Motivation:

As a student seeking a project focused on data analysis, I propose researching my own Overwatch 2 gameplay hero statistics. Analyzing my own performance increases my level of interest and engagement in the project. Understanding my strengths and weaknesses directly motivates me to examine the data more deeply to determine my areas for development.

Data source:

Obtained the Overwatch 2 gameplay hero statistics data from the OverFast API, accessible at the following website: https://overfast-api.tekrop.fr/. The API is maintained by Valentin "TeKrop" PORCHET, and you can reach out to him at valentin.porchet@proton.me. The source code and documentation for the OverFast API are available on GitHub at https://github.com/TeKrop/overfast-api, released under the MIT License. The API employs FastAPI and Beautiful Soup, utilizing nginx as a reverse proxy and Redis for caching. Through a Refresh-Ahead cache system, it minimizes calls to Blizzard pages, ensuring efficient and accurate data retrieval. For exploring and testing API calls, Swagger UI is available at https://overfast-api.tekrop.fr/docs, and you can monitor the status on the status page.

```python
import requests

url = "https://overfast-api.tekrop.fr/players/WhoisKA-2138/stats/summary?gamemode=quickplay&platform=pc"

response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    print(data)
else:
    print(f"Error: {response.status_code}")
    print(response.text)
```

The provided Python code showcases the use of the `requests` library to access Overwatch 2 player statistics through the OverFast API. The script begins by importing the `requests` library, a commonly employed tool for making HTTP requests in Python. It then defines the API URL, specifying parameters such as the player's username, the desired game mode (quickplay), and the platform (pc). A GET request is initiated using `requests.get(url)`, and the response from the server is stored in the `response` variable. The script checks if the response

status code is 200, indicating a successful request. In case of success, the JSON-formatted data is processed into a Python dictionary using `response.json()` and printed. If an error occurs (status code other than 200), the script prints an error message along with the status code and any additional error information. This code snippet provides a foundational example of how to programmatically retrieve Overwatch 2 player statistics from the OverFast API using Python.

Data analysis:

In the exploratory data analysis (EDA) of the Overwatch 2 gameplay statistics, several analyses were conducted to uncover valuable insights from the dataset.

```
high_winrate_heroes = df[df['winrate'] > 50]
print(high_winrate_heroes)
```

This analysis focused on identifying heroes I played who have a win rate exceeding 50%. The resulting high_winrate_heroes DataFrame contains only those heroes who have demonstrated a success rate above the threshold.

```
top8_heroes = df.nlargest(8, 'games_played')

# Display the filtered DataFrame
print(top8_heroes)
```

Another aspect of the analysis involved identifying the top 8 heroes based on the number of games I played. The top8_heroes DataFrame showcases the heroes that have been played most frequently by me.

```
selected_columns = ['heroes', 'time_played', 'kda']
filtered_data = df[selected_columns]
print(filtered_data)
```

To streamline the presentation of information, a subset of columns was selected, including 'heroes', 'time_played', and 'kda'. The resulting filtered_data DataFrame provides a condensed

view of essential gameplay statistics for each hero, offering a clear representation of the time played and kill-death ratio. This subset is useful for a quick overview of key metrics without overwhelming the viewer with unnecessary details.

Collectively, these EDA analyses help to provide a deeper understanding of the gameplay dynamics of my Overwatch 2 data, shedding light on heroes with exceptional win rates, the most-played heroes, and a focused display of key statistics for strategic decision-making.

Hypothesis 1: Positive Correlation

Null Hypothesis (H0): There is no significant positive correlation between the time played with a hero and their K/D ratio.
Alternative Hypothesis (H1): There is a significant positive correlation between the time played with a hero and their K/D ratio.
Explanation: Player named Kerem Atahan who spend more time playing a particular hero may demonstrate a higher K/D ratio, suggesting that increased experience and proficiency with the hero positively influence performance.

```
1 correlation, _ = pearsonr(data['time_played'], data['kda'])
2 print(f'Correlation: {correlation}')
```

The correlation coefficient ranges from -1 to 1, where 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 indicates no linear relationship.
The correlation coefficient (-0.06666946377236559) indicates a weak negative correlation between time played with a hero and their K/D ratio. However, it's essential to perform hypothesis testing to determine whether this correlation is statistically significant.

```
# Perform the Pearson correlation test
stat, p_value = pearsonr(data['time_played'], data['kda'])


alpha = 0.05

if p_value < alpha:
    print("Reject the null hypothesis: There is a significant positive correlation.")
else:
    print("Fail to reject the null hypothesis: There is no significant positive correlation.")

```

The provided code conducts a hypothesis test for the Pearson correlation between 'time_played' and 'kda' in the Overwatch 2 gameplay statistics. The null hypothesis assumes no significant positive correlation between these two variables.

The result "Fail to reject the null hypothesis: There is no significant positive correlation" suggests that, based on the provided data, there is not enough evidence to conclude that there is a significant positive correlation between the time played with a hero and their K/D ratio.

```
from sklearn.cluster import KMeans
df_w_predictions = df.copy()
model = KMeans(n_clusters=5, random_state=42)
df_w_predictions["cluster"] = model.fit_predict(df_wo_hero_names)
df_w_predictions[["heroes", "cluster"]]
```

Clustering analysis was performed using the KMeans algorithm from the scikit-learn library. The primary goal was to identify inherent patterns and groupings within the Overwatch 2 gameplay statistics.

```
# Visualization
import umap
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8,6))

standard_embedding = umap.UMAP(random_state=42).fit_transform(df_wo_hero_names)
scatter = plt.scatter(standard_embedding[:, 0], standard_embedding[:, 1], c=df_w_predictions.cluster.astype(int), s=40, cmap='Spectral');
for i, txt in enumerate(df_w_predictions['heroes']):
    plt.text(standard_embedding[i, 0], standard_embedding[i, 1]+0.05, txt, fontsize=8, ha='center', va='bottom')

# Add colorbar for better understanding of the clusters
plt.colorbar(scatter, label='Cluster')
plt.show()
```

To visualize the clustering results, Uniform Manifold Approximation and Projection (UMAP) was employed. UMAP is a dimensionality reduction technique that is particularly effective for visualizing high-dimensional data in a lower-dimensional space.

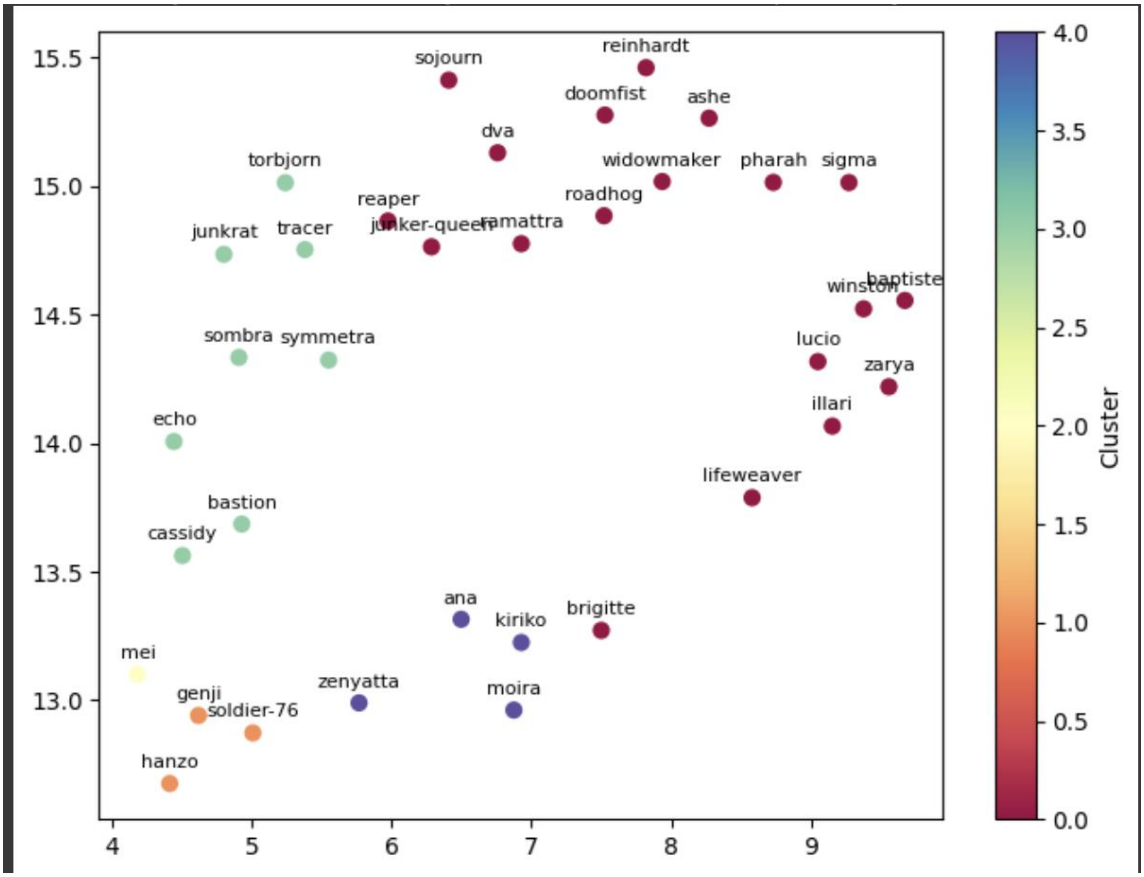
Findings:

	heroes	games_played
9	genji	74
10	hanzo	60
34	zenyatta	35
17	mei	30
26	soldier-76	29
5	cassidy	20
27	sombra	18
0	ana	17

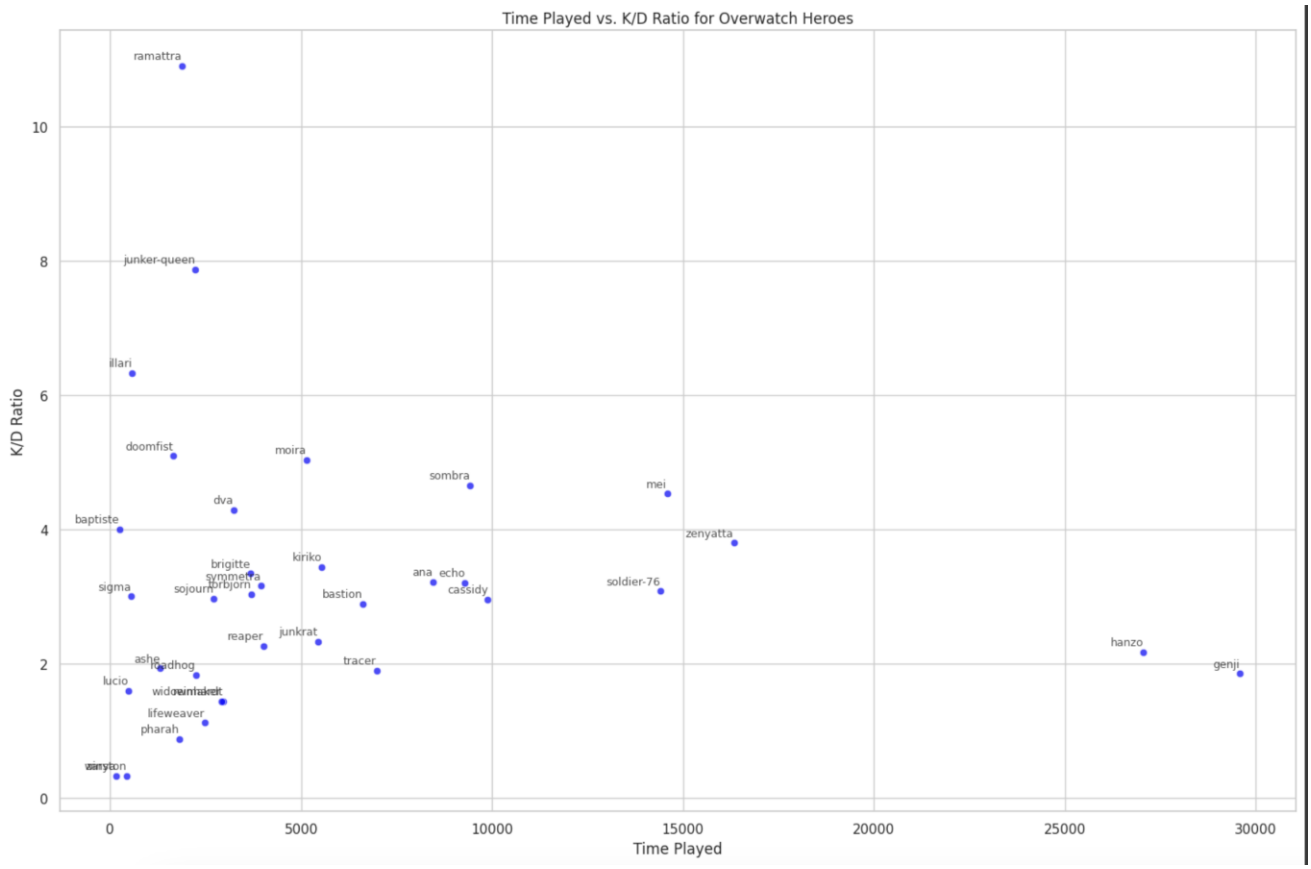
	heroes	kda
20	ramattra	10.90
12	junker-queen	7.87
11	illari	6.33
7	doomfist	5.09
18	moira	5.03
27	sombra	4.65
17	mei	4.53
6	dva	4.29
2	baptiste	4.00
34	zenyatta	3.80

	heroes	time_played	kda
0	ana	8444	3.22
1	ashe	1304	1.93
2	baptiste	239	4.00
3	bastion	6615	2.89
4	brigitte	3674	3.34
5	cassidy	9891	2.96
6	dva	3228	4.29
7	doomfist	1645	5.09
8	echo	9293	3.20
9	genji	29585	1.85
10	hanzo	27060	2.17
11	illari	575	6.33
12	junker-queen	2235	7.87
13	junkrat	5443	2.33
14	kiriko	5544	3.44
15	lifeweaver	2470	1.12
16	lucio	477	1.60
17	mei	14586	4.53
18	moira	5151	5.03
19	pharah	1809	0.88
20	ramattra	1890	10.90
21	reaper	4019	2.26
22	reinhardt	2968	1.44
23	roadhog	2236	1.83
24	sigma	549	3.00
25	sojourn	2718	2.97
26	soldier-76	14414	3.08
27	sombra	9427	4.65
28	symmetra	3945	3.16
29	torbjorn	3699	3.03
30	tracer	6976	1.90
31	widowmaker	2904	1.44
32	winston	426	0.33
33	zarya	155	0.33
34	zenyatta	16334	3.80

Heroes like "genji" and "hanzo" have high time played, indicating they are frequently chosen by me. However, some heroes like "zarya" and "winston" have lower time played, suggesting they are less commonly selected by me. Support heroes like "moira" and "zenyatta" demonstrate strong KDAs, showcasing their ability to contribute both offensively and defensively.



Based on the cluster information, the Orange Cluster (Genji, Hanzo, Soldier-76) comprises the heroes that I play most frequently, as they are clustered in the bottom left. The Purple Cluster includes heroes like Ana and Zenyatta, which I frequently played in the support role. The Green Cluster represents heroes I used to play more often, but nowadays, I don't play them as much. Finally, the Red Cluster consists of heroes I rarely play in Overwatch 2.



Even though I have a high K/D ratio with the hero named Ramattra, I observe that my playtime is relatively small. Therefore, when considering the average performance in terms of both playtime and K/D ratio, the heroes Mei, Soldier, and Zenyatta appear to be my best-performing choices.

Limitations and future work:

To make my Overwatch 2 data project even better, I can do a few things. First, I can try to get more types of data from different sources to get a fuller picture of the game. This might mean looking into other places that have information about heroes, maps, or game modes. Adding data over time could help me see how things change and improve my analysis. I could also look at more specific details, like a player's rank or how they perform in competitive games. In the future, I might explore using advanced tools like machine learning to predict things or find hidden patterns in the data. Keeping my dataset up-to-date will help me see the latest trends in gameplay.