

Street Fighter 6 Rank Distribution Explorer

RANK WEB SCRAPER AND VISUALIZER

BY: BINARYXX SUNE

Table of Contents

Background and Context

Project Details

Additional Project Details

Overview of Features

Process Walkthrough

- Web Scraping
- Visualizations

Reflection

- Problems Encountered and Addressed
- Potential Problems Avoided
- Improvements Not Implemented
- Future Improvements to Implement

Background and Context

STREET FIGHTER 6 OVERVIEW AND TERMINOLOGY

Street Fighter 6

Street Fighter 6 (SF6) is the latest iteration of the franchise

- Released on June 2nd, 2023, by CAPCOM
- Record high peak concurrent players for the fighting game genre

The video game is popular for its 1v1 battles

- Victory is determined when a player's health depletes to zero
 - When time runs out, whoever has the most health wins

Street Fighter 6 (cont.)

SF6 features a story mode and versus mode

- Online play consists of casual and ranked matches in versus mode

Ranked matches implement skill-based matchmaking and a rank system

- The rank system assigns titles to represent the player's skill level
 - Includes rank assignment mechanics to ensure fair matchmaking

Official SF6 Terminology

League Points (LP)

- Points determining the rank title assigned
- Awarded for winning and deducted for losing ranked matches

Calibrated

- Players who have completed their placement matches
 - Placement matches are the first 10 ranked matches for a given character
- Starting rank title assigned after the placement matches

Official SF6 Terminology (cont.)

League

- Title representing a skill rating group
- E.g., Bronze 4 is in the Bronze League

Rank

- Number representing a subgroup in each league
- Each league, except for Master League, has 5 ranks
- Within the same league, the 5th rank has the most LP
- E.g., Bronze 4 is the 4th Rank in the Bronze League

Code Terminology

Some terminology used in the code doesn't align with official SF6 terminology

Rank

- The full title which is made up of league and rank
- E.g., Bronze 4 is a rank

Rank Group

- All ranks within the same league
 - Synonymous to SF6's term of leagues
- E.g., Rookie 1, ..., Rookie 5 is in the Rookie Rank Group

Code Terminology (cont.)

Tier

- Refers to subgroups within a rank group
 - Synonymous to SF6's term of rank
- E.g., Rookie 1 is the 1st tier of the Rookie Rank Group

Tier Group

- Group with the same tier but differing rank groups
 - In official SF6 terms, same ranks but different leagues
- E.g., Tier Group 1 consists of Rookie 1, Iron 1, ..., Diamond 1
 - The Master league does not have a rank; included in Tier Group 1

Project Details

FUNCTION AND SCOPE

What Does It Do

Retrieves SF6's up-to-date player count for each rank and creates visualizations of the distribution

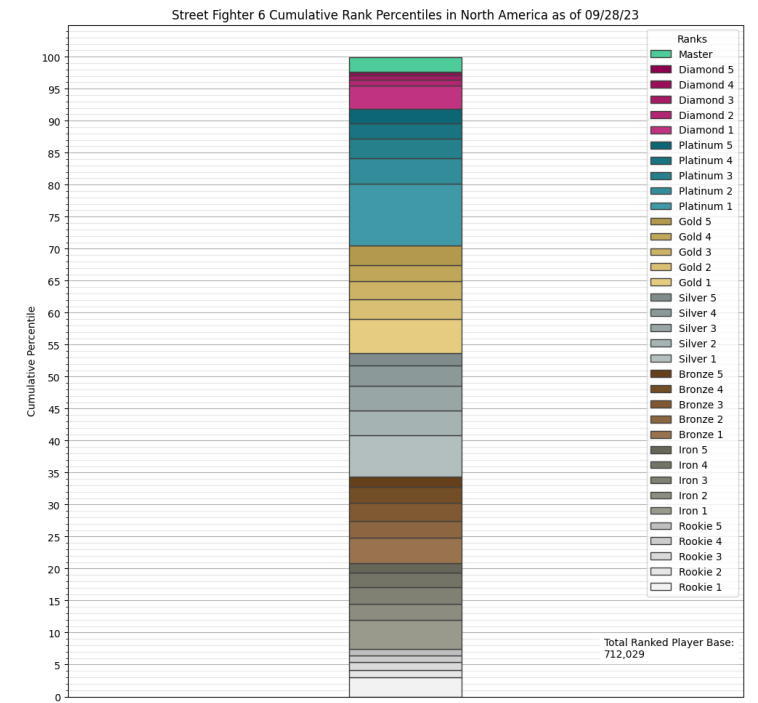
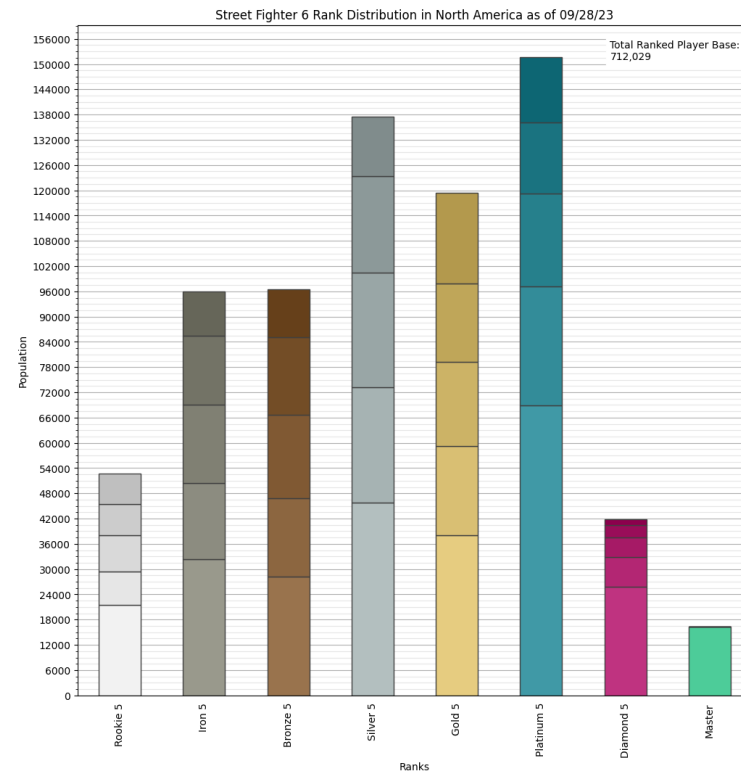
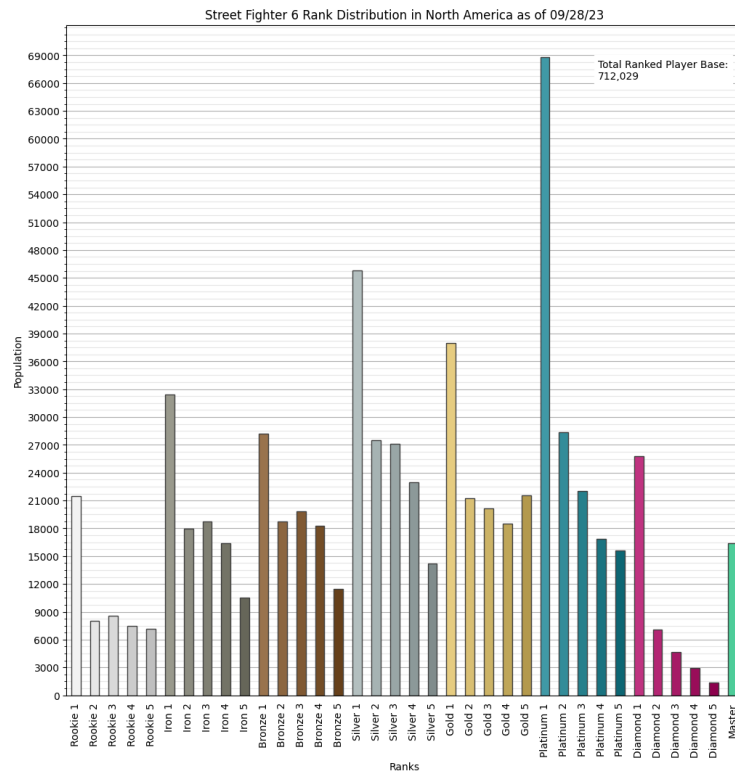
Accesses the SF6 leaderboard, filters by rank, records the player population

- Filters for players for the North America region

Provides 3 visualizations with their accompanying date frames:

- Bar chart of the distribution of players across ranks
- Stacked bar chart of the distribution of players across rank groups
- Cumulative bar chart of the ranks and their percentiles





Preview of Visualizations

Clearer view available in the Code Walkthrough section of the presentation

Limitations

Only users with calibrated characters selected for queue

- Filtering for both “League” and “Max LP Characters” are not allowed
 - Misses users with calibrated characters but has an uncalibrated character selected

Only users in the North America region

- Proximity is considered in matchmaking to minimize latency
 - Better relevance of trends for players queuing in North America

Inactive users are included in the dataset

- No built-in filters provided by the web page
- Workaround alternative is too resource demanding and slow

Additional Project Details

MOTIVATIONS TO START AND PROJECT IMPACT

Motivations

General public interest in prevailing trends in competitive video games

Outdated summaries on SF6's ranked player base

- Weeks old available summaries
- Information needed for the summaries is in the SF6 leaderboards

Application of data analysis and visualization skills

- Automate data collection through web scraping
- Summarize and present data through graphs

Benefits

PLAYER POINT OF VIEW:

Provide another basis for personal goal setting

- Compare player skill level in relation to the player base
- Identify oversaturated ranks to understand potentially wider ranges of skill level

CAPCOM POINT OF VIEW:

Monitor player distribution across ranks for effective matchmaking

- Identify unexpectedly high concentration of players in specific ranks

Improved in-game tournament entry restrictions for fairer brackets

Overview of Features

WEB SCRAPING AND VISUALIZATION FEATURES

A solid blue horizontal bar spanning the width of the slide at the bottom.

Web Scraping Features

Explicit waiting strategy

- Waits until an element from the expected page is displayed
 - Avoids checking too early or waiting too long

CAPCOM log in credentials in a separate text file

- Ensures private information is secure if the code is viewed

Addresses daily popup message, if applicable

- Occurs only for the first log in of the day

General Chart Features

Self adjusting Y tick intervals and title

- Reduce manual adjustments of formatting for each run

Graphs have horizontal grids for easy Y value reference

- Major and minor grids enabled with distinguishable shades

Differentiated bar colors for each rank

- Rank groups matches the corresponding rank color in-game
- Higher rank tiers have darker shades of its color

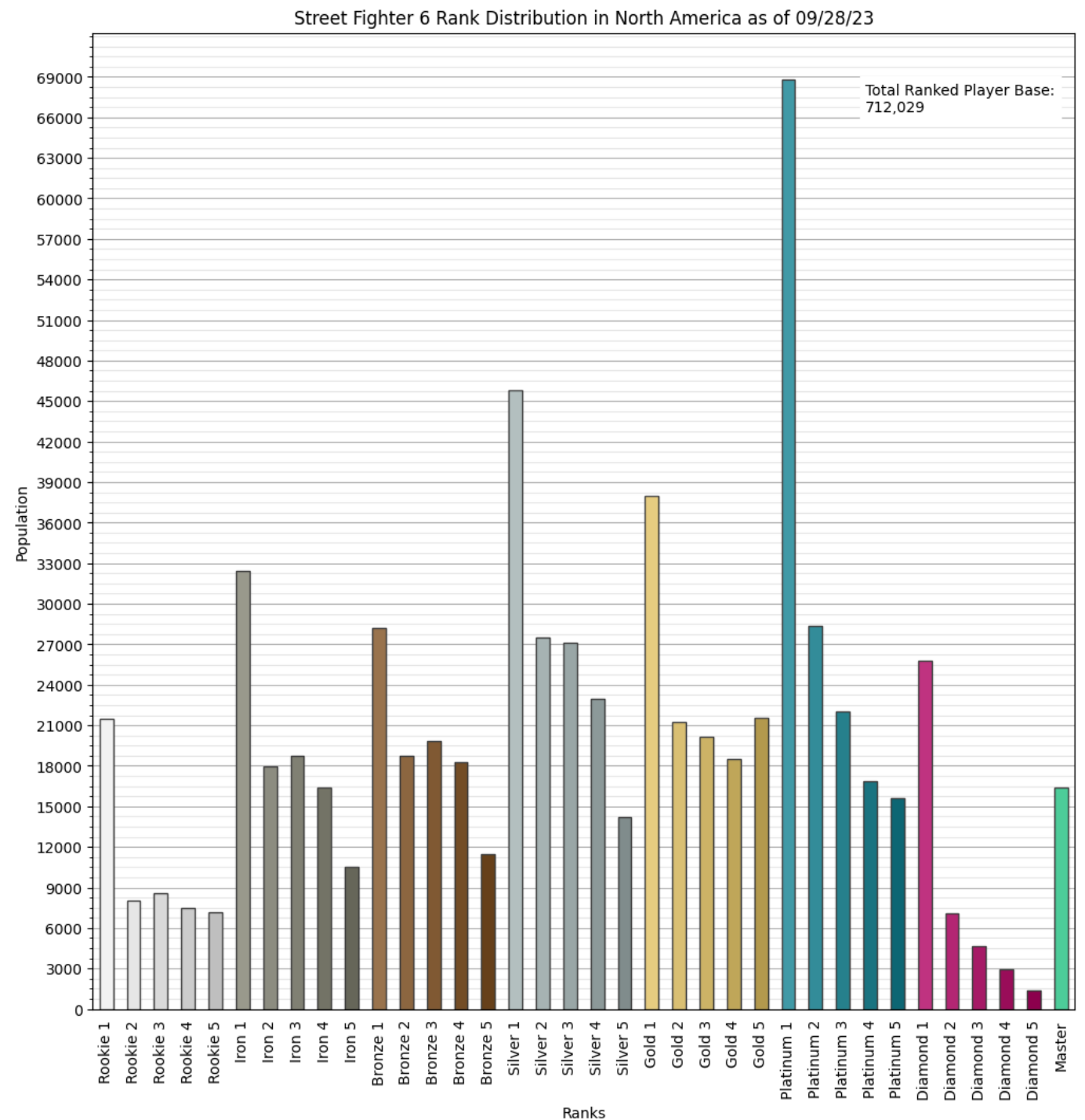
Total ranked player count textbox for context

Bar Chart Features

Each rank is an X tick

- Easily compare each rank population

Rank groups distinguishable by changes in color

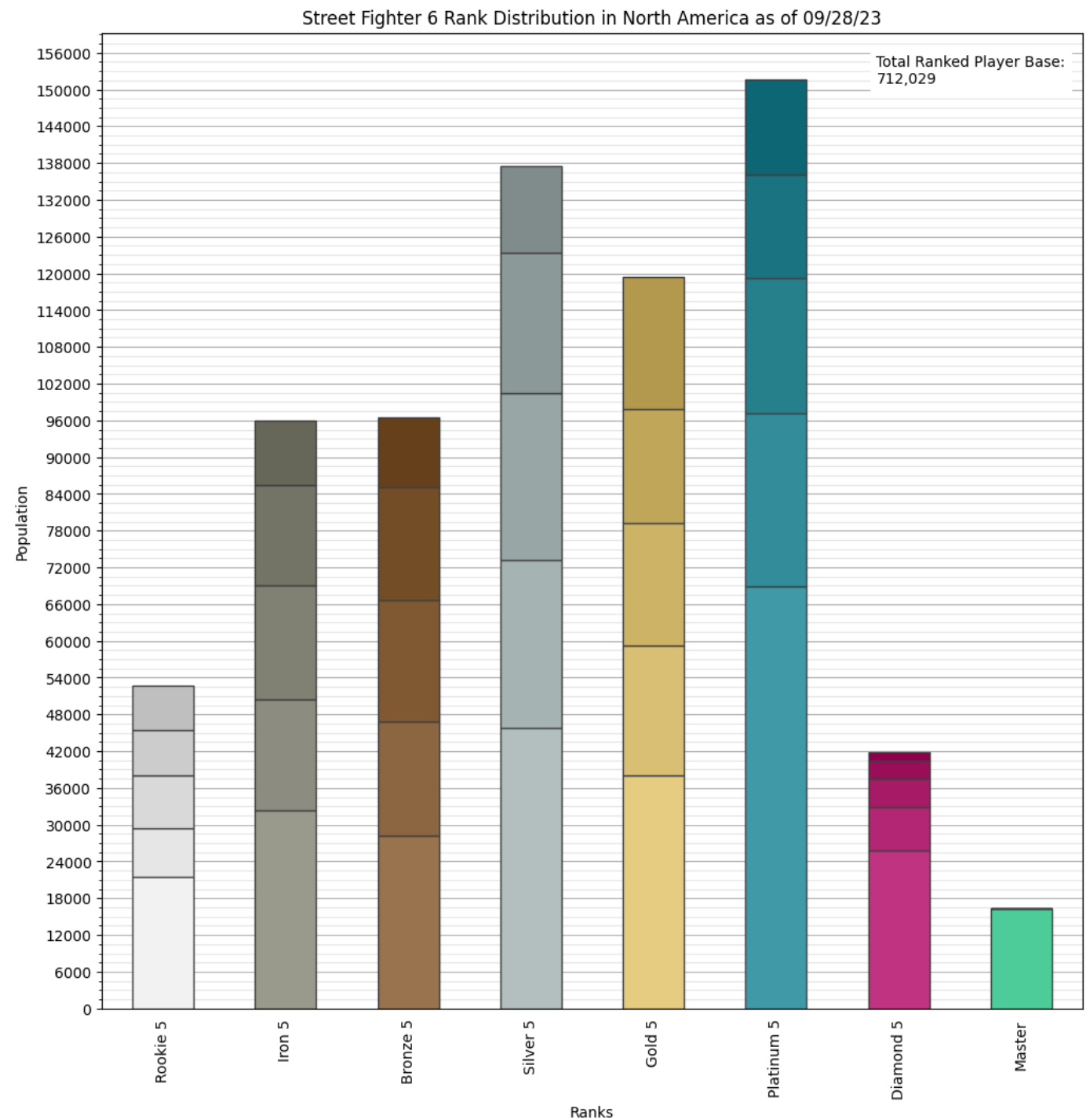


Stacked Bar Chart Features

Each rank group is an X tick

Provides a compact view of the rank distribution

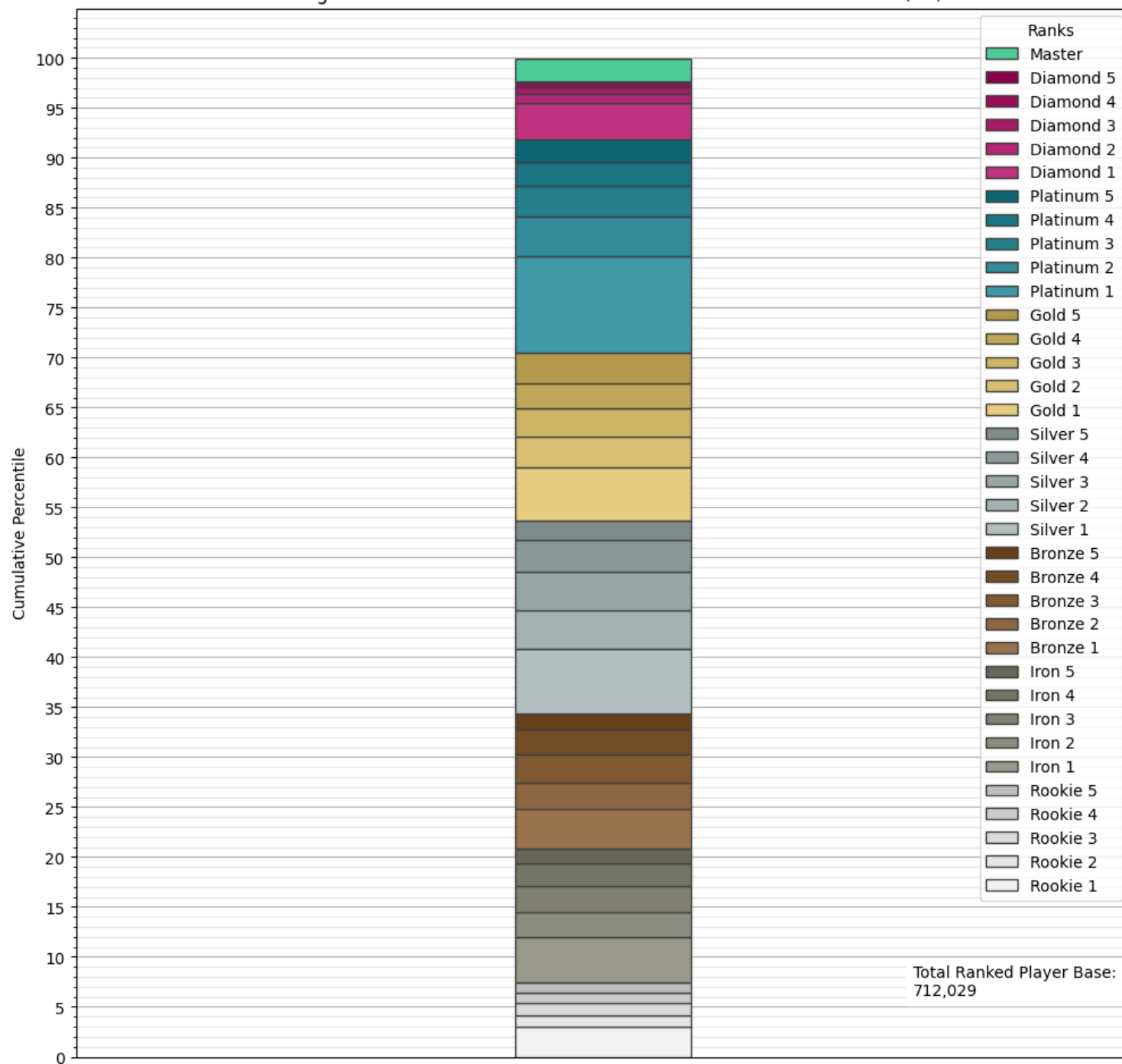
- Still informs how much each tier makes up a rank group
- Different shades allows for differentiation between rank tiers



Percentile Chart Features

Each bar represents a rank's
percentage of the population

- Also conveys what percentile each rank is
- Shows the percent of the population with a lower rank



Process Walkthrough

STEP BY STEP OVERVIEW OF THE CODE AT WORK

Language, Platform, and Libraries

Coded with Python in Jupyter Notebook in VS Code

Python libraries used:

- Selenium
- Pandas
- NumPy
- Matplotlib
- DateTime

PyCharm Community for suggestions on code readability

GitHub Repository

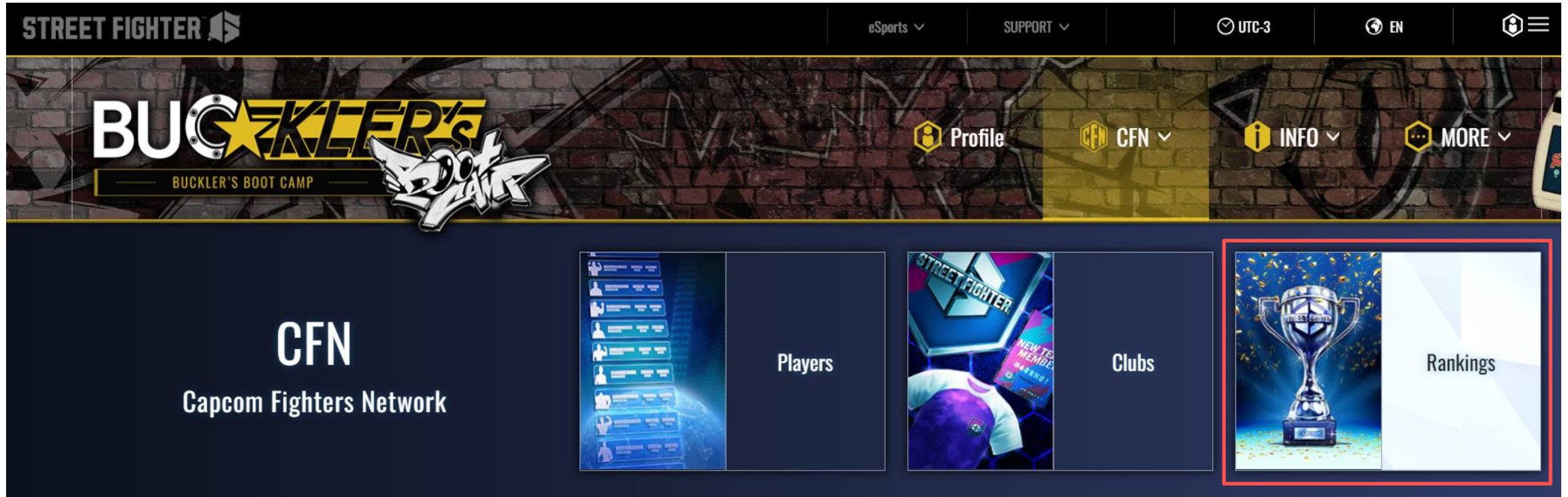
For a complete view of the code:

- Scan the QR code
- [Click this link](#)



Web Scraping

COLLECTING THE RANK POPULATIONS



Accessing the SF6 Leaderboards

1. Initialize the desired web browser
2. Access the URL Link for the website's "Rankings" page

Please enter your country/region and date of birth.

Country/Region	<input type="text" value="United States"/>
Date of Birth	<input type="text" value="Jan"/> <input type="text" value="1"/> <input type="text" value="2000"/>

Next >
BACK

Please enter your country/region and date of birth.

Country/Region	<input type="text" value="Select Country/Region"/>
Date of Birth	<input type="text" value="Month:"/> <input type="text" value="Day:"/> <input type="text" value="Year:"/>

Next >
BACK


Completing Age Check Form

1. Select the dropdown element
2. Input the desired select value
3. Repeat steps 1 and 2 for all dropdowns
4. Click the “Next” button



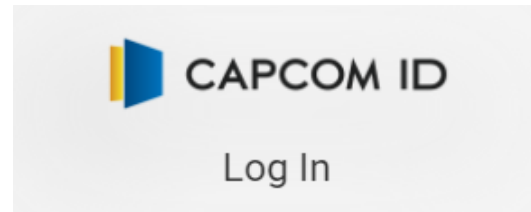
Log In Sign Up

 yours@example.com


 your password

Don't remember your password?

LOG IN >



Log In Sign Up

 username@mailserver.domain



Don't remember your password?

LOG IN >

Completing the Log In Page Form

1. Select the email textbox
2. Input the email credentials provided in the separate text file
3. Repeat steps 1 and 2 for the password
4. Click the "LOG IN" button



Close the Daily Message Popup if It Appears

1. Select the "X" button
2. Click button

| League Points

Group

League All

Character All

Region All

Crossplay All

Filters

| League Points

Group

League All

Character All

Region North America

Crossplay All

Filters

Leaderboard Filters: Region

1. Select the “Region” filter dropdown
2. Input the value for “North America”
 - Input for region persists through searches

League Points

Group

League: All

Character: All

Region: North America

Crossplay: All

Filters

League Points

Group

League: Rookie 1

Character: All

Region: North America

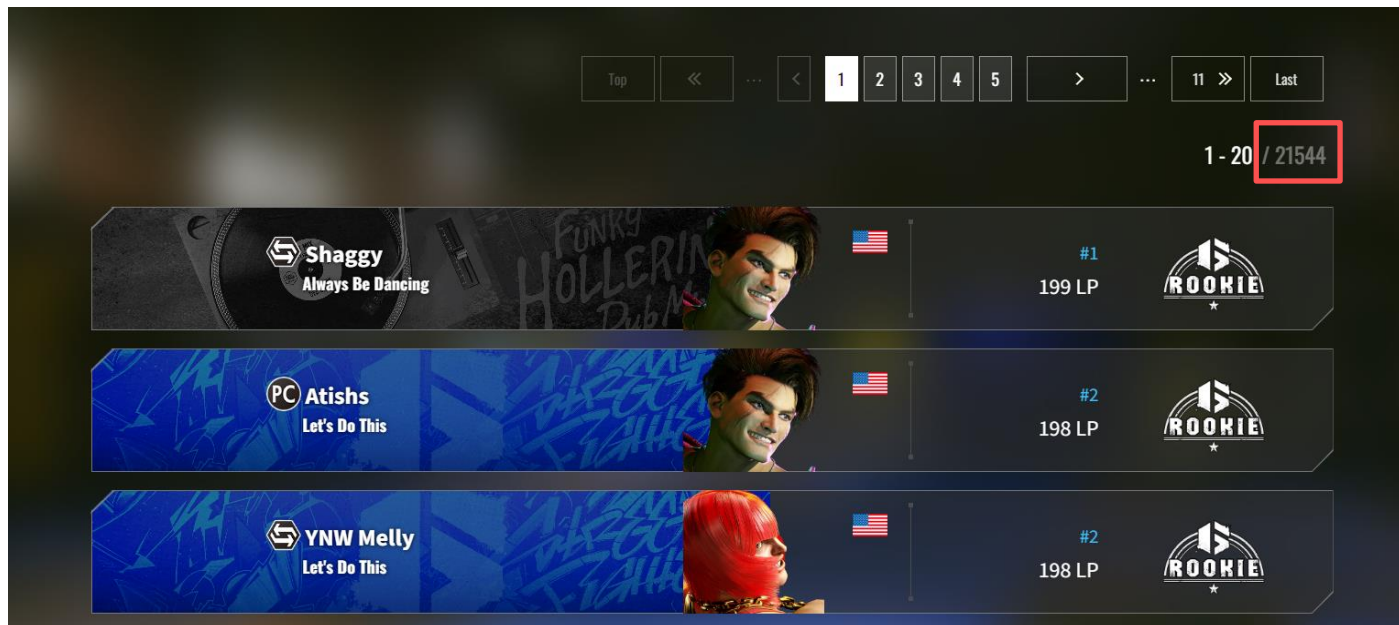
Crossplay: All

Filters

Leaderboard Filters: Rank

1. Select the “League” filter dropdown
2. Input the value for the desired rank
3. Click the “Filters” button

Rank Count Collection



4. Record the total player count for the specified rank
 - Store as a dictionary with ranks as keys and population as values

Repeat steps 1 to 4 for each rank

Web Scrapping Process in Action



Video Link: <https://youtu.be/6Vo-rcMH9p0>

Visualizations

CREATING DATA FRAMES AND CHARTS

```
rank_df = pd.DataFrame([rank_population.keys(), rank_population.values()])
```

1

```
rank_df = rank_df.transpose()
```

```
rank_df = rank_df.rename(columns={0: "Rank", 1: "Population"})
```

```
rank_df = rank_df.astype({"Population": "int32"})
```

```
adj_rank_df = rank_df.drop([0]) # Remove "All"
```

1a

```
adj_rank_df = adj_rank_df.reset_index(drop=True)
```

```
total_player_base = int(rank_population["All"])
```

2

```
total_player_base_text = f"Total Ranked Player Base:\n{total_player_base:,}"
```

```
display(adj_rank_df)
```

3

```
print(total_player_base_text)
```

Creating the Rank Data Frame

1. Create a data frame from the dictionary of ranks and populations
 - a. Remove the "All" item since it is not a rank
2. Create a message to display the total ranked player count
3. Show the data frame and the total ranked player count

Rank Data Frame

	Rank	Population			
			18	Silver 4	23356
0	Rookie 1	21553	19	Silver 5	14388
1	Rookie 2	8100	20	Gold 1	38533
2	Rookie 3	8586	21	Gold 2	21538
3	Rookie 4	7523	22	Gold 3	20444
4	Rookie 5	7172	23	Gold 4	18873
5	Iron 1	32629	24	Gold 5	22095
6	Iron 2	18045	25	Platinum 1	70066
7	Iron 3	18800	26	Platinum 2	28956
8	Iron 4	16572	27	Platinum 3	22482
9	Iron 5	10593	28	Platinum 4	17248
10	Bronze 1	28379	29	Platinum 5	15961
11	Bronze 2	18893	30	Diamond 1	26650
12	Bronze 3	19966	31	Diamond 2	7304
13	Bronze 4	18456	32	Diamond 3	4762
14	Bronze 5	11614	33	Diamond 4	3072
15	Silver 1	46380	34	Diamond 5	1443
16	Silver 2	27745	35	Master	17045
17	Silver 3	27457	Total Ranked Player Base:		
18	Silver 4	23356	722,672		

```
# Create a dictionary with the rank's RGB values; used as a base for the color palette
```

1

```
rank_rgb = {"Rookie": (0.95, 0.95, 0.95),  
            "Iron": (0.6, 0.6, 0.55),  
            "Bronze": (0.6, 0.45, 0.3),  
            "Silver": (0.7, 0.75, 0.75),  
            "Gold": (0.9, 0.8, 0.5),  
            "Platinum": (0.25, 0.6, 0.65),  
            "Diamond": (0.75, 0.2, 0.5),  
            "Master": (0.3, 0.8, 0.6)}  
  
rank_rgb_array = np.asarray(list(rank_rgb.values()))
```

```
rank_color_palette = []  
for rank in range(7):  
    for tier in range(5):  
        rank_tier_darkness = tier/20  
        new_rgb = rank_rgb_array[rank] - rank_tier_darkness # Make each tier a darker shade of the hue  
        rank_color_palette.append(new_rgb)  
  
    new_rgb = []  
  
rank_color_palette.append(rank_rgb["Master"])
```

2

```
# Calculate the max y value and use it as a basis to get the right intervals for y ticks of the graph  
rounded_max_total_player_count = round(adj_rank_df["Population"].max(), -3)  
number_of_y_tick_intervals = 25  
y_tick_intervals = int(round(rounded_max_total_player_count / number_of_y_tick_intervals, -3))  
upper_y_tick = rounded_max_total_player_count + y_tick_intervals
```

3

Set Up for the Rank Bar Chart

1. Create a dictionary for the RGB values of each rank group
2. Lower RGB values based on tier to darken the color of their bar
3. Calculate the Y ticks for the graph

```
fig, ax = plt.subplots()
```

1

```
adj_rank_df.plot(kind="bar",  
                 x="Rank",  
                 y="Population",  
                 yticks=range(0, upper_y_tick, y_tick_intervals),  
                 ax=ax,  
                 title="Street Fighter 6 Rank Distribution in North America as of " + (date.today()).strftime("%m/%d/%y"),  
                 color=rank_color_palette,  
                 edgecolor=(0.25, 0.25, 0.25),  
                 grid=True,  
                 legend=False)
```

2

```
ax.set_axisbelow(True)  
ax.grid(axis="x", visible=False)  
ax.grid(axis="y", which="minor", color=(0.9, 0.9, 0.9))  
ax.minorticks_on()  
  
fig.set_size_inches(12, 12)  
  
plt.figtext(0.735, 0.818, total_player_base_text, {"backgroundcolor":"white"})  
  
plt.tick_params(axis="x", which="both", bottom=False)  
  
plt.xlabel("Ranks")  
plt.ylabel("Population")  
  
plt.show()
```

Creating the Rank Bar Chart

1. Create a figure and axes object
2. Plot the bar chart

```
fig, ax = plt.subplots()

adj_rank_df.plot(kind="bar",
                  x="Rank",
                  y="Population",
                  yticks=range(0, upper_y_tick, y_tick_intervals),
                  ax=ax,
                  title="Street Fighter 6 Rank Distribution in North America as of " + (date.today()).strftime("%m/%d/%y"),
                  color=rank_color_palette,
                  edgecolor=(0.25, 0.25, 0.25),
                  grid=True,
                  legend=False)
```

```
ax.set_axisbelow(True)
ax.grid(axis="x", visible=False)
ax.grid(axis="y", which="minor", color=(0.9, 0.9, 0.9))
ax.minorticks_on()
```

2a

```
fig.set_size_inches(12, 12)
```

2b

```
plt.figtext(0.735, 0.818, total_player_base_text, {"backgroundcolor":"white"})
```

2c

```
plt.tick_params(axis="x", which="both", bottom=False)
```

2d

```
plt.xlabel("Ranks")
plt.ylabel("Population")
```

2e

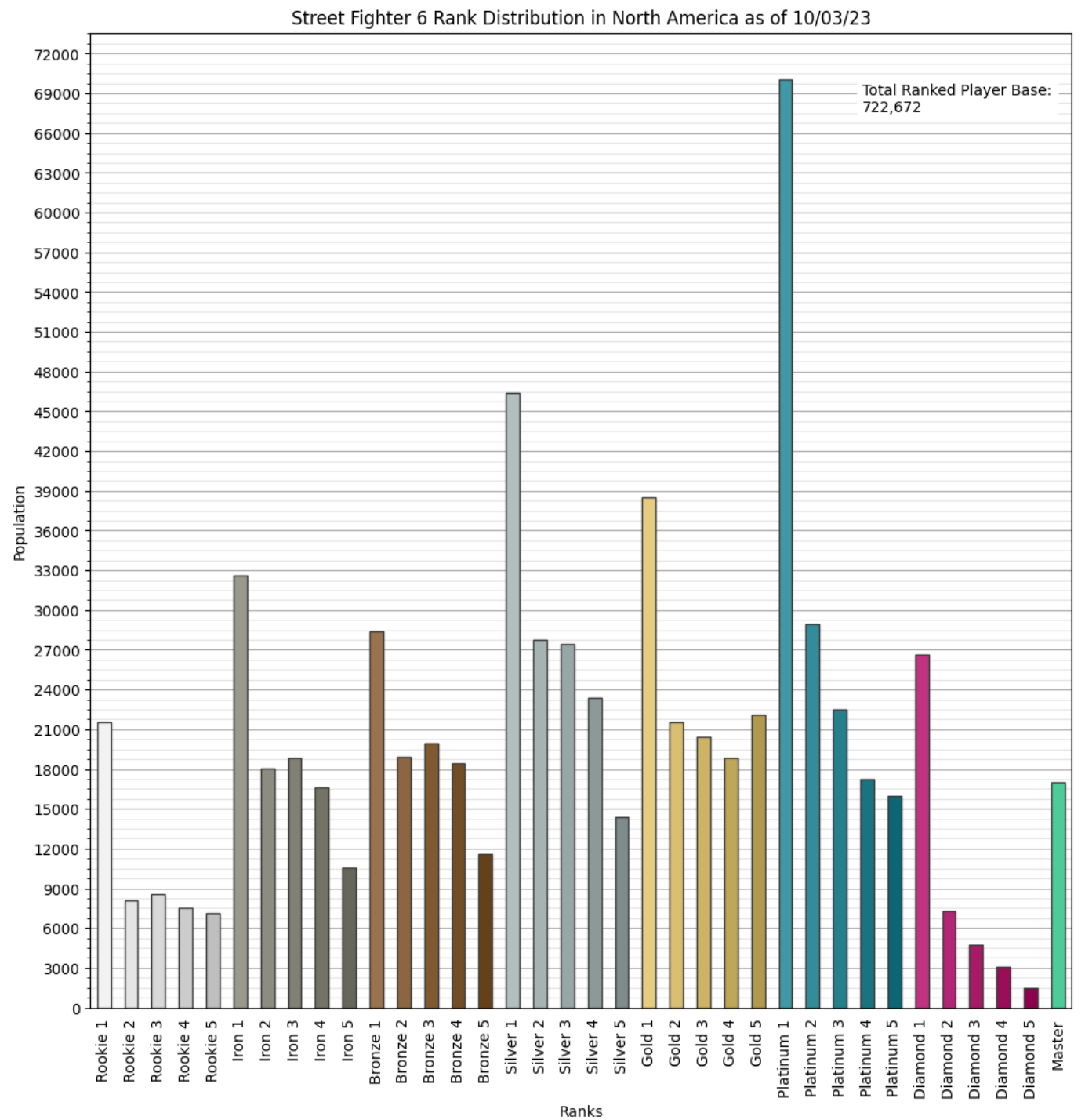
```
plt.show()
```

2f

Creating the Rank Bar Chart (cont.)

- Create major and minor grids in the background
- Set the figure size
- Add a textbox with the total ranked player count
- Remove the X ticks
- Add the X and Y labels
- Display the bar chart

Rank Bar Chart



```
# Create a new dataframe with tier groups of ranks e.g. Rookie 1, Bronze 1, ..., Diamond 1, etc.
```

1

```
def group_ranks(tier):  
    """  
    Group ranks that end with a specified number together (Rookie 1, Iron 1, Bronze 1...)   
    Master rank has the population values for the 1st tiered groups; 0 for the rest  
    """  
    tier_df = adj_rank_df[(adj_rank_df["Rank"].str[-1] == str(tier))]  
  
    if tier == 1:  
        tier_df.loc[np.inf] = adj_rank_df[(adj_rank_df["Rank"] == "Master")].values.tolist()[0]  
    else:  
        tier_df.loc[np.inf] = ["Master", 0] # Add Master rank to maintain a uniform array shape; No tiers in Master rank  
  
    tier_df = tier_df.reset_index(drop=True)  
  
    return tier_df
```

```
rank_tiers = []  
for i in range(1, 6):  
    rank_tiers.append(group_ranks(i))
```

2

```
for df in rank_tiers:  
    display(df)  
    print(f"Total Ranked Player Base:\n{total_player_base:,}")
```

3

Creating the Rank Group Data Frame

1. Define a function to create a data frame for a tier groups
2. Apply the function 5 times for the 5 tier groups
3. Show the data frame and the total ranked player count

Rank Group Data Frame

	Rank	Population
0	Rookie 1	21553
1	Iron 1	32629
2	Bronze 1	28379
3	Silver 1	46380
4	Gold 1	38533
5	Platinum 1	70066
6	Diamond 1	26650
7	Master	17045

Total Ranked Player Base:
722,672

	Rank	Population
0	Rookie 2	8100
1	Iron 2	18045
2	Bronze 2	18893
3	Silver 2	27745
4	Gold 2	21538
5	Platinum 2	28956
6	Diamond 2	7304
7	Master	0

Total Ranked Player Base:
722,672

	Rank	Population
0	Rookie 4	7523
1	Iron 4	16572
2	Bronze 4	18456
3	Silver 4	23356
4	Gold 4	18873
5	Platinum 4	17248
6	Diamond 4	3072
7	Master	0

Total Ranked Player Base:
722,672

	Rank	Population
0	Rookie 3	8586
1	Iron 3	18800
2	Bronze 3	19966
3	Silver 3	27457
4	Gold 3	20444
5	Platinum 3	22482
6	Diamond 3	4762
7	Master	0

Total Ranked Player Base:
722,672

	Rank	Population
0	Rookie 5	7172
1	Iron 5	10593
2	Bronze 5	11614
3	Silver 5	14388
4	Gold 5	22095
5	Platinum 5	15961
6	Diamond 5	1443
7	Master	0

Total Ranked Player Base:
722,672

```
fig, ax = plt.subplots()
```

1

```
stacked_bar_starting_bottom_height = np.zeros(8)
```

2

```
# Create new RGB values; the order is different from previous graph's color palette
```

```
stacked_bar_palettes = [[] for _ in range(5)]
```

```
for tier in range(5):
```

```
    for rank in range(8):
```

```
        rank_tier_darkness = tier/20 # Change the value of the rgb to make it darker for each rank tier in the same rank group
```

```
        new_rgb = rank_rgb_array[rank] - rank_tier_darkness
```

```
        stacked_bar_palettes[tier].append(new_rgb)
```

```
    new_rgb = [] # reset rgb values for every rank
```

3

Set Up for the Rank Group Stacked Bar Chart

1. Create a figure and axes object
2. Create an array of zeroes
 - Used for the bars' bottoms' heights for the first tier group
3. Create the color palette for the stacked bar chart
 - The previous color palette order is different

1

```
# Plot the bar graph for each group tier, plotting the following tier group's bars where the bars' heights end (y value)
```

```
for idx, tier_group in enumerate(rank_tiers):
```

1a

```
    tier_group.plot(kind="bar",
                    ax=ax,
                    x="Rank",
                    y="Population",
                    bottom=stacked_bar_starting_bottom_height,
                    title="Street Fighter 6 Rank Distribution in North America as of " + (date.today()).strftime("%m/%d/%y"),
                    color=stacked_bar_palettes[idx],
                    edgecolor=(0.25, 0.25, 0.25),
                    legend=False,
                    grid=True)
```

```
    stacked_bar_starting_bottom_height += rank_tiers[idx]["Population"].values # update the starting height so that the next tier starts where the previous tier ends
```

1b

```
stacked_max_y_values_array = stacked_bar_starting_bottom_height # At the end of the loop, the last tier group is added to the bottom heights, creating an array of the max y values
stacked_max_y_value = int(np.max(stacked_max_y_values_array))
stacked_y_tick_interval_number = 25
stacked_y_tick_intervals = int(round(stacked_max_y_value / stacked_y_tick_interval_number, -3))
stacked_max_y_tick = stacked_max_y_value + stacked_y_tick_intervals
ax.set_yticks(range(0, stacked_max_y_tick, stacked_y_tick_intervals))
```

Creating the Rank Group Stacked Bar Chart

1. Plot the chart
 - a. Plot each rank group's bars
 - b. Adjust the bottom height value to stack the bars

```
# Plot the bar graph for each group tier, plotting the following tier group's bars where the bars' heights end (y value)
for idx, tier_group in enumerate(rank_tiers):
    tier_group.plot(kind="bar",
                    ax=ax,
                    x="Rank",
                    y="Population",
                    bottom=stacked_bar_starting_bottom_height,
                    title="Street Fighter 6 Rank Distribution in North America as of " + (date.today()).strftime("%m/%d/%y"),
                    color=stacked_bar_palettes[idx],
                    edgecolor=(0.25, 0.25, 0.25),
                    legend=False,
                    grid=True)

    stacked_bar_starting_bottom_height += rank_tiers[idx]["Population"].values # update the starting height so that the next tier starts where the previous tier ends

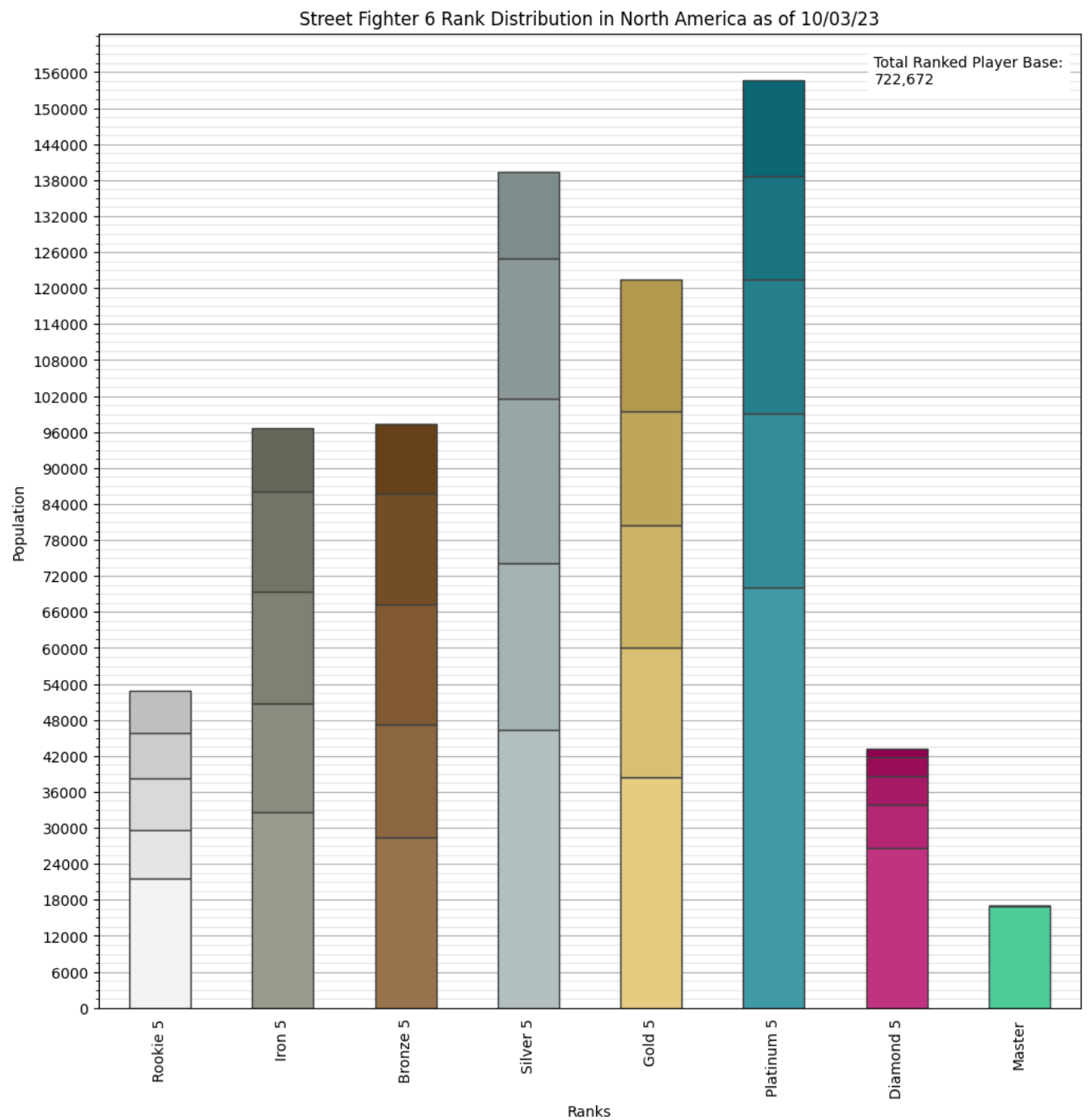
stacked_max_y_values_array = stacked_bar_starting_bottom_height # At the end of the loop, the last tier group is added to the bottom heights, creating an array of the max y values
stacked_max_y_value = int(np.max(stacked_max_y_values_array))
stacked_y_tick_interval_number = 25
stacked_y_tick_intervals = int(round(stacked_max_y_value / stacked_y_tick_interval_number, -3))
stacked_max_y_tick = stacked_max_y_value + stacked_y_tick_intervals
ax.set_yticks(range(0, stacked_max_y_tick, stacked_y_tick_intervals))
```

2

Creating the Rank Group Stacked Bar Chart (cont.)

2. Calculate the Y tick intervals
 - Omitted 10 lines of code in the screenshot
 - Repeated code included in the first chart such as background grids

Ranked Group Stacked Bar Chart



```
rank_population_sum = sum(list(adj_rank_df["Population"].values))
```

```
rank_list = list(adj_rank_df["Rank"].values)
```

1

```
def percentile_calculator(population):
```

2a

2

```
    percentile = round((population / rank_population_sum)*100, 2)
```

```
    return percentile
```

```
rank_percentile_list = adj_rank_df["Population"].apply(percentile_calculator)
```

2b

```
rank_cumulative_percentile_list = round(rank_percentile_list.cumsum(),1)
```

2c

```
rank_percentile_df = pd.DataFrame([rank_list, rank_cumulative_percentile_list]).transpose()
```

```
rank_percentile_df = rank_percentile_df.rename({0:"Rank", 1:"Cumulative Percentile"}, axis=1)
```

```
display(rank_percentile_df)
```

```
print(total_player_base_text)
```

Cumulative Rank Percentile Data Frame

1. Create a list of ranks
2. Calculate the cumulative percentile of each rank
 - a. Define a function to calculate the percentile of a rank
 - b. Apply the function to each rank
 - c. Calculate the cumulative percentile of each rank


```
rank_population_sum = sum(list(adj_rank_df["Population"].values))
rank_list = list(adj_rank_df["Rank"].values)
```

```
def percentile_calculator(population):
    percentile = round((population / rank_population_sum)*100, 2)
    return percentile
```

```
rank_percentile_list = adj_rank_df["Population"].apply(percentile_calculator)
rank_cumulative_percentile_list = round(rank_percentile_list.cumsum(),1)
```

```
rank_percentile_df = pd.DataFrame([rank_list, rank_cumulative_percentile_list]).transpose()
rank_percentile_df = rank_percentile_df.rename({0:"Rank", 1:"Cumulative Percentile"}, axis=1)
```

```
display(rank_percentile_df)
print(total_player_base_text)
```

3

4

Cumulative Rank Percentile Data Frame (cont.)

3. Create a data frame of the ranks and their cumulative sum
4. Display data frame and total player base

Cumulative Rank Percentile Data Frame

Rank		Cumulative Percentile			
0	Rookie 1	3.0	18	Silver 4	51.4
1	Rookie 2	4.1	19	Silver 5	53.4
2	Rookie 3	5.3	20	Gold 1	58.8
3	Rookie 4	6.3	21	Gold 2	61.7
4	Rookie 5	7.3	22	Gold 3	64.6
5	Iron 1	11.8	23	Gold 4	67.2
6	Iron 2	14.3	24	Gold 5	70.2
7	Iron 3	16.9	25	Platinum 1	80.0
8	Iron 4	19.2	26	Platinum 2	84.0
9	Iron 5	20.7	27	Platinum 3	87.1
10	Bronze 1	24.6	28	Platinum 4	89.5
11	Bronze 2	27.2	29	Platinum 5	91.7
12	Bronze 3	30.0	30	Diamond 1	95.4
13	Bronze 4	32.6	31	Diamond 2	96.4
14	Bronze 5	34.2	32	Diamond 3	97.0
15	Silver 1	40.6	33	Diamond 4	97.5
16	Silver 2	44.4	34	Diamond 5	97.7
17	Silver 3	48.2	35	Master	100.0
18	Silver 4	51.4	Total Ranked Player Base: 722,672		

```
fig, ax = plt.subplots()
```

1

```
percentile_max = 101 # Over 100 to have the tick label for 100 Y value
```

2

```
percentile_intervals = 5
```

```
percentile_starting_bottom_height = 0
```

```
for i in rank_percentile_df.index:
```

3a

3

```
    percentile_df = pd.DataFrame([rank_list[i], rank_percentile_list[i]]).transpose()
```

```
    percentile_df.plot(kind="bar",
```

```
        ax=ax,
```

```
        yticks=range(0, percentile_max, percentile_intervals),
```

```
        width=0.1,
```

```
        bottom=percentile_starting_bottom_height,
```

```
        title="Street Fighter 6 Cumulative Rank Percentiles in North America as of " + (date.today()).strftime("%m/%d/%y"),
```

```
        color=rank_color_palette[i],
```

```
        edgecolor=(0.25, 0.25, 0.25),
```

```
        grid=True)
```

```
    percentile_starting_bottom_height += rank_percentile_list[i] # Update the starting bottom height
```

3b

```
handles, _ = ax.get_legend_handles_labels()
```

```
labels = rank_list
```

```
ax.legend(handles[::-1], labels[::-1], title="Ranks")
```

Creating the Cumulative Rank Percentile Chart

1. Create a figure and axes object
2. Initialize variables with the graph's parameters
3. Plot the chart
 - a. Plot each group's bar using a loop
 - b. Adjust the starting bottom height value to stack the bars

```

fig, ax = plt.subplots()
percentile_max = 101 # Over 100 to have the tick label for 100 Y value
percentile_intervals = 5
percentile_starting_bottom_height = 0

for i in rank_percentile_df.index:
    percentile_df = pd.DataFrame([rank_list[i], rank_percentile_list[i]]).transpose()
    percentile_df.plot(kind="bar",
                        ax=ax,
                        yticks=range(0, percentile_max, percentile_intervals),
                        width=0.1,
                        bottom=percentile_starting_bottom_height,
                        title="Street Fighter 6 Cumulative Rank Percentiles in North America as of " + (date.today()).strftime("%m/%d/%y"),
                        color=rank_color_palette[i],
                        edgecolor=(0.25, 0.25, 0.25),
                        grid=True)
    percentile_starting_bottom_height += rank_percentile_list[i] # Update the starting bottom height

handles, _ = ax.get_legend_handles_labels()
labels = rank_list
ax.legend(handles[::-1], labels[::-1], title="Ranks")

```

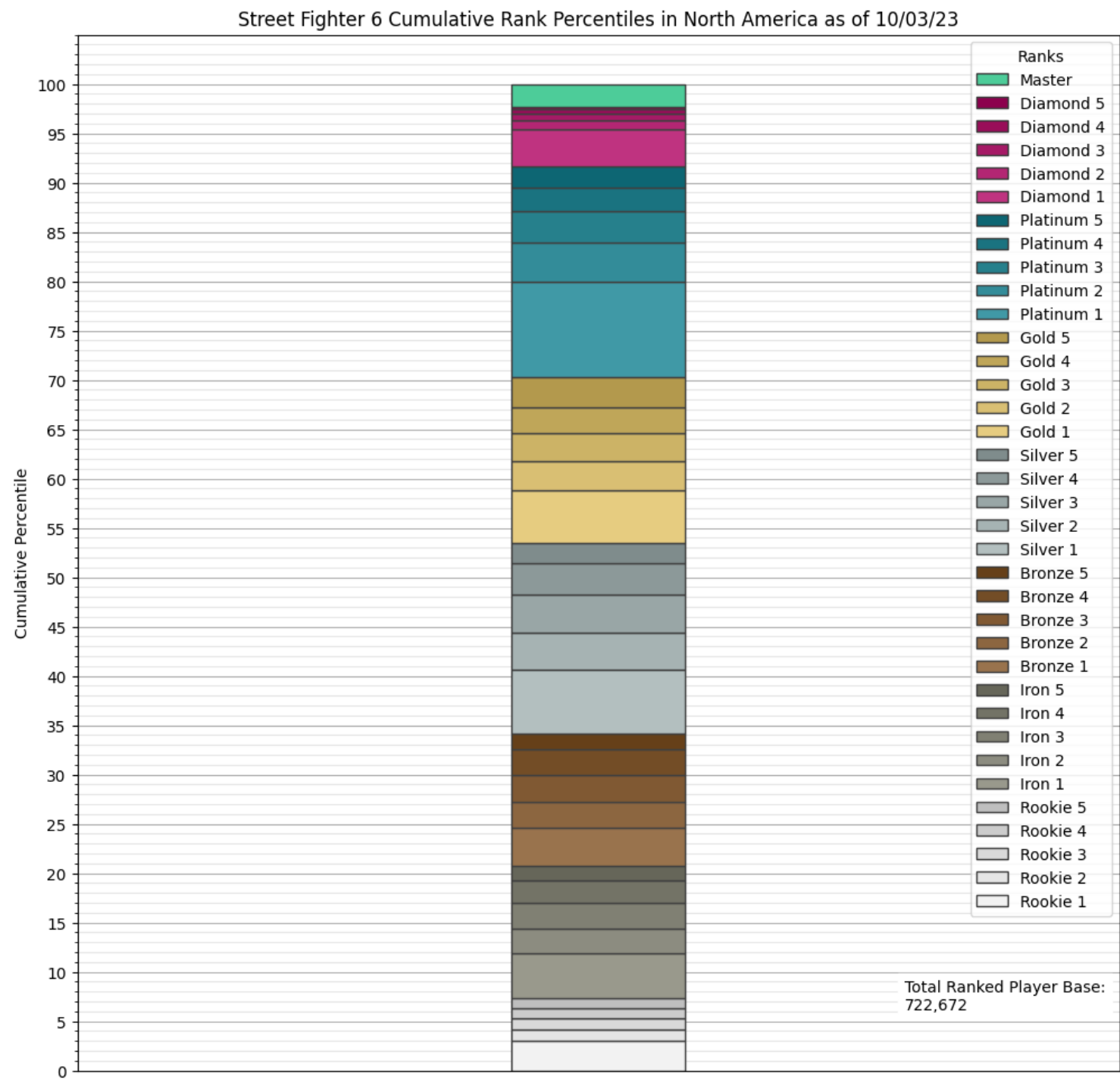
4

4a

Creating the Cumulative Rank Percentile Chart (cont.)

4. Change the legend to match the order as the graph
 - a. Reverse the order of the handles and labels
- Omitted 9 lines of code
 - Repeated code included in the first chart such as background grids

Cumulative Rank Percentile Chart



Reflection

LESSONS LEARNED AND IMPROVEMENTS TO IMPLEMENT

Problems Encountered and Addressed

CHALLENGES RESOLVED AND SKILLS EXPANDED



Used the Wrong Tools for the Job

Originally planned on using BeautifulSoup to web scrape

BeautifulSoup was not suitable for the tasks required

- Web scrapers are sent to a location and age check
- POST requests from BeautifulSoup could not complete the form

Switched to Selenium for the web scraping task

- Provided a simple solution to completing the forms
- Able to learn an alternate web scraping library for greater versatility

Required New Skills to Complete the Task

The basics of Selenium to web scrape such as:

- Navigating web pages and selecting elements
- Implementing an effective waiting strategy
- Acting on an element and requesting element information

Deeper Matplotlib knowledge for improved graphs such as:

- Plotting stacked bar charts
- Adjusting parameters of objects such as legends and grids
- Bridging the relationship between the Pandas plot function and Matplotlib documentation

Lacked Clear Direction

The absence of a finalized plan led to otherwise preventable issues

Lack of naming conventions led to unclear variable references

- A variable's purpose could not be easily inferred from its name

Frequently referenced values and processes were not optimized

- Functions to address repeating code patterns
- More appropriate containers to easily store and access values
 - Using dictionaries over lists when necessary

Potential Problems Avoided

CONDITIONS WHICH ALLOWED FOR AN EASIER EXPERIENCE

Cumbersome Learning Process

Straightforward transition from BeautifulSoup and Selenium

- Selenium had quality documentation and a “Getting Started” guide
- Transferrable skills in HTML and BeautifulSoup facilitated learning

Experience with NumPy and Pandas reduced potential errors and inefficiencies

- Also lessened the workload for learning

Unknown Issues

Review and revision process with different points of focus

- Revision for code efficiency revealed redundant or repeating code and processes
- Reference to PEP8 style guide for better code readability

Posted visuals in Reddit for additional review

- Inquiries about the project details were clarified in the notebook or presentation
- Identified mistakes and requests for additional information addressed

Improvements Not Implemented

MINOR ISSUES THAT WON'T ADDRESSED FOR EXTERNAL REASONS

Coding for the Purposes of Learning

Unnecessary complexity of code for the scope of the project

- Practice for larger sets of data and different variations of the project
 - The simplest and most efficient process is preferred otherwise

Unnecessary comments for personal reference

- Used for reference in future Selenium projects if needed
 - Explanation of the code's function is unnecessary for experienced programmers

Future Improvements to Implement

FEATURES AND IMPROVEMENTS TO BE MADE
WITH ADDITIONAL TIME

New Features for Added Value

Dashboard functionality to allow users to select parameters

- Users can choose the filter(s) applied and visualization(s) presented

Visualizations to show the trending characters in different skill levels

- In alignment with understanding the meta of the game
 - Allow CAPCOM to tweak overwhelming strong and weak characters
 - Allow players to train for common characters in their rank

Adjustments to Improve Quality

Completely dynamic graphs to prevent formatting issues

- Textbox of total ranked population is static

Web scrape without showing the process in a browser

- Only useful during development

Further improve code readability and efficiency

Condense accompanying slide presentation

- Created to be read, not presented
- Create separate slides for each major tick mark in the visualization walkthrough
 - Recreate code to show the graph being built step-by-step

End of Presentation

TIME FOR QUESTIONS, COMMENTS, OR CONCERNS

Recap of Topics Covered for Reference

Background and Context

Project Details

Additional Project Details

Overview of Features

Process Walkthrough

- Web Scraping
- Visualizations

Reflection

- Problems Encountered and Addressed
- Potential Problems Avoided
- Improvements Not Implemented
- Future Improvements to Implement