GAME ANALYSIS

USING SQL

TITLE: DECODING GAME BEHAVIOR

BY- MOSTAQUL ARIF

You can connect with me on :

CONTENTS OF THIS PRESENTATION

- Overview
- Data Source and Methodology
- Uploading CSVs into MySQL
- Data Cleaning & Transformation
- Insight Slides (SQL queries to solve all 16 Problem statements & their significance)
- Conclusion

Overview

- Delving into the realm of gaming to analyze player behavior.
- Studying gameplay data to identify key metrics and patterns.
- Understanding how game elements influence player strategies.
- Providing insights to help developers enhance gaming experiences.
- Embarking on an exciting journey of decoding game behavior together!

DATA SOURCE & METHODOLOGY

The data for this project was provided by ***Mentorness***, which served as the primary source for our analysis. The datasets were imported into a ***MySQL*** database management system, ensuring a reliable and efficient storage solution for the game's data. As part of the data preparation process, certain columns were ***restructured*** or modified to facilitate easier analysis and querying. This step was crucial in ensuring that the data was clean, relevant, and easy to work with.

The methodology involved solving 16 problem statements using SQL queries. These problem statements, also provided by ***Mentorness***, guided the analysis and helped uncover ***key insights*** into game behavior. The use of SQL and the MySQL database system allowed for ***robust data manipulation*** and querying capabilities, enabling a thorough exploration of the dataset.

This approach ensured a systematic and rigorous analysis of the data, leading to valuable insights and conclusions.

Uploading CSVs into MySQL

To upload CSV files into MySQL, you can follow these steps:

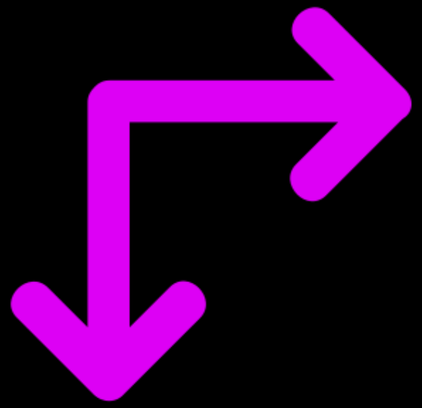
- Preparation: Ensure that you have the CSV files ready and accessible.
- Database Creation: Create a database in MySQL where you want to import the CSV data.

```
CREATE DATABASE Game_Analysis;
```

- Data Import: Use MySQL's LOAD DATA INFILE statement to import the data from CSV files into your MySQL tables.

Data Cleaning & Transformation

Use These Queries To Clean And Transform The Data



Rename table player_details to pd

```
ALTER TABLE player_details RENAME TO pd;
```

Drop the column 'myunknowncolumn' from the pd table

```
ALTER TABLE pd DROP COLUMN myunknowncolumn;
```

Modify the data type of 'L1_Status' column to varchar(30)

```
ALTER TABLE pd MODIFY COLUMN L1_Status VARCHAR(30);
```

Modify the data type of 'L2_Status' column to varchar(30)

```
ALTER TABLE pd MODIFY COLUMN L2_Status VARCHAR(30);
```

Modify the data type of 'P_ID' column to int and add it as the primary key

```
ALTER TABLE pd MODIFY COLUMN P_ID INT PRIMARY KEY;
```

Rename table level_details2 to ld

```
ALTER TABLE level_details2 RENAME TO ld;
```

Drop the column 'myunknowncolumn' from the ld table

```
ALTER TABLE ld DROP COLUMN myunknowncolumn;
```

Change the data type of 'timestamp' column to datetime and rename it to 'start_datetime'

```
ALTER TABLE ld CHANGE COLUMN timestamp start_datetime DATETIME;
```

Modify the data type of 'Dev_Id' column to varchar(10)

```
ALTER TABLE ld MODIFY COLUMN Dev_Id VARCHAR(10);
```

Modify the data type of 'Difficulty' column to varchar(15)

```
ALTER TABLE ld MODIFY COLUMN Difficulty VARCHAR(15);
```

Add a composite primary key consisting of P_ID, Dev_id, and start_datetime columns to the ld table

```
ALTER TABLE ld ADD PRIMARY KEY(P_ID, Dev_id, start_datetime);
```

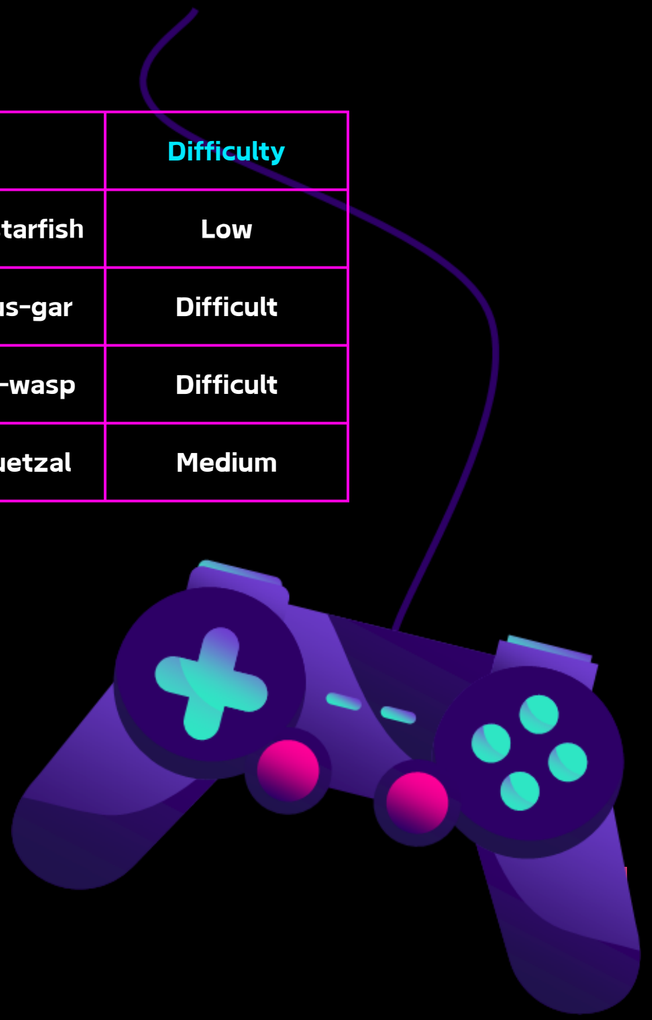
Insights, Queries & their Significance

01

EXTRACT 'P-ID','DEV-ID','Pname' and 'Difficulty_level' of all players at level 0

```
SELECT
   ld.P_ID, ld.Dev_ID, pd.PName,ld.Difficulty
FROM
   ld
JOIN
   pd ON ld.P_ID = pd.P_ID
WHERE
   ld.Level = 0;
```

P_ID	Dev_ID	PName	Difficulty
211	bd_017	breezy-indigo-starfish	Low
300	zm_015	lanky-asparagus-gar	Difficult
310	bd_015	gloppy-tomato-wasp	Difficult
358	zm_013	skinny-grey-quetzal	Medium

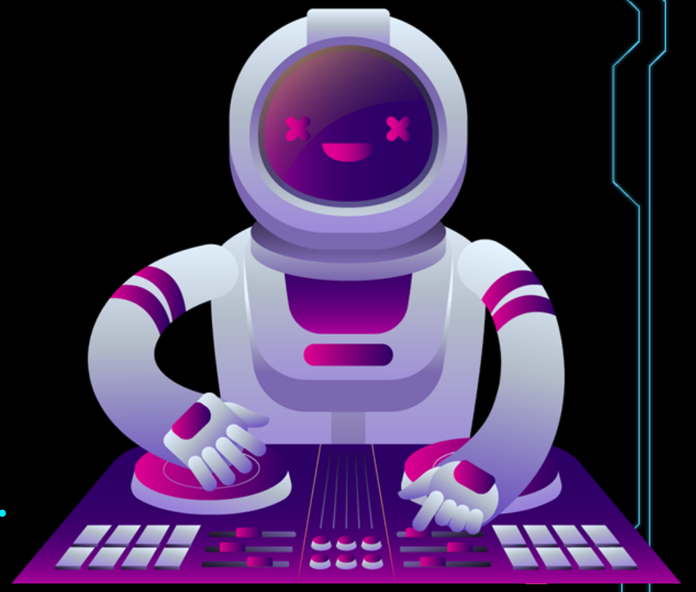


Significance: Identifying players at Level 0 helps to identify new or inexperienced gamers, which allows for targeted support or marketing campaigns.

02 Find ‘Level1_code’ wise ‘Avg_Kill_Count’ where lives_earned is 2 and at least 3 stages are crossed

```
SELECT
   pd.L1_Code,
   ROUND(AVG(id.Kill_count), 2) AS Avg_Kill
FROM
   id
JOIN
   pd ON id.P_ID = pd.P_ID
WHERE
   id.lives_earned = 2
   AND id.Stages_crossed >= 3
GROUP BY
   pd.L1_Code;
```

L1_Code	Avg_Kill
war_zone	19.29
bulls_eye	22.25
speed_blitz	19.33



Significance: The average 'Kill_Count' where 'lives_earned' is 2 and at least 3 levels are completed can provide information on player performance under these specific conditions.

03

Find the total number of stages crossed at each difficulty level where for Level2 with players use zm_series devices. Arrange the result in decreasing order of the total number of stages crossed.

```
SELECT
   ld.Difficulty,
   COUNT(ld.Stages_crossed) AS Total_Stages_Crossed
FROM
   ld
JOIN
   pd ON ld.P_ID = pd.P_ID
WHERE
   ld.Level = 2
   AND ld.Dev_ID LIKE 'zm_%'
GROUP BY
   ld.Difficulty
ORDER BY
   Total_Stages_Crossed DESC;
```

Difficulty	Total_Stages_Crossed
Difficult	7
Medium	6
Low	2

Significance: Knowing the total number of stages crossed at each difficulty level for Level 2 with players using `zm_series` devices can help understand player engagement and device performance.

04 Extract P_ID and the total number of unique dates for those players who have played games on multiple days.

```
SELECT
    Id.P_ID,
    COUNT(DISTINCT Id.start_datetime) AS Total_Unique_Dates
FROM
    Id
GROUP BY
    Id.P_ID
HAVING
    Total_Unique_Dates > 1
ORDER BY
    Total_Unique_Dates DESC;
```

P_ID	Total_Unique_Dates
683	7
211	6
300	5
483	5
590	5

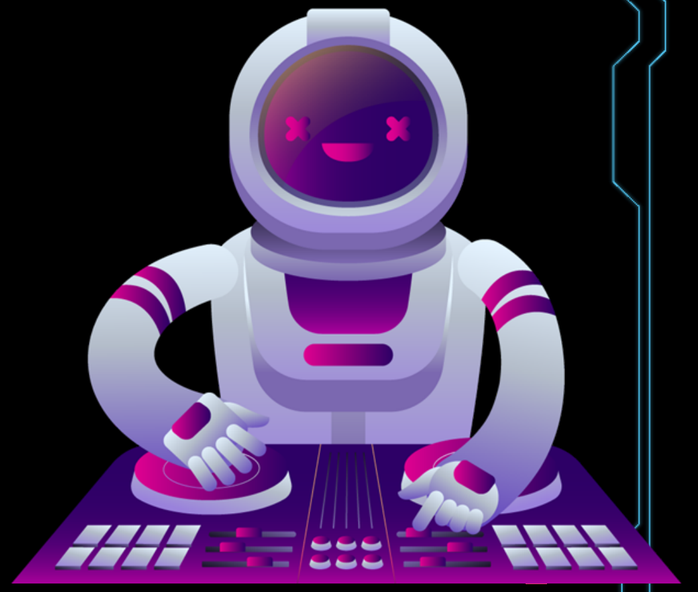
Significance: Identifying players who have played games on multiple days can help understand player retention and engagement over time.

05

Find ‘P_ID’ and level-wise sum of ‘kill_counts’ where ‘kill_count' is greater than the average kill count for the Medium difficulty.

```
SELECT
   Id.P_ID, Id.Level, SUM(Id.kill_Count) AS Total
FROM
   Id
WHERE
   Id.kill_Count > (
      SELECT AVG(Id.kill_Count)
      FROM Id
      WHERE Id.Difficulty = 'Medium'
   )
GROUP BY
   Id.P_ID, Id.Level
ORDER BY
   Id.Level;
```

P_ID	Level	Total
211	0	20
310	0	34
558	0	21
632	0	45
211	1	55

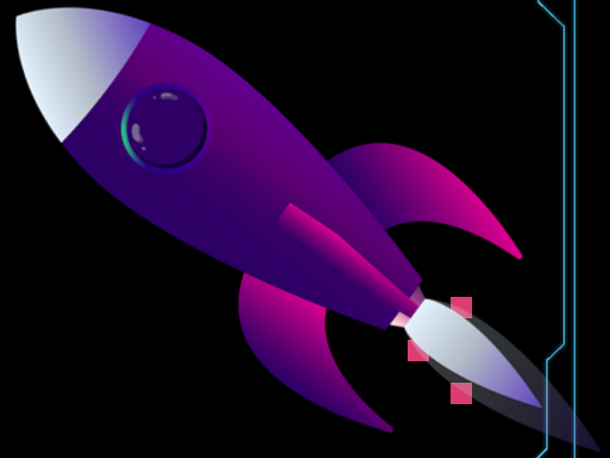


Significance: Finding players who have a `kill_count` greater than the average for Medium difficulty can help identify skilled players.

06 Find ‘Level’ and its corresponding ‘Level code’ wise sum of lives earned excluding level 0. Arrange in ascending order of the level.

```
SELECT
   Id.Level,
   CASE
      WHEN Id.level = 1 THEN pd.l1_code
      WHEN Id.level = 2 THEN pd.l2_code
      ELSE NULL
   END AS Level_Code,
   SUM(Id.Lives_Earned) AS Total_Lives_Earned
FROM
   Id JOIN  pd ON Id.P_ID = pd.P_ID
WHERE
   Id.Level > 0
GROUP BY
   Id.Level, Level_Code
ORDER BY
   Id.Level;
```

Level	Level_Code	Total_Lives_Earned
1	bulls_eye	5
1	leap_of_faith	0
1	speed_blitz	7
1	war_zone	11
2	cosmic_vision	12
2	resurgence	11
2	splippery_slope	28



Significance: Understanding the sum of lives earned at each level, excluding Level 0, can provide insights into player survival rates at different levels.

07 Find Top 3 scores based on each ‘dev_id’ and rank them in increasing order using Row_Number. Display difficulty as well.

```
SELECT
   dev_id, score, difficulty, rn
FROM (
   SELECT
      Id.Dev_ID, Id.Score, Id.Difficulty,
      ROW_NUMBER() OVER (PARTITION BY Id.dev_id ORDER BY Id.score DESC) AS rn
   FROM
      Id
) AS t
WHERE
   rn <= 3;
```

dev_id	score	difficulty	rn
bd_013	5300	Difficult	1
bd_013	4570	Difficult	2
bd_013	3370	Difficult	3
bd_015	5300	Difficult	1
bd_015	3200	Low	2

Significance: Ranking the top 3 scores based on each `Dev_ID` can help understand device performance and player skill.

08 Find the first_login datetime for each device id

```
SELECT
    Id.Dev_ID,
    MIN(Id.start_datetime) AS First_Login
FROM
    Id
GROUP BY
    Id.Dev_ID
ORDER BY
    First_Login;
```

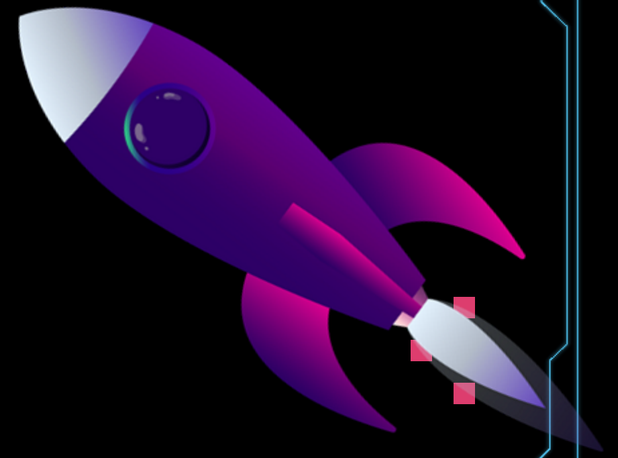
Dev_ID	First_Login
bd_013	11-10-2022 2:23
rf_013	11-10-2022 5:20
rf_017	11-10-2022 9:28
zm_013	11-10-2022 13:00
zm_015	11-10-2022 14:05

Significance: Knowing the `first_login` datetime for each device ID can provide insights into device usage patterns.

09 Find the Top 5 scores based on each difficulty level and rank them in increasing order using Rank. Display dev_id as well.

```
SELECT
    Difficulty, score, dev_id, rn
FROM (
    SELECT
        Id.Difficulty, Id.Score, Id.Dev_ID,
        RANK() OVER (PARTITION BY Id.Difficulty ORDER BY Id.Score DESC) AS rn
    FROM
        Id
) AS t
WHERE
    rn <= 5
ORDER BY
    Difficulty, rn;
```

Difficulty	Score	Dev_ID	rn
Difficult	5500	zm_017	1
Difficult	5500	zm_017	1
Difficult	5300	bd_013	3
Difficult	5300	bd_015	3
Difficult	5140	rf_017	5



Significance: Ranking the top 5 scores based on each difficulty level can help understand player performance across different difficulty levels.

10

Find the device ID that is first logged in (based on start_datetime) for each player(p_id). Output should contain player id, device id, and first login datetime.

```
SELECT
   ld.P_ID,
   ld.Dev_ID,
   MIN(ld.start_datetime) AS First_Login
FROM
   ld
GROUP BY
   ld.P_ID, ld.Dev_ID;
```

P_ID	Dev_ID	First_Login
211	bd_013	12-10-2022 18.30
211	bd_017	12-10-2022 13.23
211	rf_013	13-10-2022 5.36
211	rf_017	15-10-2022 11.41
211	zm_015	13-10-2022 22.30

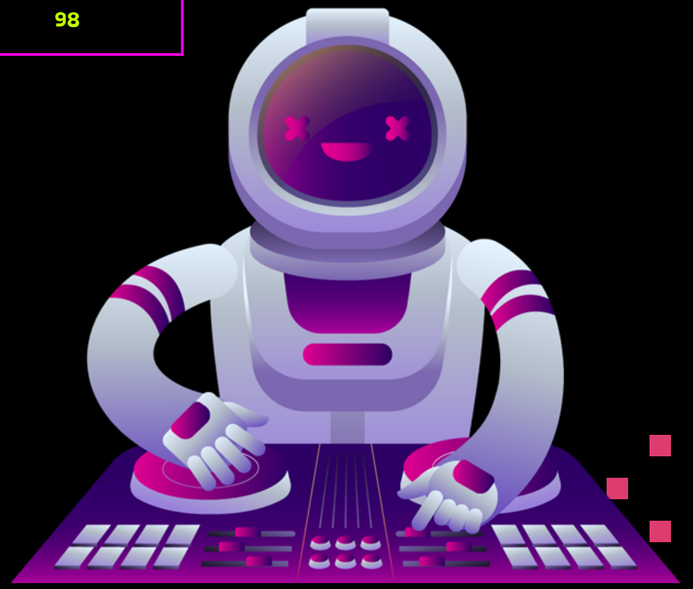
Significance: Identifying the device ID that is first logged in for each player can provide insights into player device preferences

11 For each player and date, determine how many `kill_counts` were played by the player so far.
a) Using window functions

```
SELECT
    ld.P_ID,
    ld.start_datetime,
    SUM(ld.Kill_Count) OVER (PARTITION BY ld.p_id
ORDER BY ld.start_datetime) AS total_kills_so_far
FROM
    ld;
```

P_ID	start_datetime	total_kills_so_far
211	12-10-2022 13.23	20
211	12-10-2022 18.30	45
211	13-10-2022 5.36	75
211	13-10-2022 22.30	89
211	14-10-2022 8.56	98

Significance: Determining how many `kill_counts` were played by the player so far can help understand player progress and engagement.



11 For each player and date, determine how many `kill_counts` were played by the player so far.
b) Without window functions

```
SELECT
   ldl.P_ID, ldl.start_datetime,
   SUM(ldl.Kill_Count) AS total_kills_so_far
FROM
   ld ldl
      JOIN
   ld ld2 ON ldl.P_ID = ld2.P_ID
      AND ldl.start_datetime >= ld2.start_datetime
GROUP BY ldl.P_ID , ldl.start_datetime
```

P_ID	start_datetime	total_kills_so_far
211	12-10-2022 18.30	50
211	13-10-2022 5.36	90
211	15-10-2022 11.41	90
211	13-10-2022 22.30	56
211	14-10-2022 8.56	45

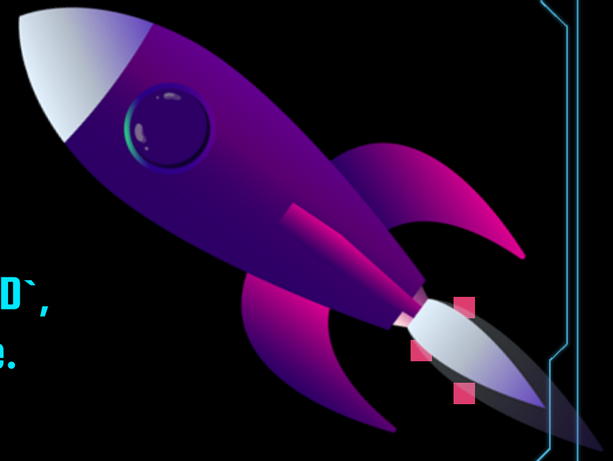
Significance: Determining how many `kill_counts` were played by the player so far can help understand player progress and engagement.

12

Find the cumulative sum of stages crossed over `start_datetime` for each `P_ID`, excluding the most recent `start_datetime`.

```
SELECT
    Id.P_ID,
    Id.start_datetime,
    Id.Stages_crossed,
    SUM(Id.Stages_crossed) OVER (
        PARTITION BY Id.p_id
        ORDER BY Id.start_datetime
        ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING
    ) AS cumulative_sum
FROM
    Id;
```

P_ID	start_datetime	Stages_crossed	cumulative_sum
211	12-10-2022 13.23	4	NULL
211	12-10-2022 18.30	5	4
211	13-10-2022 5.36	5	9
211	13-10-2022 22.30	5	14
211	14-10-2022 8.56	7	19



Significance: Finding the cumulative sum of stages crossed over `start_datetime` for each `P_ID`, excluding the most recent `start_datetime`, can provide insights into player progress over time.

13 Extract the top 3 highest sums of scores for each `Dev_ID` and the corresponding `P_ID`.

```
SELECT
    dev_id,p_id,total, rn
FROM (
    SELECT
        Id.Dev_ID, Id.P_ID, SUM(Id.Score) AS Total,
        RANK() OVER (PARTITION BY Id.Dev_ID ORDER BY SUM(Id.Score) DESC) AS rn
    FROM
        Id
    GROUP BY
        Id.Dev_ID, Id.P_ID
) AS t
WHERE
    rn <= 3;
```

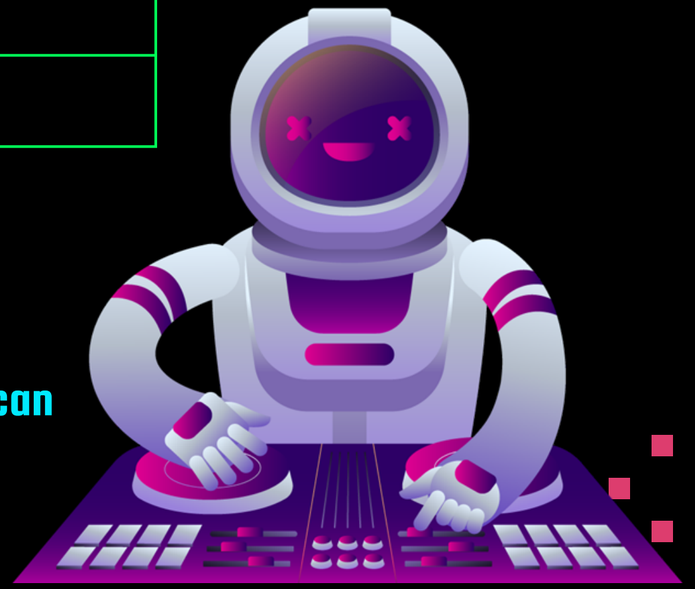
dev_id	p_id	total	rn
bd_013	224	9870	1
bd_013	310	3370	2
bd_013	211	3200	3
bd_015	310	5300	1
bd_015	683	3200	2

Significance: Extracting the top 3 highest sums of scores for each `Dev_ID` and the corresponding `P_ID` can help understand player performance and device performance.

14 Find players who scored more than 50% of the average score scored by the sum of scores for each player_id

```
SELECT
   P_ID, SUM(Score) AS Total_Score
FROM
   Id
GROUP BY
   P_ID
HAVING
   SUM(Score) > (
      SELECT 0.5 * AVG(Score)
      FROM Id
   )
ORDER BY
   Total_Score DESC;
```

P_ID	Total_Score
683	18140
483	17230
224	16310
310	13810
429	13220



Significance: Identifying players who scored more than 50% of the average score can help identify above-average players.

15 Create a stored procedure to find top n headshots_count based on each dev_id and rank them in increasing order using Row_Number. Display difficulty as well.

```
DELIMITER $$
CREATE PROCEDURE Top_N_Headshots(
    IN n INT
)
BEGIN
    SELECT
        dev_id, headshots_count, difficulty, ranking
    FROM (
        SELECT
            dev_id, headshots_count, difficulty,
            ROW_NUMBER() OVER (PARTITION BY dev_id ORDER BY headshots_count DESC) AS ranking
        FROM
            Id
    ) AS ranked
    WHERE
        ranking <= n;
END$$
DELIMITER ;

CALL Top_N_Headshots(3);
```

dev_id	headshots_count	difficulty	ranking
bd_013	30	Difficult	1
bd_013	30	Difficult	2
bd_013	25	Difficult	3
bd_015	30	Difficult	1
bd_015	30	Difficult	2
bd_015	20	Low	3

Significance: Creating a stored procedure to find the top ‘n’ ‘headshots count’ based on each ‘Dev_ID’ can provide a reusable tool for analgzing player performance.

16 Create a function to return the sum of Score for a given player_id.

```
DELIMITER $$
CREATE FUNCTION Total_Score(
   playerId INT
)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
   DECLARE totalScore INT;

   SELECT
      SUM(Score) INTO totalScore
   FROM
      Id
   WHERE
      P_ID = playerId;

   RETURN totalScore;
END$$
DELIMITER ;

SELECT Total_Score(683);
```

Total_Score (683)
18140

Significance: The Total_Score function simplifies the process of calculating the total score for a specified player ID, offering a concise and efficient method for obtaining this information in SQL queries.

CONCLUSION

Our project harnessed Game Analysis data, leveraging MySQL for robust database management. We meticulously prepared and analyzed the data using SQL, addressing 15 key problem statements to unveil critical insights into gaming behavior.

These insights, encompassing player profiling, retention strategies, and performance assessment, offer actionable implications for player support and game refinement. Our project underscores SQL's versatility in handling large datasets and emphasizes the importance of systematic analysis in shaping industry strategies.

The outcomes of this project hold significant potential to inform future strategies and drive decision-making within the gaming industry, showcasing the value of rigorous data analysis within MySQL environments.

THANKS!