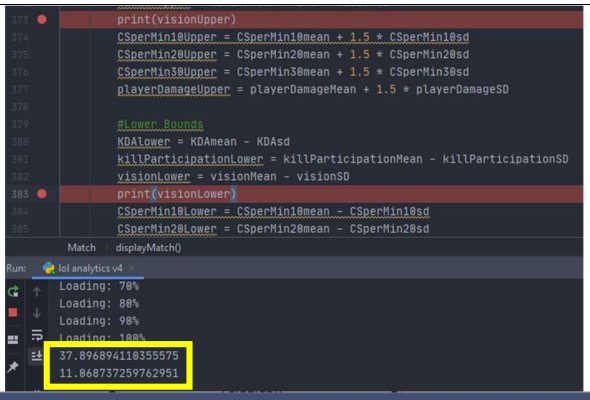


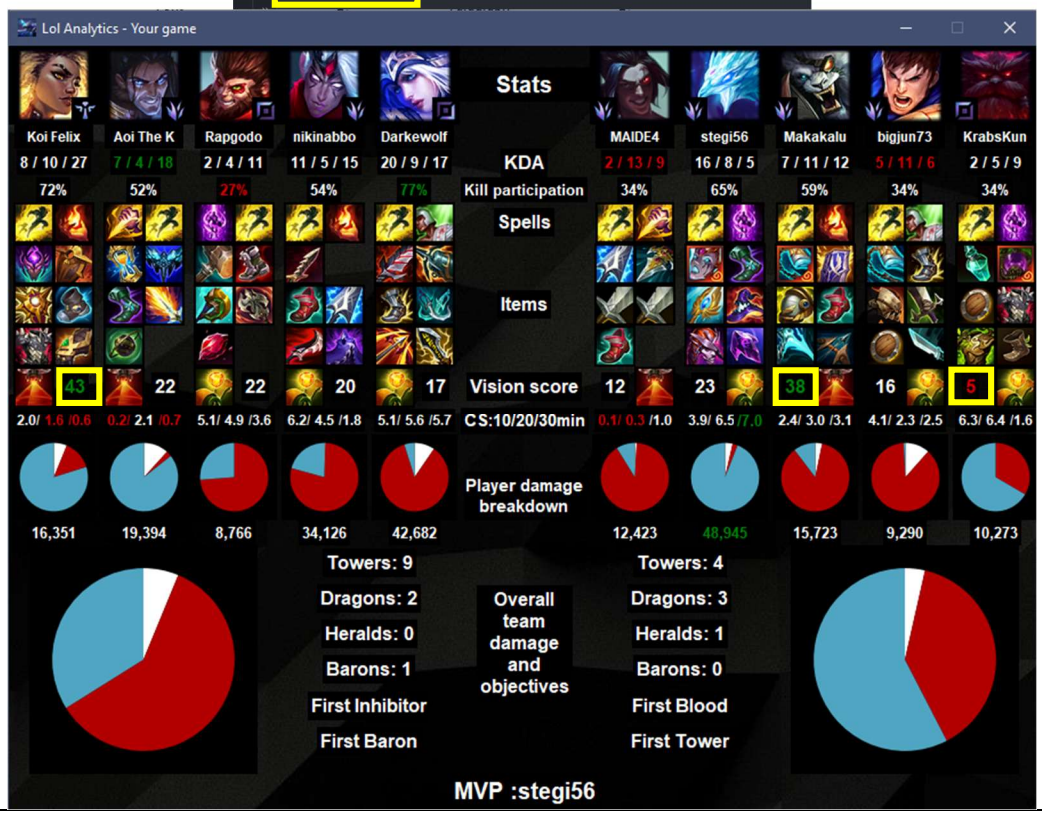
Test 15 - CS score displayed correctly in chunks [9 - output]	
Context	I could not find a way to use more than 1 colour on a single text object in tKinter. But I still wanted to display the CS at different points in the game and compare it to the standard deviation from the mean so that it can be indicated whether did well early game or later. To do this I had to split each section into parts but still fit in a single column.
Test	Observe CS row
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	Expected, a single player can have more than 1 colour in their CS score. The CS score needs to still look like it is a single phrase and it cannot overlap with other columns.
Results	



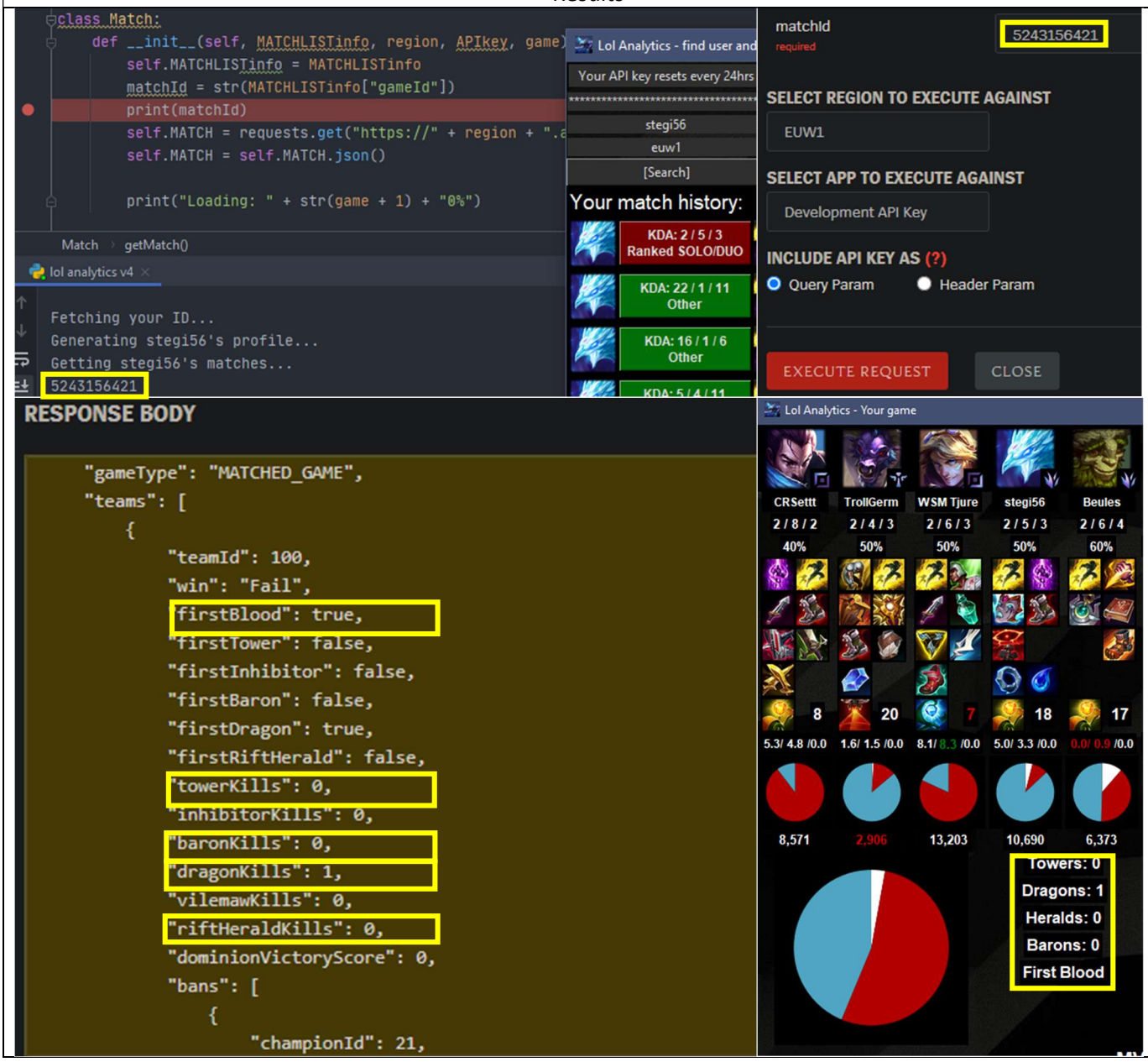
Test 16 - Player damage and pie charts [7,9 match data processing and output]	
Context	The player damage number is a sum of true, magic and physical damage given by the API and the pie chart is made using these three values
Test	Print the damage stored for each player
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	The sum of the damage needs to be the same as under pie chart. The ratio should also be represented correctly in pie chart. So for the first player in this case their damage was: 69 true, 417 physical and 10900 magic. The total should be 11,386 and on the pie chart white(true damage) should be almost invisible, red(physical damage) a small portion and blue(magic damage) should cover the majority of the pie chart.
Results	

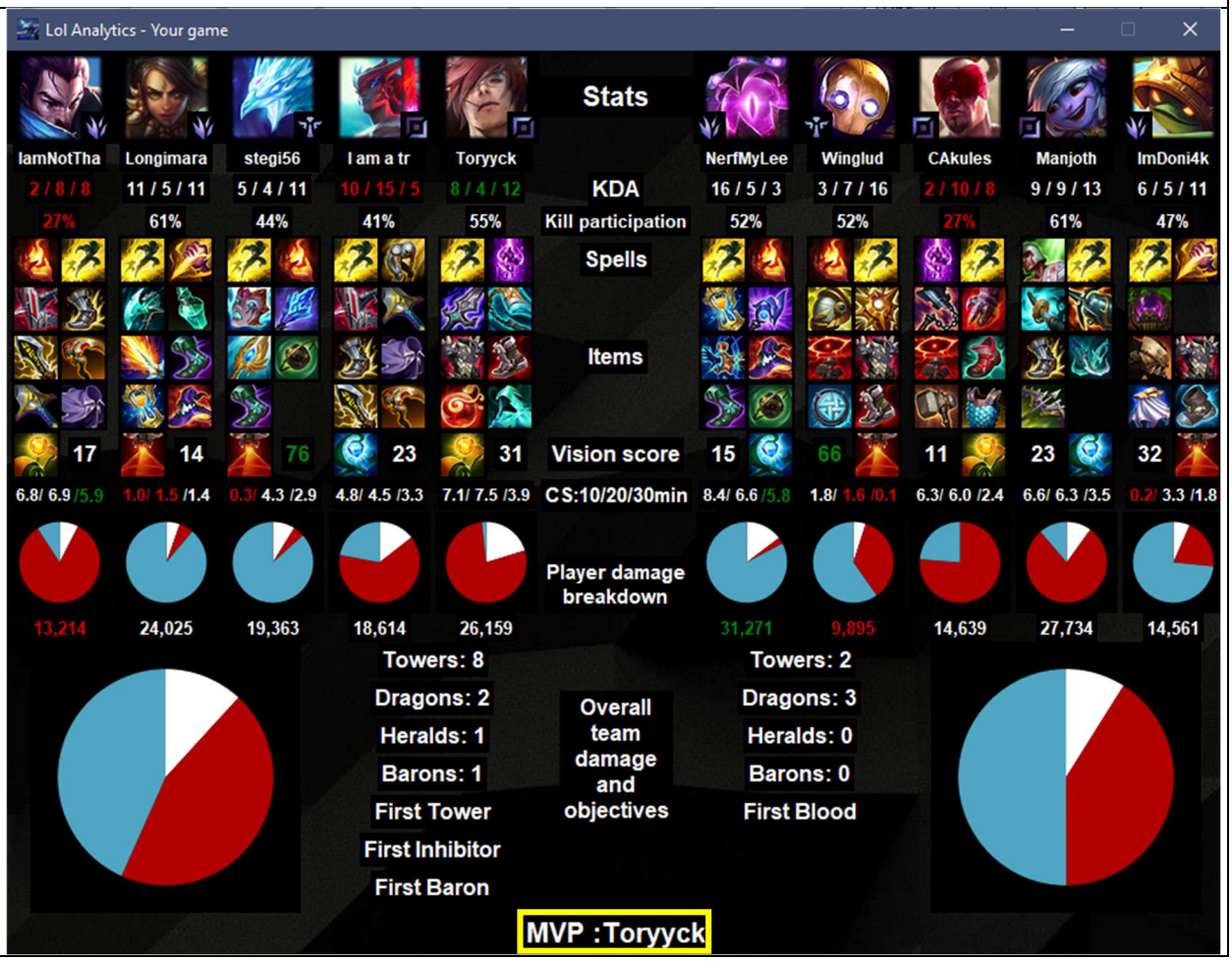
Test 17 - Flagging exeptionaly good/poor performance [7,9 match data processing and output]	
Context	In order to see if a player performs exceptionally well, a mean and standard deviation in each statistic is calculated for each team. After this and upper bound of 1.5 standard deviations from the mean and a lower bound of 1 standard deviation below the mean are created. Before a player statistic is displayed it is compared to the bound in that statistic for the players team and the color red is assigned if the player performed significantly below the average or green if significantly above.
Test	Print upper and lower bound for team 2 vision score and check colour coding
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	Expected, since the same method is used to calculate means, standard deviations and bounds – only one statistic needs to be observed. For this the vision score will be observed as it is the easiest to read and this has players withing both the upper and lower bound. I will use the statistics function to verify the calculations in my calculator. When I typed in the data the mean is 21.8 and standard deviation 11.23. The upper bound should be 21.8 + 10.73 * 1.5 = 37.895. And lower should be 21.8 – 10.73 = 11.07. These results from my calculator should match the results produced by the print statements. Anyone above or below these numbers should be flagged red or green.
Results	





Test 18 - Objectives displayed correctly [9 - output]	
Context	It can be useful to see which objectives were taken during the game and could have given a team more edge to the other
Test	Observe matchID on riot API website and software objectives output
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	API and output should match
Results	



Test 19 – MVP algorithm [8 – MVP algorithm]	
Context	The MVP algorithm takes all data for each player and awards points if they perform the best in for that game and double if are within the upper bound. Points for each statistic vary depending on the significance of that statistic.
Test	Observe players points list inside the MVP function
Input	Checkbox selected API key: RGAPI-add51e5c-8142-46e3-a9c7-b03438fa2e3c Username: stegi56 Region: euw1
Expected	Expected, points should be allocated correctly and the function should return the greatest or one of the greatest values in the list. If there are equal values randomisation does not need to be implemented as players are given random player indexes for each match already. Since the list corresponds to them the first greatest value is also a random greatest value.
Results	
List	[['IamNotThaFather', 2], ['Longimara', 2], ['stegi56', 4], ['I am a triangle', 0], ['Toryyck', 5], ['NerfMyLeePlz', 5], ['Winglud', 0], ['CAkules', 0], ['Manjoth', 0], ['ImDoni4k', 0]]
Output	

Evaluation

Summary:

Overall this project has been mostly a success, the project is complete, the software functions as intended and passes my tests. The project lacks in output data as was described in the project analysis, this is due to me underestimating how long it would take to implement each data point in a meaningful way – I ended up leaving most of the data out and only included data that matters the most such as damage and objectives. My client is mostly satisfied but commented there could be more features included such as flag boundary adjustment and team gold to be displayed.

Objectives:

- API connection
  - This is a foundation of the project and was implemented with high reliability and features invalid input detection to ensure that the software works with the API.
  - Passed tests (2,3,4).
- Error messages
  - This was implemented early on in the project and proved extremely useful when trying different input configurations with accidental typos to test my code.
  - Passed tests(6,7,8).
- Autofill feature
  - My client was impressed once I implemented this feature as then they did not have to type their details again each time they wanted to use the software.
  - Passed test (1).
- Profile output
  - Uses correct images for the indexes returned by the API
  - Detects when player does not have a rank/s and uses correct placeholders to prevent a crash
  - Calculates winrate correctly
  - Passed tests (9,10)
- Matches assigned to classes
  - Ininitialises matches into classes correctly and only completes match calculation if the class is called to which increases performance
  - Passed test (5)
- History output
  - Intuitive to use
  - Buttons show enough information for matches to be identifiable but not so much that they take too much space
  - Displayed in a way that is easy to read and compare
  - Not all data was used like intended at the start as it would be impractical and would take too much space
  - Passed tests (11,12)
- Data processing
  - All formulae were tested and work as expected
  - Meets critearia of processing features such as kill participation and flag system to point out player's weaknesses
  - Passed tests (13,14,16,17)
- MVP algorithm
  - Uses all data points that are displayed like intended and determines a top player using a points based system
  - Passed test (19)

- Match output
    - All data is outputted in a manner that is expected, easy to read and easy to understand for league of legends players.
    - The column design makes it easy to compare different players in the match
    - Passed tests (13,15,16,17,18)

Client feedback:

- Adjust colours to create a dark mode aesthetic and make easier to read
    - An overhaul on the background and text colour was made, to be dark themed and less straining on the eyes
    - An adjustment was also made to the match button background red/green colours to make the white text more readable
- Feature to adjust upper and lower bounds
    - This was not implemented due to the amount of time it takes to add data to the GUI but would definitely be considered in future development.
- Display player and team gold difference
    - This was not implemented intialy due to concerns about providing too much data but I have come to realise this is an important data point I have missed out, looking back at my initial survey.

Future development:

- Display player and team gold
    - Requested by client
    - Important data point
- Adjustable flag bounds
    - The client wants to be able to set a standard of what is good and poor performance themselves as right now it is a fixed value of 1 standard deviation below mean and 1.5 standard deviations above
    - A feature for them to adjust the upper and lower bound will be implemented
- Match caching feature
    - Matches can be stored on a database with the key being the matchID and content being the Match class
    - If a known matchID is spotted in the Match list, instead of calling the API for match data and completing data processing the match class that has already been initialised would be loaded

Objectives:

No	Objective	Performance criteria
1	**Establish connection to API and retrieve all data required.**	-UI to get player info -Use info to send request to API -Scrub return data from link and convert from json to py dictionary -Using player ID stored in dictionary send request to API for match history -Store match history in a dictionary -Using player ID request player profile from API -Store profile in a dictionary -Take match ID that user selected and make request to API -Store match data to a dictionary
2	**Error message**	-Display an appropriate error message indication which input the user has incorrectly entered
3	**Autofill feature**	- Save user inputs if the opt in to do so via checkbox -Inputs should be loaded when program is opened again
4	**Output profile**	-Display user rank/s and LP -Display user icon -Display username -Calculate and display winrate
5	**Assign each match to class**	-Class should be initialised with matchID -Initialy class must retrieve , player champion, KDA, win/loss, items, queue type
6	**Match history output**	-Display each match as a button with the match class assigned to it -Each match button must show: • Player champion • KDA • Win/Loss • Items • Queue type
7	**Process data**	-Calculate damage given/taken ratios for individual players and team as a whole and present in a pie chart -Kill participation % take each player's kill/assist scores, add together and compare to total team kills as % -Calculate number of objectives taken, take score for each objective – dragon kills, herald kills, baron kills and add together -Algorithm that suggests where to improve each data point will have an average made for each game to which each player's data points will be compared to, if data point falls 1 standard deviation below average it will be flagged red, 1.5 standard deviations above average it will be flagge green.
8	**Write MVP algorithm**	-Use data given about each player in match to flag a player as the MVP using a points-based system for being best in various statistics that game.
9	**Output match data in tables using tKinter and matplotlib**	-Each player needs to have their own row, rows to be group into two teams and ordered by MVP score within teams. Data points to be in columns so that they can be easily compared to other players top down. -Not everything can be represented intuitively as a numbers so damage ratios will need to be presented in pie charts. True damage – white, physical damage – red, magic damage – blue.