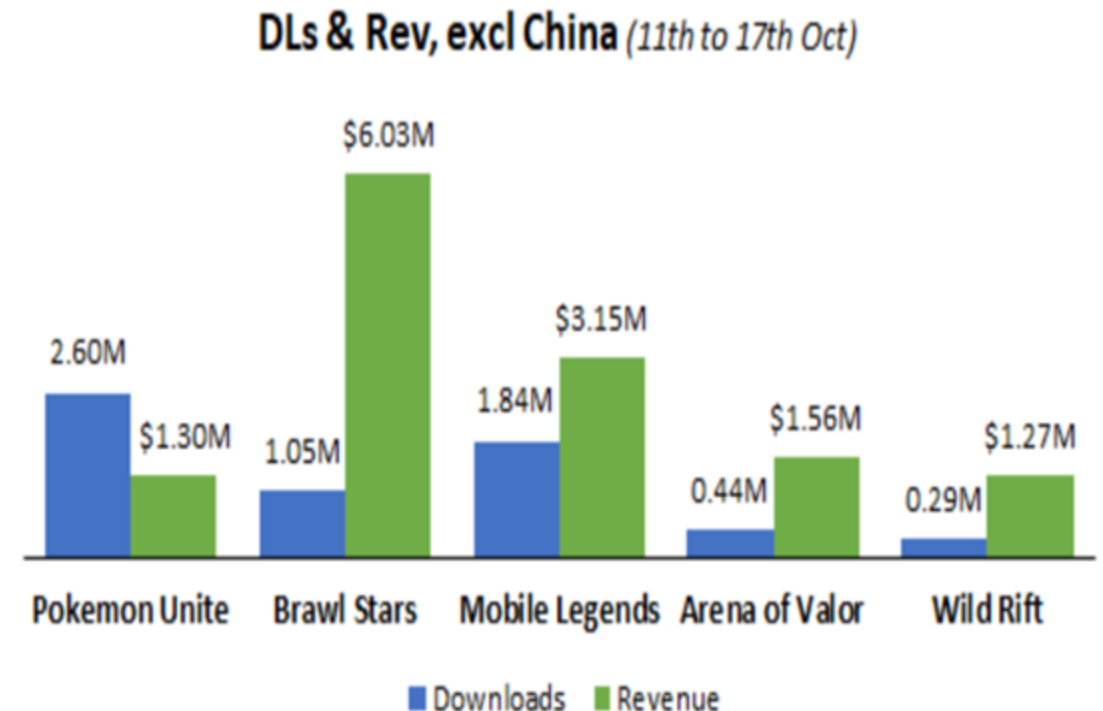
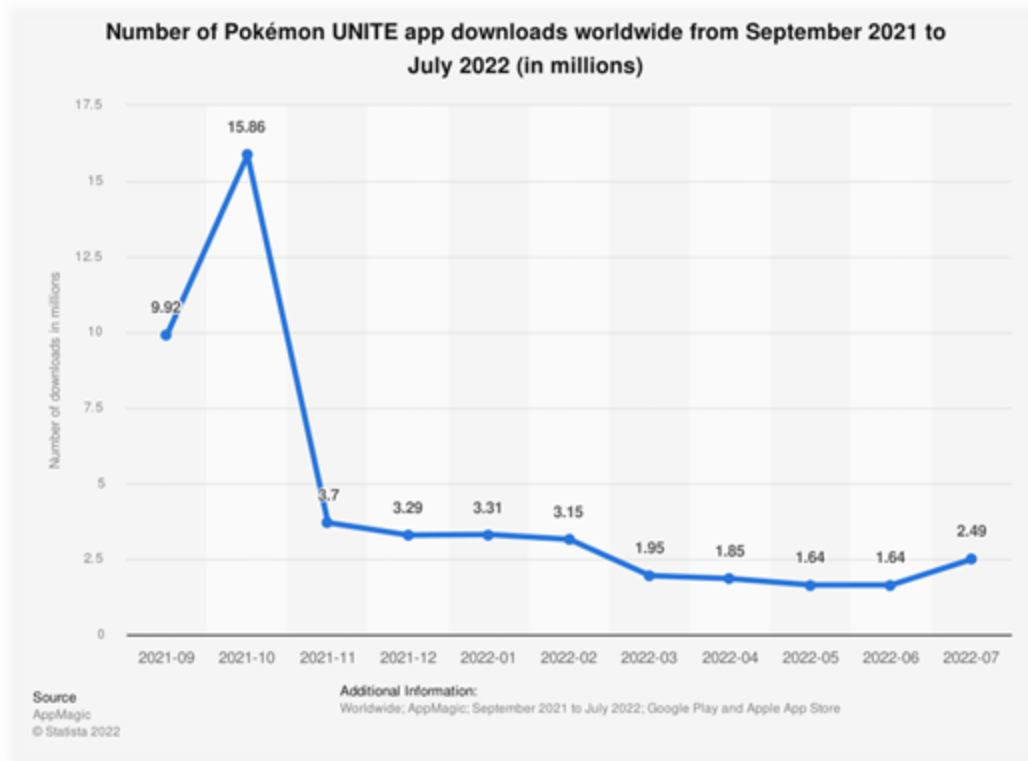


Building & Analyzing Pokémon UNITE's Database

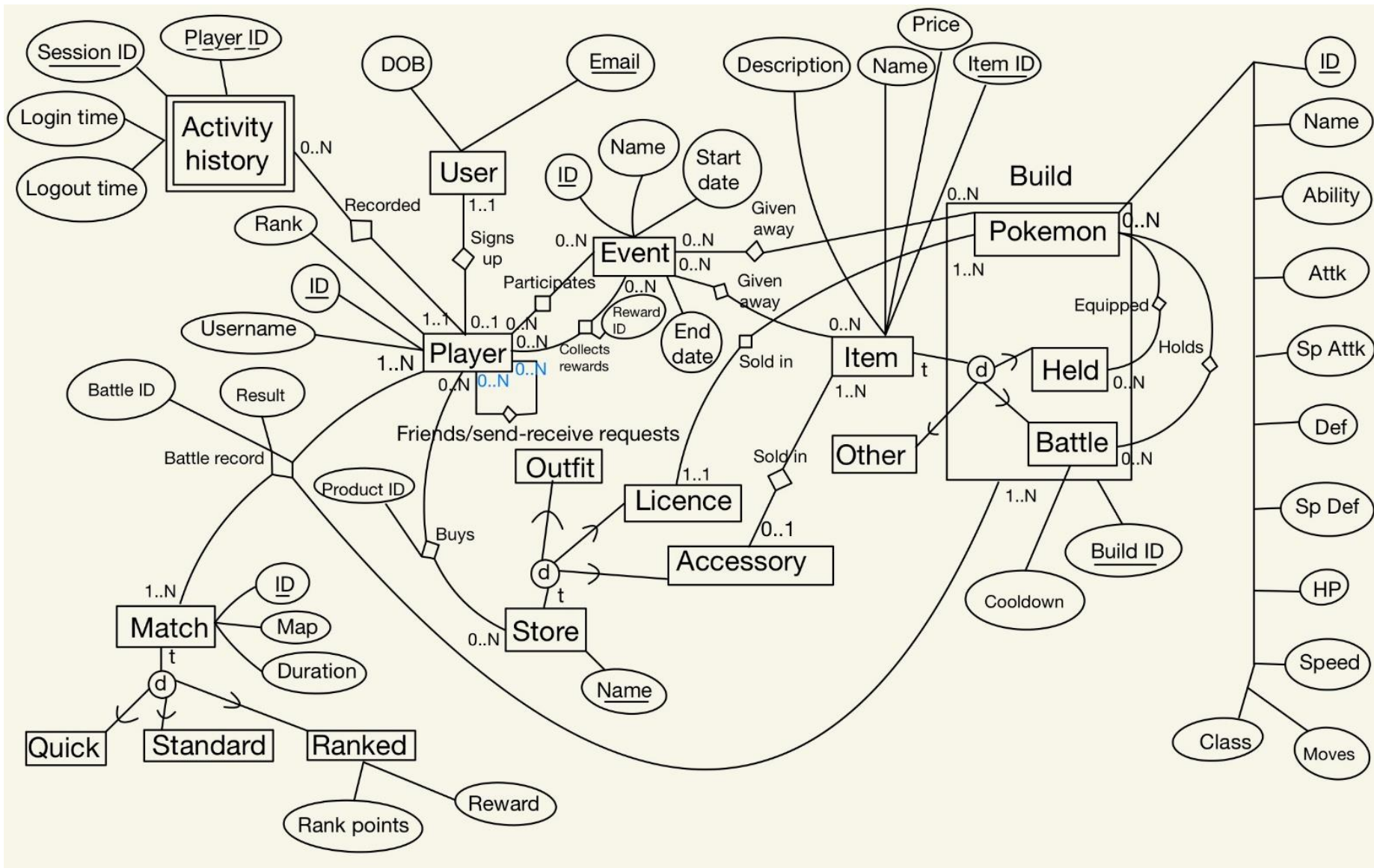


Problem Statement

The Pokémon UNITE team invests thousands of dollars in building and maintaining a free-to-play game containing millions of users. **Poor customer experience leads to low retention rates and lower monetization opportunities, negatively impacting the business' return on investment (ROI) and profit margins.** A successful solution would be to continuously improve the game mechanics in subsequent patch updates by analyzing game data and player behavior.



Conceptual Model



Scope of Analytics

Activity History (Session ID, Player ID, Login Time, Logout Time), User (Email, DOB), Player (ID, Rank, *Email*)

- Can be used to track retention rates and user activity
- Analytics on player data can be utilized to target certain cohorts of customers to spend more time on the platform for better monetization opportunities & customer retention

Battle Record (Battle ID, Match ID, Build ID, Player ID, Result)

- Can be used to evaluate build IDs, Pokémon pick rates, and win rates
- This data can be used to balance the game, resulting in a better user experience

Player – Event – Participates (Player ID, Event ID)

- Can be used to get an understanding of what types of events have maximum participation
- Borrowing ideas from events that work out well gives a reason for users to spend time on the game through rewards and this maximizes monetization opportunity

Player – Event – Rewards (Player ID, Event ID, Reward ID)

- Can be used to analyze what kinds of rewards motivate users

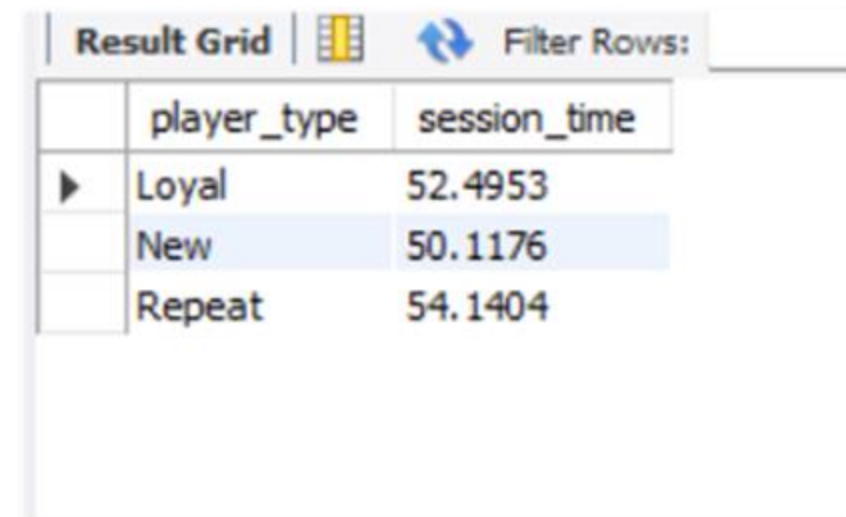
Player – Store – Buys (Player ID, Store Name, Product ID)

- Can be used to get a sense of top-selling products and concepts, so that the company can invest resources in developing similar mechanics

Querying the Database with SQL

Analytical Purpose: Analysing the average time spent on the game by type of players from different ranks

```
select case when b.rank < 10 then 'New'
        when b.rank between 11 and 20 then 'Repeat'
        when b.rank > 20 then 'Loyal'
        end as player_type,
        avg(TIMESTAMPDIFF(MINUTE, login_time, logout_time)) as session_time
from activity_history a
left join player b
    on a.player_id = b.ID
group by case when b.rank <= 10 then 'New'
            when b.rank between 11 and 20 then 'Repeat'
            when b.rank > 20 then 'Loyal'
            end;
```



The screenshot shows a 'Result Grid' interface with a table containing three rows of data. The columns are 'player_type' and 'session_time'. The rows represent 'Loyal', 'New', and 'Repeat' player types with their respective average session times in minutes.

	player_type	session_time
▶	Loyal	52.4953
	New	50.1176
	Repeat	54.1404

Querying the Database with SQL

Analytical Purpose: Determining the top 3 performing builds. By identifying this, the build can be nerfed or modified to balance the game. Here, the build should be part of the minimum number of battles (threshold value of 6 in this sample) to judge the quality of the build accurately.

```
with base as
(select build_id,
      count(case when result="Win" then battle_id else null end) as wins,
      count(case when result="Lose" then battle_id else null end) as loses,
      count(battle_id) as battles,
      count(case when result="Win" then battle_id else null end)/count(battle_id) as win_rate
 from game.battle_records
 group by build_id
 having battles>6)
```

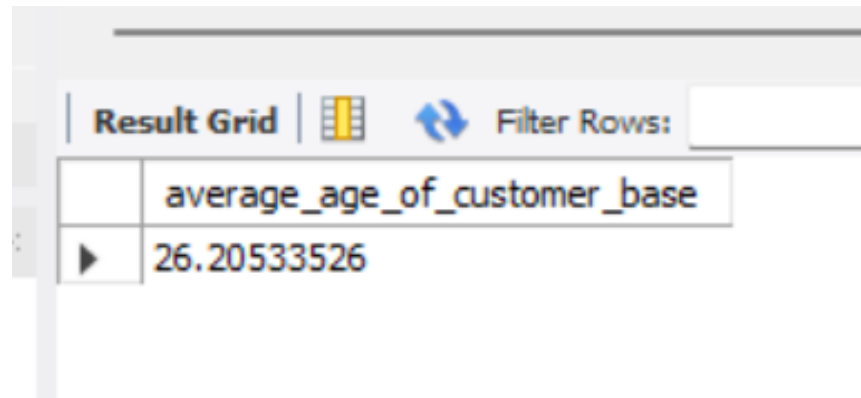
```
select a.*,
      c.pokemon_id,
      d.name as pokemon_name,
      c.move_set
 from base a
 left join game.build_use c
   on a.build_id = c.build_id
 left join game.pokemon d
   on c.pokemon_id = d.id
 where 3 > (select count(*)
            from base b
           where a.win_rate < b.win_rate)
 order by win_rate desc;
```

Result Grid		Filter Rows:		Export:	Wrap Cell Content:			
	build_id	wins	loses	battles	win_rate	pokemon_id	pokemon_name	move_set
▶	87	7	0	9	0.7778	22	Talonflame	Flame Charge - Fly
	3	7	2	9	0.7778	1	Absol	Night Slash - Sucker Punch
	10	10	2	13	0.7692	3	Blissey	Egg Bomb - Safeguard

Querying the Database with SQL

Analytical Purpose: Determining the average age of the game's customer base

```
select avg(age) as average_age_of_customer_base  
from (select email,  
            datediff(CURDATE(), DOB)/365 as age  
      from game.user) a;
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid contains one row with the column name 'average_age_of_customer_base' and the value '26.20533526'. Above the grid, there are icons for a grid, a refresh button, and a 'Filter Rows:' input field.

	average_age_of_customer_base
▶	26.20533526

Implementing NoSQL

Finding the average price of Pokémon by class:

```
db.pokemon.aggregate([{$match: {'price':{ $ne: null} } },{$group:{_id: "$class", avg_price:{ $avg: "$price"}}}]);
```

```
> db.pokemon.aggregate([
  {$match:
    {'price':
      { $ne: null} } },
  {$group:
    {_id: "$class",
     avg_price:{ $avg: "$price"}}
  });
< { _id: 'Attacker', avg_price: 8400 }
  { _id: 'All Rounder', avg_price: 8800 }
  { _id: 'Defender', avg_price: 7500 }
  { _id: 'Supporter', avg_price: 7500 }
  { _id: 'Speedster', avg_price: 8666.666666666666 }
Pokemongame >
```

Finding the DOB of the youngest player:

```
db.User.aggregate({$group: {_id: null, youngest_player_DOB: {$max: "$DOB"}}});
```

```
> db.User.aggregate({$group: {_id: null, youngest_player_DOB: {$max: "$DOB"}}});
< { _id: null, youngest_player_DOB: '2009/12/05' }
Pokemongame >
```

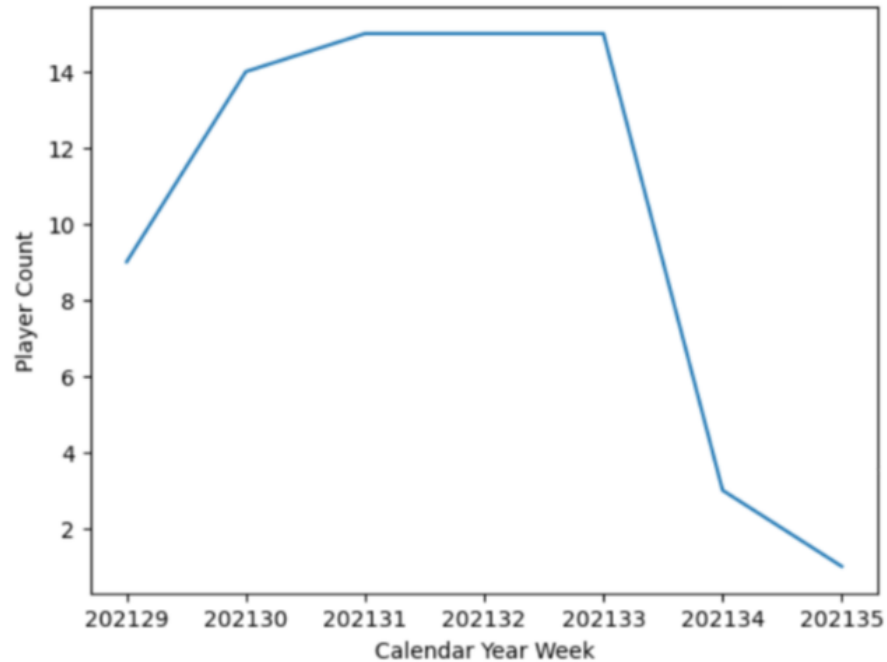

Implementing NoSQL

Finding the list of Users who were born after 2000:

```
db.User.find({DOB: {$gte: "2000/01/01"}});
```

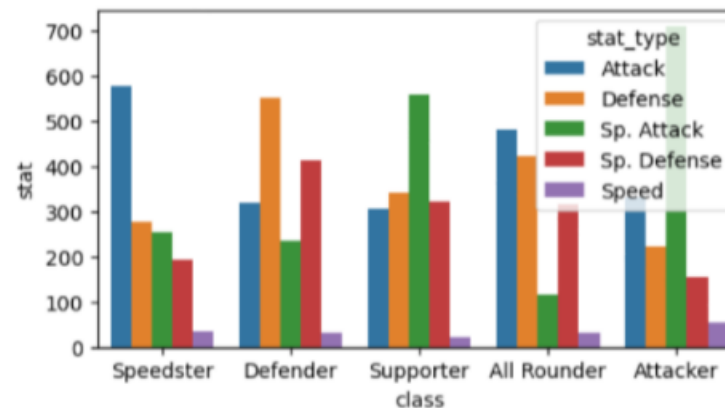
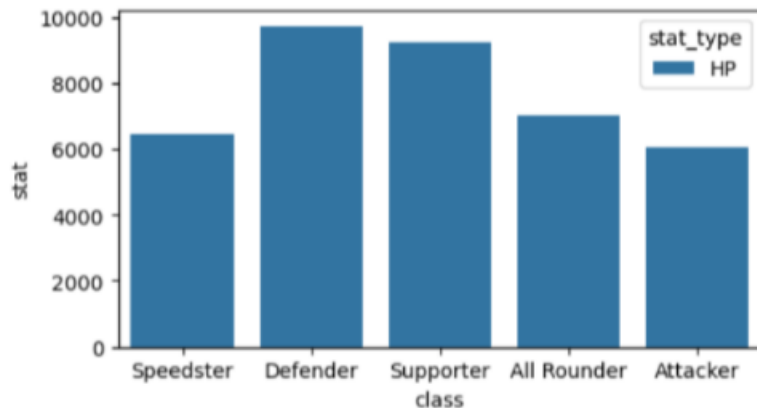
```
db.User.find({DOB: {$gte: "2000/01/01"}});
< { _id: ObjectId("639125557f2511d40309ede8"),
  email: 'skylar21@hotmail.com',
  DOB: '2008/06/19' }
{ _id: ObjectId("639125557f2511d40309ede9"),
  email: 'jada90@yahoo.com',
  DOB: '2007/10/10' }
{ _id: ObjectId("639125557f2511d40309eded"),
  email: 'emmalee.heller57@gmail.com',
  DOB: '2007/11/07' }
{ _id: ObjectId("639125557f2511d40309edee"),
  email: 'brooke.mckenzie@hotmail.com',
  DOB: '2008/03/08' }
{ _id: ObjectId("639125557f2511d40309edef"),
  email: 'jayme_harris@yahoo.com',
  DOB: '2000/10/15' }
{ _id: ObjectId("639125557f2511d40309edf0"),
  email: 'leonel_cruickshank37@yahoo.com',
  DOB: '2001/06/16' }
{ _id: ObjectId("639125557f2511d40309edf1"),
  email: 'idell.blanda98@gmail.com',
  DOB: '2009/12/05' }
```

Visualizing in Python



The line plot on the left tracks the overall users playing the game over the weeks. It can be noticed that the number of active users has declined over time. This would help the company keep track of the activity and evaluate how the strategies implemented impact the activity over time.

The bar graphs below are the average stats of Pokémon by class. It can be utilized by comparing the stats of a given Pokémon to its class averages and evaluating the deviation from the average. This could help balance the game, which in turn would result in a better user experience.



POKÉMON UNITE



Thank You!