FAF GAME ANALYSIS

Big Data Architecture and Governance

Group 3:

Sanika Saraff

Trupti Raut

Shiva Shankar Ananthula

Dataset Summary

FAF (Forged Alliance Forever) is a gaming community driven project designed to facilitate online play for the game Supreme Commander: Forged Alliance. It is being supported by various fan volunteers who maintain the community, add unofficial patches, maps, mods and release regular context for the FAF Supreme Commander.

FAF game has a scrapable API tracking every game ever played by a player, and a parseable binary replay of every command issued by each player in every game.

Our dataset is picked from Kaggle which is the online community for finding and publishing various datasets and the dataset was published there by scraping the game data of around 10M FAF games about who played who, on which map, what's the scoreboard etc.

There were 5 datasets related to FAF all of which were used for creating a central graph database for storing player, game, map, command statistics for analysis purposes. The datasets were extracted from the website https://www.kaggle.com/code/yanivaknin/introducing-faf-datasets/data?scriptVersionId=100892524&select=playerstats.csv, were of the comma separated values (.csv) format and were named as below-

- issue_cmds.feb22.csv
- issue_cmds.jan22.csv
- issue_cmds.mar22.csv
- issue_stats.csv
- playerstats.csv

Data Profiling Instructions

Our first step towards understanding the datasets is data profiling which is the process of examining the data available from an existing information source and collecting statistics or informative summaries about that data. We have profiled our datasets using Python, Jupyter Notebook and below are the instructions for the same:

1. Download Python (follow instructions on Download Python | Python.org)
2. Download and Install Anaconda distribution depending on your operating system (follow instructions on Anaconda | Anaconda Distribution)
3. Open Anaconda Navigator
4. Select and Install Jupyter Lab environment from the menu
5. Open the Jupyter Lab or Jupyter notebook by selecting from the menu in Anaconda navigator or running the command 'jupyter notebook' in the command prompt. This opens the Jupyter web interface

6. Install various libraries which will be required for profiling (!pip install pandas , !pip install numpy , !pip install pandas-profiling
7. Download the 5 datasets from the link https://www.kaggle.com/code/yanivaknin/introducing-faf-datasets/data?scriptVersionId=100892524&select=playerstats.csv
8. Ensure that the downloaded csv files are in the same folder as the Jupyter notebook
9. Run Profiling.ipynb script and <output>.html profiling file will be created for 5 datasets as given below in the same path

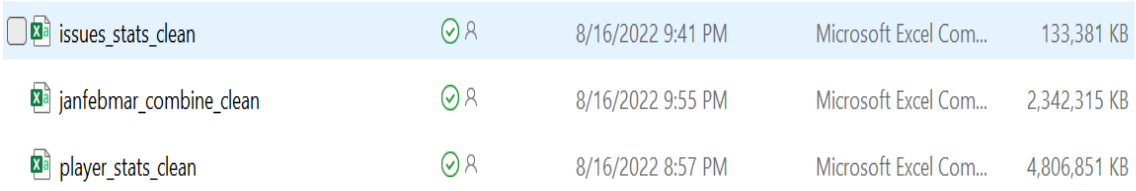
9.1.

10. Open <output>.html on browser to view data profile report

Data Wrangling/Cleansing Instructions

Our next step will be cleaning the datasets and getting them to usable formats before loading to Neo4j graph database. Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable. Detailed instructions on how the data was cleaned are mentioned in the Jupyter notebook Data Wrangling and Cleaning.ipynb and below are the instructions for using the same:

1. Download the file Data Wrangling and Cleaning.ipynb and save it in the same data path as all team 3 projects
2. We will be using Jupyter notebook for data wrangling and cleaning and installing Jupyter has been already covered in our previous Data Profiling section
3. Make sure pandas, numpy library is installed (should have been installed during data profiling)
4. Run Data Wrangling and Cleaning.ipynb script
5. Once the script is finished running, issues_stats_clean.csv , janfebmar_combine_clean.csv , player_stats_clean.csv files will be created in the same path as the Jupyter notebook



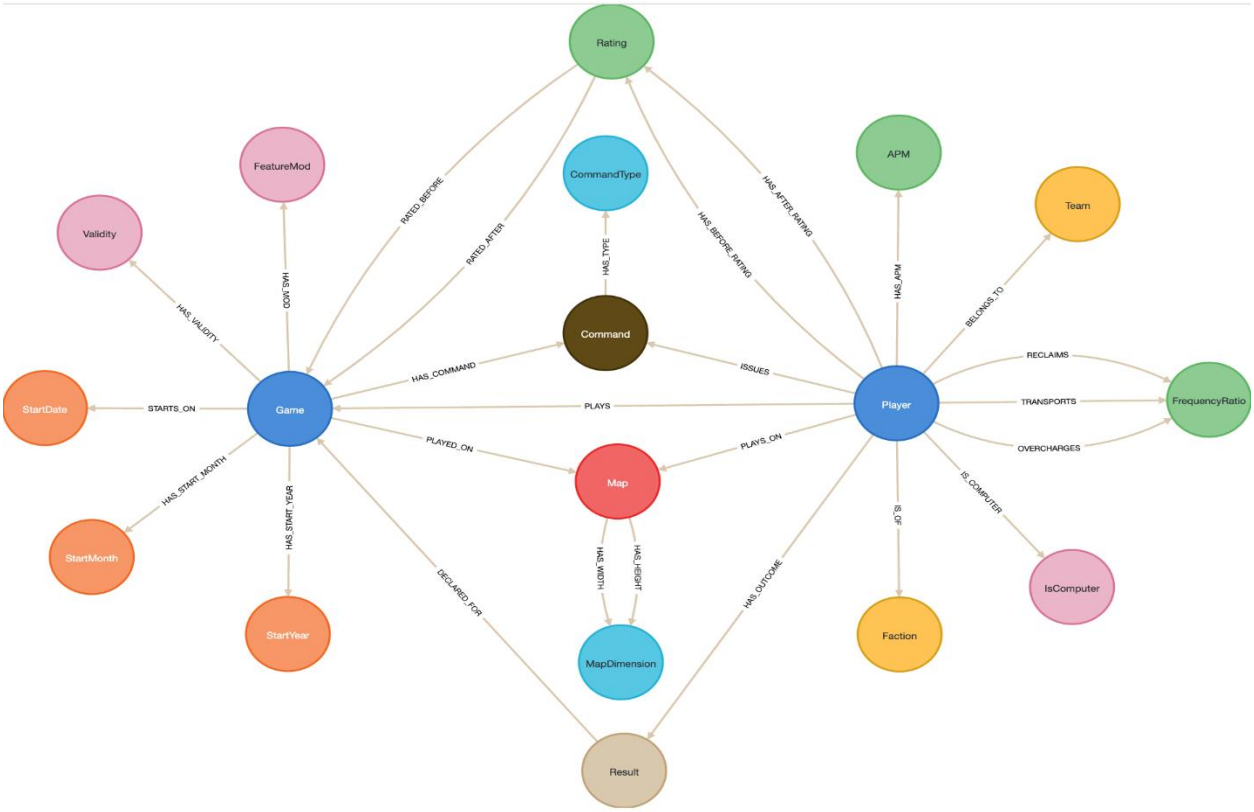
6. Use these new cleaned datasets for database/visualization

Neo4J Database Intallation Instructions:

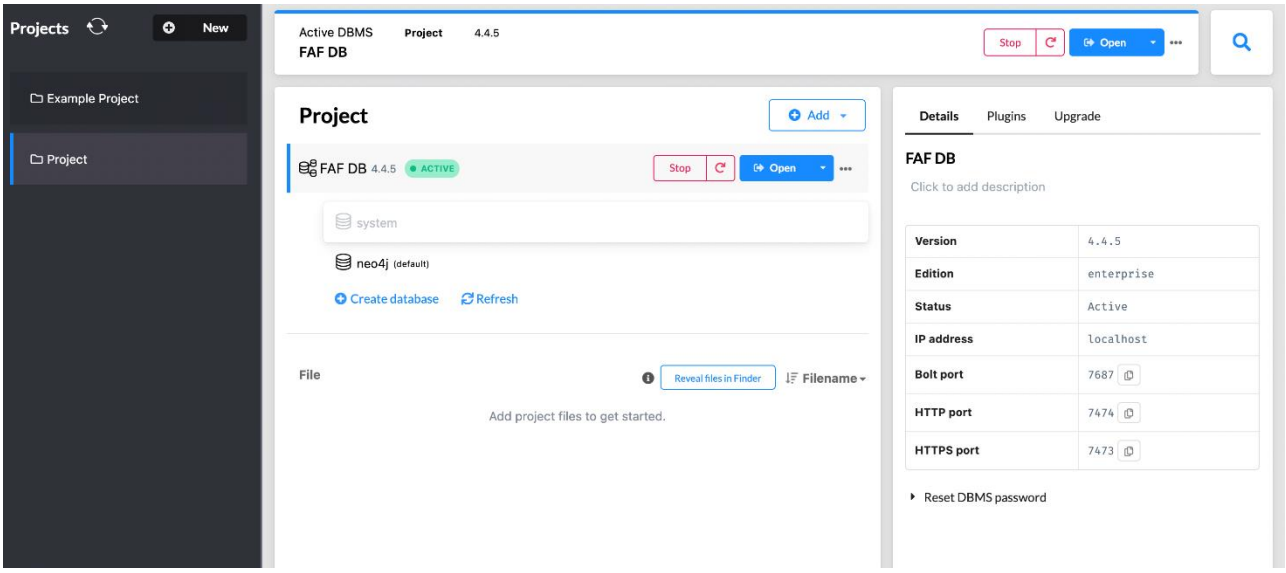
We will be using Neo4j graph database for loading all our FAF data.

1. Download and Install Neo4j Graph database using the link [Neo4j Desktop Download | Free Graph Database Download]
2. Follow the instructions for installation as per your operating system
3. After installing review the user instructions and start loading the data in Neo4j as per the loading documentation below.

Graph Database Design:



Neo4J Database Loading Instructions:



1. Create a new Database in Neo4j named FAF DB – Click on Add button under Project --> Give database name as FAF DB, username and password and select Create
2. Copy the cleaned csv files into the DBMS import folder of the database created.
3. Start the database and open the database.
4. Download the Neo4j Scripts named ‘Cypher queries.txt’ and run the queries by copying and pasting in the Neo4j browser
5. The scripts will run one by one and will take some time due to the large dataset size
6. Once the scripts have been completed running the graph database will be created

Constraint Creation:

```
// Uniqueness constraints
CREATE CONSTRAINT ON (frequencyratio:FrequencyRatio) ASSERT frequencyratio.FrequencyRatio is UNIQUE;
CREATE CONSTRAINT ON (player:Player) ASSERT player.PlayerId IS UNIQUE;
CREATE CONSTRAINT ON (game:Game) ASSERT game.GameId is UNIQUE;
CREATE CONSTRAINT ON (apm:APM) ASSERT apm.IssueAPM is UNIQUE;
CREATE CONSTRAINT ON (map:Map) ASSERT map.MapId is UNIQUE;
CREATE CONSTRAINT ON (mapdimension:MapDimension) ASSERT mapdimension.MapDimension is UNIQUE;
CREATE CONSTRAINT ON (command:Command) ASSERT command.CommandName is UNIQUE;
CREATE CONSTRAINT ON (commandtype:CommandType) ASSERT commandtype.CommandType is UNIQUE;
CREATE CONSTRAINT ON (faction:Faction) ASSERT faction.FactionName is UNIQUE;
CREATE CONSTRAINT ON (iscomputer:IsComputer) ASSERT iscomputer.IsComputer is UNIQUE;
CREATE CONSTRAINT ON (validity:Validity) ASSERT validity.Validity is UNIQUE;
CREATE CONSTRAINT ON (featuremod:FeatureMod) ASSERT featuremod.FeatureModName is UNIQUE;
CREATE CONSTRAINT ON (rating:Rating) ASSERT rating.Rating is UNIQUE;
CREATE CONSTRAINT ON (team:Team) ASSERT team.TeamId is UNIQUE;
CREATE CONSTRAINT ON (result:Result) ASSERT result.Result is UNIQUE;
CREATE CONSTRAINT ON (startdate:StartDate) ASSERT startdate.StartDate is UNIQUE;
CREATE CONSTRAINT ON (startmonth:StartMonth) ASSERT startmonth.StartMonth is UNIQUE;
CREATE CONSTRAINT ON (startyear:StartYear) ASSERT startyear.StartYear is UNIQUE;

//
```

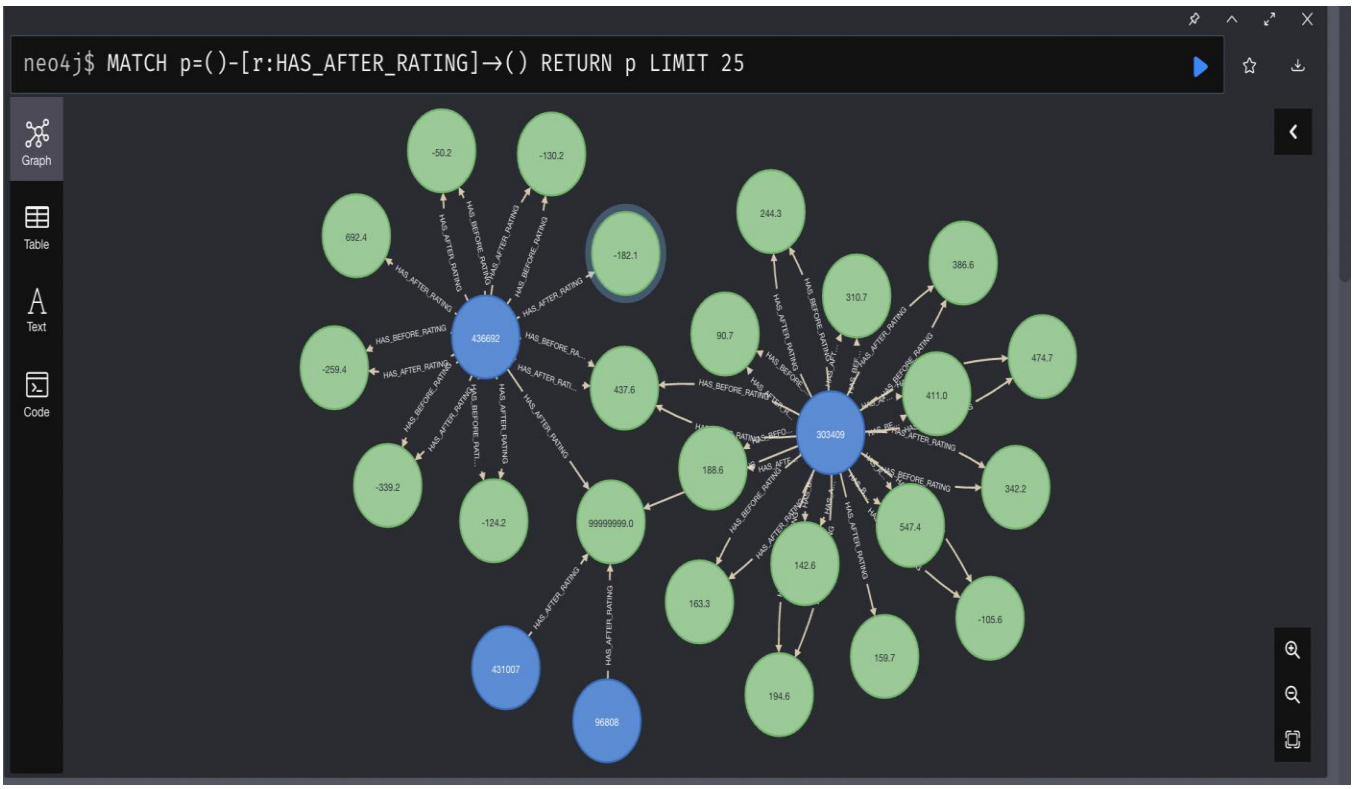
Node Creation:

```
// Create Player node

:auto USING PERIODIC COMMIT 500

LOAD CSV With HEADERS FROM 'file:///player_stats_clean.csv' AS row

MERGE

  (player:Player {PlayerId:row.player_id})

ON CREATE SET player.Login=row. player_login;

//


//Creating Game Node

:auto USING PERIODIC COMMIT 1000

LOAD CSV With HEADERS FROM 'file:///player_stats_clean.csv' AS row

MERGE (game:Game {GameId:row.game_id});
//

//Creating Map Node

:auto USING PERIODIC COMMIT 1000

LOAD CSV With HEADERS FROM 'file:///player_stats_clean.csv' AS row

MERGE (map:Map {MapId:row.map_id})

    ON CREATE SET map.MapName=row.map_name;

//
```


neo4j$
neo4j$ MATCH (n:Command) RETURN n LIMIT 25
Graph
Table
Text
Code
AssistMove
Nuke
Sacrifice
KillSelf
TransportRever...
Move
Attack
Patrol
Stop

Relationships Establishing:

```
Relations
//  Player plays game relation
:auto USING PERIODIC COMMIT 500
LOAD CSV With HEADERS FROM 'file:///player_stats_clean.csv' AS row
     MATCH (player:Player {PlayerId:row.player_id})
     MATCH (game:Game {GameId:row.game_id})
     MERGE (player)-[:PLAYS]->(game);
//

//  Player issues command relation
:auto USING PERIODIC COMMIT 500
LOAD CSV With HEADERS FROM 'file:///janfebmar_combine_clean.csv' AS row
     MATCH (player:Player {PlayerId:row.player_id})
     MATCH (command:Command {CommandName:row.command_name})
     MERGE (player)-[:ISSUES]->(command);
//
```



Neo4J Python Data Visualization Instructions:

We will be creating FAF data visualizations using Python's Matplotlib library using Jupyter notebook.

1. Download and install Python, Anaconda, Jupyter notebook if not already downloaded and follow instructions in the Data Profiling steps
2. Open jupyter notebook by running command jupyter notebook on Anaconda command prompt
3. Install the necessary python packages as below using command !pip install <package_name> in Jupyter notebook
    a. Pandas
    b. Numpy
    c. Matplotlib
    d. Neo4j
4. Navigate to the folder containing dataset for team 3 and open Data Visualizations.ipynb Ensure all files are in the same folder.
5. We have made a connection to Neo4j FAF DB to query the database using the following line of code. Make sure you enter your own Neo4j credentials previously set on your local laptop in place of the url, user and pwd fields in the second line below

In [4]:
```
from neo4j import GraphDatabase
class Neo4jConnection:

    def __init__(self, uri, user, pwd):
        self.__uri = uri
        self.__user = user
        self.__pwd = pwd
        self.__driver = None
        try:
            self.__driver = GraphDatabase.driver(self.__uri, auth=(self.__user, self.__pwd))
        except Exception as e:
            print("Failed to create the driver:", e)

    def close(self):
        if self.__driver is not None:
            self.__driver.close()

    def query(self, query, db=None):
        assert self.__driver is not None, "Driver not initialized!"
        session = None
        response = None
        try:
            session = self.__driver.session(database=db) if db is not None else self.__driver.session()
            response = list(session.run(query))
        except Exception as e:
            print("Query failed:", e)
        finally:
            if session is not None:
                session.close()
        return response
```

In [5]:
```
conn = Neo4jConnection(uri="bolt://localhost:7687", user="neo4j", pwd="root")
```

6. Run the code
7. Wait until code fully loads and IP address appears at the bottom of the page
8. Navigate to IP address and wait for dashboard to fully load
9. Visualizations are displayed

Metadata:

We have extracted the technical meta data for our database from Neo4j by running the below commands in the Neo4j browser. The business metadata has also been explained below.

- CALL apoc.meta.data()
- CALL apoc.meta.schema() YIELD value as schemaMap

```
UNWIND keys(schemaMap) as label

WITH label, schemaMap[label] as data

WHERE data.type = "node"

UNWIND keys(data.properties) as property

WITH label, property, data.properties[property] as propData

RETURN label,

property,

propData.type as type,

propData.indexed as isIndexed,

propData.unique as uniqueConstraint,

propData.existence as existenceConstraint
```

Technical Metadata:

Detailed technical metadata has been given in the files named metadata.xlsx

label	property	type	isIndexed	Unique Constraint	Existence Constraint
Rating	Rating	FLOAT	TRUE	TRUE	FALSE
Result	Result	STRING	TRUE	TRUE	FALSE
StartDate	Startdate	STRING	FALSE	FALSE	FALSE
StartMonth	StartMonth	STRING	TRUE	TRUE	FALSE
CommandType	CommandType	STRING	TRUE	TRUE	FALSE
IsComputer	IsComputer	STRING	TRUE	TRUE	FALSE
StartYear	StartYear	INTEGER	TRUE	TRUE	FALSE
FeatureMod	FeatureModName	STRING	TRUE	TRUE	FALSE
Team	TeamId	INTEGER	TRUE	TRUE	FALSE
Faction	FactionName	STRING	TRUE	TRUE	FALSE
FrequencyRatio	FrequencyRatio	STRING	TRUE	TRUE	FALSE

Player	PlayerId	INTEGER	TRUE	TRUE	FALSE
Player	Login	STRING	FALSE	FALSE	FALSE
MapDimension	MapDimension	INTEGER	TRUE	TRUE	FALSE
Validity	Validity	STRING	TRUE	TRUE	FALSE
Game	GameId	STRING	TRUE	TRUE	FALSE
Command	CommandName	STRING	TRUE	TRUE	FALSE
Map	MapId	INTEGER	TRUE	TRUE	FALSE
Map	MapName	STRING	FALSE	FALSE	FALSE
APM	IssueAPM	STRING	TRUE	TRUE	FALSE

Business Metadata:

From a business perspective, the dataset and visualizations helps in understanding the FAF game related data from a high level, where we could draw conclusions on different metrics for the dataset as follows:

1. Players with highest number of games played.
2. Most popular maps where the players have played, with this we could make further more enhancements to the maps to increase the players.
3. Determine the most frequently used commands.
4. Determine the various statistics related data like Transport ratio, Over change ratio, Actions per minute etc.

attribute	**business term**	**description**
Rating	after_rating	The rating of this player, after the game
Rating	before_rating	The rating of this player, at the start of the game
Result	player_result	Final result of the player in the game
Startdate	game_start_time	The database recorded date of the game's start
CommandType	type	The type of command given; there are many command types in the game
IsComputer	player_is_computer	Whether or not the player was a computer AI.
FeatureModName	game_featuremod_name	It is the feature modification name of the game

TeamId	player_team	The team ID of the player
FactionName	player_faction | faction	Which faction ("race" or "culture") the player played as.
Frequency Ratio	reclaim_ratio	What proportion of issue commands were `reclaim` commands (obtain excess resources left on the map).
Frequency Ratio	overcharge_ratio	What proportion of issue commands were "Overcharge" commands (https://supcom.fandom.com/wiki/Overcharge_Cannon)
FrequencyRatio	transport_ratio	What proportion of issue commands were `transport` commands (use transport units to load/unload/ferry etc
PlayerId	player_id | faf_player_id	The ID of the player in FAF
Login	player_login | login	Players login ID
MapDimension	map_width	It is a dimension, width of map.
MapDimension	map_height	It is a dimension, height of map.
Validity	game_validity	Validity of the games player. It alter players rating in the ladder.
GameId	id | game_id	ID of the "Game" model in the API
CommandName	command_name	The name of the specific `issue` or `factory_issue` command that was given.
MapId	map_id	ID of Map in FAF
MapName	map_name	Human-readable name of the map in the game
IssueAPM	issue_apm	Actions Per Minute for `issue` type commands (most common command type, telling game units what to do).

Data Challenges encountered and how we resolved them?

Challenges	Mitigation
1. Issue with profiling on the local server due to huge size of the dataset.	Used different machine with a better configuration to perform profiling
2. Lack of knowledge about FAF Supreme Commander game	Researched thoroughly about the FAF game commands and rules.
3. Command Ratio related columns are highly skewed and when observing distribution.	Classified the ratio data into categories.

4. Incorrect datatypes in the neo4j while loading leading to incorrect minimum and maximum values of the fields.	Corrected the datatypes in the database after loading.
5. Faced Java Heap Space memory exception while uploading the data related to the game node due to the data being large.	Updated the neo4j memory configuration by increasing memory allocated.

Risks and Issues:

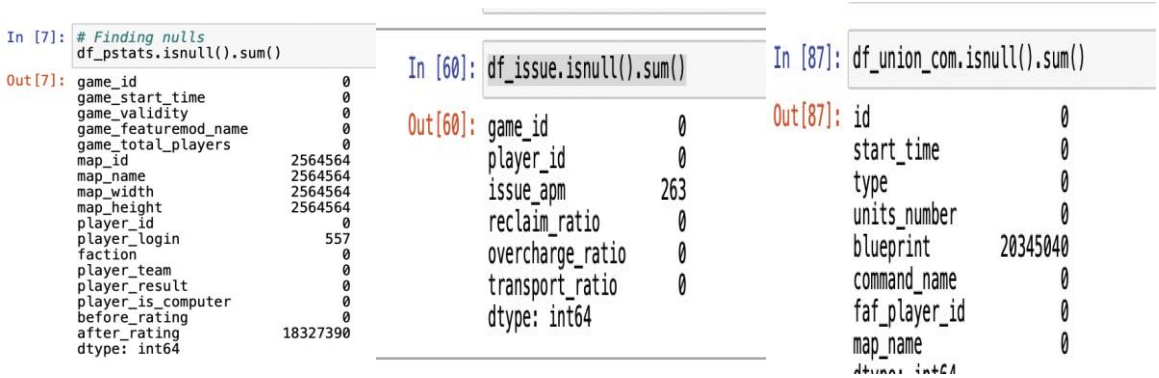
Type	Description	Mitigation
Risk	Less knowledge of cypher (Neo 4J)	Watched and learnt through online tutorials
Issue	Lack of Knowledge about the FAF game among team members which could lead to incorrect analysis	Performed prior research and understanding of FAF game dataset.
Issue	Metadata were vague	Researched and tried to make sense of the data and relation between entities
Issue	Dataset is too large for profiling and loading into database.	Tried loading data on a better and high- performance system

System Integration and User Acceptance Testing

Test 1 : Checking Count of Null Values Before and After Loading

There were 5 datasets (playerstats – 1 file, issuestats – 1 file, command stats – 3 files combined into 1 file) in all which were used as data sources. The number of null values for every dataset was handled by replacing them with a standard value defined by the team.

Before Handling Null Values



After Handling Null Values

Test 2 : Checking unique Counts for Nodes before and after loading

From the screenshots below which were taken from our Profiling files and Neo4j after running queries, it is evident that the number of distinct counts for all the nodes remains same before and after the data load.

Run in Neo4j Browser: CALL apoc.meta.stats() YIELD labels RETURN labels

Neo4j

```
CALL apoc.meta.stats() YIELD labels
RETURN labels
```

labels


```
{
  "FrequencyRatio": 10,
  "Player": 270421,
  "Rating": 33111,
  "MapDimension": 12,
  "Result": 5,
  "StartDate": 3781,
  "Validity": 23,
  "Game": 10811711,
  "StartMonth": 12,
  "Command": 31,
  "CommandType": 2,
  "IsComputer": 2,
  "StartYear": 11,
  "FeatureMod": 15,
  "Team": 16,
  "Map": 19025,
  "APM": 11,
  "Faction": 5
}
```


Other unique values are available in the profiling files in the document. Please refer them for data before load.

Unique Values before load

game_id
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	10811711	Minimum	535
Distinct (%)	33.6%	Maximum	17405720

map_id
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	19024	Minimum	1
Distinct (%)	0.1%	Maximum	22655

player_id
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	270421	Minimum	5
Distinct (%)	0.8%	Maximum	449760

game_validity

Distinct	23

game_featuremod_name

Distinct	15

player_faction

Distinct	5

Test 3 : Checking minimum and maximum counts for Nodes before and after loading

The minimum and maximum counts for all the nodes before and after data loading matched exactly.

```
MATCH (player:Player)
RETURN min(player.PlayerId) AS min, max(player.PlayerId) AS max
```

	min	max
1	5	449760

player_id
Real number (ℝ≥0)

Distinct	270421	Minimum	5
Distinct (%)	0.8%	Maximum	449760

```
MATCH (map:Map)
RETURN min(map.MapId) AS min, max(map.MapId) AS max
```

	min	max
1	-1	22655

```
In [62]: #Testing map id min max
         df_pstats['map_id'].min()
Out[62]: -1.0

In [63]: #Testing map id min max
         df_pstats['map_id'].max()
Out[63]: 22655.0
```

game_id
Real number (ℝ≥0)

Distinct	10811711	Minimum	535
Distinct (%)	33.6%	Maximum	17405720

Test 4 : Checking Neo4j data and before load data

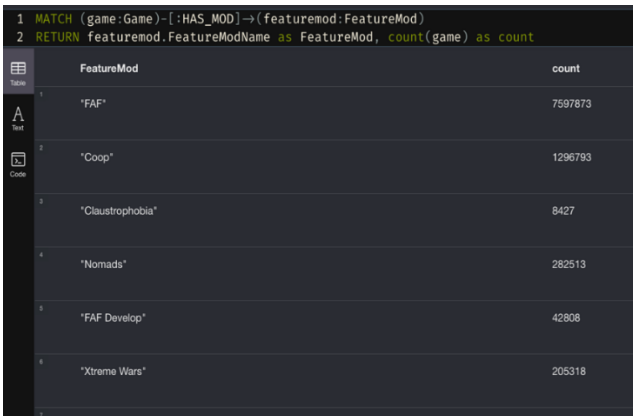
Run the Neo4j queries given before every screenshot in the Neo4j browser and then run the Python testing only cells for before load tests in the Data Wrangling and Cleaning.ipynb notebook(Make sure to follow installation steps in Data Profiling step). The test cases have passed as the values of the queries below before data load and after data load are same.

A : Count of games as per featuremod

Neo4j after load

MATCH (game:Game)-[:HAS_MOD]->(featuremod:FeatureMod)

RETURN featuremod.FeatureModName as FeatureMod, count(game) as count;



```
1 MATCH (game:Game)-[:HAS_MOD]→(featuremod:FeatureMod)
2 RETURN featuremod.FeatureModName as FeatureMod, count(game) as count
```

	FeatureMod	count
1	"FAF"	7597873
2	"Coop"	1296793
3	"Claustrophobia"	8427
4	"Nomads"	282513
5	"FAF Develop"	42808
6	"Xtreme Wars"	205318

Python test before load

```
In [57]: test3 = t_game.sort_values(['game_id','game_featuremod_name'],ascending=True).groupby('game_featuremod_name').
```

```
In [58]: test3
```

Out[58]:

	game_featuremod_name	game_id
4	FAF	7597873
1	Coop	1296793
9	Ladder1v1	1144443
11	Nomads	282513
14	Xtreme Wars	205318
5	FAF Beta Balance	105633
12	Phantom-X	66335
6	FAF Develop	42808
3	Equilibrium	26924
8	LABwars	12826
10	Murder Party	11824
0	Claustrophobia	8427
7	King of the Hill	7025
13	Vanilla	1714
2	Diamond	1255

B : Count of distinct players who are computers and who aren't

Neo4j After loading

MATCH (player:Player)-[:IS_COMPUTER]->(iscomputer:IsComputer)

RETURN iscomputer.IsComputer as IsCompurer, count(player) as count;

```
1 MATCH (player:Player)-[:IS_COMPUTER]→(iscomputer:IsComputer)
2 RETURN iscomputer.IsComputer as IsCompurer, count(player) as count
```

Table / Text / Code

	IsCompurer	count
1	"False"	270420
2	"True"	13

Python before Loading

```
In [52]: test2 = t2.sort_values(['player_id','player_is_computer'],ascending=True).groupby('player_is_computer').count(
In [53]: test2
```

Out[53]:

	player_is_computer	player_id
0	False	270420
1	True	13

C : Count of distinct players who played for a team

Neo4j After Loading

MATCH (player:Player)-[:BELONGS_TO]->(team:Team)

RETURN team.TeamId AS team, COUNT(player) AS count_player ORDER BY count_player DESC;

```
1 MATCH (player:Player)-[:BELONGS_TO]→(team:Team)
2 RETURN team.TeamId AS team, COUNT(player) AS count_player ORDER BY count_player DESC
```

Table / Text / Code

	team	count_player
1	1	218273
2	2	210786
3	3	134163
4	0	49824
5	4	36995
6	5	26537

Python before loading

```
In [47]: t1 = t.sort_values(['player_id','player_team'],ascending=True).groupby('player_team').count().reset_index().so

In [48]: t1
```

Out[48]:

	player_team	player_id
2	1	218273
3	2	210786
4	3	134163
1	0	49824
5	4	36995
6	5	26537
7	6	21845
8	7	20172
9	8	4974
10	9	3887
11	10	1148
12	11	269
13	12	126
0	-1	8
14	13	8
15	14	1

End Users for Dashboard: Community members of the FAF Game and Data Analysts at FAF Company

- Which Columns are used for dimensions and columns that are used for measurement.

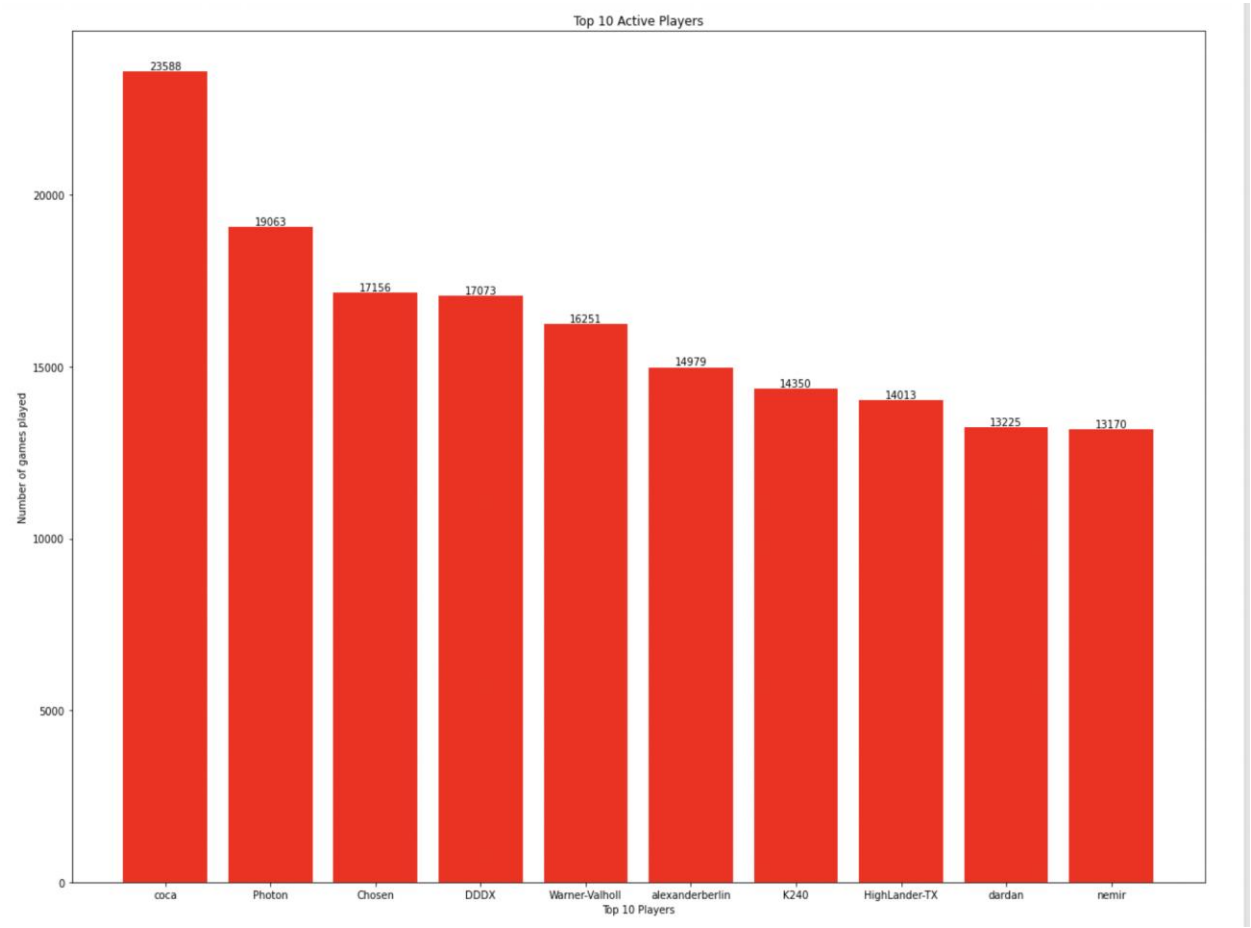
Dimensions : Player , Game, Map, Command, Faction, FeatureMod,

Measures : Before Rating, After Rating, IssueAPM, Dates, Frequency Ratios

- How would you generate new dimensions?
    - Using IssueAPM and dividing it into different categories as per the value, we can generate a new dimension for category of IssueAPM.
    - Using Overcharge ratio, transport ratio, reclaim ratio and dividing it into different categories as per the value, we can generate new dimensions for frequency category of these dimensions
- Who would use this dashboard and how do they benefit from your dashboard?
    - Members of the FAF Game community would need to analyze the data being generated to determine various KPI's for them to develop new enhancements to the multiplayer game.
    - Players of the FAF game who follow this community can also use these dashboards to compare their game with other top players and improve their gameplay.
- What value would be generated using this dashboard?
    - The data such as the top 10 popular maps played would be more useful to determine the nature of maps the user's are more inclined to and these are the areas where the developers could capitalize on business enhancements. And also new players would get to know which map is more popular.
    - New players can go through the stats of the most games played players to understand their game and instigate their statistics.
    - The end user's would be able to fetch the most frequently used commands in the game across the dataset to figure out the best command.
    - The classification of the player statistics would make it easier to find various ratio's with respect to player.
    - These visuals will assist them in identifying and classifying games and player stats for further analysis.

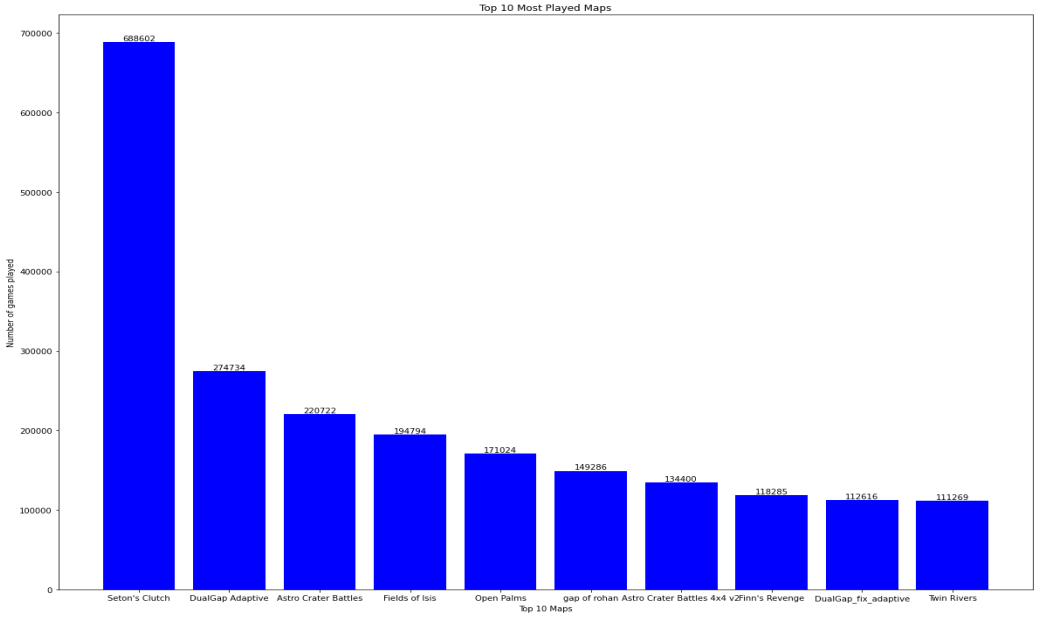
Visualization 1 : 10 Players who played the most number of games

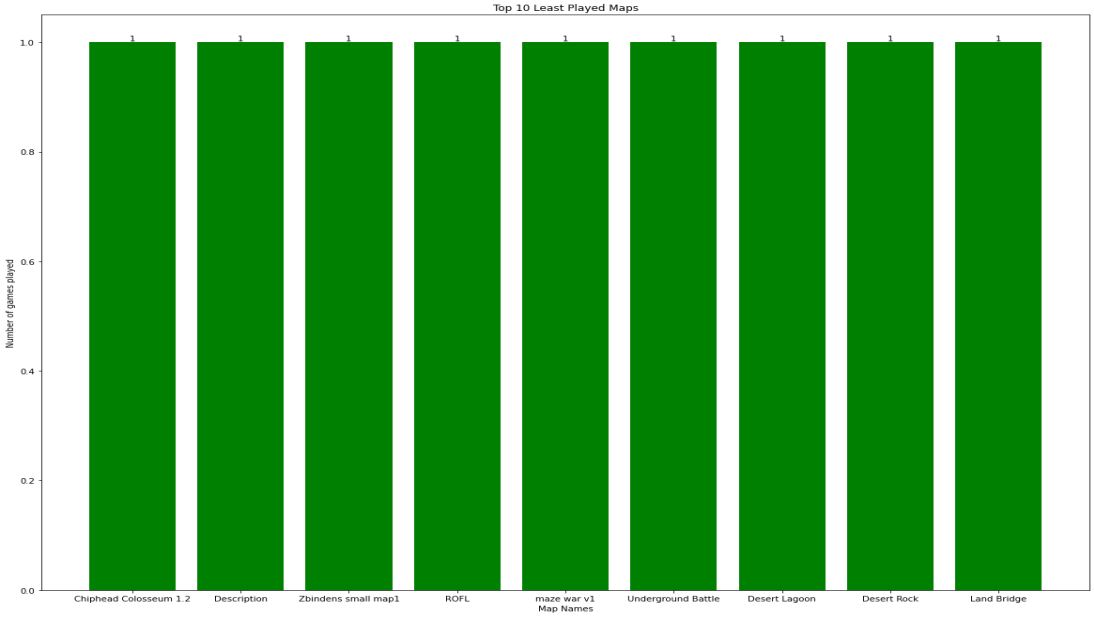
This bar graph gives the users an idea of who played the highest number of FAF games till date. It depicts the players who are most actively playing the game and will be useful for the other players who wish to improve their game as they can investigate the statistics of these active players and understand their gameplay. This graph can also be useful for the community members at FAF to determine who are the highly engaged players and further they can analyze what type of games/maps/commands are being used by them to enhance the future modifications for games based on these. We can see that the player with login name 'coca' has played highest number of games which is 23,588 till date which is June 2022.



Visualization 2 : Top 9 Most Played Maps

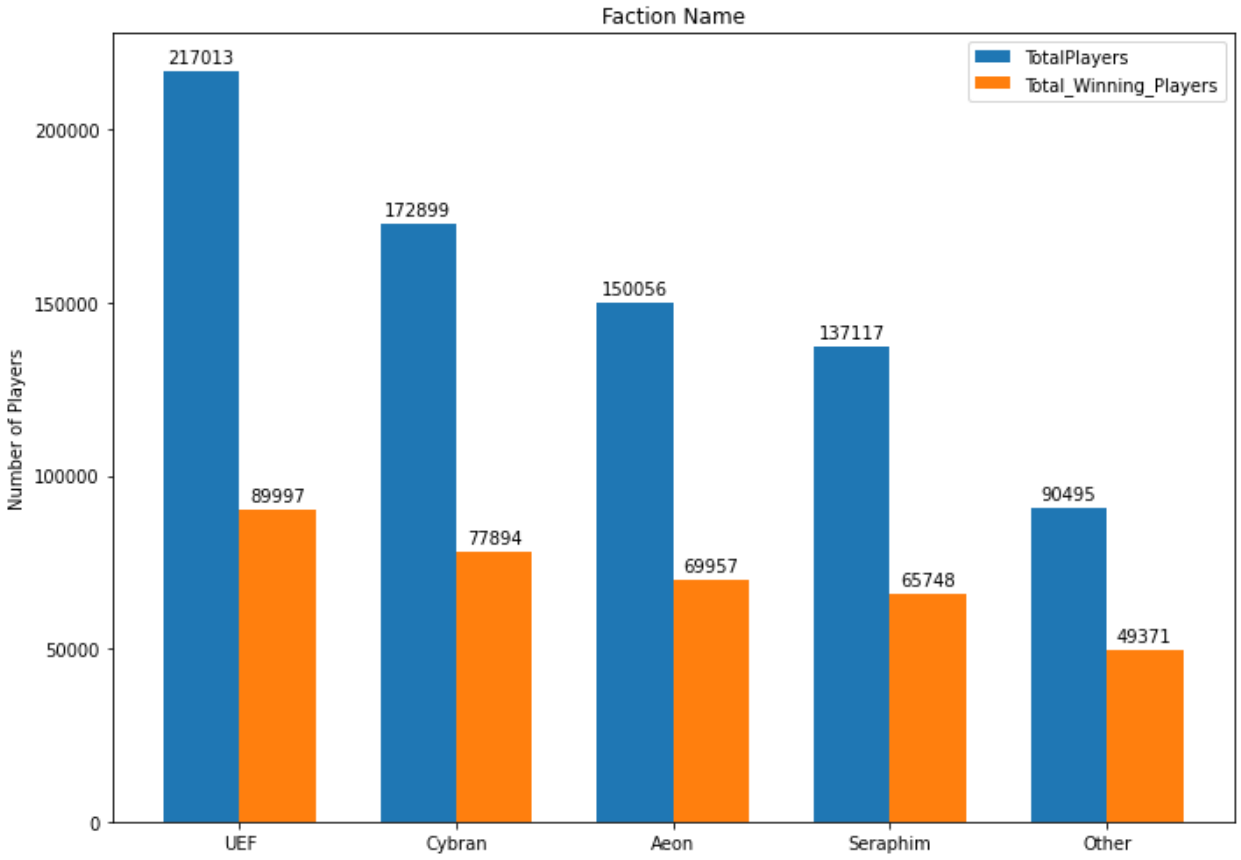
This bar graph shows us the most popular maps in FAF. These maps out of the 19025 available maps in FAF have been played on the most by players. We can see that the Secton's Clutch was played in 688,602 games till date, and it tops as one of the most popular maps with a huge margin from the rest of the maps. This data can be useful for players in selecting the maps to play on as it helps them understand which map is the most trending one. The community developers can also use this information to understand which map performs the best and the least and this information can be used to improve or upgrade existing maps accordingly.





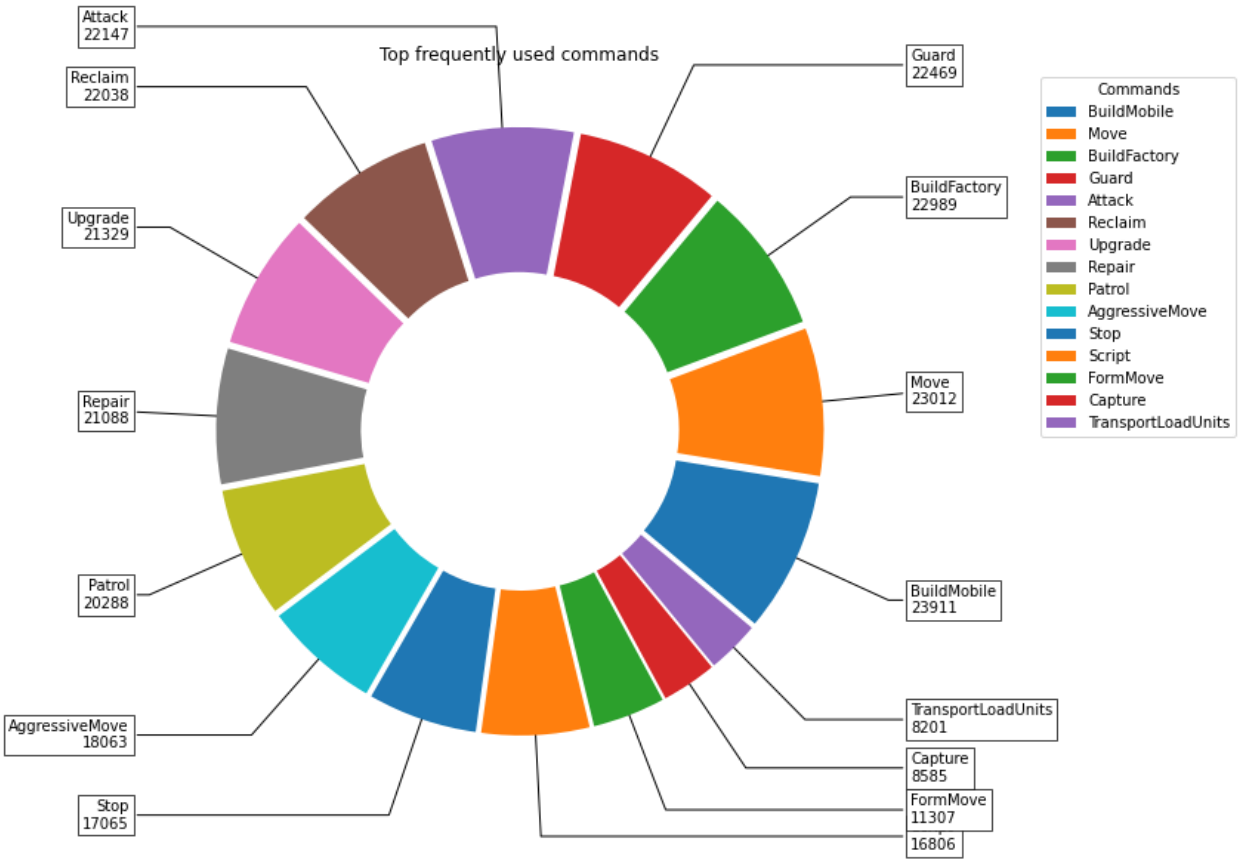
Visualization 3 : Player Statistics by Faction

The below bar graph helps us understand how many players played as various factions out of the total available factions. Further it also tells us how many players out of the total won the game by playing as that faction. Faction is like a race/ military group which the player can choose to play during the game. Every faction has different number of military units, weapons to choose from and the below information will be useful for players to understand what number of players were using a particular faction and how many out of them won the game to choose their faction before playing. This data can also be used by the community members to decide which faction is a winning faction and what utilities need to be added to the underperforming factions to improve the game and increase the difficulty.



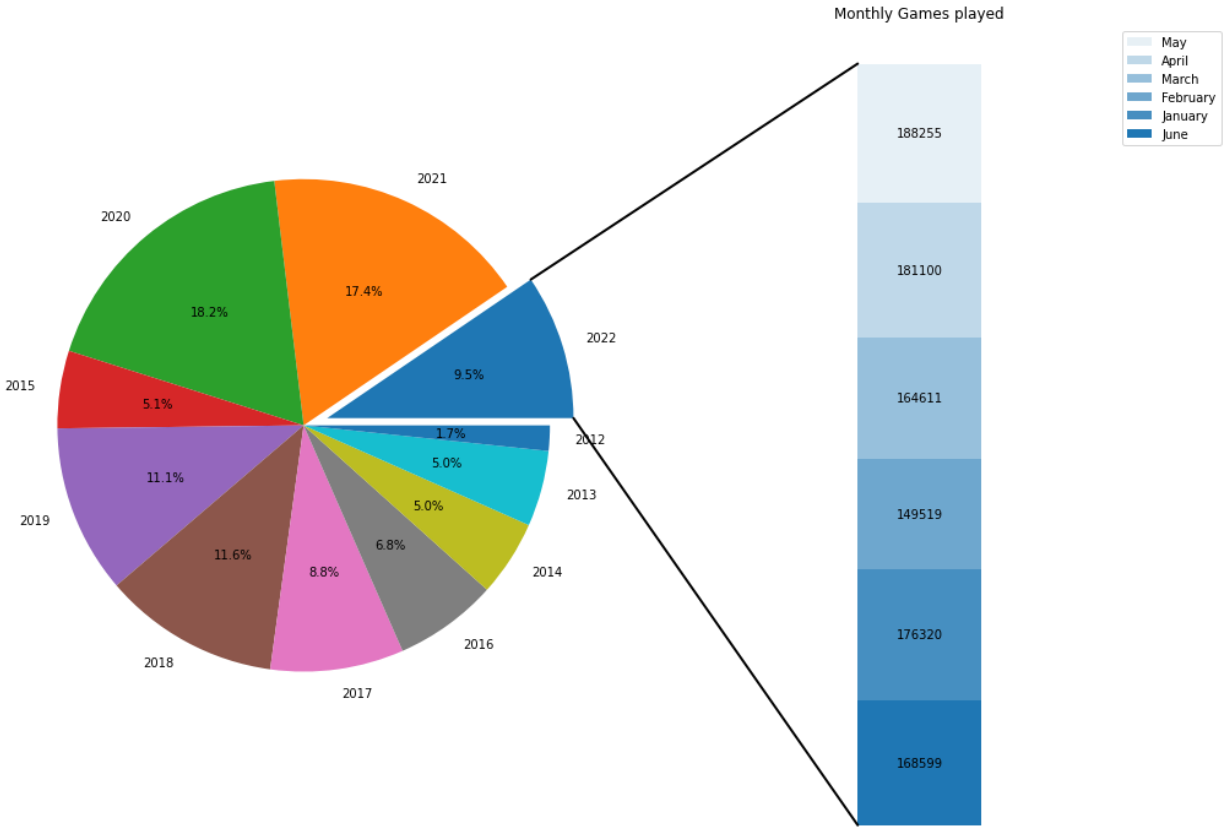
Visualization 4 : Most Frequently Used Commands by Players

This donut chart helps us understand the frequency of use of different types of commands by players. The most used commands out of the available set is BuildMobile. This data can be used by the players to understand what commands are popularly used in a game and compare their command use with the rest of the community to improve their game.



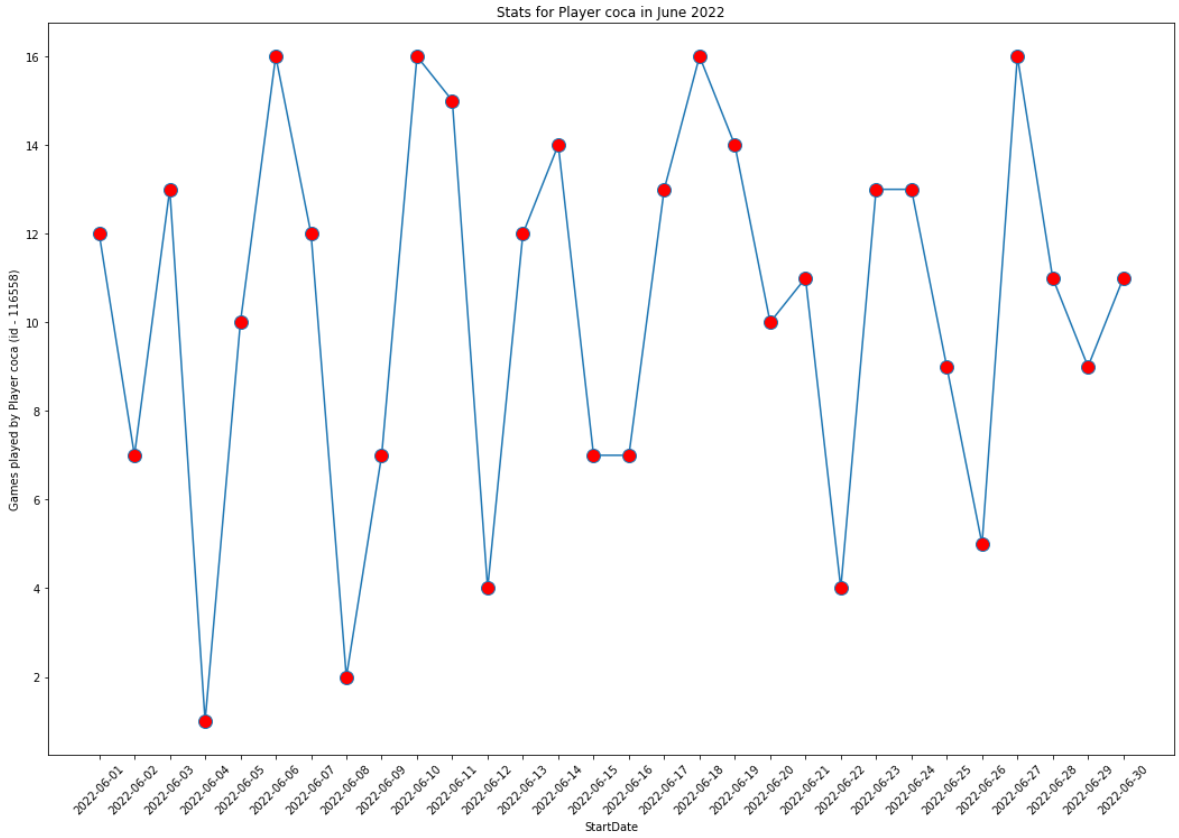
Visualization 5 : Games Played by Year and Month

This pie chart shows what percent of the total games played till date was played in every year starting 2012 to 2022. Here, we have exploded the percentage of games for 2022 year to explain the number of games played in detail. The bar on the right depicts the number of games played in every month of 2022. This graph is useful for the community members to find in which year more games were played and shows the distribution of games played by months for the most recent year. This can be used in deciding how the game is performing overall year after year and which year had the maximum games played. Further, this information can be used to find out what modifications were prevalent in that year which lead to more number of players playing the game.



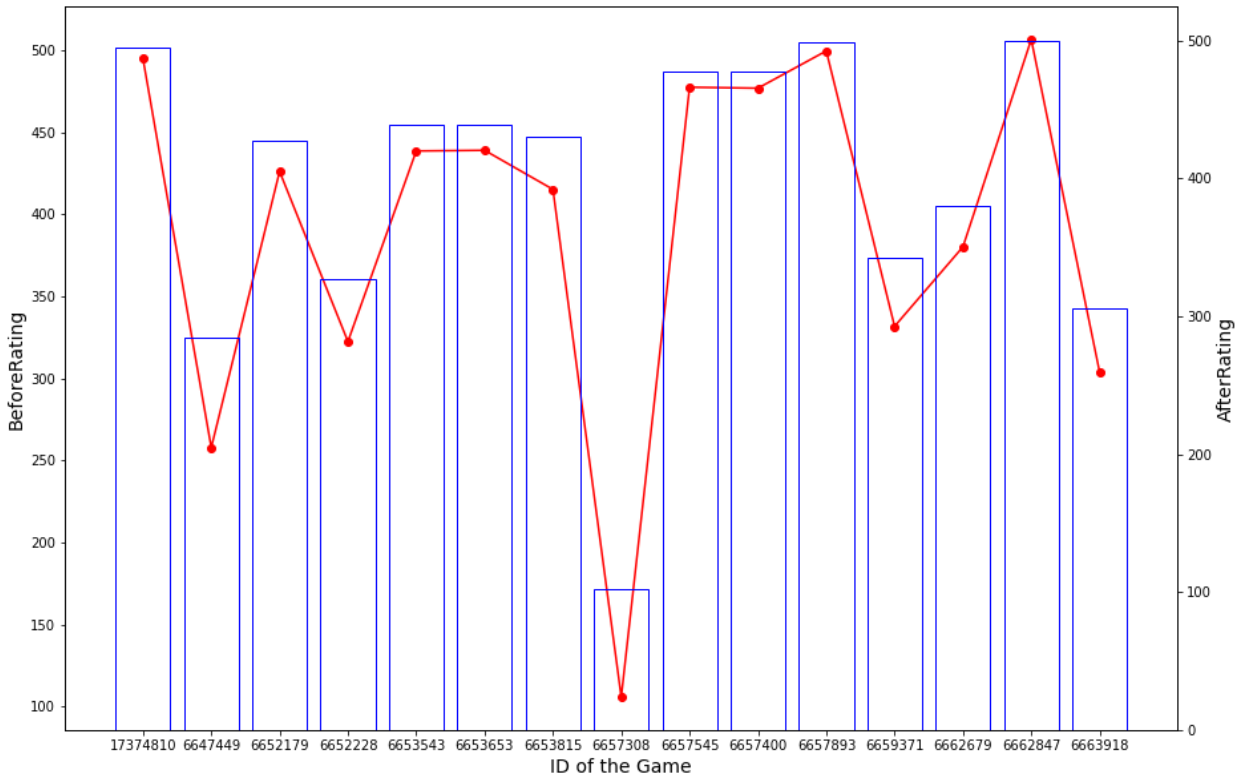
Visualization 6 : Games Played by Player Coca In the Month of June

This time series graph shows the total number of games played by the player with login id coca daily in the month of June 2022. We chose this player because they have played the greatest number of games to date. This visualization will be useful for the player to understand his statistics and improve his gameplay. You can also change the attribute PlayerId in the code to look for a particular player's statistics. This can be used to track the player's activity and in future compare it's performance.



Visualization 7 : Ratings of Player Coca In the Month of June

This bar-line graph is helpful for a player to understand their ratings before and after a particular game they played and how it has improved or reduced over different games played. We selected the player with login coca here whose before and after ratings have been displayed for different games. You can also change the PlayerId and GameId in the code to view details for the supplied fields. The players can also check other player ratings to compare with their own for improving their game. The community members can also get an insight from this graph as they can track the before and after ratings for different players to understand the difficulty level of various games.



Visualization 8 : Range of Actions per minute of the players

This horizontal bar diagram is for the command type name issue and shows the actions taken by total players per minute. This has been divided into categories and we can see that most players have an APM between 0 to 25. This data can be useful for the players to determine how many issue command actions are being done by other players and how much they need to improve on their speed of using this command to win.