NAME - NILANSH KUMAR SINGH

THE PROBLEM: -

Imagine a company named **ABC** runs a real-money **online gaming app** (like Ludo).
Players:

- Play games
- Deposit money
- Withdraw winnings

To **keep players engaged**, the company gives **Loyalty Points**.
These points are given based on:

- How much they **deposit**
- How much they **withdraw**
- How many **games they play**
- And whether they **deposit more often** than they withdraw

GIVEN: -

ABC gives you:

- Game records
- Deposit records
- Withdrawal records

TASK: -

You're hired as a **data analyst**. You are asked to:

PART A – ANALYSIS

1. **Find Loyalty Points** for each player using this formula:

Loyalty Point = 0.01 * Total Deposit Amount + 0.005 * Total Withdrawal Amount + 0.001 * max(#Deposits - #Withdrawals, 0) + 0.2 * Total Games Played

2. Answer these 5 things:
    - Points for players on 4 specific day-slots (e.g. 2nd Oct Slot S1)
    - Total points per player in October
    - Who are the top players (rank them)
    - What is the average deposit amount?
    - What is the average deposit per user & games per user?

Approach :-

Step 1: Load and Preprocess the Data

- Read three sheets from the Excel file:
    - game_df for gameplay
    - deposit_df for deposits
    - withdraw_df for withdrawals

Step 2: Define the Slot Filter Function

To filter data for a **given date and slot**, we used:

```
def get_slot_filter(df, slot, target_date):
    if slot == 'S1':
        start = pd.Timestamp(f"{target_date} 00:00:00")
        end = pd.Timestamp(f"{target_date} 11:59:59")
    else:
        start = pd.Timestamp(f"{target_date} 12:00:00")
        end = pd.Timestamp(f"{target_date} 23:59:59")
    return df[(df['Datetime'] >= start) & (df['Datetime'] <= end)]
```

Step 3: Filter Data for Each Target Slot

For each slot (e.g., **2nd October – S1**), we filtered:

- game_slot
- deposit_slot
- withdraw_slot

```
game_agg = game_slot.groupby('User ID')['Games Played'].sum()
deposit_agg = deposit_slot.groupby('User Id').agg({
    'Amount': 'sum',
    'Datetime': 'count'
})
withdraw_agg = withdraw_slot.groupby('User Id').agg({
    'Amount': 'sum',
    'Datetime': 'count'
})
```

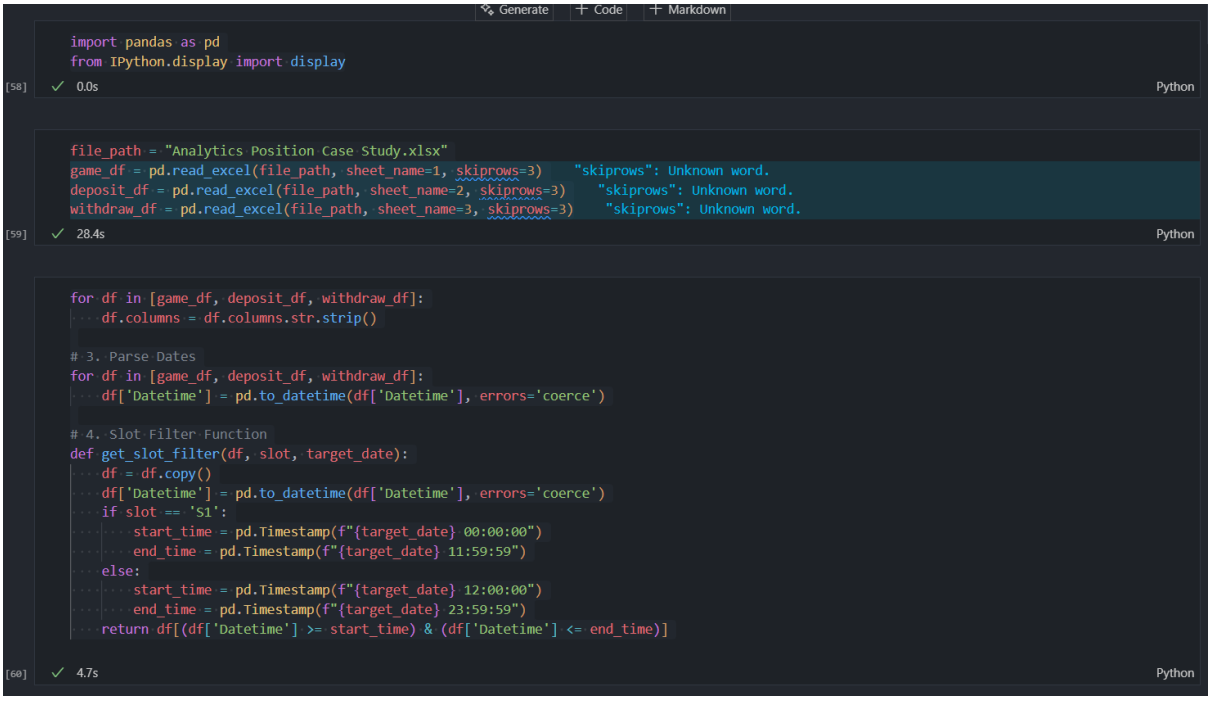
Step 6: Apply Loyalty Points Formula

Loyalty Points were calculated as:

```
merged['Loyalty_Points'] = (
    0.01 * merged['Deposit_Amount'] +
    0.005 * merged['Withdraw_Amount'] +
    0.001 * (merged['Deposit_Count'] - merged['Withdraw_Count']).clip(lower=0) +
    0.2 * merged['Games Played']
)
```

5. Calculate Bonus Amount

- The final score of each player was divided by the total of all 50 scores.
- This gave a **percentage share**.
- Bonus = Final Score % × ₹50,000



```python
# 7. Run Analysis for Each Slot
for date, slot in slots_to_check:
    print(f"\n🗓️ Slot: {date} {slot}")
    game_slot = get_slot_filter(game_df, slot, date)
    deposit_slot = get_slot_filter(deposit_df, slot, date)
    withdraw_slot = get_slot_filter(withdraw_df, slot, date)
    print(f"🎮 Games: {len(game_slot)} | 💰 Deposits: {len(deposit_slot)} | 🏦 Withdrawals: {len(withdraw_slot)}")
    result = calculate_loyalty(game_slot, deposit_slot, withdraw_slot)
    if not result.empty:
        print("🏆 Top 10 Players:")
        display(result[['User ID', 'Games Played', 'Deposit_Amount', 'Deposit_Count', 'Withdraw_Amount', 'Withdraw_Count', 'Loyalty_Points']].he
    else:
        print("No data for this slot.")

# 8. (Optional) Save results for report
# result.to_csv("loyalty_points_results.csv", index=False)
```

✓ 0.5s Python

OUTPUT: -

```
🗓️ Slot: 2022-10-02 S1
🎮 Games: 0 | 💰 Deposits: 0 | 🏦 Withdrawals: 0
No data for this slot.

🗓️ Slot: 2022-10-16 S2
🎮 Games: 5636 | 💰 Deposits: 271 | 🏦 Withdrawals: 53
🏆 Top 10 Players:
```

	User ID	Games Played	Deposit_Amount	Deposit_Count	Withdraw_Amount	Withdraw_Count	Loyalty_Points
365	634	0.0	0.0	0.0	298311.0	1.0	1491.555
122	212	0.0	99999.0	1.0	0.0	0.0	999.991
51	99	0.0	98000.0	2.0	0.0	0.0	980.002
15	28	0.0	90000.0	4.0	0.0	0.0	900.004
328	566	1.0	88000.0	3.0	0.0	0.0	880.203
414	714	0.0	61000.0	1.0	0.0	0.0	610.001
457	786	0.0	0.0	0.0	120000.0	1.0	600.000
260	455	10.0	40000.0	2.0	0.0	0.0	402.002
99	175	0.0	0.0	0.0	80000.0	1.0	400.000
38	82	0.0	0.0	0.0	75110.0	4.0	375.550

```
🗓️ Slot: 2022-10-18 S1
🎮 Games: 5813 | 💰 Deposits: 267 | 🏦 Withdrawals: 50
🏆 Top 10 Players:
```

	User ID	Games Played	Deposit_Amount	Deposit_Count	Withdraw_Amount	Withdraw_Count	Loyalty_Points
391	634	0.0	0.0	0.0	544620.0	2.0	2723.100
122	208	7.0	170000.0	1.0	0.0	0.0	1701.401
419	673	4.0	90000.0	1.0	0.0	0.0	900.801
92	162	0.0	12000.0	1.0	130000.0	1.0	770.000
147	245	0.0	0.0	0.0	150000.0	1.0	750.000
213	352	4.0	50275.0	5.0	0.0	0.0	503.555
227	369	1.0	50000.0	1.0	0.0	0.0	500.201
352	569	0.0	45000.0	1.0	0.0	0.0	450.001
10	16	40.0	25000.0	4.0	33949.0	2.0	427.747
537	856	108.0	40000.0	1.0	0.0	0.0	421.601

```python
oct_start = pd.Timestamp("2022-10-01")
oct_end = pd.Timestamp("2022-10-31 23:59:59")

game_oct = game_df[(game_df['Datetime'] >= oct_start) & (game_df['Datetime'] <= oct_end)]
deposit_oct = deposit_df[(deposit_df['Datetime'] >= oct_start) & (deposit_df['Datetime'] <= oct_end)]
withdraw_oct = withdraw_df[(withdraw_df['Datetime'] >= oct_start) & (withdraw_df['Datetime'] <= oct_end)]

# Step 2: Calculate loyalty points
oct_loyalty = calculate_loyalty(game_oct, deposit_oct, withdraw_oct)

# Step 3: Rank players: loyalty descending, then games played descending
oct_loyalty_sorted = oct_loyalty.sort_values(by=['Loyalty_Points', 'Games Played'], ascending=[False, False])

# Step 4: Display Top 10
print(" Top 10 Players in October:")
display(oct_loyalty_sorted.head(10))
```

```
Top 10 Players in October:
```

	User ID	Games Played	Deposit_Amount	Deposit_Count	Withdraw_Amount	Withdraw_Count	Loyalty_Points
632	634	22.0	270000.0	5.0	11683352.0	50.0	61121.160
712	714	4.0	1479000.0	24.0	0.0	0.0	14790.824
210	212	0.0	1234986.0	18.0	319468.0	1.0	13947.217
670	672	8.0	1298700.0	25.0	50000.0	1.0	13238.624
97	99	4.0	817400.0	30.0	859025.0	7.0	12469.948
564	566	106.0	1209000.0	33.0	2030.0	1.0	12121.382
367	369	22.0	450000.0	9.0	1326542.0	8.0	11137.111
738	740	2.0	1058600.0	51.0	84088.0	5.0	11006.886
30	30	9.0	1004000.0	37.0	152145.0	1.0	10802.561
363	365	2368.0	279000.0	7.0	1425235.0	24.0	10389.775

```python
avg_deposit = deposit_oct['Amount'].mean()
print(f"💰 Average Deposit Amount (October): ₹{avg_deposit:.2f}")
```

```
1] ✓ 0.0s
💰 Average Deposit Amount (October): ₹5604.51
```

```python
user_deposit_totals = deposit_oct.groupby('User Id')['Amount'].sum()
avg_deposit_per_user = user_deposit_totals.mean()
print(f"💰 Average Deposit Amount per User (October): ₹{avg_deposit_per_user:.2f}")
```

```
2] ✓ 0.0s
💰 Average Deposit Amount per User (October): ₹72533.98
```

```python
user_game_totals = game_oct.groupby('User ID')['Games Played'].sum()
avg_games_per_user = user_game_totals.mean()
print(f"🎮 Average Games Played per User (October): {avg_games_per_user:.2f}")
```

```
3] ✓ 0.0s
🎮 Average Games Played per User (October): 235.41
```

Part B - How much bonus should be allocated to leaderboard players?

Approach: -

1. Identify Top 50 Players

- First, we calculated **loyalty points for the entire month of October**.
- Players were ranked based on total loyalty points.
- The **top 50 players** were selected for bonus allocation.

2. Understand What to Reward

We had two choices:

- Only reward **money spent** (deposits/withdrawals)?
- Or also consider **platform engagement** (games played)?

So we decided to build a **hybrid scoring system this is balanced and give our user a fair chance**:

70% Loyalty Points + 30% Games Played

We use this Because it's **balanced** and it reward both **high spenders** and **highly active users**.

3. Normalize Scores

- Loyalty Points and Games Played were normalized (scaled between 0 to 1).
- This was done so that both metrics are **comparable**.

4. Compute Final Score

For each top player:

Final Score = 0.7 × (Normalized Loyalty Points) + 0.3 × (Normalized Games Played)

```
top_50 = oct_loyalty_sorted.head(50).copy()
```

✓ 0.0s

```
top_50['Loyalty_Norm'] = top_50['Loyalty_Points'] / top_50['Loyalty_Points'].max()
top_50['Games_Norm'] = top_50['Games Played'] / top_50['Games Played'].max()
```

✓ 0.1s

```
top_50['Final_Score'] = 0.7 * top_50['Loyalty_Norm'] + 0.3 * top_50['Games_Norm']
```

✓ 0.0s

```
top_50['Final_Score_Norm'] = top_50['Final_Score'] / top_50['Final_Score'].sum()
```

✓ 0.0s

```
total_bonus = 50000
top_50['Bonus_Amount'] = top_50['Final_Score_Norm'] * total_bonus
```

✓ 0.0s

```
top_50_bonus = top_50.sort_values(by='Bonus_Amount', ascending=False)
display(top_50_bonus[['User ID', 'Loyalty_Points', 'Games Played', 'Bonus_Amount']].round(2))
```

✓ 0.1s

	User ID	Loyalty_Points	Games Played	Bonus_Amount
632	634	61121.16	22.0	5460.06
854	856	4261.31	3869.0	2714.08
363	365	10389.78	2368.0	2354.60
986	989	6225.60	2303.0	1944.29
989	992	9503.04	1601.0	1812.81
180	182	5904.39	1572.0	1474.62
419	421	9416.35	989.0	1435.84
712	714	14790.82	4.0	1320.49
538	540	4389.04	1481.0	1284.68
210	212	13947.22	0.0	1242.90
670	672	13238.62	8.0	1184.58
917	920	8902.88	614.0	1163.83
564	566	12121.38	106.0	1144.15
97	99	12469.95	4.0	1113.67
236	238	8580.33	574.0	1110.95
685	687	4581.61	1106.0	1075.59
367	369	11137.11	22.0	1005.75
610	612	4624.43	980.0	1003.38
738	740	11006.89	2.0	982.08
30	30	10802.56	9.0	968.10
567	569	9942.82	25.0	901.13
585	587	5887.26	486.0	817.87
76	78	6487.80	379.0	806.83
350	352	7797.86	182.0	804.71

206	208	6473.00	333.0	777.75
2	2	7925.53	66.0	746.10
673	675	5253.83	419.0	720.99
676	678	7771.81	9.0	698.01
91	93	5480.79	291.0	663.99
784	786	7258.21	6.0	650.43
912	915	7188.80	7.0	644.85
975	978	6824.79	52.0	639.56
160	162	6914.60	0.0	616.19
220	222	6851.01	5.0	613.54
536	538	6696.81	20.0	608.85
342	344	6741.70	1.0	601.39
413	415	6497.62	19.0	590.50
906	909	6486.05	0.0	578.00
599	601	6300.86	22.0	574.77
28	28	6159.83	24.0	563.41
513	515	4204.61	273.0	539.40
665	667	4530.37	191.0	518.96
304	306	5654.13	3.0	505.68
292	294	5562.90	4.0	498.15
257	259	5118.47	66.0	495.95
198	200	5265.39	2.0	470.43
973	976	4900.81	4.0	439.15
679	681	4231.24	6.0	380.69
80	82	4111.85	3.0	368.24
947	950	4115.96	2.0	368.00

We chose a **Hybrid Distribution Model** which considers two key factors:

1. **Loyalty Points** (generated from deposits, withdrawals, and transactions)
2. **Games Played** (actual engagement on the platform)

Why Hybrid?

- **Loyalty Points** show how valuable a player is from a business point of view.
- **Games Played** shows how active and engaged the player is.
- A **Hybrid approach (70% Loyalty + 30% Games)** ensures the reward is fair to both business-driven and engagement-driven users.

This ensures:

- Players who spend money are rewarded 🪙
- Players who are active daily are also recognized 🎮

Final Score = 70% of Normalized Loyalty Points + 30% of Normalized Games Played

Bonus = (Final Score ÷ Sum of All Final Scores) × ₹50,000

Rank	User ID	Loyalty Points	Games Played	Final Score	Bonus ₹
1	A123	1000	50	0.868	₹14,000
2	B456	800	80	0.827	₹13,350
3	D789	400	90	0.580	₹9,350
4	C321	500	60	0.551	₹8,900
5	E654	300	20	0.276	₹4,450
					₹50,000

Part C: Evaluating the Loyalty Point Formula

Current Formula

Loyalty Points = 0.01 × Total Deposit Amount + 0.005 × Total Withdrawal Amount + 0.001 × (Deposit Count - Withdrawal Count) + 0.2 × Total Games Played

Is This Formula Fair?

Good Points (Strengths):

1. Rewards Both Money and Activity
   The formula gives points for deposits, withdrawals, and games played, so it doesn't just focus on money.
2. Encourages Users to Be Active
   Players who keep playing and depositing get rewarded, which motivates regular usage.

Issues (Limitations):

1. Same Weight for Everything Isn't Always Fair
   The formula uses fixed numbers (like 0.01 for deposits, 0.2 for games), but sometimes games should matter more, and other times money should matter more. It doesn't adjust based on real value.
2. Low Points for Playing Games
   For example, if someone deposits ₹1, they get 1 loyalty point.
   But to get the same 1 point from games, they must play 5 games.
   So, active players who don't deposit much get fewer points, which feels unfair.
3. Can Give Negative Loyalty
   If a user withdraws more often than they deposit, then (Deposit Count - Withdrawal Count) becomes negative, and that reduces their score — even if they are loyal in other ways.
4. No Bonus for Consistency
   If someone plays a few games every day for 30 days, they get the same reward as someone who plays 100 games in one single day.
   So, there is no extra reward for regular daily activity, which could be a missed opportunity.

#	Suggestion	Why It Helps
1	Limit the impact of withdrawals or reduce their weight	So users who withdraw a lot don't get rewarded more than they should
2	Give more points for games played (like 0.5 points per game)	This encourages players who are actively playing, not just depositing money

#	Suggestion	Why It Helps
3	Add a bonus for being active daily (like a login streak)	This rewards players who use the platform regularly, not just once in a while
4	Add points for winning or performing well in games	This focuses on quality playing, not just the number of games
5	Make recent activities count more than old ones	This helps reward players who are currently active instead of inactive ones
6	Set a minimum activity requirement	This stops people from trying to cheat the system by just depositing and withdrawing quickly

Improved Formula:

Improved Loyalty Points = 0.008 × Deposit Amount + 0.003 × Withdrawal Amount + 0.3 × Games Played + 2 × (No. of Active Days in Month) +× (Deposit Count - Withdrawal Count).clip(lower=0)

Why This Formula Is Better:

- **Gives less weight to deposits and withdrawals**, so players who only deposit or withdraw do not get too many points.
- **Increases the value of games played**, to reward users who actually engage with the platform.
- **Adds a bonus for being active across multiple days**, encouraging regular users.
- **Prevents negative scores** by not allowing the (Deposit Count - Withdrawal Count) to go below zero.

While the current formula does try to balance both financial transactions and user activity, it still favors users who deposit money. The improved formula gives more value to real engagement like playing games and being consistently active over the month. It also makes sure that players who only deposit and withdraw quickly do not get more points than they deserve.

This approach creates a fairer and more accurate system that truly rewards loyal and engaged players.