SoniqsDataEntry

April 17, 2022

This project is an analysis of the Rainbow 6 Siege esports team Soniqs from playoffs on Feb 22nd, 2022. I want to scrape the website Siegegg for the match data, and create visualizations that can give me insight into the team's dynamic.

These are the packages I am using for this web scraping exercise. I need requests to request the data from the url. I need pandas to create the dataframe. I use beautiful soup to scrape the website for the text in the tables and pprint to visualize the html in jupyter. I use cycle from itertools to cycle through my column list and create a dictionary with all the data from each column. I use seaborn to create my visualizations. I have matplotlib to save my images to png to have my visualizations easily accessible.

[1]:
```
import requests
import pandas as pd
from bs4 import BeautifulSoup
from pprint import pprint
from itertools import cycle
import seaborn as sns
import matplotlib.pyplot as plt
```

I initialize the url that I will be using to scrape the web. I request the url and get the status code. 200 means that the request was successful. I then parse the url with BeautifulSoup and state the website is in html.

[2]:
```
url = 'https://siege.gg/matches/7196-invitational-intl-elevate-vs-soniqs'
r = requests.get(url)
print(r.status_code) #200 means the website is working
soup = BeautifulSoup(r.content, 'html') #I want to parse the website in html
```

```
200
```

After seeing the 200 above I know that the website is ready to be scraped. I look at the html layout of the website to find where the table is located on the webpage. The table I want is named playertable, so I access it by using soup.find to locate the exact table. I want to separate the table headers from the table data, and this can be done by selecting only the first row of the table. I print out the rows to make sure these are the columns I want. I see that the first few rows on the webpage are blank columns, so I cut them out by slicing only the columns that contain data.

[3]:
```
table = soup.find(id="playertable") #this is the id of the table on the website
header_row = table.find_all('tr')[0] #get all column names
pprint(header_row.text.split(' ')[2:-1]) #get the columns that contain data
```

```
['Rating',
 'K-D',
 '(+/-)',
 'Entry',
 '(+/-)',
 'KOST',
 'KPR',
 'SRV',
 '1vX',
 'Plants',
 'HS%',
 'Atk',
 'Def',
 'Team']
```

Once the columns I want are sliced, I need to add a column to the front that has the player's names. The column with the player names in the actual table is blank, so I add this column in myself with the insert function. After I print out this new column list, I see that everything is ready and I can begin working on pulling the data from the table.

[4]:
```
column_names = []
new_column = 'Player' #add the player column for the player's names
for each in header_row('th'):
    column_names.append(each.text)
column_names = column_names[1:]
column_names.insert(0, new_column)
print(column_names) #these are the columns that match the table on the website
```

```
['Player', 'Rating', 'K-D (+/-)', 'Entry (+/-)', 'KOST', 'KPR', 'SRV', '1vX',
'Plants', 'HS%', 'Atk', 'Def', 'Team']
```

The first step is to pull the rows of the table that contain the data and not the headers. This is all rows except the first row. I iterate through each row and pull the text data from each cell. The text data has a bunch of empty space before and after to fit the webpage. This can be removed with string comprehension stripping each cell.

[5]:
```
data = table.find_all('tr')[1:] #get all rows from the table

row_data = []
for each in data:
    for x in each('td'):
        row_data.append(x.text) #get text from each cell
        row_data = [x.strip(' ') for x in row_data] #cleans whitespace


print(row_data)
```

```
['Rexen', '1.05', '25-25 (+0)', '2-6 (-4)', '64%', '0.69', '31%', '2', '0',
'76%', 'Ace', 'Jager', '125', 'Gryxr', '1.35', '39-24 (+15)', '7-3 (+4)', '72%',
```

```
'1.08', '33%', '2', '1', '36%', 'Twitch', 'Mira', '125', 'Supr', '0.85', '13-22
(-9)', '1-2 (-1)', '64%', '0.36', '39%', '0', '4', '23%', 'Thermite', 'Smoke',
'125', 'Sapper', '1.03', '24-23 (+1)', '1-0 (+1)', '72%', '0.67', '36%', '0',
'2', '38%', 'Thermite', 'Smoke', '357', 'Nay..Pew', '0.86', '22-26 (-4)', '6-5
(+1)', '56%', '0.61', '28%', '0', '0', '29%', 'Maverick', 'Mute', '357', 'DCH',
'0.82', '18-25 (-7)', '1-1 (+0)', '61%', '0.50', '31%', '0', '1', '39%',
'Sledge', 'Jager', '357', 'Kanzen', '0.95', '22-25 (-3)', '4-1 (+3)', '72%',
'0.61', '31%', '0', '0', '50%', 'Buck', 'Kaid', '125', 'sprOnigiri', '0.83',
'24-28 (-4)', '5-11 (-6)', '56%', '0.67', '22%', '0', '0', '63%', 'Iana',
'Aruni', '357', 'Yeti', '0.96', '25-25 (+0)', '6-4 (+2)', '61%', '0.69', '31%',
'0', '0', '64%', 'Maverick', 'Mute', '125', 'Nerix', '1.09', '32-23 (+9)', '3-3
(+0)', '56%', '0.89', '36%', '0', '0', '53%', 'Jackal', 'Mira', '357']
```

The final step to creating the dataframe is creating a dictionary with the keys being the column headers and the values correlating to the data within each column. I make a list by zipping each column to their respective values. I use the itertools function cycle to repeat the shorter list of columns to the longer list of data.

[6]:
```python
full_table = []
full_table = (zip(cycle(column_names),row_data)) #create tuples of column names␣
↪matched to the rows under the columns
full_table = list(full_table)
full_table
```

[6]:
```
[('Player', 'Rexen'),
 ('Rating', '1.05'),
 ('K-D (+/-)', '25-25 (+0)'),
 ('Entry (+/-)', '2-6 (-4)'),
 ('KOST', '64%'),
 ('KPR', '0.69'),
 ('SRV', '31%'),
 ('1vX', '2'),
 ('Plants', '0'),
 ('HS%', '76%'),
 ('Atk', 'Ace'),
 ('Def', 'Jager'),
 ('Team', '125'),
 ('Player', 'Gryxr'),
 ('Rating', '1.35'),
 ('K-D (+/-)', '39-24 (+15)'),
 ('Entry (+/-)', '7-3 (+4)'),
 ('KOST', '72%'),
 ('KPR', '1.08'),
 ('SRV', '33%'),
 ('1vX', '2'),
 ('Plants', '1'),
 ('HS%', '36%'),
 ('Atk', 'Twitch'),
```

```
('Def', 'Mira'),
('Team', '125'),
('Player', 'Supr'),
('Rating', '0.85'),
('K-D (+/-)', '13-22 (-9)'),
('Entry (+/-)', '1-2 (-1)'),
('KOST', '64%'),
('KPR', '0.36'),
('SRV', '39%'),
('1vX', '0'),
('Plants', '4'),
('HS%', '23%'),
('Atk', 'Thermite'),
('Def', 'Smoke'),
('Team', '125'),
('Player', 'Sapper'),
('Rating', '1.03'),
('K-D (+/-)', '24-23 (+1)'),
('Entry (+/-)', '1-0 (+1)'),
('KOST', '72%'),
('KPR', '0.67'),
('SRV', '36%'),
('1vX', '0'),
('Plants', '2'),
('HS%', '38%'),
('Atk', 'Thermite'),
('Def', 'Smoke'),
('Team', '357'),
('Player', 'Nay..Pew'),
('Rating', '0.86'),
('K-D (+/-)', '22-26 (-4)'),
('Entry (+/-)', '6-5 (+1)'),
('KOST', '56%'),
('KPR', '0.61'),
('SRV', '28%'),
('1vX', '0'),
('Plants', '0'),
('HS%', '29%'),
('Atk', 'Maverick'),
('Def', 'Mute'),
('Team', '357'),
('Player', 'DCH'),
('Rating', '0.82'),
('K-D (+/-)', '18-25 (-7)'),
('Entry (+/-)', '1-1 (+0)'),
('KOST', '61%'),
('KPR', '0.50'),
```

```
('SRV', '31%'),
('1vX', '0'),
('Plants', '1'),
('HS%', '39%'),
('Atk', 'Sledge'),
('Def', 'Jager'),
('Team', '357'),
('Player', 'Kanzen'),
('Rating', '0.95'),
('K-D (+/-)', '22-25 (-3)'),
('Entry (+/-)', '4-1 (+3)'),
('KOST', '72%'),
('KPR', '0.61'),
('SRV', '31%'),
('1vX', '0'),
('Plants', '0'),
('HS%', '50%'),
('Atk', 'Buck'),
('Def', 'Kaid'),
('Team', '125'),
('Player', 'sprOnigiri'),
('Rating', '0.83'),
('K-D (+/-)', '24-28 (-4)'),
('Entry (+/-)', '5-11 (-6)'),
('KOST', '56%'),
('KPR', '0.67'),
('SRV', '22%'),
('1vX', '0'),
('Plants', '0'),
('HS%', '63%'),
('Atk', 'Iana'),
('Def', 'Aruni'),
('Team', '357'),
('Player', 'Yeti'),
('Rating', '0.96'),
('K-D (+/-)', '25-25 (+0)'),
('Entry (+/-)', '6-4 (+2)'),
('KOST', '61%'),
('KPR', '0.69'),
('SRV', '31%'),
('1vX', '0'),
('Plants', '0'),
('HS%', '64%'),
('Atk', 'Maverick'),
('Def', 'Mute'),
('Team', '125'),
('Player', 'Nerix'),
```

```
('Rating', '1.09'),
('K-D (+/-)', '32-23 (+9)'),
('Entry (+/-)', '3-3 (+0)'),
('KOST', '56%'),
('KPR', '0.89'),
('SRV', '36%'),
('1vX', '0'),
('Plants', '0'),
('HS%', '53%'),
('Atk', 'Jackal'),
('Def', 'Mira'),
('Team', '357')]
```

I iterate through the entire list of key value pairs and create a dictionary out of the pairs.

[7]:
```
full_dict = {}
for i in full_table:
    full_dict.setdefault(i[0],[]).append(i[1]) #make a dictionary that matches␣
 ↪the tuples to the columns
print(full_dict)
```

```
{'Player': ['Rexen', 'Gryxr', 'Supr', 'Sapper', 'Nay..Pew', 'DCH', 'Kanzen',
'sprOnigiri', 'Yeti', 'Nerix'], 'Rating': ['1.05', '1.35', '0.85', '1.03',
'0.86', '0.82', '0.95', '0.83', '0.96', '1.09'], 'K-D (+/-)': ['25-25 (+0)',
'39-24 (+15)', '13-22 (-9)', '24-23 (+1)', '22-26 (-4)', '18-25 (-7)', '22-25
(-3)', '24-28 (-4)', '25-25 (+0)', '32-23 (+9)'], 'Entry (+/-)': ['2-6 (-4)',
'7-3 (+4)', '1-2 (-1)', '1-0 (+1)', '6-5 (+1)', '1-1 (+0)', '4-1 (+3)', '5-11
(-6)', '6-4 (+2)', '3-3 (+0)'], 'KOST': ['64%', '72%', '64%', '72%', '56%',
'61%', '72%', '56%', '61%', '56%'], 'KPR': ['0.69', '1.08', '0.36', '0.67',
'0.61', '0.50', '0.61', '0.67', '0.69', '0.89'], 'SRV': ['31%', '33%', '39%',
'36%', '28%', '31%', '31%', '22%', '31%', '36%'], '1vX': ['2', '2', '0', '0',
'0', '0', '0', '0', '0', '0'], 'Plants': ['0', '1', '4', '2', '0', '1', '0',
'0', '0', '0'], 'HS%': ['76%', '36%', '23%', '38%', '29%', '39%', '50%', '63%',
'64%', '53%'], 'Atk': ['Ace', 'Twitch', 'Thermite', 'Thermite', 'Maverick',
'Sledge', 'Buck', 'Iana', 'Maverick', 'Jackal'], 'Def': ['Jager', 'Mira',
'Smoke', 'Smoke', 'Mute', 'Jager', 'Kaid', 'Aruni', 'Mute', 'Mira'], 'Team':
['125', '125', '125', '357', '357', '357', '125', '357', '125', '357']}
```

Once the dictionary is made, I can create a dataframe out of my dictionary.

[8]:
```
ele_sqs = pd.DataFrame(full_dict) #make the dictionary into a df
ele_sqs
```

[8]:

```
       Player Rating   K-D (+/-) Entry (+/-) KOST   KPR  SRV 1vX Plants  HS%  \
0       Rexen   1.05  25-25 (+0)    2-6 (-4)  64%  0.69  31%   2      0  76%
1       Gryxr   1.35 39-24 (+15)    7-3 (+4)  72%  1.08  33%   2      1  36%
2        Supr   0.85  13-22 (-9)    1-2 (-1)  64%  0.36  39%   0      4  23%
3      Sapper   1.03  24-23 (+1)    1-0 (+1)  72%  0.67  36%   0      2  38%
```

4	Nay..Pew	0.86	22-26 (-4)	6-5 (+1)	56%	0.61	28%	0	0	29%
5	DCH	0.82	18-25 (-7)	1-1 (+0)	61%	0.50	31%	0	1	39%
6	Kanzen	0.95	22-25 (-3)	4-1 (+3)	72%	0.61	31%	0	0	50%
7	sprOnigiri	0.83	24-28 (-4)	5-11 (-6)	56%	0.67	22%	0	0	63%
8	Yeti	0.96	25-25 (+0)	6-4 (+2)	61%	0.69	31%	0	0	64%
9	Nerix	1.09	32-23 (+9)	3-3 (+0)	56%	0.89	36%	0	0	53%

	Atk	Def	Team
0	Ace	Jager	125
1	Twitch	Mira	125
2	Thermite	Smoke	125
3	Thermite	Smoke	357
4	Maverick	Mute	357
5	Sledge	Jager	357
6	Buck	Kaid	125
7	Iana	Aruni	357
8	Maverick	Mute	125
9	Jackal	Mira	357

The dataframe requires a lot of cleaning before analysis can be done. I want to create new columns for the k/d column into separate kills and death columns. I take the other part of the k/d column into its own column called KDMargin. KDMargin is the difference between kills and deaths that a given player has gotten in a series. I do this same split with the entry column. This column is split into first blood and first death columns. These columns help explain if the player is an entry player that gets the first kill in a round, or if the player dies first in each round. The column EntryMargin explains the amount of times a given player has gotten first blood compared to dying first. This column is important to include due to the volatility of the game once a player has died. I then remove the % sign off each percentage based column. This allows me to change the columns that are numbers into numerpic data types instead of string types like they are when they are pulled. I also change the team's number designation to their team name. In this example, 125 refers to Soniqs team and 357 refers to Elevate.

[9]:
```
ele_sqs[['Kills','Deaths']] = ele_sqs['K-D (+/-)'].str.split('-',n=1,expand=True) #split the kill death into 2
ele_sqs[['Deaths','KDMargin']] = ele_sqs['Deaths'].str.split('(',n=1,expand=True)
ele_sqs['KDMargin'] = ele_sqs['KDMargin'].str.replace(')','') #clean the columns
ele_sqs[['FB','FD']] = ele_sqs['Entry (+/-)'].str.split('-', n=1,expand=True) #split entry into first blood and first death
ele_sqs[['FD','EntryMargin']] = ele_sqs['FD'].str.split('(', n=1,expand=True)
ele_sqs['EntryMargin'] = ele_sqs['EntryMargin'].str.replace(')','')
ele_sqs['KOST'] = ele_sqs['KOST'].str.replace('%','') #remove the % sign to change the data type
ele_sqs['HS%'] = ele_sqs['HS%'].str.replace('%','')
ele_sqs['SRV'] = ele_sqs['SRV'].str.replace('%','')
ele_sqs['Team'] = ele_sqs['Team'].replace(['125'],'Soniqs') #map team name to the number they represent
```

```python
ele_sqs['Team'] = ele_sqs['Team'].replace(['357'],'Elevate')
cols =␣
 ↪['Rating','KOST','KPR','SRV','1vX','Plants','HS%','Kills','Deaths','KDMargin','FB','FD','En
 ↪#columms to change data type
ele_sqs[cols] = ele_sqs[cols].apply(pd.to_numeric) #change data types
ele_sqs
```

[9]:

```
       Player  Rating    K-D (+/-) Entry (+/-)  KOST   KPR  SRV  1vX  Plants  \
0       Rexen    1.05   25-25 (+0)    2-6 (-4)    64  0.69   31    2       0
1       Gryxr    1.35  39-24 (+15)    7-3 (+4)    72  1.08   33    2       1
2        Supr    0.85   13-22 (-9)    1-2 (-1)    64  0.36   39    0       4
3      Sapper    1.03   24-23 (+1)    1-0 (+1)    72  0.67   36    0       2
4     Nay..Pew   0.86   22-26 (-4)    6-5 (+1)    56  0.61   28    0       0
5         DCH    0.82   18-25 (-7)    1-1 (+0)    61  0.50   31    0       1
6      Kanzen    0.95   22-25 (-3)    4-1 (+3)    72  0.61   31    0       0
7  sprOnigiri    0.83   24-28 (-4)   5-11 (-6)    56  0.67   22    0       0
8        Yeti    0.96   25-25 (+0)    6-4 (+2)    61  0.69   31    0       0
9       Nerix    1.09   32-23 (+9)    3-3 (+0)    56  0.89   36    0       0

   HS%       Atk     Def     Team  Kills  Deaths  KDMargin  FB  FD  EntryMargin
0   76       Ace   Jager   Soniqs     25      25         0   2   6           -4
1   36    Twitch    Mira   Soniqs     39      24        15   7   3            4
2   23  Thermite   Smoke   Soniqs     13      22        -9   1   2           -1
3   38  Thermite   Smoke  Elevate     24      23         1   1   0            1
4   29  Maverick    Mute  Elevate     22      26        -4   6   5            1
5   39    Sledge   Jager  Elevate     18      25        -7   1   1            0
6   50      Buck    Kaid   Soniqs     22      25        -3   4   1            3
7   63      Iana   Aruni  Elevate     24      28        -4   5  11           -6
8   64  Maverick    Mute   Soniqs     25      25         0   6   4            2
9   53    Jackal    Mira  Elevate     32      23         9   3   3            0
```

I want only data from the Soniqs team, so I divide their data into a different dataframe. I also check their datatypes to make sure every column is changed correctly. This concludes the ETL of this dataframe. I can now use this clean dataframe for visualization.

[10]:

```python
sqs = ele_sqs[ele_sqs.Team =='Soniqs'] #take only Soniqs players
sqs.dtypes
```

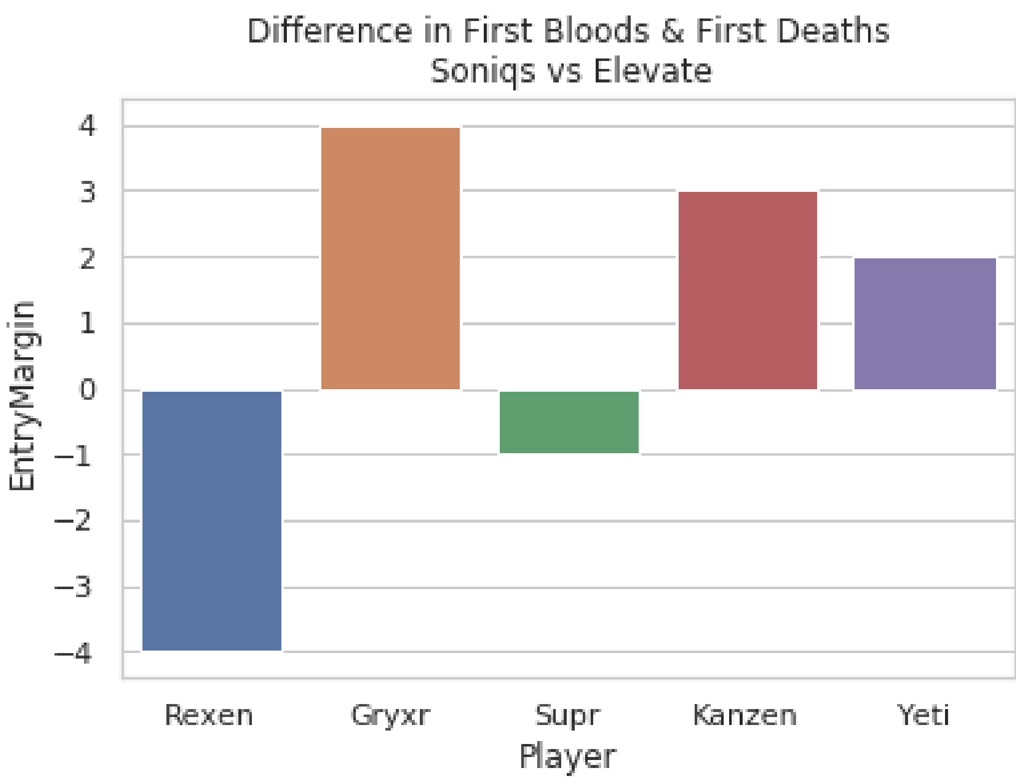
[10]:

```
Player          object
Rating         float64
K-D (+/-)       object
Entry (+/-)     object
KOST             int64
KPR            float64
SRV              int64
1vX              int64
Plants           int64
```

```
HS%                int64
Atk               object
Def               object
Team              object
Kills              int64
Deaths             int64
KDMargin           int64
FB                 int64
FD                 int64
EntryMargin        int64
dtype: object
```

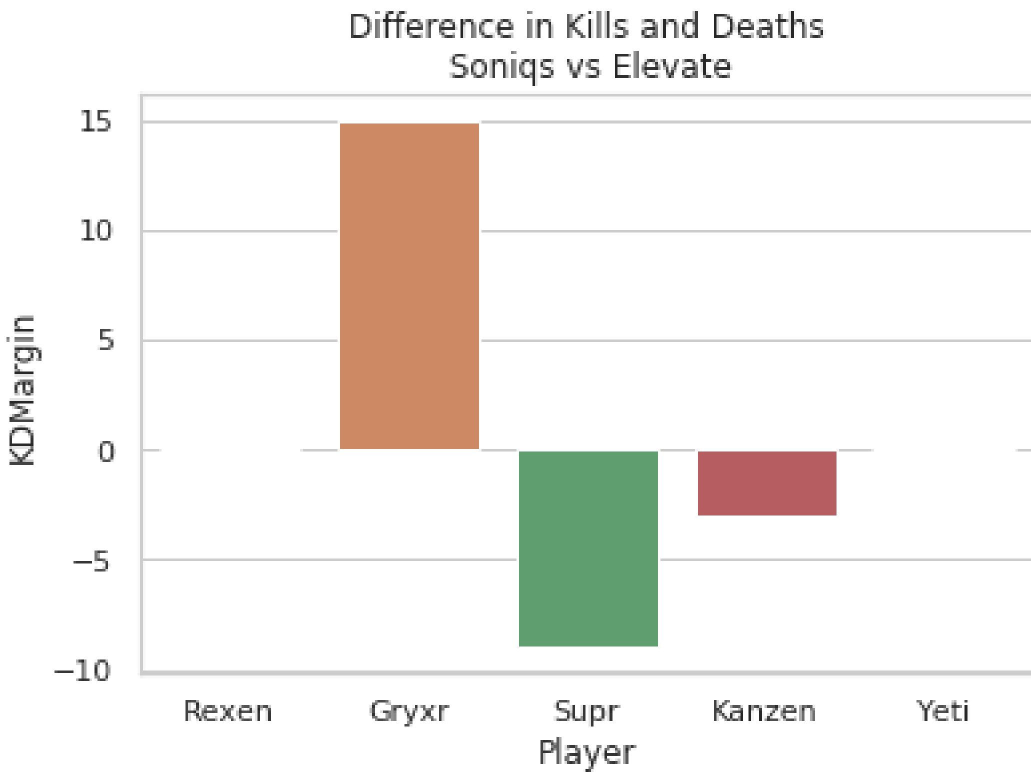
For these visualizations I want gridlines to make the graph easily understandable at a glance. This first graph is a barchart of First Bloods and First Deaths. This Entry Margin is important to understand, because the higher scores mean a better team player that can open up sites for the team. In this scenario, Gryxr performed the best while Rexen has a bad series in terms of holding and taking sites.

[11]:
```
sns.set(style="whitegrid") #make a background for the visualizations
entry_frags = sns.barplot(x='Player',y='EntryMargin', data=sqs).
 ↪set(title='Difference in First Bloods & First Deaths\n Soniqs vs Elevate')
entry_frags
plt.savefig('sqs_elevate_entry_frags.png')
```



KD margin provides info if the player got more or less kills in relation to thier deaths in the series. The goal for a player is to go even throughout a series. Going 1 for 1 in a gunfight is the baseline for any round. In this series against Elevate Gryxr had the largest Kill to Death margin. This could imply that Gryxr carried the game for Soniqs, or that Gryxr was in a good position to trade his other teammates.

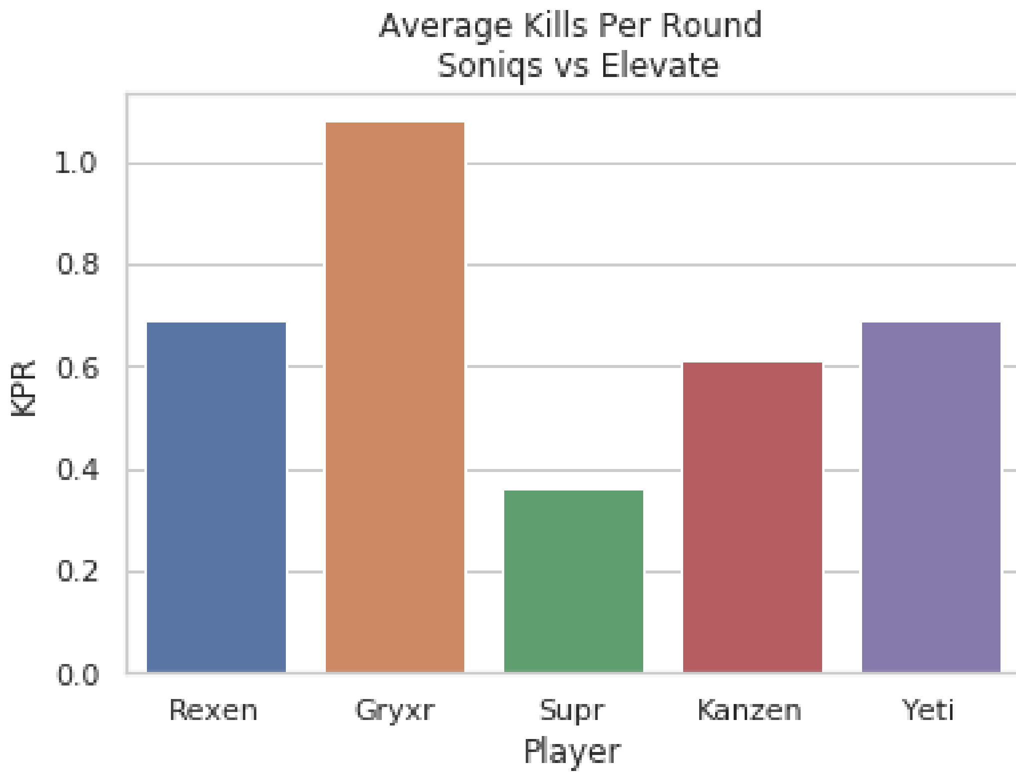
[12]:
```
kd_margin = sns.barplot(x='Player',y='KDMargin', data=sqs).
 ↪set(title='Difference in Kills and Deaths\n Soniqs vs Elevate')
kd_margin
plt.savefig('sqs_elevate_kd_margin.png')
```



This last plot backs up the Kill Death graph above. This graph shows average kills per round, and again Gryxr is back on top with over a 1.0 average kills per round. This means that Gryxr was likely killing 1-2 per round they played in this series. This graph shows that the other players were also providing kills, but they were also dying just as much. After looking at all 3 graphs, you can see that Gryxr performed the best on Soniqs this series. I now want to pull a couple more series from the R6 playoff where Soniqs plays against other teams. This could help me decide if Gryxr is always this good or if this was a fluke series.

[37]:
```
kpr = sns.barplot(x='Player',y='KPR', data=sqs).set(title='Average Kills Per␣
 ↪Round\n Soniqs vs Elevate')
```

```
kpr
plt.savefig('sqs_elevate_kpr.png')
```



Now that the data has been pulled and reviewed, I think I can do a faster job by automating some of the steps. I broke this into 2 functions that can pull the data and transform it into an uncleaned dataframe. The data needs to be cleaned manually for this scenario, because scraping from a website can provide different outcomes that need different functions to clean the data.

This first function pull_table_data requires a url to pull the columns and data from the website. This is the same process I used above, but now automated to return the column and data from any match given the url. The data is not cleaned yet, so the next function pulls the text from each cell of the data.

[14]:
```
def pull_table_data(url):
    '''pull the uncleaned column names and table rows'''
    r = requests.get(url)
    print(r.status_code)
    soup = BeautifulSoup(r.content, 'html')
    table = soup.find(id="playertable")
    header_row1 = table.find_all('tr')[0]
    nc_table_data = table.find_all('tr')[1:]
    return header_row1, nc_table_data
```

This second function get_columns_and_rows takes the uncleaned data table and returns a dataframe. The functions pulls the text and removes the white spaces before and after each cell entry. Then transforms the list of data into a dataframe that can be cleaned further with any specific needs.

[15]:
```
def get_columns_and_rows(columns,rows):
    '''take the uncleaned data and transform it into a dataframe'''
    column_names = []
    new_column = 'Player'
    for each in columns('th'):
        column_names.append(each.text)
        column_names = column_names[1:]
        column_names.insert(0, new_column)
    row_data = []
    for each in rows:
        for x in each('td'):
            row_data.append(x.text)
            row_data = [x.strip(' ') for x in row_data]
    full_table = []
    full_table = (zip(cycle(column_names),row_data))
    full_table = list(full_table)
    full_dict = {}
    for i in full_table:
        full_dict.setdefault(i[0],[]).append(i[1])
    df = pd.DataFrame(full_dict)
    return df
```

This calls the first function with the url to the Soniqs vs Damwon game on Feb 22, 2022. I had the function print the status code of the website, so I would still know that the website is working when I see a 200 as a response.

[16]:
```
url1 = 'https://siege.gg/matches/7201-invitational-intl-soniqs-vs-dwg-kia'
nc_td = pull_table_data(url1)
```

```
200
```

This next call to the get_columns_and_rows function uses the stored data from the first function. The headers for the columns are stored in the first indexed function call and the data for the rows is stored in the second index.

[17]:
```
df = get_columns_and_rows(nc_td[0],nc_td[1])
```

This is what the dataframe looks like after both functions are called. The data looks good, but there is more that I want to change on my own. I want to separate the kills/deaths, and the first bloods/first deaths columns again. I want to identify the team and change the number to the proper team. In this case 125 is Soniqs and 120 is Damwon. I want to change the columns to numeric values and remove the % signs from the columns that contain them. Once all this is finished, I have a cleaned dataset that I can use for analysis.

[18]:
```
df #this is a preview of the table as it looks on the website
```

[18]:
```
    Player Rating     K-D (+/-) Entry (+/-) KOST   KPR  SRV 1vX Plants  HS%  \
0    Rexen   1.04   21-18 (+3)    4-1 (+3)  58%  0.88  25%   0      0  40%
1     yass   0.98   18-18 (+0)    3-4 (-1)  75%  0.75  25%   0      0  75%
2    Gryxr   1.61  30-13 (+17)    6-1 (+5)  88%  1.25  46%   1      1  37%
3     Supr   0.69   5-17 (-12)    1-0 (+1)  63%  0.21  29%   0      4  40%
4   Kanzen   1.26   18-14 (+4)    2-2 (+0)  71%  0.75  42%   1      0  59%
5     Yeti   0.95   13-16 (-3)    3-4 (-1)  71%  0.54  33%   0      1  50%
6 Woogiman   0.91   14-16 (-2)    1-5 (-4)  54%  0.58  33%   1      3  50%
7    coted   0.92   16-18 (-2)    1-1 (+0)  63%  0.67  25%   0      2  20%
8      RIN   0.82   14-19 (-5)    0-3 (-3)  58%  0.58  21%   0      0  57%
9  CATsang   0.88   16-18 (-2)    3-3 (+0)  58%  0.67  25%   0      1  69%

       Atk    Def Team
0   Twitch  Wamai  125
1     Iana  Alibi  120
2    Finka   Mute  125
3 Thermite  Smoke  125
4     Buck  Smoke  125
5 Maverick  Wamai  125
6 Thermite  Smoke  120
7   Hibana   Mute  120
8    Zofia  Jager  120
9     Buck  Aruni  120
```

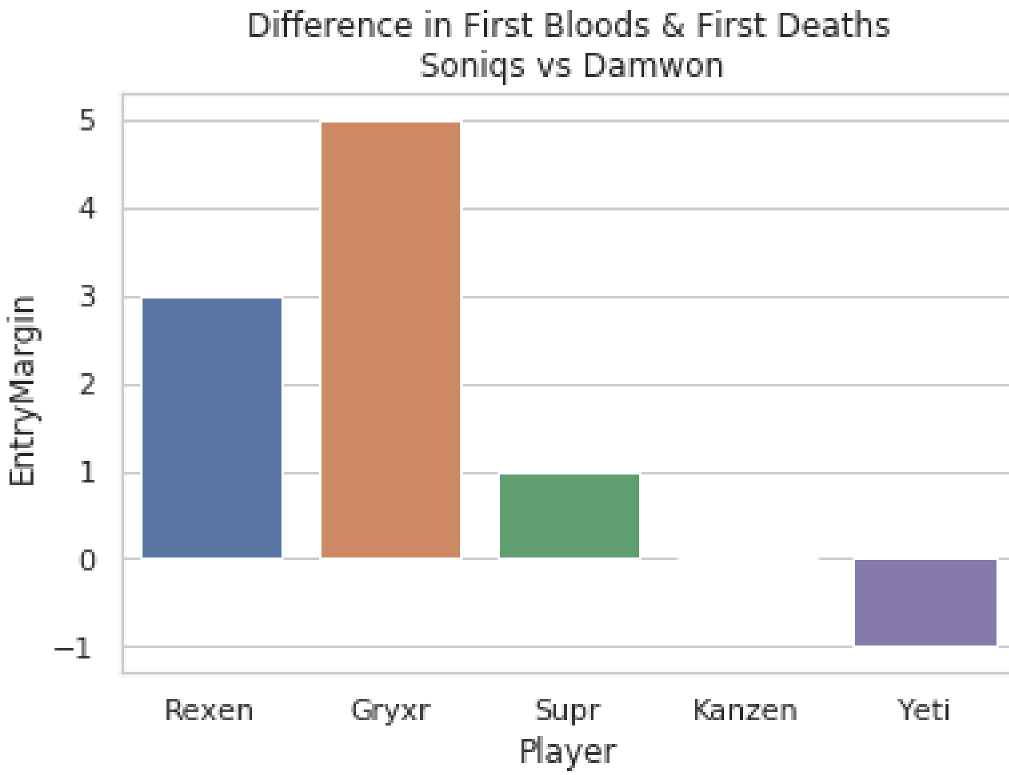
[19]:
```
df[['Kills','Deaths']] = df['K-D (+/-)'].str.split('-', n=1,expand=True)
df[['Deaths','KDMargin']] = df['Deaths'].str.split('(', n=1,expand=True)
df['KDMargin'] = df['KDMargin'].str.replace(')','')
df[['FB','FD']] = df['Entry (+/-)'].str.split('-', n=1,expand=True)
df[['FD','EntryMargin']] = df['FD'].str.split('(', n=1,expand=True)
df['EntryMargin'] = df['EntryMargin'].str.replace(')','')
df['KOST'] = df['KOST'].str.replace('%','')
df['HS%'] = df['HS%'].str.replace('%','')
df['SRV'] = df['SRV'].str.replace('%','')
df['Team'] = df['Team'].replace(['125'],'Soniqs')
df['Team'] = df['Team'].replace(['120'],'Damwon')
cols =␣
 ↪['Rating','KOST','KPR','SRV','1vX','Plants','HS%','Kills','Deaths','KDMargin','FB','FD','En
df[cols] = df[cols].apply(pd.to_numeric)
```

For this project I am still only looking at Soniqs, so I split the dataset into only the Soniqs players. I decided to plot the same graphs as above hoping to find some relationship between the games by looking at the same players.

[20]:
```
dam_sqs = df
sqs1 = dam_sqs[dam_sqs.Team =='Soniqs']
```

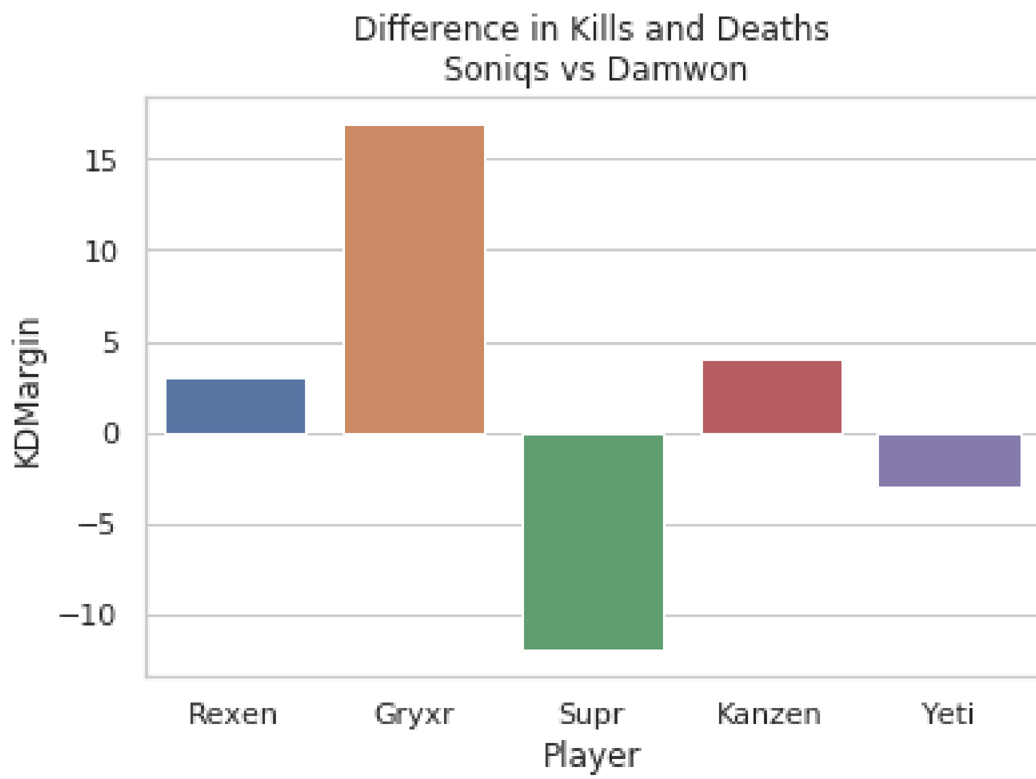
This first graph shows that Gryxr is still on top for entry kills. At this point I have an idea that Gryxr might be a consistent player since he was on top for all the other graphs from the Soniqs vs Elevate games above.

[21]:
```
entry_frags = sns.barplot(x='Player',y='EntryMargin', data=sqs1).
 ↪set(title='Difference in First Bloods & First Deaths\n Soniqs vs Damwon')
entry_frags
plt.savefig('sqs_dw_entry_frags.png')
```



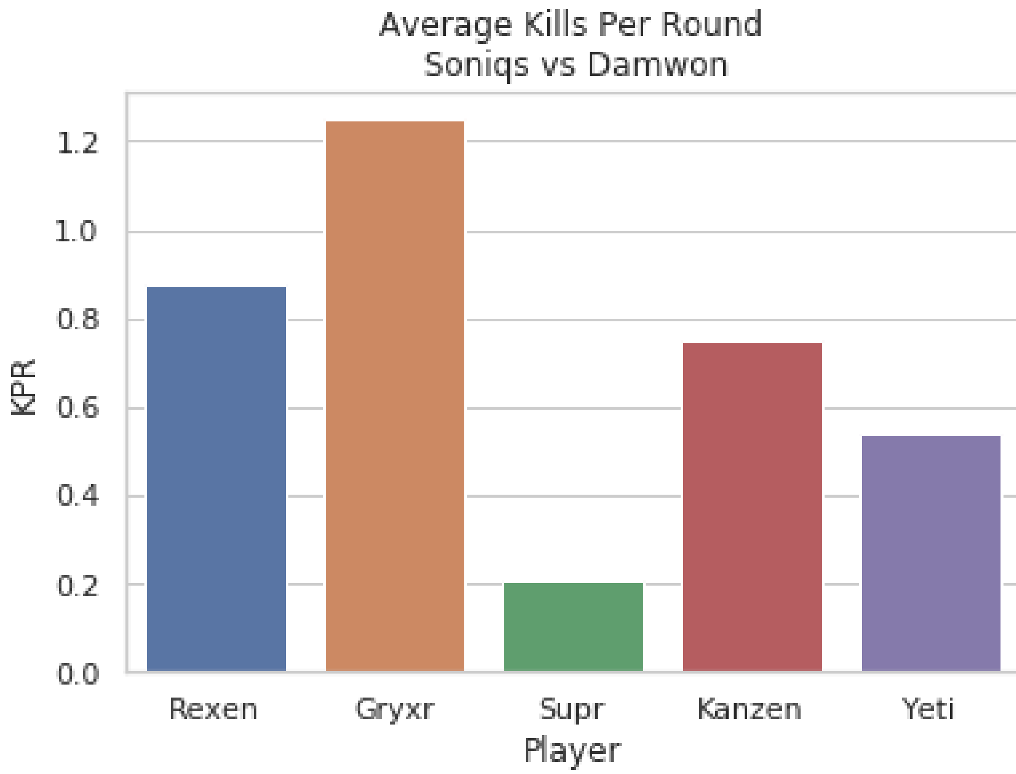
The Kill Death margin is almost the exact same as the series vs Elevate. All players have almost even kill death margin, but Gryxr stands out.

[22]:
```
kd_margin = sns.barplot(x='Player',y='KDMargin', data=sqs1).
 ↪set(title='Difference in Kills and Deaths\n Soniqs vs Damwon')
kd_margin
plt.savefig('sqs_dw_kd_margin.png')
```



My last graph for this series further emphasizes the performance of the players. It seems everyone had a better series against Damwon than against Elevate with the exception of Supr.

[23]:
```
kpr = sns.barplot(x='Player',y='KPR', data=sqs1).set(title='Average Kills Per␣
↪Round\n Soniqs vs Damwon')
kpr
plt.savefig('sqs_dw_kpr.png')
```



This last function call pulls the data from the playoff games against Empire. This last series goes to further visualize the performance of the Soniqs players.

[24]:
```
url2 = 'https://siege.gg/matches/7206-invitational-intl-team-empire-vs-soniqs'
nc_td = pull_table_data(url2)
df = get_columns_and_rows(nc_td[0],nc_td[1])
```

200

[25]:
```
df[['Kills','Deaths']] = df['K-D (+/-)'].str.split('-', n=1,expand=True)
df[['Deaths','KDMargin']] = df['Deaths'].str.split('(', n=1,expand=True)
df['KDMargin'] = df['KDMargin'].str.replace(')','')
df[['FB','FD']] = df['Entry (+/-)'].str.split('-', n=1,expand=True)
df[['FD','EntryMargin']] = df['FD'].str.split('(', n=1,expand=True)
df['EntryMargin'] = df['EntryMargin'].str.replace(')','')
df['KOST'] = df['KOST'].str.replace('%','')
df['HS%'] = df['HS%'].str.replace('%','')
df['SRV'] = df['SRV'].str.replace('%','')
df['Team'] = df['Team'].replace(['125'],'Soniqs')
df['Team'] = df['Team'].replace(['63'],'Empire')
cols =␣
 ↪['Rating','KOST','KPR','SRV','1vX','Plants','HS%','Kills','Deaths','KDMargin','FB','FD','En
```

```
df[cols] = df[cols].apply(pd.to_numeric)
```

[26]:
```
emp_sqs = df
sqs2 = emp_sqs[emp_sqs.Team =='Soniqs']
sqs2
```

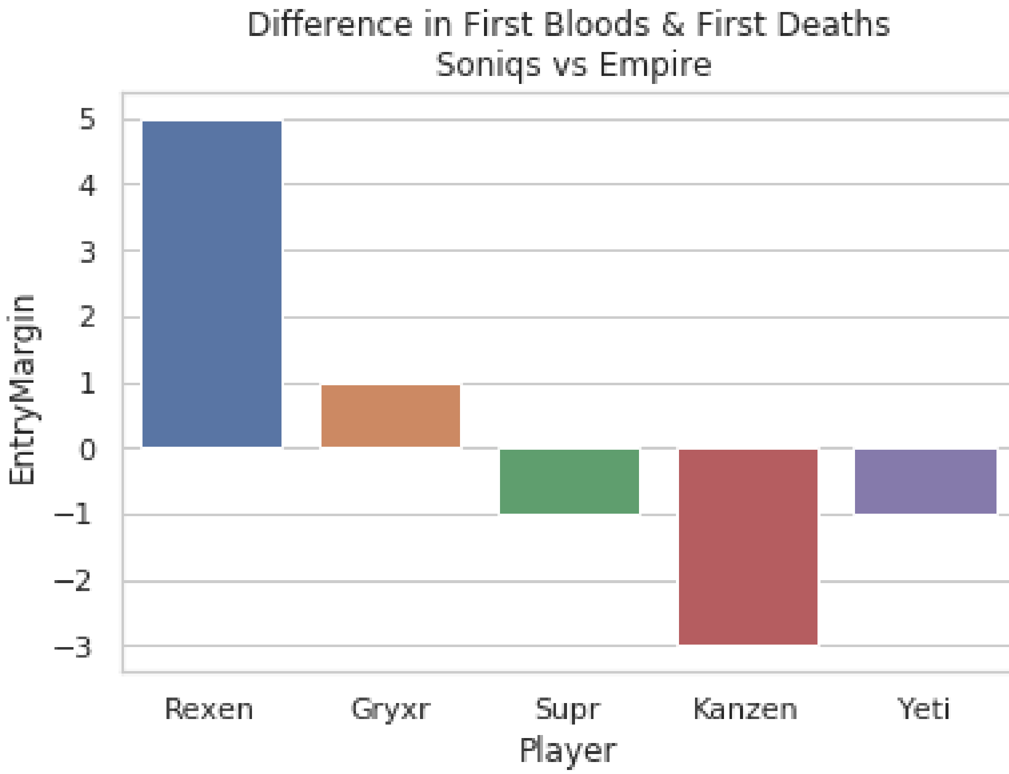
[26]:

	Player	Rating	K-D (+/-)	Entry (+/-)	KOST	KPR	SRV	1vX	Plants	HS%
0	Rexen	1.29	36-25 (+11)	9-4 (+5)	74	1.03	29	1	0	40
1	Gryxr	1.07	31-26 (+5)	4-3 (+1)	63	0.89	26	0	0	55
2	Supr	0.77	16-28 (-12)	0-1 (-1)	60	0.46	20	1	1	27
3	Kanzen	0.60	15-29 (-14)	3-6 (-3)	34	0.43	17	0	1	50
5	Yeti	0.82	18-26 (-8)	2-3 (-1)	60	0.51	26	0	0	33

	Atk	Def	Team	Kills	Deaths	KDMargin	FB	FD	EntryMargin
0	Finka	Aruni	Soniqs	36	25	11	9	4	5
1	Finka	Mira	Soniqs	31	26	5	4	3	1
2	Ace	Smoke	Soniqs	16	28	-12	0	1	-1
3	Sledge	Wamai	Soniqs	15	29	-14	3	6	-3
5	Hibana	Mute	Soniqs	18	26	-8	2	3	-1

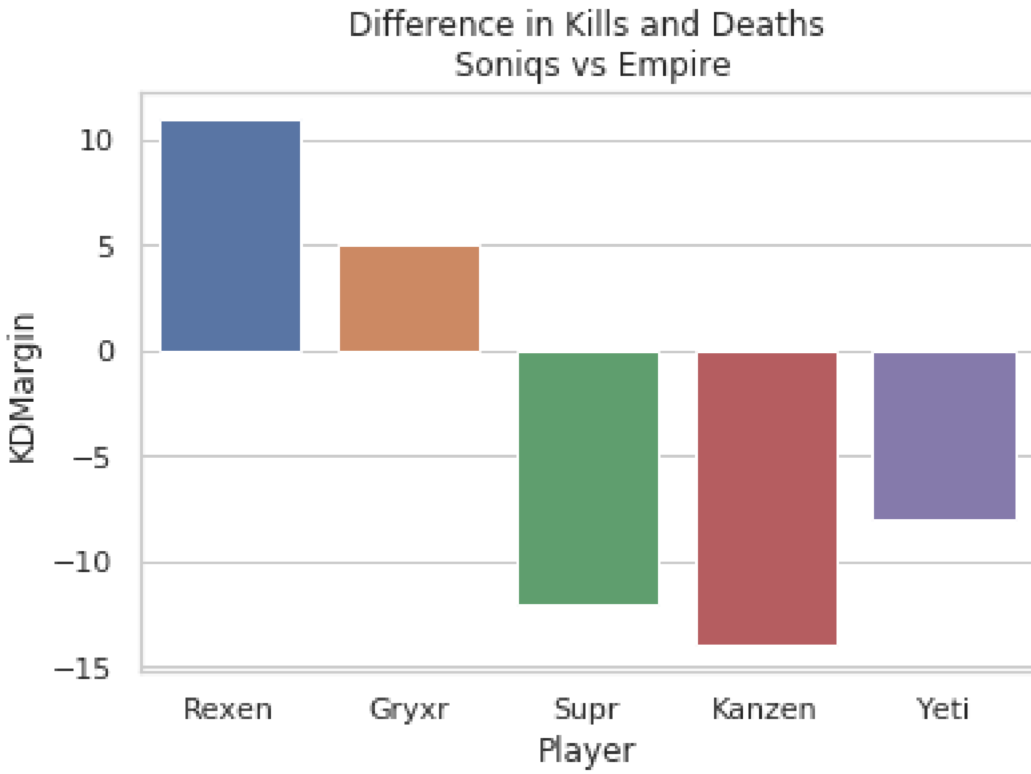
This match seems like the whole team struggled a bit to find their footing. Rexen had a great game with 5 more first bloods than first deaths. Gryxr still managed to perform with a positive margin in these games as well.

[38]:
```
entry_frags = sns.barplot(x='Player',y='EntryMargin', data=sqs2).
 ↪set(title='Difference in First Bloods & First Deaths\n Soniqs vs Empire')
entry_frags
plt.savefig('sqs_emp_entry_frags.png')
```

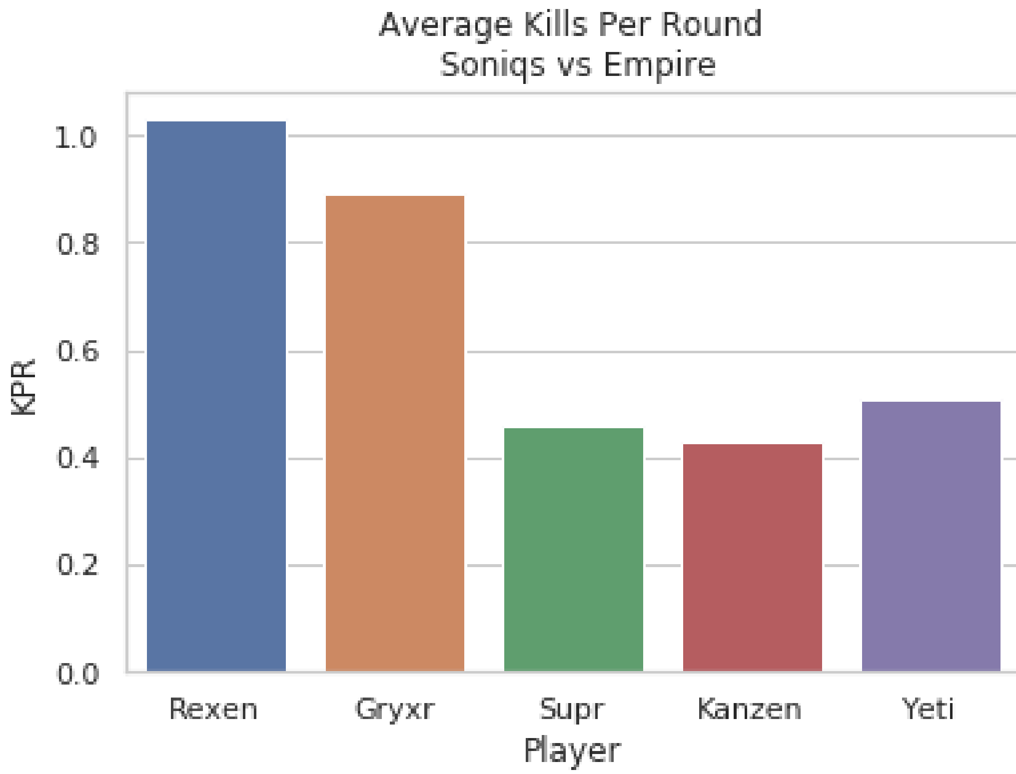


The next 2 graphs just futher explain the series against Empire. Supr has not performed very well in any of the games observed. This is a small sample size, but with more analysis Supr might be looking like a weak link.

[39]:
```
kd_margin = sns.barplot(x='Player',y='KDMargin', data=sqs2).
 ↪set(title='Difference in Kills and Deaths\n Soniqs vs Empire')
kd_margin
plt.savefig('sqs_emp_kd_margin.png')
```



[40]:
```
kpr = sns.barplot(x='Player',y='KPR', data=sqs2).set(title='Average Kills Per␣
↪Round\n Soniqs vs Empire')
kpr
plt.savefig('sqs_emp_kpr.png')
```



Conclusion: After looking at these 3 series, I can see that Gryxr is consistantly the best player for Soniqs. He is getting the highest amount of entry kills and still getting more kills than deaths each round. This means that Gryxr is creating space for his team while also being a threat himself. If I were to be playing against Soniqs, I would be looking at what Gryxr does each round to find out more about his playstyle. Shutting him down would be a good way to take down Soniqs. Seeing where Gryxr plays on each map or which Operator he plays could give the opportunity to counter him. Rexen had a a few ups and downs in the playoffs, but looks to be a strong player.

The next part of this project comes from data taken from the gameplay VODs and round by round information from Siegegg. I created 3 datasets that covered the round info, the picks and bans of maps and operators. I am looking to find out a little bit more about how the team performs, and what other teams do to counter Soniqs.

[30]:
```
game_sum = pd.read_csv('SoniqsPrepDocGameSummary.csv')
game_sum
```

[30]:
```
    SeriesId  GameId    Map  Round Objective1 Objective2   Winner  \
0          1       1  Villa      1    Aviator       Game  Elevate
1          1       1  Villa      2     Trophy     Statue  Elevate
2          1       1  Villa      3    Kitchen     Dining  Elevate
3          1       1  Villa      4    Aviator       Game  Elevate
4          1       1  Villa      5     Trophy     Statue  Elevate
..       …       …      …      …          …          …
```

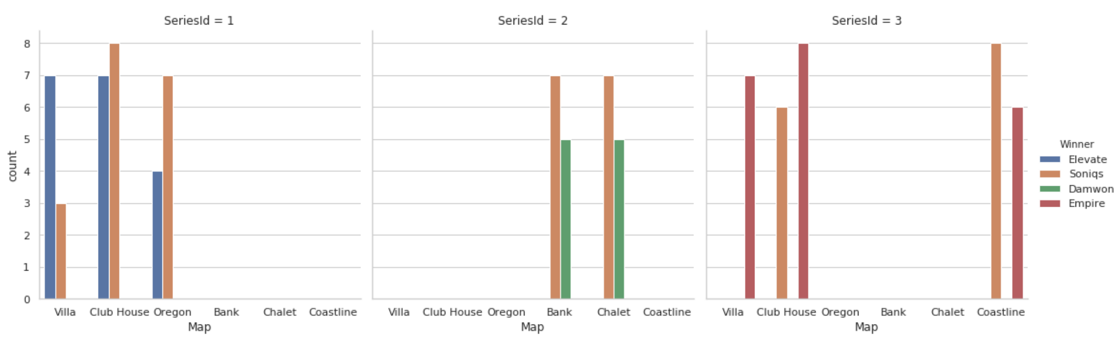
```
90          3        3   Villa       3    Kitchen     Dining   Empire
91          3        3   Villa       4    Aviator       Game   Empire
92          3        3   Villa       5     Trophy     Statue   Empire
93          3        3   Villa       6    Kitchen     Dining   Empire
94          3        3   Villa       7    Aviator       Game   Empire

    Soniqs Score  Opponent Score Attacker Defender
0              0               1   Soniqs  Elevate
1              0               2   Soniqs  Elevate
2              0               3   Soniqs  Elevate
3              0               4   Soniqs  Elevate
4              0               5   Soniqs  Elevate
..           …               …      …        …
90             0               3   Soniqs   Empire
91             0               4   Soniqs   Empire
92             0               5   Soniqs   Empire
93             0               6   Soniqs   Empire
94             0               7   Empire   Soniqs

[95 rows x 11 columns]
```

This first plot comes from the round by round dataset that covers all 3 playoff games from above. The plot shows which team won on each given map and how many rounds were played. Soniqs played significantly more rounds against Empire and Elevate compared to Damwon. The only repeating maps that were played across 2 series were Villa and Club House.

[31]:
```
round_per_map = sns.catplot(x='Map', hue='Winner', col='SeriesId',␣
↪data=game_sum, kind='count')
round_per_map
plt.savefig('rounds_per_map.png')
```



This dataset is the map picks and bans in order by team. For the next visualization I want to see what maps were banned.

[32]:
```python
map_bans = pd.read_csv('SoniqsPrepDocMapBans.csv')
map_bans
```

[32]:

	SeriesId	Team	MapVetos	Ban	Pick	Decider
0	1	Soniqs	Kafe	1	0	0
1	1	Elevate	Coastline	1	0	0
2	1	Soniqs	Villa	0	1	0
3	1	Elevate	Club House	0	1	0
4	1	Soniqs	Bank	1	0	0
5	1	Elevate	Chalet	1	0	0
6	1	Both	Oregon	0	0	1
7	2	Soniqs	Villa	1	0	0
8	2	Damwon	Club House	1	0	0
9	2	Soniqs	Bank	0	1	0
10	2	Damwon	Chalet	0	1	0
11	2	Soniqs	Coastline	1	0	0
12	2	Damwon	Kafe	1	0	0
13	2	Both	Oregon	0	0	1
14	3	Empire	Chalet	1	0	0
15	3	Soniqs	Kafe	1	0	0
16	3	Empire	Coastline	0	1	0
17	3	Soniqs	Club House	0	1	0
18	3	Empire	Oregon	1	0	0
19	3	Soniqs	Bank	1	0	0
20	3	Both	Villa	0	0	1

[33]:
```python
sqs_map_bans = map_bans.copy()[map_bans['Ban']==1]
sqs_map_bans
```

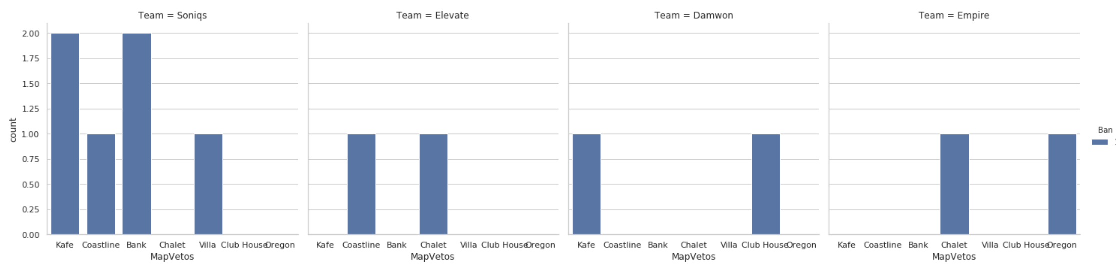
[33]:

	SeriesId	Team	MapVetos	Ban	Pick	Decider
0	1	Soniqs	Kafe	1	0	0
1	1	Elevate	Coastline	1	0	0
4	1	Soniqs	Bank	1	0	0
5	1	Elevate	Chalet	1	0	0
7	2	Soniqs	Villa	1	0	0
8	2	Damwon	Club House	1	0	0
11	2	Soniqs	Coastline	1	0	0
12	2	Damwon	Kafe	1	0	0
14	3	Empire	Chalet	1	0	0
15	3	Soniqs	Kafe	1	0	0
18	3	Empire	Oregon	1	0	0
19	3	Soniqs	Bank	1	0	0

In this visualization there are repeat bans for Bank and Kafe. This means that Soniqs banned these in multiple series. Kafe and Bank might be a weak point for Soniqs that they choose to ban because they do not want to play. Looking at the other team's bans against Soniqs, I notice that the maps are similar to Soniqs' bans. The opposing teams also ban Chalet, this may imply that

Soniqs is good at playing Chalet. A consideration for picks and bans could be whether the team was on attack or defense to start the game. Some maps may be attack or defense sided.

[41]:
```python
map_bans_by_team = sns.catplot(x='MapVetos', hue='Ban',col='Team',
 ↪data=sqs_map_bans, kind='count')
map_bans_by_team
plt.savefig('map_bans.png')
```



The last dataset is Operator pick and bans by team. The most notable part of this dataset is that Soniqs banned Valkyrie and Thatcher every game. This signals to me that they do not want to play against these operators. The other teams banned an assortment of operators each series. This makes it hard to tell which operators would be best to ban against Soniqs' players.

[35]:
```python
op_bans = pd.read_csv('SoniqsPrepDocOperatorBans.csv')
op_bans
```

[35]:

	SeriesId	GameId	Team	Bans
0	1	1	Elevate	Finka
1	1	1	Soniqs	Hibana
2	1	1	Soniqs	Valkyrie
3	1	1	Elevate	Goyo
4	1	1	Both	Thorn
5	1	2	Elevate	Finka
6	1	2	Soniqs	Thatcher
7	1	2	Soniqs	Valkyrie
8	1	2	Elevate	Wamai
9	1	2	Both	Thorn
10	1	3	Elevate	Finka
11	1	3	Soniqs	Jackal
12	1	3	Soniqs	Valkyrie
13	1	3	Elevate	Wamai
14	1	3	Both	Thorn
15	2	1	Damwon	Nokk
16	2	1	Soniqs	Hibana
17	2	1	Soniqs	Kaid
18	2	1	Damwon	Mira
19	2	1	Both	Thorn

20	2	2	Damwon	Zero
21	2	2	Soniqs	Thatcher
22	2	2	Soniqs	Valkyrie
23	2	2	Damwon	Mira
24	2	2	Both	Thorn
25	3	1	Empire	Nomad
26	3	1	Soniqs	Hibana
27	3	1	Soniqs	Smoke
28	3	1	Empire	Valkyrie
29	3	1	Both	Thorn
30	3	2	Soniqs	Thatcher
31	3	2	Empire	Maverick
32	3	2	Empire	Kaid
33	3	2	Soniqs	Valkyrie
34	3	2	Both	Thorn
35	3	3	Empire	Thatcher
36	3	3	Soniqs	Hibana
37	3	3	Soniqs	Kaid
38	3	3	Empire	Valkyrie
39	3	3	Both	Thorn

[44]:
```
sqs_bans = op_bans[op_bans.Team=='Soniqs']
sqs_bans
```

[44]:

	SeriesId	GameId	Team	Bans
1	1	1	Soniqs	Hibana
2	1	1	Soniqs	Valkyrie
6	1	2	Soniqs	Thatcher
7	1	2	Soniqs	Valkyrie
11	1	3	Soniqs	Jackal
12	1	3	Soniqs	Valkyrie
16	2	1	Soniqs	Hibana
17	2	1	Soniqs	Kaid
21	2	2	Soniqs	Thatcher
22	2	2	Soniqs	Valkyrie
26	3	1	Soniqs	Hibana
27	3	1	Soniqs	Smoke
30	3	2	Soniqs	Thatcher
33	3	2	Soniqs	Valkyrie
36	3	3	Soniqs	Hibana
37	3	3	Soniqs	Kaid

[45]:
```
enemy_team_bans = op_bans[op_bans.Team!='Soniqs']
enemy_team_bans
```

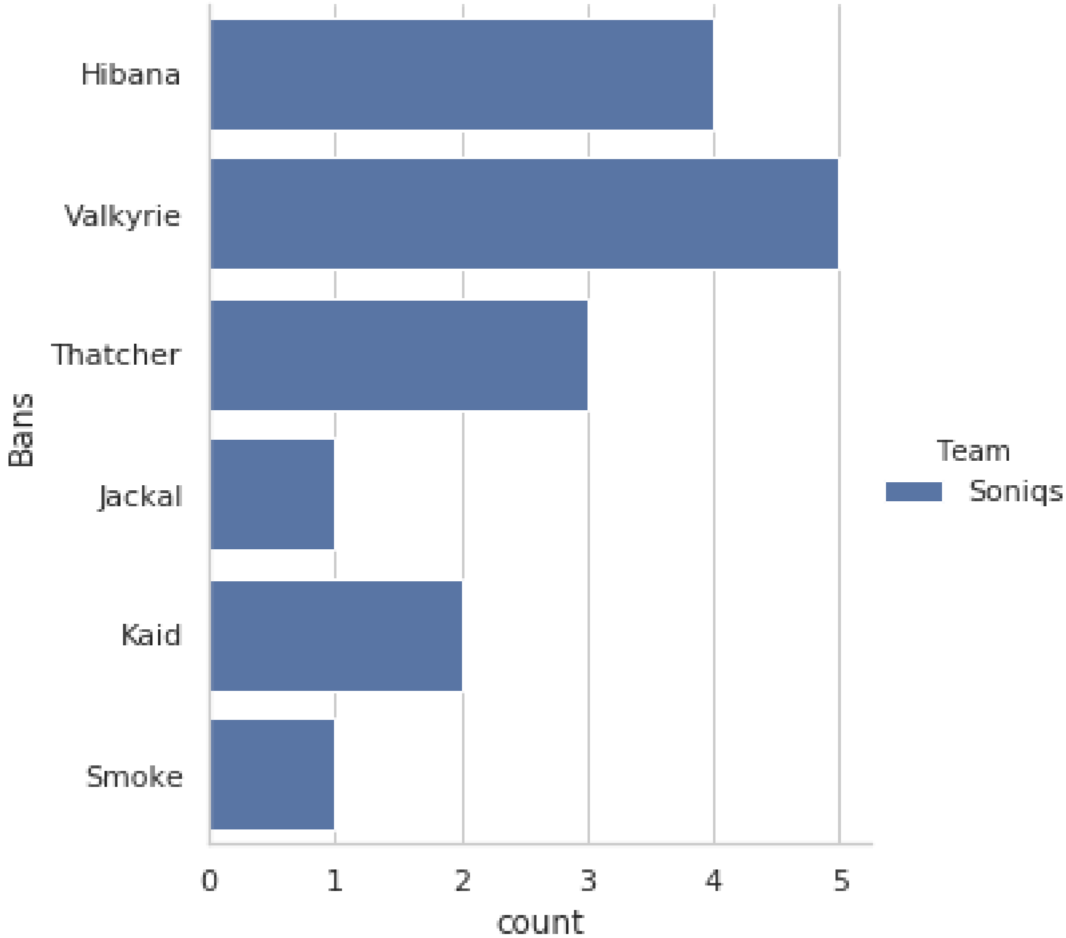
[45]:

	SeriesId	GameId	Team	Bans
0	1	1	Elevate	Finka

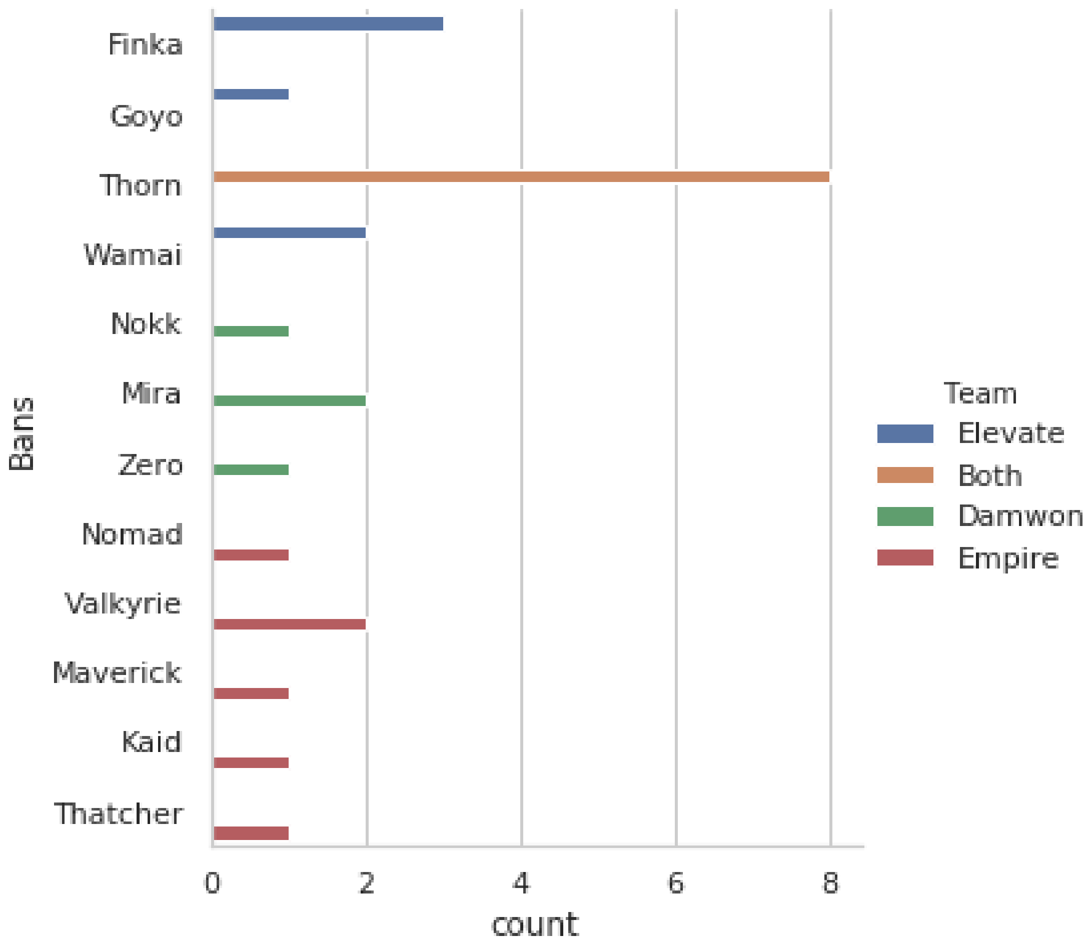
```
3           1        1  Elevate      Goyo
4           1        1     Both     Thorn
5           1        2  Elevate     Finka
8           1        2  Elevate     Wamai
9           1        2     Both     Thorn
10          1        3  Elevate     Finka
13          1        3  Elevate     Wamai
14          1        3     Both     Thorn
15          2        1   Damwon      Nokk
18          2        1   Damwon      Mira
19          2        1     Both     Thorn
20          2        2   Damwon      Zero
23          2        2   Damwon      Mira
24          2        2     Both     Thorn
25          3        1   Empire     Nomad
28          3        1   Empire  Valkyrie
29          3        1     Both     Thorn
31          3        2   Empire  Maverick
32          3        2   Empire      Kaid
34          3        2     Both     Thorn
35          3        3   Empire  Thatcher
38          3        3   Empire  Valkyrie
39          3        3     Both     Thorn
```

[46]:

```
sqs_op_bans = sns.catplot(y='Bans', hue='Team', data=sqs_bans, kind='count')
sqs_op_bans
plt.savefig('sqs_op_bans.png')
```



[47]:
```
enemy_op_bans = sns.catplot(y='Bans', hue='Team', data=enemy_team_bans,␣
 ↪kind='count')
enemy_op_bans
plt.savefig('enemy_op_bans.png')
```



After visualizing data from series to series, I can see trends from Soniqs. Soniqs looks to ban Valkyrie, Thatcher for operators and Kafe, Bank for maps. They have playoff practice games on Club House and Villa. This means that these games can be reviewed before playing against Soniqs to find playstyle patterns on those maps.