Data Wrangling on BF4 Dataset

Step 1: Web Scraping

The BF4 dataset is split into two parts. The first part, which contains all player rankings are stored on the webpage of the site, whereas the second part containing all player stat details can be obtained via API call.

In order to gather the first part of the data, we need to first obtain the final page of the website that we are scraping. We can do this by looking at the web link and clicking on a few pages. We can see that by clicking on page 11, the link ends with “?start=500”, if we were to click on page 12, the link ends with “?start=550”. With this information we can infer that each page contains 50 lines of player data.

https://bf4stats.com/leaderboards/xone_player_score?start=500
https://bf4stats.com/leaderboards/xone_player_score?start=550

We can try changing the ending of the page to say “?start=55500”, which means we’d like to start from entry 55500, however, we get a blank page with no data. The good thing is that this reveals the total pages that exist on the site, we can use that number to create a loop as to how many pages need to be scraped. Since each page contains 50 entries, we know that we need to manipulate the web link to end with “?start= (page number) * 50”. Then we can use beautifulsoup and urlrequest to commence our operation.

Step 2: API testing and File Downloading

By reviewing https://bf4stats.com/api , we can create the url link that is needed to gather player stat data using the player’s name, which we had already obtained via web scraping. However, before we go on about looping through the player list we have, we should first evaluate the individual player file size that we can generate via API. By using our API, we first obtain the player information, then we store that information as a pkl file. A pickle file is a file created via a python module that allows objects to be serialized to files on disk and deserialized back into the program at runtime. It’s an easy way to store python objects and can be un-pickled using python when the information is needed.

By storing one API call file as pkl format, we can see that the file size is around 980kb per player. Meaning that we’d get about 38GB of data if we were to collect all player data. We know have two choices in handling data of this size, one is to create a MongoDB database and store this data in NoSQL format, the other is to save this data to Google Cloud Services, just in case we need to involve the usage of big data tools such as Cloud Dataproc. We decided to go with the latter decision in this case. We first set up GCP and then create a bucket where all the pkl files will be stored. Then we go through a list of player_id and use that player_id to make the api calls. The files will be stored on the local machine temporarily then dumped into the GCP bucket.

Step 3: Processing the Data

To process the data, we need to re-download the player data from the GCP bucket, pick the correct columns that contains data that can be analyzed using machine learning and statistics, then save the data into a pandas dataframe and csv file. We will need to install a few tools on windows, which are the “googlecloud SDK” and windows command prompt extension “coreutils”. Using these tools we can get a grasp of how big our data is, how many objects are contained within and a list of the object names.

After downloading one single file, we use json_normalize to transpose some of the data. We discovered that some nested json that were normalized contained their own table indexes. In order to utilize that data, we need to transform the index to be included in the header, converting a 3 row * 3 column table to a 9 column * 1 row table. For instance, if we had a table that contained “kills, accuracy, use_time” for “AK47, M4, AR-15”, we would convert that table to only include one row but have headers as AK47_kills, AK47_accuracy, AK47_use_time, M4_kills, M4_accuracy, M4_use_time etc. Moving on to data selection, the player data will include 5 parts, player information, player basic stats, play modes, weapon category and usage, vehicle category and usage. There are also other data contained within the player file, but most of them are extensions of the 5 parts above.

After writing all required functions, we will iterate through the list of object names we obtained via command prompt and use it to process and transform our data. The data is first stored in a pandas dataframe and then saved as a csv. During this process, in order to save space, we replace all NA values with a number 999.99. We might also run into connection issues and errors along the way, but since we have the list of player names, we can save what we collected first, evaluate what went wrong during the process, make adjustments, then continue to iterate through the list.

Step 4: Data Wrangling

Since we downloaded the data in five parts, we can row bind the 5 parts together into 1 and load them as a dataframe. The file only uses 114MB now because of the data processing we did in step 3. We can reindex this dataframe and drop any unneeded columns. Then we describe this data and see if there are any unusual information.

There are certain columns that contains data where the min and max are the same. Usually this kind of data doesn't really contribute to analysis so we use loop to find all of them and have them dropped from the dataframe. Then we look for duplicate datasets. We found one entry that was collected twice using the value_count function, which auto-sorts counts by highest to lowest. We can remove the duplicate data set. We also notice that the 75% and 25% of the data for bestStreak is somehow the same, we plot bestStreak out to see what's going on and find that only a small percentage of players are able to make high bestStreak numbers whereas the majority of players only gain 0 to 5 bestStreak numbers, thus we delete this column and other columns similar to it where the 75% and 25% data are the same. Since in the previous processing step, we used 999.99 to replace all NA values, we can try to find which columns contain this value to check for missing values.

Since no value is returned, we can conclude that our dataframe doesn't contain any missing values. Video game data usually have a high level of completeness and missing data doesn't occur if the game's data collection process is well designed. Lastly, we merge this dataframe with our web scraped data to obtain our final data frame. We can now use this data frame for statistical and machine learning analysis.