Tom Duong

CIS 4130 CMWA

Prof. Holowczak

5/17/2024

https://github.com/TQD02/CIS4130_Tom_Duong

I. Project Proposal

An interest in games lies in many of us, including myself, but only few have found significance in the data behind those games. I grew up playing the game Clash Royale, and seeing a big dataset from the top players sparked a fire in me. From Clash Royale Wiki, the game is “the fast-paced brawler where you collect cards and duel players in real time. Destroy your opponent's Crown Towers, but be sure to defend your own.” Each match includes 2 players, each possessing a deck of 8 character cards they can deploy for a certain amount of elixir.

The dataset that I would be using could be assessed at
https://www.kaggle.com/datasets/s1m0n38/clash-royale-games/data

The dataset in question is statistics from matches of top players from September 2022 to December 2023 Each month represents one season of the game which adds up to 15 seasons of game data. This data set contains 23 columns which includes: information of when the match took place and game mode. It also contains information regarding 2 players in each match (11 columns each): their unique ids, trophies before the match, the number of crowns left and their 8 specific cards on their deck. I intend on predicting the number of crowns left in matches based on the average elixir cost of each player’s deck using a supervised learning model.

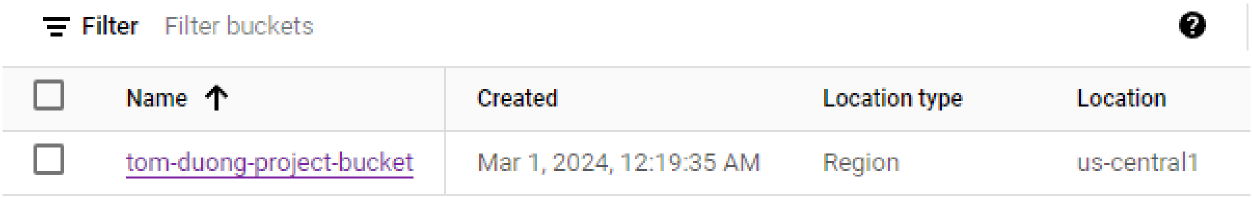
II. Data Transfer and Storage:

- Refer to Appendix A
- Download dataset

```
$ kaggle datasets download -d s1m0n38/clash-royale-games
$ unzip -j clash-royale-games '*.csv'
```

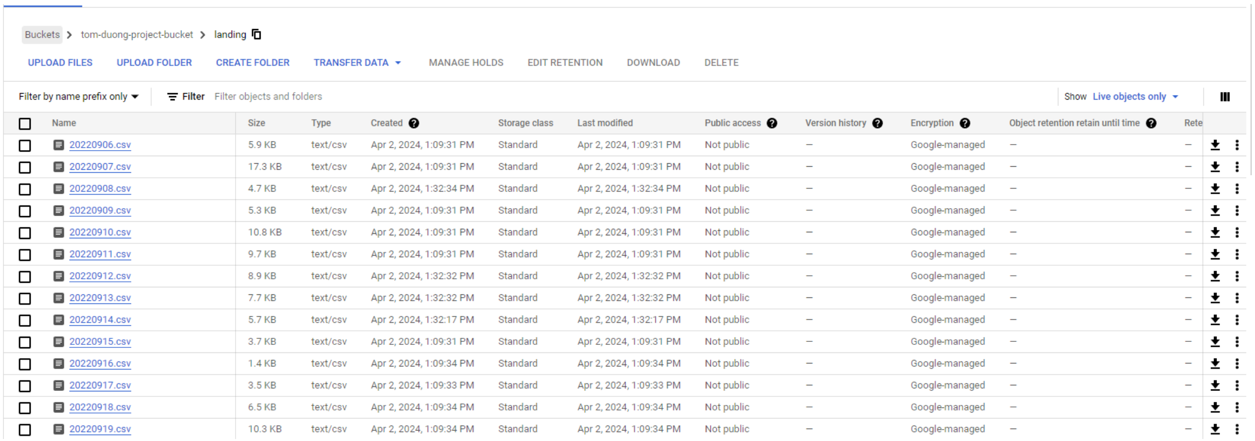
- Create bucket and relocating dataset to said bucket

```
$ gcloud storage buckets create gs://my-project-bucket
--project=extended-cable-413602 --default-storage-class=STANDARD
--location=us-central1 --uniform-bucket-level-access
$ gcloud auth login
$ gcloud config set project extended-cable-413602
$ gcloud storage cp /home/quang200211/pythondev/*.csv
gs://tom-duong-project-bucket/landing
```



Filter Filter buckets

□	Name ↑	Created	Location type	Location
□	tom-duong-project-bucket	Mar 1, 2024, 12:19:35 AM	Region	us-central1



Buckets > tom-duong-project-bucket > landing

Show Live objects only

□	Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Object retention retain until time	Rete
□	20220906.csv	5.9 KB	text/csv	Apr 2, 2024, 1:09:31 PM	Standard	Apr 2, 2024, 1:09:31 PM	Not public	–	Google-managed	–	–
□	20220907.csv	17.3 KB	text/csv	Apr 2, 2024, 1:09:31 PM	Standard	Apr 2, 2024, 1:09:31 PM	Not public	–	Google-managed	–	–
□	20220908.csv	4.7 KB	text/csv	Apr 2, 2024, 1:32:34 PM	Standard	Apr 2, 2024, 1:32:34 PM	Not public	–	Google-managed	–	–
□	20220909.csv	5.3 KB	text/csv	Apr 2, 2024, 1:09:31 PM	Standard	Apr 2, 2024, 1:09:31 PM	Not public	–	Google-managed	–	–
□	20220910.csv	10.8 KB	text/csv	Apr 2, 2024, 1:09:31 PM	Standard	Apr 2, 2024, 1:09:31 PM	Not public	–	Google-managed	–	–
□	20220911.csv	9.7 KB	text/csv	Apr 2, 2024, 1:09:31 PM	Standard	Apr 2, 2024, 1:09:31 PM	Not public	–	Google-managed	–	–
□	20220912.csv	8.9 KB	text/csv	Apr 2, 2024, 1:32:32 PM	Standard	Apr 2, 2024, 1:32:32 PM	Not public	–	Google-managed	–	–
□	20220913.csv	7.7 KB	text/csv	Apr 2, 2024, 1:32:32 PM	Standard	Apr 2, 2024, 1:32:32 PM	Not public	–	Google-managed	–	–
□	20220914.csv	5.7 KB	text/csv	Apr 2, 2024, 1:32:17 PM	Standard	Apr 2, 2024, 1:32:17 PM	Not public	–	Google-managed	–	–
□	20220915.csv	3.7 KB	text/csv	Apr 2, 2024, 1:09:31 PM	Standard	Apr 2, 2024, 1:09:31 PM	Not public	–	Google-managed	–	–
□	20220916.csv	1.4 KB	text/csv	Apr 2, 2024, 1:09:34 PM	Standard	Apr 2, 2024, 1:09:34 PM	Not public	–	Google-managed	–	–
□	20220917.csv	3.5 KB	text/csv	Apr 2, 2024, 1:09:33 PM	Standard	Apr 2, 2024, 1:09:33 PM	Not public	–	Google-managed	–	–
□	20220918.csv	6.5 KB	text/csv	Apr 2, 2024, 1:09:34 PM	Standard	Apr 2, 2024, 1:09:34 PM	Not public	–	Google-managed	–	–
□	20220919.csv	10.3 KB	text/csv	Apr 2, 2024, 1:09:34 PM	Standard	Apr 2, 2024, 1:09:34 PM	Not public	–	Google-managed	–	–

III. Exploratory Data Analysis and Data Cleaning:

- Refer to Appendix B + C

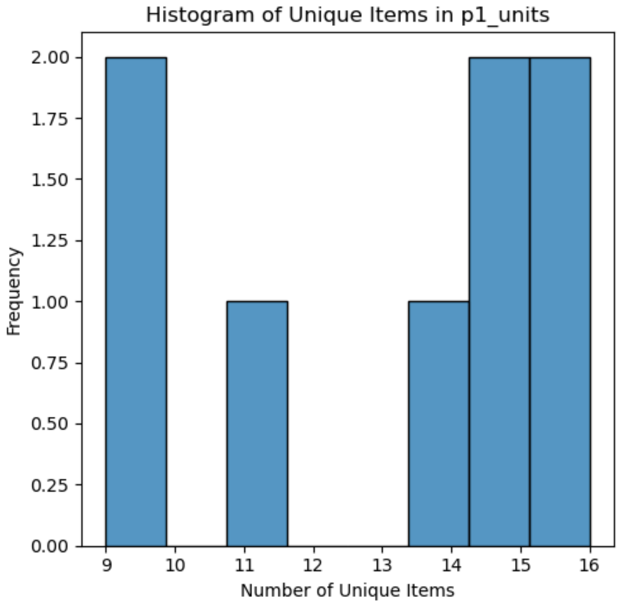
For the EDA, I decided to take a look at the first two files in my landing folder.

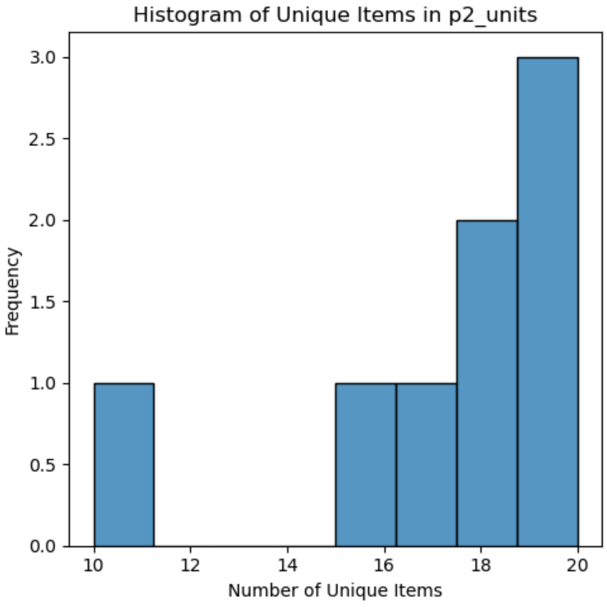
For the result I have obtained the following:

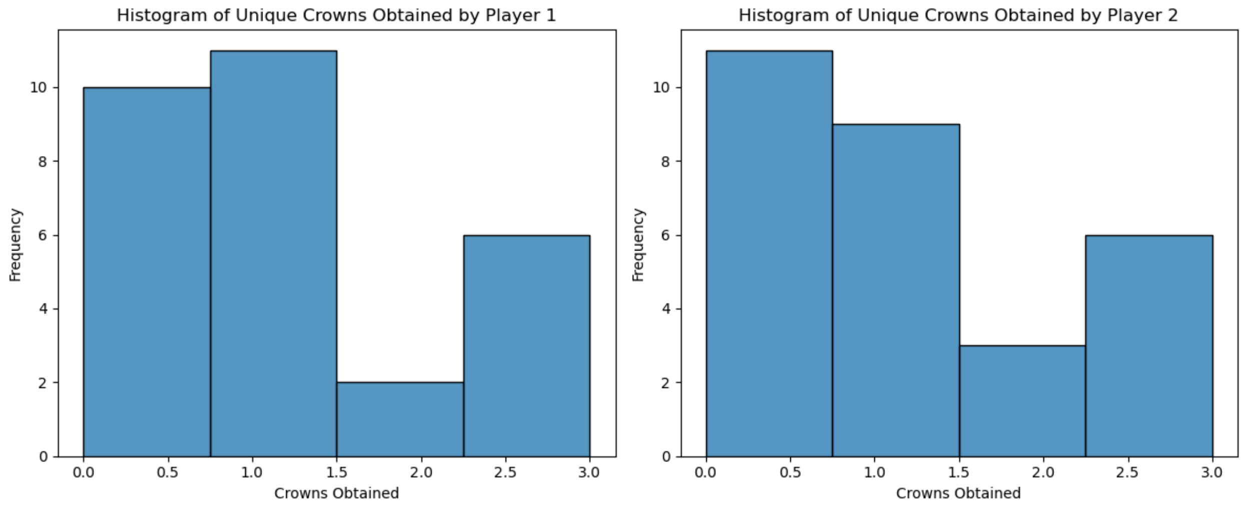
file 20220906.csv

```
match_ID  p1_trophies  p1_crowns_obtained      p1_unit1  \
count  2.900000e+01    29.000000          29.000000  2.900000e+01
mean   7.200001e+07  5779.655172          1.137931  2.600001e+07
std    1.535652e+01   689.179444          1.125171  7.355170e+00
min    7.200001e+07  5000.000000          0.000000  2.600000e+07
25%    7.200001e+07  5001.000000          0.000000  2.600001e+07
50%    7.200001e+07  6070.000000          1.000000  2.600001e+07
75%    7.200001e+07  6571.000000          2.000000  2.600002e+07
max    7.200007e+07  6623.000000          3.000000  2.600004e+07
          p1_unit7  ...  p2_trophies  p2_crowns_obtained      p2_unit1  \
count  2.900000e+01  ...    29.000000          29.000000  2.900000e+01
mean   2.786208e+07  ...  5746.793103          1.137931  2.600001e+07
std    5.157489e+05  ...   631.458141          1.156477  9.520266e+00
min    2.600004e+07  ...  5002.000000          0.000000  2.600000e+07
25%    2.800000e+07  ...  5034.000000          0.000000  2.600001e+07
50%    2.800000e+07  ...  5920.000000          1.000000  2.600001e+07
```

```
75%    2.800001e+07  ...  6196.000000           2.000000  2.600002e+07
max    2.800001e+07  ...  6632.000000           3.000000  2.600005e+07
          p2_unit2      p2_unit3      p2_unit4      p2_unit5      p2_unit6  \
count  2.900000e+01  2.900000e+01  2.900000e+01  2.900000e+01  2.900000e+01
mean   2.600002e+07  2.600003e+07  2.600004e+07  2.620694e+07  2.710347e+07
std    1.132665e+01  1.275672e+01  1.574684e+01  4.912857e+05  8.169758e+05
min    2.600001e+07  2.600002e+07  2.600002e+07  2.600002e+07  2.600004e+07
25%    2.600002e+07  2.600002e+07  2.600003e+07  2.600005e+07  2.600008e+07
50%    2.600002e+07  2.600003e+07  2.600004e+07  2.600006e+07  2.700000e+07
75%    2.600003e+07  2.600004e+07  2.600006e+07  2.600007e+07  2.800000e+07
max    2.600005e+07  2.600006e+07  2.600008e+07  2.800001e+07  2.800001e+07
          p2_unit7      p2_unit8
count  2.900000e+01  2.900000e+01
mean   2.789656e+07  2.800001e+07
std    4.092431e+05  3.892015e+00
min    2.600006e+07  2.800000e+07
25%    2.800000e+07  2.800001e+07
50%    2.800000e+07  2.800001e+07
75%    2.800001e+07  2.800002e+07
max    2.800001e+07  2.800002e+07
```



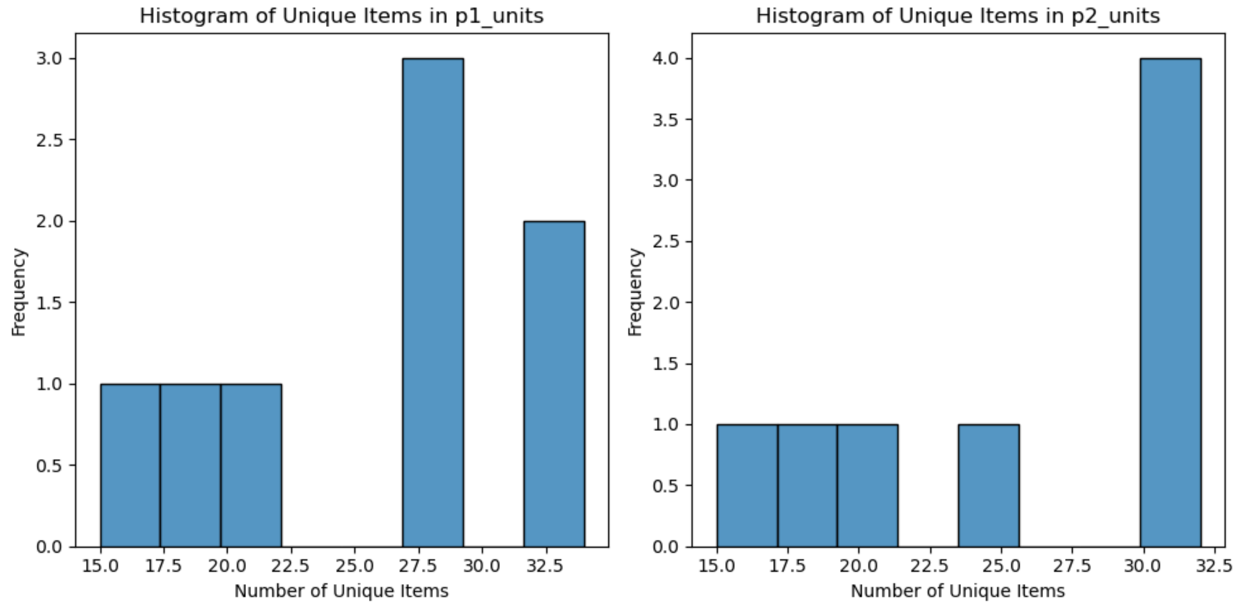


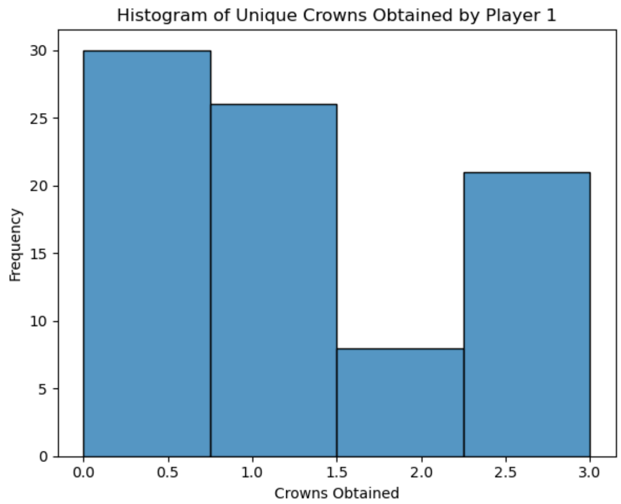


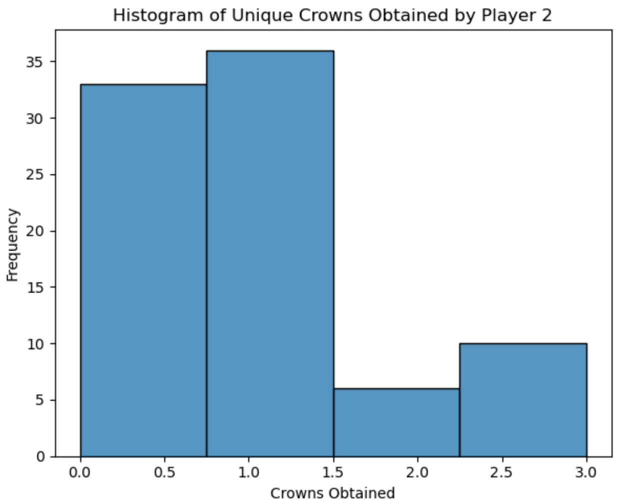
```
Working on file 20220907.csv
           match_ID  p1_trophies  p1_crowns_obtained      p1_unit1  \
count  8.500000e+01    85.000000           85.000000  8.500000e+01   
mean   7.200013e+07  6157.388235            1.235294  2.600001e+07   
std    1.390923e+02   613.515749            1.181676  1.087872e+01   
min    7.200001e+07  5030.000000            0.000000  2.600000e+07   
25%    7.200001e+07  5675.000000            0.000000  2.600001e+07   
50%    7.200001e+07  6570.000000            1.000000  2.600001e+07   
75%    7.200029e+07  6630.000000            2.000000  2.600002e+07   
max    7.200029e+07  6763.000000            3.000000  2.600005e+07   
           p1_unit2      p1_unit3      p1_unit4      p1_unit5      p1_unit6  \
count  8.500000e+01  8.500000e+01  8.500000e+01  8.500000e+01  8.500000e+01   
mean   2.600002e+07  2.600004e+07  2.602358e+07  2.625886e+07  2.689415e+07   
std    1.371529e+01  1.530130e+01  1.524709e+05  4.405662e+05  8.731672e+05   
min    2.600000e+07  2.600001e+07  2.600002e+07  2.600002e+07  2.600003e+07   
25%    2.600001e+07  2.600002e+07  2.600004e+07  2.600004e+07  2.600006e+07   
50%    2.600002e+07  2.600003e+07  2.600005e+07  2.600006e+07  2.700000e+07   
75%    2.600003e+07  2.600005e+07  2.600006e+07  2.700000e+07  2.800000e+07   
max    2.600006e+07  2.600007e+07  2.700001e+07  2.700001e+07  2.800001e+07   
           p1_unit7  ...  p2_trophies  p2_crowns_obtained      p2_unit1  \
count  8.500000e+01  ...    85.000000           85.000000  8.500000e+01   
mean   2.784707e+07  ...  6025.705882            0.917647  2.600001e+07   
std    5.001284e+05  ...   621.405240            0.966237  1.380765e+01   
min    2.600004e+07  ...  5005.000000            0.000000  2.600000e+07   
25%    2.800000e+07  ...  5331.000000            0.000000  2.600001e+07   
50%    2.800000e+07  ...  6226.000000            1.000000  2.600001e+07   
75%    2.800000e+07  ...  6620.000000            1.000000  2.600002e+07   
max    2.800001e+07  ...  6736.000000            3.000000  2.600005e+07   
           p2_unit2      p2_unit3      p2_unit4      p2_unit5      p2_unit6  \
count  8.500000e+01  8.500000e+01  8.500000e+01  8.500000e+01  8.500000e+01   
mean   2.600002e+07  2.600003e+07  2.604710e+07  2.630592e+07  2.692944e+07   
std    1.413469e+01  1.551436e+01  2.130131e+05  5.122935e+05  8.835314e+05   
```

```
min    2.600000e+07  2.600001e+07  2.600002e+07  2.600002e+07  2.600003e+07
25%    2.600001e+07  2.600002e+07  2.600004e+07  2.600004e+07  2.600006e+07
50%    2.600002e+07  2.600003e+07  2.600005e+07  2.600006e+07  2.700001e+07
75%    2.600003e+07  2.600004e+07  2.600006e+07  2.700000e+07  2.800000e+07
max    2.600005e+07  2.600006e+07  2.700001e+07  2.800001e+07  2.800001e+07
           p2_unit7      p2_unit8
count  8.500000e+01  8.500000e+01
mean   2.787060e+07  2.797648e+07
std    4.827451e+05  2.169247e+05
min    2.600003e+07  2.600006e+07
25%    2.800000e+07  2.800001e+07
50%    2.800000e+07  2.800001e+07
75%    2.800001e+07  2.800001e+07
max    2.800002e+07  2.800002e+07

[8 rows x 21 columns]
gs://tom-duong-project-bucket/landing Columns with null values
[]
gs://tom-duong-project-bucket/landing Number of Rows with null values: 0
gs://tom-duong-project-bucket/landing Integer data type columns: Index(['match_ID', 'p1_trophies',
'p1_crowns_obtained', 'p1_unit1', 'p1_unit2',
       'p1_unit3', 'p1_unit4', 'p1_unit5', 'p1_unit6', 'p1_unit7', 'p1_unit8',
       'p2_trophies', 'p2_crowns_obtained', 'p2_unit1', 'p2_unit2', 'p2_unit3',
       'p2_unit4', 'p2_unit5', 'p2_unit6', 'p2_unit7', 'p2_unit8'],
      dtype='object')
```







From the results, the amount of games played amongst top players seems to fluctuate hugely on a daily basis, going from 20s to 80s. Amongst the top players, the highest percent of crowns obtained lies around 0 and 1. Given the similarity in skills among top players, this is understandable. I also tried to look into the number of unique units that players bring but it seems to bring relatively similar results amongst the two files, further analysis on a greater number of files could possibly bring a better result.

As for the process of data cleaning, there were no headers for any of the slides, as a result I have to read the data into a dataframe with pre-set column headers. The units are marked by their unique ids. I first scraped through the JSON file containing all the unit info, and systematically replace those ids with corresponding unit names. This is the result:

```
date  match_ID     p1_tag  p1_trophies  \
0  2022-09-06 00:06:53+00:00  72000006  2UP28CLJP        5217
1  2022-09-06 00:51:49+00:00  72000006   YULUVJCG        5560
2  2022-09-06 01:27:32+00:00  72000007  R29J9J9LU        5015
3  2022-09-06 01:43:10+00:00  72000007   9ULPU0QL        6600
4  2022-09-06 01:53:52+00:00  72000007  8YJC92L0P        6064
5  2022-09-06 03:18:28+00:00  72000007  LVJVVQGVQ        5001
6  2022-09-06 03:22:51+00:00  72000007  LVJVVQGVQ        5001
7  2022-09-06 04:49:30+00:00  72000007  LVJVVQGVQ        5001
8  2022-09-06 04:53:13+00:00  72000007  LVJVVQGVQ        5001
9  2022-09-06 05:11:45+00:00  72000007  LVJVVQGVQ        5001
10 2022-09-06 05:13:58+00:00  72000006   LG8GYVPU        6070
11 2022-09-06 05:15:20+00:00  72000007  LVJVVQGVQ        5001
12 2022-09-06 05:21:59+00:00  72000006  8YVCPCUQC        5034
13 2022-09-06 05:31:59+00:00  72000007  LVJVVQGVQ        5001
14 2022-09-06 05:37:43+00:00  72000007  LVJVVQGVQ        5001
```

```
15 2022-09-06 07:04:39+00:00  72000006   LYUGL0RJ        6571
16 2022-09-06 13:39:25+00:00  72000006  820Y8CLJU        6171
17 2022-09-06 13:42:39+00:00  72000006  820Y8CLJU        6142
18 2022-09-06 13:54:25+00:00  72000006  P0RG0Q9CL        6099
19 2022-09-06 13:58:49+00:00  72000006    PGRLLJR        6108
20 2022-09-06 14:03:55+00:00  72000006   9C0CQLRG        6228
21 2022-09-06 14:07:05+00:00  72000006   UQYQY9U0        6103
22 2022-09-06 16:11:02+00:00  72000006   R0VVVLJQ        5000
23 2022-09-06 19:52:53+00:00  72000006   LYUGL0RJ        6597
24 2022-09-06 19:57:53+00:00  72000006   LYUGL0RJ        6623
25 2022-09-06 21:46:04+00:00  72000007  YPC0VJYR2        6600
26 2022-09-06 21:52:57+00:00  72000007  YPC0VJYR2        6600
27 2022-09-06 22:03:12+00:00  72000066  YPC0VJYR2        6600
28 2022-09-06 22:07:30+00:00  72000066  YPC0VJYR2        6600

    p1_crowns_obtained        p1_unit1          p1_unit2          p1_unit3  \
0                    0           Giant             Witch          Valkyrie   
1                    1     Baby Dragon        Lumberjack       Mega Minion   
2                    0         P.E.K.K.A        Battle Ram    Electro Wizard   
3                    3       Skeletons         Musketeer         Hog Rider   
4                    0   Giant Skeleton       Royal Giant            Hunter   
5                    2     Baby Dragon    Giant Skeleton       Royal Ghost   
6                    1   Giant Skeleton       Royal Ghost           Zappies   
7                    3   Giant Skeleton       Royal Ghost           Zappies   
8                    2   Giant Skeleton       Royal Ghost           Zappies   
9                    1   Giant Skeleton       Royal Ghost           Zappies   
10                   1         Balloon        Barbarians         Hog Rider   
11                   0   Giant Skeleton       Royal Ghost           Zappies   
12                   3           Golem       Baby Dragon            Prince   
13                   1   Giant Skeleton       Royal Ghost           Zappies   
14                   0          Bomber    Mini P.E.K.K.A              Bats   
15                   1         Balloon        Barbarians        Lava Hound   
16                   1           Golem       Baby Dragon    Mini P.E.K.K.A   
17                   0           Golem       Baby Dragon    Mini P.E.K.K.A   
18                   1  Inferno Dragon  Elite Barbarians    Electro Dragon   
19                   1       Skeletons         Musketeer         Hog Rider   
20                   1  Mini P.E.K.K.A    Inferno Dragon       Mega Knight   
21                   1       Skeletons    Giant Skeleton            Hunter   
22                   0           Witch        Ice Spirit  Elite Barbarians   
23                   3         Balloon        Barbarians        Lava Hound   
24                   3         Balloon        Barbarians        Lava Hound   
25                   0       Skeletons    Giant Skeleton         Hog Rider   
26                   0  Giant Skeleton       Royal Ghost           Zappies   
27                   3       Skeletons          Valkyrie        Ice Wizard   
28                   0       Skeletons          Valkyrie        Ice Wizard   
```

```
       p1_unit4         p1_unit5  ... p2_trophies p2_crowns_obtained  \
0        Bomber      Dark Prince  ...        5272                  1
1   Night Witch    Electro Giant  ...        5532                  0
2        Bandit      Royal Ghost  ...        6600                  1
3   Mighty Miner          Mortar  ...        6600                  0
4       Zappies        Fisherman  ...        6600                  1
5    Royal Hogs        Fisherman  ...        5002                  1
6    Royal Hogs        Fisherman  ...        5002                  3
7    Royal Hogs        Fisherman  ...        5002                  0
8    Royal Hogs        Fisherman  ...        5002                  1
9    Royal Hogs        Fisherman  ...        5002                  0
10       Bowler       Lumberjack  ...        5989                  0
11   Royal Hogs        Fisherman  ...        5002                  1
12  Night Witch             Bats  ...        5005                  0
13   Royal Hogs        Fisherman  ...        5034                  0
14 Electro Giant Elixir Collector ...        5034                  1
15        Miner   Inferno Dragon  ...        6632                  2
16  Mega Minion      Night Witch  ...        6112                  0
17  Mega Minion      Night Witch  ...        6209                  3
18  Elixir Golem    Battle Healer ...        6170                  3
19  Dark Prince             Bats  ...        6196                  2
20  Magic Archer    Skeleton King ...        6167                  0
21  Royal Ghost          Zappies  ...        6193                  3
22       Bandit            Tesla  ...        5068                  3
23        Miner   Inferno Dragon  ...        6505                  0
24        Miner   Inferno Dragon  ...        6535                  2
25  Fire Spirit     Archer Queen  ...        5947                  3
26   Royal Hogs        Fisherman  ...        5920                  1
27        Tesla           X-Bow  ...        5410                  0
28        Tesla           X-Bow  ...        5915                  1

        p2_unit1         p2_unit2        p2_unit3       p2_unit4  \
0      Skeletons        Musketeer       Hog Rider     Ice Spirit
1  Skeleton Army           Wizard  Giant Skeleton         Sparky
2  Skeleton Army      Royal Giant         Zappies      Fisherman
3        Balloon         Valkyrie          Guards          Miner
4      Skeletons   Giant Skeleton     Fire Spirit   Archer Queen
5          Giant           Prince     Dark Prince          Miner
6  Skeleton Army    Mini P.E.K.K.A  Giant Skeleton    Royal Giant
7  Mini P.E.K.K.A      Dark Prince          Sparky    Mega Minion
8      Skeletons   Giant Skeleton       Hog Rider    Fire Spirit
9  Giant Skeleton      Royal Ghost         Zappies     Royal Hogs
10   Dark Prince   Inferno Dragon         Zappies  Golden Knight
11 Royal Recruits          Zappies  Flying Machine     Royal Hogs
12       Minions  Three Musketeers      Lumberjack         Bandit
13        Bomber      Dark Prince  Inferno Dragon   Mother Witch
```

```
14       Skeletons      Baby Dragon  Giant Skeleton          Hunter
15   Spear Goblins            Miner            Bats  Wall Breakers
16          Knight         Princess       Ice Spirit    Goblin Gang
17         P.E.K.K.A      Dark Prince  Electro Wizard       Ram Rider
18           Golem      Baby Dragon  Mini P.E.K.K.A     Mega Minion
19           Golem      Baby Dragon  Mini P.E.K.K.A     Mega Minion
20           Golem      Baby Dragon  Mini P.E.K.K.A     Mega Minion
21           Golem      Baby Dragon  Mini P.E.K.K.A     Mega Minion
22    Minion Horde      Goblin Gang     Executioner     Cannon Cart
23       Skeletons         Valkyrie       Hog Rider     Fire Spirit
24       Skeletons        Musketeer       Hog Rider      Ice Spirit
25     Fire Spirit            Miner      Battle Ram     Mega Minion
26        Valkyrie           Guards           Miner            Bats
27           Witch       Barbarians          Prince          Wizard
28  Giant Skeleton       Lumberjack  Electro Wizard      Royal Hogs
          p2_unit5         p2_unit6        p2_unit7        p2_unit8
0        Ice Golem           Cannon        Fireball         The Log
1      Mega Knight     Magic Archer   Inferno Tower          Arrows
2    Skeleton King     Mother Witch        Fireball         The Log
3      Executioner       Bomb Tower             Zap         Tornado
4           Cannon     Goblin Drill        Fireball         The Log
5           Hunter         Fireball             Zap         The Log
6        Fisherman      Firecracker        Fireball  Barbarian Barrel
7   Electro Wizard     Goblin Giant            Rage             Zap
8     Archer Queen           Cannon         The Log      Earthquake
9        Fisherman     Mother Witch          Arrows          Mirror
10       Tombstone           Poison       Graveyard  Barbarian Barrel
11     Goblin Cage         Fireball          Arrows  Barbarian Barrel
12          Mirror        Lightning       Graveyard           Clone
13   Electro Giant      Goblin Cage       Lightning         Tornado
14       Fisherman         Fireball          Freeze       Graveyard
15     Mighty Miner       Bomb Tower        Fireball         The Log
16   Inferno Tower           Rocket   Goblin Barrel         The Log
17    Mother Witch           Mirror          Poison  Barbarian Barrel
18     Night Witch        Lightning         Tornado  Barbarian Barrel
19     Night Witch        Lightning         Tornado  Barbarian Barrel
20     Night Witch        Lightning         Tornado  Barbarian Barrel
21     Night Witch        Lightning         Tornado  Barbarian Barrel
22  Skeleton Barrel  Flying Machine     Firecracker        Fireball
23    Archer Queen           Cannon        Fireball         The Log
24       Ice Golem           Cannon        Fireball         The Log
25     Night Witch           Cannon          Poison         The Log
26    Archer Queen           Mortar          Poison         The Log
27       Hog Rider  Elite Barbarians          Rocket          Mirror
28    Magic Archer     Archer Queen          Mirror         Tornado

[29 rows x 24 columns]
```

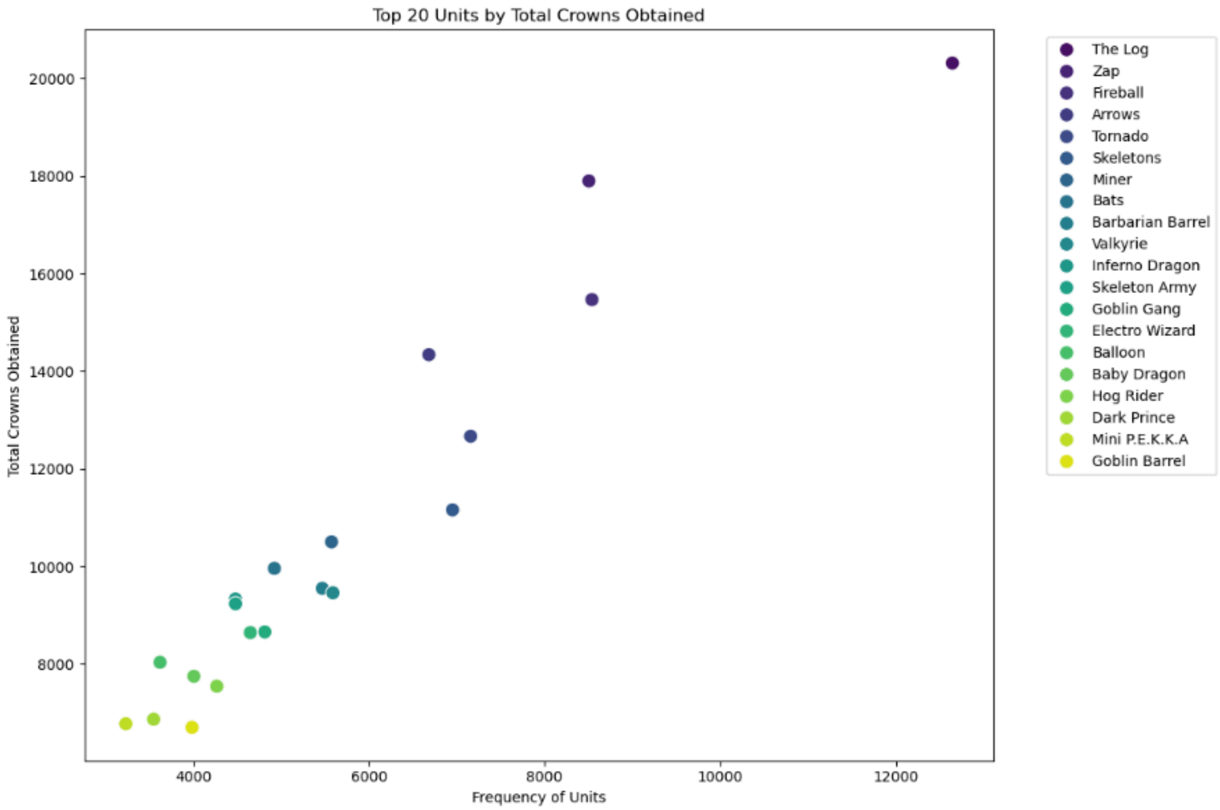
IV. Feature Engineering:

- Refer to Appendix D

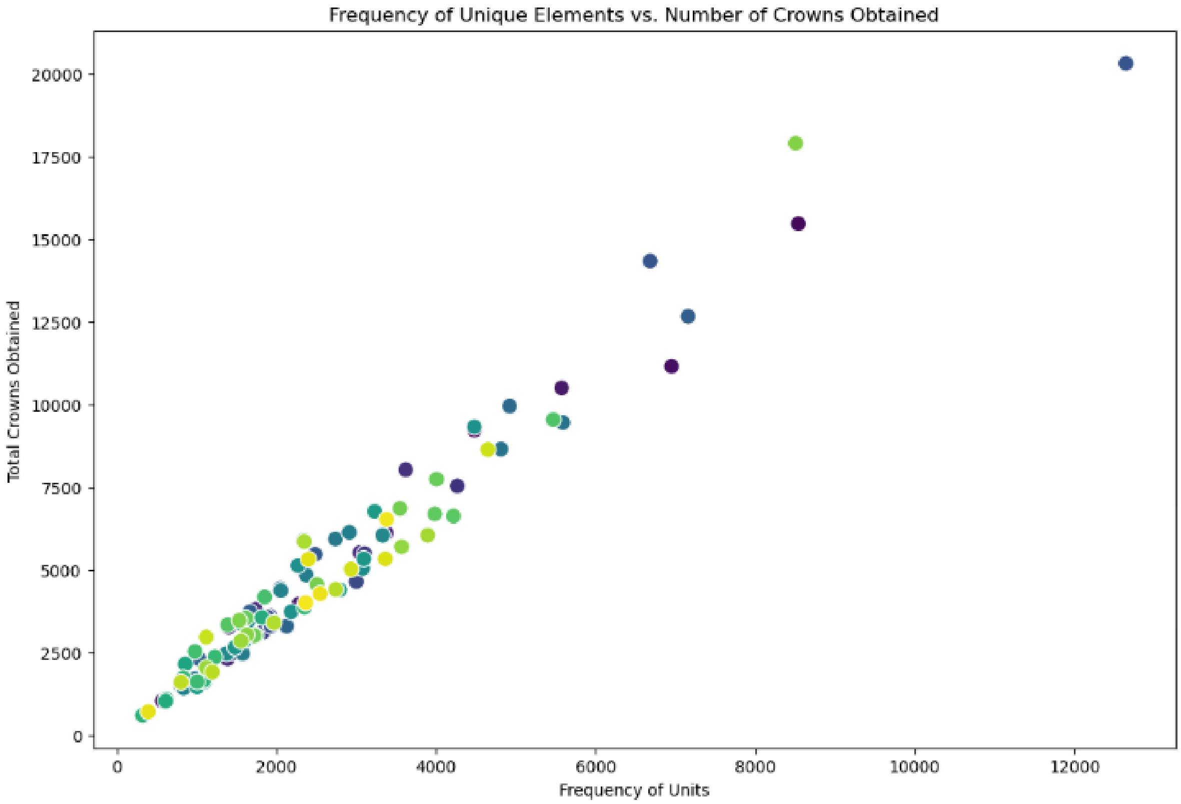
I began by loading and preprocessing data from a CSV file stored in Google Cloud Storage. Next, string indexing, one-hot encoding, and vector assembly were applied to prepare the data. After splitting the transformed DataFrame into training and testing datasets, a logistic regression model was trained on the training data. Evaluation metrics, including accuracy, precision, recall, and F1-score, were computed for each class using the test results. Additionally, cross-validation was performed to optimize model performance, with hyperparameters tuned using a grid search. Finally, the average metrics for each parameter combination in the grid were calculated, providing insights into the model's overall performance.

V. Data Visualization:

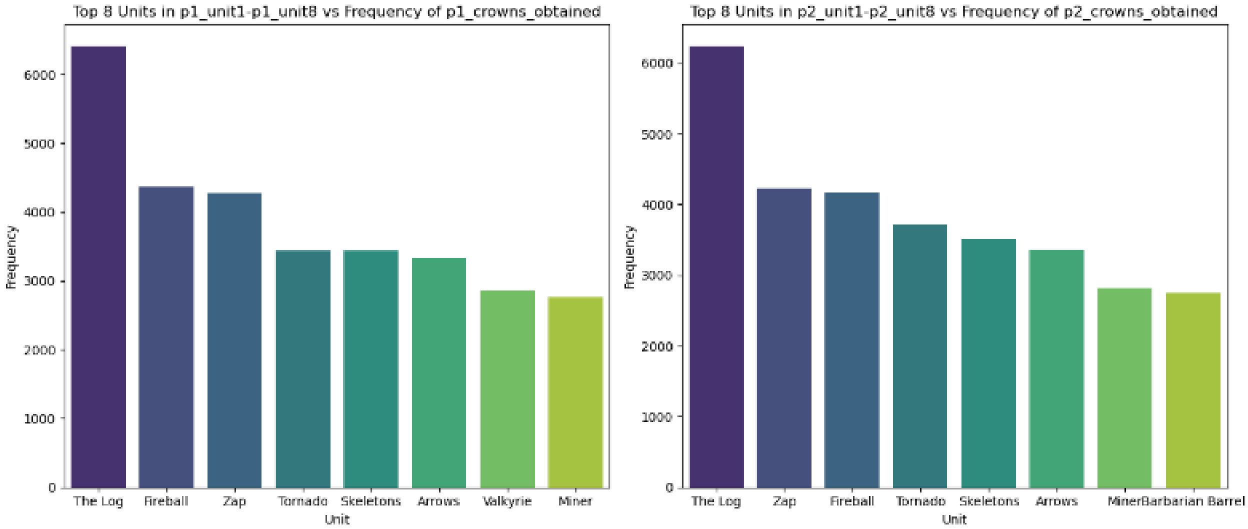
- Refer to Appendix E



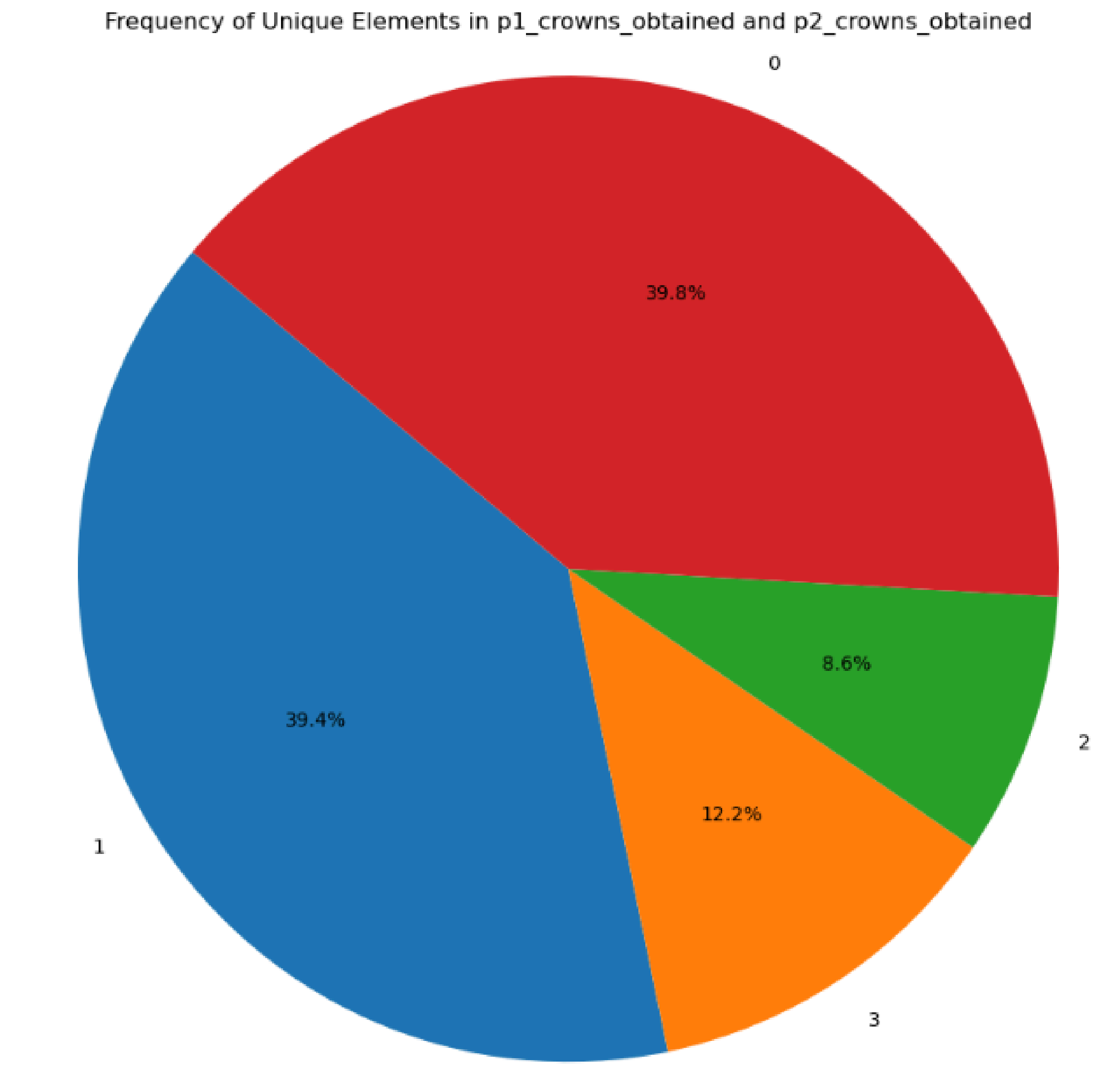
I first look at the relationship between the total number of crowns obtained and 20 most frequently appeared units. It seems they have a positive correlation.



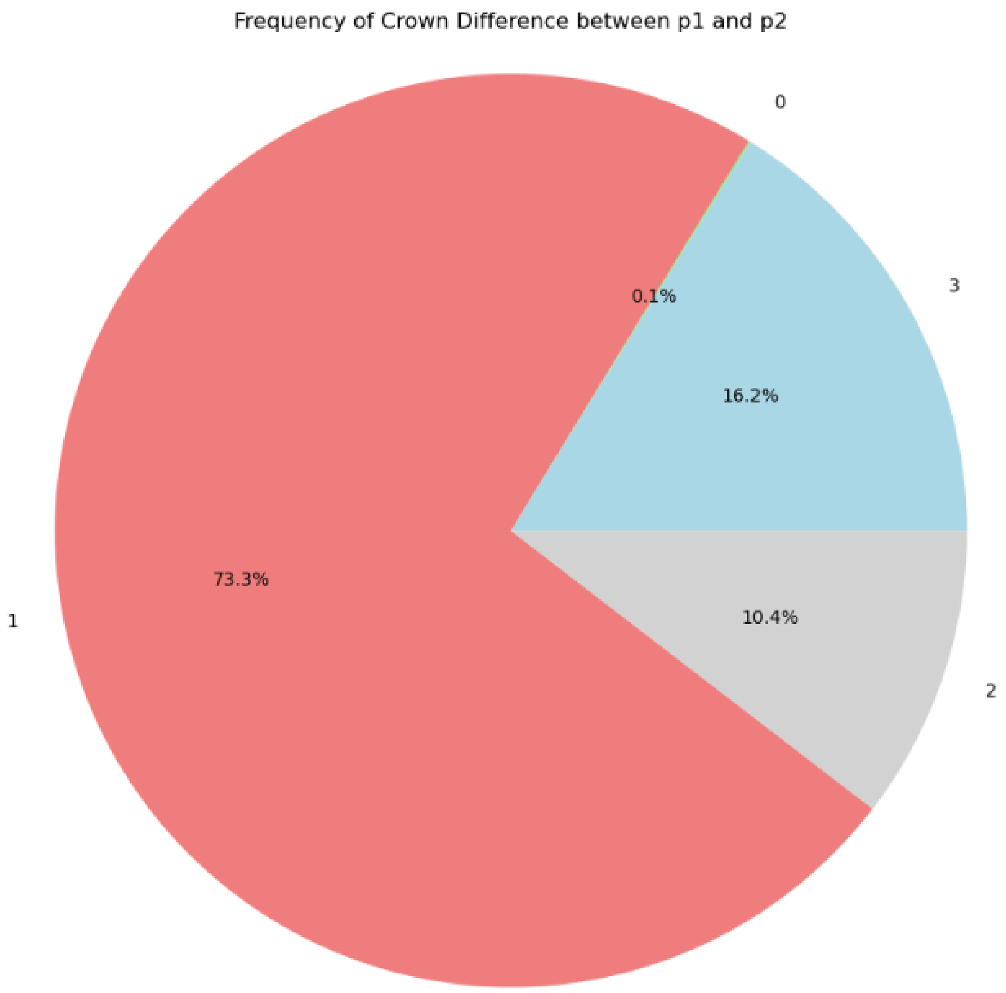
To further verify this, I look at the relationship between crowns obtained and the frequency of all the unique units. The positive correlation holds true.



I also looked into the 8 most frequently used units, seeing that each player can only have 8 cards on their deck. Not to my surprise, there is a similarity in composition amongst the highest rated players. There is only a difference of one unit between the two graphs indicating that the highest players stick closely to a “meta” - or a strategy that is commonly agreed to be the most optimal to become victorious.



I, then, looked into the number of crowns that players earned. It came as a surprise to me that almost 80% lies in 0 and 1 crowns. This goes to show the little gap that exists in the skills difference amongst the top players.



I also looked into their difference in crowns obtained in the matches between the top players. With how competitive and close-knit the skill levels are, it is logical that almost ¾ of the games come down to a difference of only 1 crown.

VI. Summary and Conclusion:

Area for improvement: I was not able to process and save the cleaned data file as a parquet file. Throughout the project, the data of top players in Clash Royale during the period of October and November in 2022. The purpose of the project is to build a prediction model to predict the number of crowns a player would obtain using the units that said player equipped. Throughout the process, The prediction model performed exceptionally in the case of players obtaining either 0 or 3 crowns, with an accuracy of 99%. The model accuracy falls behind when it comes to 1 and 2 crowns obtained with an accuracy of 71.1% and 85.7%. The average accuracy of the cross-validated model comes out to a 98.5%, which indicates robustness and effectiveness of the model. Moving forward, I intend on introducing more data to the model to keep training the models to increase the accuracy. I would also dive deeper into building a prediction model given a certain selection of units.

Appendix A.

- Download Kaggle API token
- Create a directory for Kaggle

```
$ mv kaggle.json .kaggle/
$ chmod 600 .kaggle/kaggle.json
$ ls -la .kaggle
```

- Upload kaggle token
- Install zip utilities

```
$ sudo apt install zip
$ sudo apt install zip
```

- install pip3, virtual environment tools, python virtual environment, directory, and virtual environment activation

```
$ sudo apt -y install python3-pip python3.11-venv
$ python3 -m venv pythondev
$ cd pythondev
$ source bin/activate
```

Appendix B.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

filepath = "gs://tom-duong-project-bucket/landing"

filelist = ['20220906.csv',
          '20220907.csv',
          '20220908.csv',
          '20220909.csv',
          '20220910.csv',
         ]

columns = ["date", "match_ID", "p1_tag", "p1_trophies", "p1_crowns_obtained",
         "p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
         "p1_unit6", "p1_unit7", "p1_unit8", "p2_tag", "p2_trophies",
         "p2_crowns_obtained", "p2_unit1", "p2_unit2", "p2_unit3",
         "p2_unit4", "p2_unit5", "p2_unit6", "p2_unit7", "p2_unit8"]
```

```
def perform_EDA(df: pd.DataFrame, filename : str):

    print(f"{filename} Number of records:")
    print(df.count())
    print(f"{filename} Number of variables:")
    print(df.shape[1])
    print(f"{filename} Number of duplicate records: { len(df)-len(df.drop_duplicates())}" )
    print(f"{filename} Info")
    print(df.info())
    print(f"{filename} Describe")
    print(df.describe())
    print(f"{filename} Columns with null values")
    print(df.columns[df.isnull().any()].tolist())
    rows_with_null_values = df.isnull().any(axis=1).sum()
    print(f"{filename} Number of Rows with null values: {rows_with_null_values}" )
    integer_column_list = df.select_dtypes(include='int64').columns
    print(f"{filename} Integer data type columns: {integer_column_list}")
    float_column_list = df.select_dtypes(include='float64').columns
    print(f"{filename} Float data type columns: {float_column_list}")
    unique_counts_p1 = df.loc[:, 'p1_unit1':'p1_unit8'].nunique()
    unique_counts_p2 = df.loc[:, 'p2_unit1':'p2_unit8'].nunique()
    print(f"{filename} Unique items in p1_unit1 to p1_unit8:")
    print(unique_counts_p1)
    print(f"{filename} Unique items in p2_unit1 to p2_unit8:")
    print(unique_counts_p2)
    unique_crowns_p1 = df['p1_crowns_obtained'].nunique()
    print(f"{filename} Unique items in p1_crowns_obtained: {unique_crowns_p1}")
    unique_crowns_p2 = df['p2_crowns_obtained'].nunique()
    print(f"{filename} Unique items in p2_crowns_obtained: {unique_crowns_p2}")
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    sns.histplot(unique_counts_p1, bins=len(unique_counts_p1), kde=False)
    plt.title('Histogram of Unique Items in p1_units')
    plt.xlabel('Number of Unique Items')
    plt.ylabel('Frequency')

    plt.subplot(1, 2, 2)
    sns.histplot(unique_counts_p2, bins=len(unique_counts_p2), kde=False)
    plt.title('Histogram of Unique Items in p2_units')
    plt.xlabel('Number of Unique Items')
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    sns.histplot(df['p1_crowns_obtained'], bins=unique_crowns_p1, kde=False)
    plt.title('Histogram of Unique Crowns Obtained by Player 1')
    plt.xlabel('Crowns Obtained')
    plt.ylabel('Frequency')

    plt.subplot(1, 2, 2)
    sns.histplot(df['p2_crowns_obtained'], bins=unique_crowns_p2, kde=False)
    plt.title('Histogram of Unique Crowns Obtained by Player 2')
    plt.xlabel('Crowns Obtained')
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()

for filename in filelist:
    print(f"Working on file {filename}")
    data_df = pd.read_csv(f"{filepath}/{filename}", sep = ",",header = None)
    data_df.columns = columns
    perform_EDA(data_df,filepath)
```

Appendix C.

```
import pandas as pd
import seaborn as sns
import requests
import json

filepath = "gs://tom-duong-project-bucket/landing"

filelist = ['20220906.csv']

columns = ["date", "match_ID", "p1_tag", "p1_trophies", "p1_crowns_obtained",
        "p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
        "p1_unit6", "p1_unit7", "p1_unit8", "p2_tag", "p2_trophies",
        "p2_crowns_obtained", "p2_unit1", "p2_unit2", "p2_unit3",
        "p2_unit4", "p2_unit5", "p2_unit6", "p2_unit7", "p2_unit8"]
```

```
def parse_json_from_url(url):
    response = requests.get(url)
    if response.status_code == 200:
        try:
            json_data = response.json()
            return json_data
        except json.decoder.JSONDecodeError as e:
            print("Error decoding JSON:", e)
    else:
        print("Failed to fetch data from URL:", response.status_code)

for filename in filelist:
    data_df = pd.read_csv(f"{filepath}/{filename}", sep = ",",header = None)
    data_df.columns = columns
url = "https://royaleapi.github.io/cr-api-data/json/cards_i18n.json"

ID_data = parse_json_from_url(url)

def extract_names_and_ids(parsed_data):
    names_and_ids = {}
    for item in parsed_data:
        if 'name' in item and 'id' in item:
            names_and_ids[item['id']] = item['name']
    return names_and_ids

name_id_data = extract_names_and_ids(ID_data)
print(name_id_data)

def unit_data_cleaning(df: pd.DataFrame, df2: list):
    unit_columns = ["p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
                    "p1_unit6", "p1_unit7", "p1_unit8", "p2_unit1", "p2_unit2",
                    "p2_unit3", "p2_unit4", "p2_unit5", "p2_unit6", "p2_unit7", "p2_unit8"]
    df['date'] = pd.to_datetime(data_df['date'])
    for col in unit_columns:
        df[col] = df[col].replace(df2)

unit_data_cleaning(data_df,name_id_data)
print(data_df)
```

Appendix D.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
   .appName("Read CSV into Spark DataFrame") \
   .getOrCreate()

gcs_path = "gs://tom-duong-project-bucket/cleaned/merged_data.csv"

spark_df = spark.read.csv(gcs_path, header=True, inferSchema=True)

spark_df.show(truncate=False)

from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression, LogisticRegressionModel
from pyspark.ml.evaluation import *
from pyspark.ml.tuning import *
import numpy as np

indexer = StringIndexer(inputCols = ["match_ID", "p1_tag", "p1_trophies",
"p1_crowns_obtained",
        "p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
        "p1_unit6", "p1_unit7", "p1_unit8", "p2_tag", "p2_trophies",
        "p2_crowns_obtained", "p2_unit1", "p2_unit2", "p2_unit3",
        "p2_unit4", "p2_unit5", "p2_unit6", "p2_unit7", "p2_unit8"] ,
                  outputCols = ["match_ID_index", "p1_tag_index", "p1_trophies_index",
"p1_crowns_obtained_index",
        "p1_unit1_index", "p1_unit2_index", "p1_unit3_index", "p1_unit4_index",
"p1_unit5_index",
        "p1_unit6_index", "p1_unit7_index", "p1_unit8_index", "p2_tag_index",
"p2_trophies_index",
        "p2_crowns_obtained_index", "p2_unit1_index", "p2_unit2_index", "p2_unit3_index",
        "p2_unit4_index", "p2_unit5_index", "p2_unit6_index", "p2_unit7_index",
"p2_unit8_index"])

encoder = OneHotEncoder(inputCols=["match_ID_index", "p1_tag_index", "p1_trophies_index",
"p1_crowns_obtained_index",
        "p1_unit1_index", "p1_unit2_index", "p1_unit3_index", "p1_unit4_index",
"p1_unit5_index",
        "p1_unit6_index", "p1_unit7_index", "p1_unit8_index", "p2_tag_index",
"p2_trophies_index",
        "p2_crowns_obtained_index", "p2_unit1_index", "p2_unit2_index", "p2_unit3_index",
        "p2_unit4_index", "p2_unit5_index", "p2_unit6_index", "p2_unit7_index",
"p2_unit8_index"],
 outputCols=["match_ID_vector", "p1_tag_vector", "p1_trophies_vector",
"p1_crowns_obtained_vector",
          "p1_unit1_vector", "p1_unit2_vector", "p1_unit3_vector", "p1_unit4_vector",
"p1_unit5_vector",
          "p1_unit6_vector", "p1_unit7_vector", "p1_unit8_vector", "p2_tag_vector",
"p2_trophies_vector",
          "p2_crowns_obtained_vector", "p2_unit1_vector", "p2_unit2_vector", "p2_unit3_vector",
          "p2_unit4_vector", "p2_unit5_vector", "p2_unit6_vector", "p2_unit7_vector",
"p2_unit8_vector"],
dropLast=False)

assembler = VectorAssembler(inputCols=["match_ID_vector", "p1_tag_vector",
"p1_trophies_vector", "p1_crowns_obtained_vector",
          "p1_unit1_vector", "p1_unit2_vector", "p1_unit3_vector", "p1_unit4_vector",
"p1_unit5_vector",
          "p1_unit6_vector", "p1_unit7_vector", "p1_unit8_vector", "p2_tag_vector",
"p2_trophies_vector",
          "p2_crowns_obtained_vector", "p2_unit1_vector", "p2_unit2_vector", "p2_unit3_vector",
          "p2_unit4_vector", "p2_unit5_vector", "p2_unit6_vector", "p2_unit7_vector",
"p2_unit8_vector"], outputCol="features")
sdf_pipe =  Pipeline(stages=[indexer, encoder, assembler])


transformed_sdf = sdf_pipe.fit(spark_df).transform(spark_df)

trainingData, testData =transformed_sdf.randomSplit([0.7, 0.3], seed=42)

labelColumn = "p1_crowns_obtained"

lr = LogisticRegression(labelCol=labelColumn)
model = lr.fit(trainingData)



test_results = model.transform(testData)

test_results.groupby("p1_crowns_obtained").pivot('prediction').count().sort("p1_crowns_obtaine
d").show()

TP = [2104, 1996, 423, 602]
FP = [24, 2008, 426, 13]
```

```
FN = [11, 19, 434, 11]
TN = [3038,3157,4740,4551]
def calculate_recall_precision(tp, fp, fn, tn):
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    f1_score = 2 * ((precision * recall) / (precision + recall))
    return accuracy, precision, recall, f1_score

accuracy_0, precision_0, recall_0, f1_score_0 = calculate_recall_precision(TP[0], FP[0], FN[0],
TN[0])
accuracy_1, precision_1, recall_1, f1_score_1 = calculate_recall_precision(TP[1], FP[1], FN[1],
TN[1])
accuracy_2, precision_2, recall_2, f1_score_2 = calculate_recall_precision(TP[2], FP[2], FN[2],
TN[2])
accuracy_3, precision_3, recall_3, f1_score_3 = calculate_recall_precision(TP[3], FP[3], FN[3],
TN[3])

zero = [accuracy_0, precision_0, recall_0, f1_score_0]
one = [accuracy_1, precision_1, recall_1, f1_score_1]
two = [accuracy_2, precision_2, recall_2, f1_score_2 ]
three = [accuracy_3, precision_3, recall_3, f1_score_3]

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator


lr1 = LogisticRegression(featuresCol="features", labelCol=labelColumn,
rawPredictionCol="rawPrediction")

evaluator = MulticlassClassificationEvaluator(metricName="accuracy", labelCol=labelColumn)

grid = ParamGridBuilder().build()

cv = CrossValidator(estimator=lr1, estimatorParamMaps=grid, evaluator=evaluator,
numFolds=3)

cv_model = cv.fit(trainingData)

cv_avg_metrics = cv_model.avgMetrics

cv_avg_metrics
```

Appendix E

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, array, explode, count, sum as sum_agg
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

unit_columns = ["p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
                "p1_unit6", "p1_unit7", "p1_unit8", "p2_unit1", "p2_unit2",
                "p2_unit3", "p2_unit4", "p2_unit5", "p2_unit6", "p2_unit7",
                "p2_unit8"]

combined_units_df = spark_df.select(explode(array(*[col(column) for column in
unit_columns])).alias("unit"))

unit_freq_df = combined_units_df.groupBy("unit").count().orderBy("count", ascending=False)

unit_freq_pd = unit_freq_df.toPandas()

plt.figure(figsize=(17, 9))
sns.barplot(data=unit_freq_pd, x='unit', y='count', palette='viridis')

plt.xticks(rotation=90, ha='right')
plt.xlabel('Unit')
plt.ylabel('Frequency')
plt.title('Frequency of Unique Elements in p1_unit1 to p1_unit8 and p2_unit1 to p2_unit8')
plt.tight_layout(pad=3.0)
plt.savefig("unit_frequency_plot.png")

plt.show()

combined_units_df = spark_df.select(
    explode(array(*[col(column) for column in unit_columns])).alias("unit"),
    col("p1_crowns_obtained").alias("p1_crowns"),
    col("p2_crowns_obtained").alias("p2_crowns")
)

unit_freq_crowns_df = combined_units_df.groupBy("unit").agg(
    count("unit").alias("frequency"),
    sum_agg("p1_crowns").alias("total_p1_crowns"),
    sum_agg("p2_crowns").alias("total_p2_crowns")
)
```

```
unit_freq_crowns_pd = unit_freq_crowns_df.toPandas()

unit_freq_crowns_pd['total_crowns'] = unit_freq_crowns_pd['total_p1_crowns'] +
unit_freq_crowns_pd['total_p2_crowns']

top_20_units_pd = unit_freq_crowns_pd.nlargest(20, 'total_crowns')

plt.figure(figsize=(12, 8))
sns.scatterplot(data=top_20_units_pd, x='frequency', y='total_crowns', hue='unit',
palette='viridis', s=100)

plt.xlabel('Frequency of Units')
plt.ylabel('Total Crowns Obtained')
plt.title('Top 20 Units by Total Crowns Obtained')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig("top_20_unit_frequency_vs_crowns_scatter_plot.png")

plt.show()

combined_units_df = spark_df.select(
    explode(array(*[col(column) for column in unit_columns])).alias("unit"),
    col("p1_crowns_obtained").alias("p1_crowns"),
    col("p2_crowns_obtained").alias("p2_crowns")
)

unit_freq_crowns_df = combined_units_df.groupBy("unit").agg(
    count("unit").alias("unit_frequency"),
    sum_agg("p1_crowns").alias("total_p1_crowns"),
    sum_agg("p2_crowns").alias("total_p2_crowns")
)

unit_freq_crowns_pd = unit_freq_crowns_df.toPandas()

unit_freq_crowns_pd['total_crowns'] = unit_freq_crowns_pd['total_p1_crowns'] +
unit_freq_crowns_pd['total_p2_crowns']

plt.figure(figsize=(12, 8))
sns.scatterplot(data=unit_freq_crowns_pd, x='unit_frequency', y='total_crowns', hue='unit',
palette='viridis', s=100)

plt.xlabel('Frequency of Units')
plt.ylabel('Total Crowns Obtained')
plt.title('Frequency of Unique Elements vs. Number of Crowns Obtained')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig("unit_frequency_vs_crowns_obtained_scatter_plot.png")

plt.show()

unit_columns_p1 = ["p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
                "p1_unit6", "p1_unit7", "p1_unit8"]
unit_columns_p2 = ["p2_unit1", "p2_unit2", "p2_unit3", "p2_unit4", "p2_unit5",
                "p2_unit6", "p2_unit7", "p2_unit8"]
crown_column_p1 = "p1_crowns_obtained"
crown_column_p2 = "p2_crowns_obtained"

unit_freq_p1 = spark_df.select(explode(array(*[col(column) for column in
unit_columns_p1])).alias("unit")) \
                    .groupBy("unit").agg(count("unit").alias("frequency")) \
                    .orderBy(col("frequency").desc()).limit(8)
unit_freq_p2 = spark_df.select(explode(array(*[col(column) for column in
unit_columns_p2])).alias("unit")) \
                    .groupBy("unit").agg(count("unit").alias("frequency")) \
                    .orderBy(col("frequency").desc()).limit(8)

crown_freq_p1 = spark_df.groupBy(crown_column_p1).count().orderBy(col("count").desc())
crown_freq_p2 = spark_df.groupBy(crown_column_p2).count().orderBy(col("count").desc())

unit_freq_p1_pd = unit_freq_p1.toPandas()
unit_freq_p2_pd = unit_freq_p2.toPandas()
crown_freq_p1_pd = crown_freq_p1.toPandas()
crown_freq_p2_pd = crown_freq_p2.toPandas()

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.barplot(data=unit_freq_p1_pd, x='unit', y='frequency', palette='viridis')
plt.xlabel('Unit')
plt.ylabel('Frequency')
plt.title('Top 8 Units in p1_unit1-p1_unit8 vs Frequency of p1_crowns_obtained')

plt.subplot(1, 2, 2)
sns.barplot(data=unit_freq_p2_pd, x='unit', y='frequency', palette='viridis')
plt.xlabel('Unit')
plt.ylabel('Frequency')
plt.title('Top 8 Units in p2_unit1-p2_unit8 vs Frequency of p2_crowns_obtained')
plt.tight_layout()
plt.show()

crowns_columns = ["p1_crowns_obtained", "p2_crowns_obtained"]

exploded_crowns_df = spark_df.select(explode(array(*[col(column) for column in
crowns_columns])).alias("crown"))

crown_freq_df = exploded_crowns_df.groupBy("crown").count()

labels = crown_freq_df.select("crown").toPandas()["crown"]
sizes = crown_freq_df.select("count").toPandas()["count"]

plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Frequency of Unique Elements in p1_crowns_obtained and p2_crowns_obtained')
plt.tight_layout()
plt.savefig("crowns_obtained_pie_chart.png")
plt.show()

spark_df = spark_df.withColumn("crown_difference", col("p1_crowns_obtained") -
col("p2_crowns_obtained"))

spark_df = spark_df.withColumn("grouped_difference",
                               when(abs(col("crown_difference")) == 1, "1") \
                               .when(abs(col("crown_difference")) == 2, "2") \
                               .when(abs(col("crown_difference")) == 3, "3") \
                               .otherwise(col("crown_difference")))

grouped_diff_df = spark_df.groupBy("grouped_difference").count()

grouped_diff_pd = grouped_diff_df.toPandas()

plt.figure(figsize=(8, 8))
plt.pie(grouped_diff_pd["count"], labels=grouped_diff_pd["grouped_difference"],
autopct='%1.1f%%')
plt.title('Frequency of Crown Difference between p1 and p2')
plt.axis('equal')
colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightgray']

for patch, color in zip(plt.gca().patches, colors):
    patch.set_facecolor(color)
```

```
plt.tight_layout()
plt.savefig("grouped_crown_difference_pie_chart.png")
plt.show()
```

Appendix E

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, array, explode, count, sum as sum_agg
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

unit_columns = ["p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
                "p1_unit6", "p1_unit7", "p1_unit8", "p2_unit1", "p2_unit2",
                "p2_unit3", "p2_unit4", "p2_unit5", "p2_unit6", "p2_unit7",
                "p2_unit8"]

combined_units_df = spark_df.select(explode(array(*[col(column) for column in
unit_columns])).alias("unit"))

unit_freq_df = combined_units_df.groupBy("unit").count().orderBy("count", ascending=False)

unit_freq_pd = unit_freq_df.toPandas()

plt.figure(figsize=(17, 9))
sns.barplot(data=unit_freq_pd, x='unit', y='count', palette='viridis')

plt.xticks(rotation=90, ha='right')
plt.xlabel('Unit')
plt.ylabel('Frequency')
plt.title('Frequency of Unique Elements in p1_unit1 to p1_unit8 and p2_unit1 to p2_unit8')
plt.tight_layout(pad=3.0)
plt.savefig("unit_frequency_plot.png")

plt.show()

combined_units_df = spark_df.select(
    explode(array(*[col(column) for column in unit_columns])).alias("unit"),
    col("p1_crowns_obtained").alias("p1_crowns"),
    col("p2_crowns_obtained").alias("p2_crowns")
)
```

```
unit_freq_crowns_df = combined_units_df.groupBy("unit").agg(
   count("unit").alias("frequency"),
   sum_agg("p1_crowns").alias("total_p1_crowns"),
   sum_agg("p2_crowns").alias("total_p2_crowns")
)

unit_freq_crowns_pd = unit_freq_crowns_df.toPandas()

unit_freq_crowns_pd['total_crowns'] = unit_freq_crowns_pd['total_p1_crowns'] +
unit_freq_crowns_pd['total_p2_crowns']

top_20_units_pd = unit_freq_crowns_pd.nlargest(20, 'total_crowns')

plt.figure(figsize=(12, 8))
sns.scatterplot(data=top_20_units_pd, x='frequency', y='total_crowns', hue='unit',
palette='viridis', s=100)

plt.xlabel('Frequency of Units')
plt.ylabel('Total Crowns Obtained')
plt.title('Top 20 Units by Total Crowns Obtained')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig("top_20_unit_frequency_vs_crowns_scatter_plot.png")

plt.show()

combined_units_df = spark_df.select(
   explode(array(*[col(column) for column in unit_columns])).alias("unit"),
   col("p1_crowns_obtained").alias("p1_crowns"),
   col("p2_crowns_obtained").alias("p2_crowns")
)

unit_freq_crowns_df = combined_units_df.groupBy("unit").agg(
   count("unit").alias("unit_frequency"),
   sum_agg("p1_crowns").alias("total_p1_crowns"),
   sum_agg("p2_crowns").alias("total_p2_crowns")
)

unit_freq_crowns_pd = unit_freq_crowns_df.toPandas()

unit_freq_crowns_pd['total_crowns'] = unit_freq_crowns_pd['total_p1_crowns'] +
unit_freq_crowns_pd['total_p2_crowns']

plt.figure(figsize=(12, 8))
```

```
sns.scatterplot(data=unit_freq_crowns_pd, x='unit_frequency', y='total_crowns', hue='unit',
palette='viridis', s=100)

plt.xlabel('Frequency of Units')
plt.ylabel('Total Crowns Obtained')
plt.title('Frequency of Unique Elements vs. Number of Crowns Obtained')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig("unit_frequency_vs_crowns_obtained_scatter_plot.png")

plt.show()

unit_columns_p1 = ["p1_unit1", "p1_unit2", "p1_unit3", "p1_unit4", "p1_unit5",
            "p1_unit6", "p1_unit7", "p1_unit8"]
unit_columns_p2 = ["p2_unit1", "p2_unit2", "p2_unit3", "p2_unit4", "p2_unit5",
            "p2_unit6", "p2_unit7", "p2_unit8"]
crown_column_p1 = "p1_crowns_obtained"
crown_column_p2 = "p2_crowns_obtained"

unit_freq_p1 = spark_df.select(explode(array(*[col(column) for column in
unit_columns_p1])).alias("unit")) \
               .groupBy("unit").agg(count("unit").alias("frequency")) \
               .orderBy(col("frequency").desc()).limit(8)
unit_freq_p2 = spark_df.select(explode(array(*[col(column) for column in
unit_columns_p2])).alias("unit")) \
               .groupBy("unit").agg(count("unit").alias("frequency")) \
               .orderBy(col("frequency").desc()).limit(8)

crown_freq_p1 = spark_df.groupBy(crown_column_p1).count().orderBy(col("count").desc())
crown_freq_p2 = spark_df.groupBy(crown_column_p2).count().orderBy(col("count").desc())

unit_freq_p1_pd = unit_freq_p1.toPandas()
unit_freq_p2_pd = unit_freq_p2.toPandas()
crown_freq_p1_pd = crown_freq_p1.toPandas()
crown_freq_p2_pd = crown_freq_p2.toPandas()

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.barplot(data=unit_freq_p1_pd, x='unit', y='frequency', palette='viridis')
plt.xlabel('Unit')
plt.ylabel('Frequency')
plt.title('Top 8 Units in p1_unit1-p1_unit8 vs Frequency of p1_crowns_obtained')
plt.subplot(1, 2, 2)
sns.barplot(data=unit_freq_p2_pd, x='unit', y='frequency', palette='viridis')
plt.xlabel('Unit')
plt.ylabel('Frequency')
plt.title('Top 8 Units in p2_unit1-p2_unit8 vs Frequency of p2_crowns_obtained')

plt.tight_layout()
plt.show()

crowns_columns = ["p1_crowns_obtained", "p2_crowns_obtained"]

exploded_crowns_df = spark_df.select(explode(array(*[col(column) for column in
crowns_columns])).alias("crown"))

crown_freq_df = exploded_crowns_df.groupBy("crown").count()

labels = crown_freq_df.select("crown").toPandas()["crown"]
sizes = crown_freq_df.select("count").toPandas()["count"]

plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Frequency of Unique Elements in p1_crowns_obtained and p2_crowns_obtained')
plt.tight_layout()
plt.savefig("crowns_obtained_pie_chart.png")
plt.show()

spark_df = spark_df.withColumn("crown_difference", col("p1_crowns_obtained") -
col("p2_crowns_obtained"))

spark_df = spark_df.withColumn("grouped_difference",
                               when(abs(col("crown_difference")) == 1, "1") \
                               .when(abs(col("crown_difference")) == 2, "2") \
                               .when(abs(col("crown_difference")) == 3, "3") \
                               .otherwise(col("crown_difference")))

grouped_diff_df = spark_df.groupBy("grouped_difference").count()

grouped_diff_pd = grouped_diff_df.toPandas()

plt.figure(figsize=(8, 8))
plt.pie(grouped_diff_pd["count"], labels=grouped_diff_pd["grouped_difference"],
autopct='%1.1f%%')
plt.title('Frequency of Crown Difference between p1 and p2')
plt.axis('equal')
colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightgray']

for patch, color in zip(plt.gca().patches, colors):
    patch.set_facecolor(color)

plt.tight_layout()
plt.savefig("grouped_crown_difference_pie_chart.png")
plt.show()
```