

# **Micro Leakage Detection - A Personal Finance Project**

Presented by -  
Ansuman Patro

## EXECUTIVE SUMMARY

Micro leakage in personal finance refers to small, often unnoticed outflows of money that accumulate over time, leading to significant financial inefficiencies. Identifying these leakages can empower individuals to optimize their spending and improve financial well-being. This project aims to detect such micro leakages using a data-driven approach, transforming raw financial transaction data into actionable insights.

The analysis followed a structured pipeline beginning with Python-based exploratory data analysis (EDA) and preprocessing using Jupyter Notebook. A machine learning model was then developed to identify patterns indicative of micro leakage behavior and classifying a given transaction as Leakage or Not. Following the model development, the final cleaned dataset was exported and analyzed using SQL to answer business-relevant queries and uncover deeper transactional trends. And finally an interactive Power BI dashboard to enable intuitive understanding and decision-making was made and delivered for personal finance tracking based on income, savings and expenses.

Key outcomes include identifying 4%-5% of total transactions attributed to micro leakage, highlighting dining out, medical emergencies and fuel as primary sources. Additional insights revealed temporal and behavioral trends such as “Evening based” or “Weekday based”.



- The tools and technologies used throughout the project include:
- Python (Jupyter Notebook) for data preprocessing, EDA, and ML modeling
  - SQL for structured querying and business question resolution
  - Power BI for interactive dashboarding and insight communication

This comprehensive approach provides a holistic view of financial micro leakages and establishes a strong foundation for informed personal finance optimization.



## Problem Statement

### Objective:

Detect financial micro-leakage and generate meaningful, data-backed personal finance insights.

Micro-leakage refers to small, often unnoticed financial outflows—like unused subscriptions, rounding errors, or minor impulse spends—that compound over time and impact financial health. Though seemingly insignificant individually, these transactions can lead to thousands in annual losses if left unchecked.

This project aims to detect and analyze such leakages using a three-layered approach:

- ML classification to label transactions as leakage or not,
- SQL exploration to answer key business questions, and
- Power BI visualization to present actionable insights.

## Data Overview

The dataset was synthetically generated to simulate real-world personal spending behavior across multiple categories and time blocks. It includes 1000+ transaction records with attributes relevant to financial analysis and leak detection.

### Key Features:

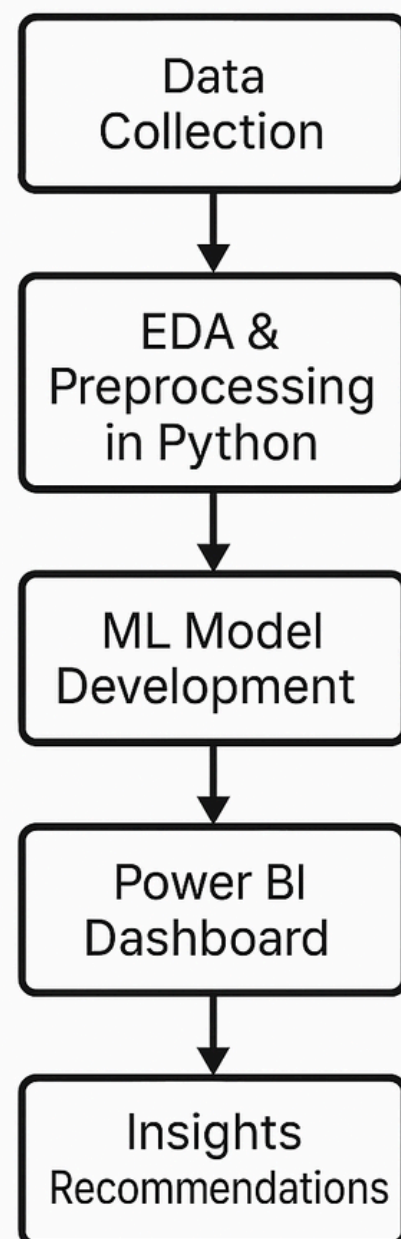
- **timestamp** – Date of transaction
- **type** – spending, expense or income
- **category** – Spending category (e.g., Rent, Dining Out)
- **value** – Amount spent
- **is\_necessary** – Whether the expense is essential
- **payment\_mode** – Cash, UPI, card, etc.
- **is\_recurring** – Monthly recurring or not
- **context** – Tag for specific purpose or event
- **time\_block** – Time of day (Morning, Evening, etc.)
- **day\_type** – Weekday vs Weekend
- **cumulative\_monthly\_spend** – Running total of monthly spend

## Sample of the Dataset

	Id	Date	type	category	value	Is_Necessary	payment_mode	is_recurring	context	time_block	day_type	cummulative	monthly spend
0	1	01-01-2023	expense	Education	1217	Yes	Debit Card	No	Planned	Afternoon	Weekend		1217.0
1	2	01-01-2023	income	Bonus	2245	No	Credit Card	No	Unplanned	Afternoon	Weekend		NaN
2	3	01-01-2023	expense	Education	525	Yes	Debit Card	No	Planned	Morning	Weekend		1742.0
3	4	02-01-2023	expense	Entertainment	194	No	UPI	No	Unplanned	Morning	Weekday		1936.0
4	5	02-01-2023	expense	Clothing	208	No	Cash	No	Seasonal	Morning	Weekday		2144.0

## Workflow

### Methodology & Workflow Overview



In our case, the dataset was synthetically generated, for industry standard, the data can be obtained from finance management apps, regular transaction data and budgeting info by the person

Feature engineering, data exploration, preprocessing and EDA on the dataset was done to understand trends, patterns and get valuable insight for further approaches

Developing a ML model based on the featured engineered columns and insights, hence, working on a model to detect whether for a given transaction - **it is a leakage transaction or not**

A personalized Dashboard was made using PowerBI, where the person can track, understand and get insights on their savings, expenses, and income in monthly basis

Finally, a number of queries and business questions were answered using SQL queries and data insights and hence got some valuable steps to follow to reduce our leakage and to save ourselves from unnecessary expenses



# Feature Engineering

Certain numbers of columns and measures were created for further analysis and model development.

```
# Daily spend sum
daily_spend = df.groupby('Date')['value'].sum().rename('daily_spend').reset_index()
df = df.merge(daily_spend, on='Date')
```

```
[588] # Category spend ratio
df = df.merge(monthly_cat, on=['year_month', 'category'])
df['category_spend_ratio'] = df['monthly_category_spend'] / df['monthly_total_spend']
```

```
[589] # Recurring expense count per month
recurring_counts = df[df['is_recurring'] == 'Yes'].groupby('year_month').size().rename('recurring_expense_count').reset_index()
df = df.merge(recurring_counts, on='year_month', how='left')
df['recurring_expense_count'] = df['recurring_expense_count'].fillna(0)
```

```
[580] # Create a Year-Month column for grouping
df['year_month'] = df['Date'].dt.to_period('M')
```

```
[581] # Monthly total spend (overall)
monthly_totals = df.groupby('year_month')['value'].sum().rename('monthly_total_spend')
df = df.merge(monthly_totals, on='year_month')
```

```
[582] # Monthly category spend
monthly_cat = df.groupby(['year_month', 'category'])['value'].sum().rename('monthly_category_spend').reset_index()
```

```
[583] # Monthly non-essential percentage
monthly_non_ess = df.groupby('year_month').apply(
    lambda x: x.loc[x['Is_Necessary']=='No', 'value'].sum() / x['value'].sum()
).rename('monthly_non_essential_pct').reset_index()

df = df.merge(monthly_non_ess, on='year_month')
```

```
[584] # Average spend per time_block
avg_time_block = df.groupby('time_block')['value'].mean().rename('avg_spend_per_time_block').reset_index()
df = df.merge(avg_time_block, on='time_block')
```

```
[585] # Average spend per day_type
avg_day_type = df.groupby('day_type')['value'].mean().rename('avg_spend_per_day_type').reset_index()
df = df.merge(avg_day_type, on='day_type')
```

```
# Extract the numeric month and year from the Date column
df['month'] = df['Date'].dt.month
df['year'] = df['Date'].dt.year
```

```
# Extract the name of the day (e.g., Monday, Tuesday) from the Date column
df['day_of_week'] = df['Date'].dt.day_name()
```

```
# Flag if the transaction occurred in the first week of the month (1st to 7th)
df['is_salary_week'] = df['Date'].dt.day.between(1, 7)
```

```
# Identify micro-spending transactions (less than ₹300)
df['is_micro_spend'] = (df['value'] < 300).astype(int)
```

```
# Detect potential financial leakages based on necessity, recurrence, or context
df['is_leakage_potential'] = (
    ((df['Is_Necessary'] == 'No') & (df['is_recurring'] == 'Yes')) |
    (df['context'].isin(['Unplanned', 'Festive', 'Emergency']))
).astype(int)
```

# Leakage Scoring System

## ✓ Leakage Scoring System (Feature Synthesis)

Define the scoring function -

Each transaction is evaluated for leakage potential based on 6 conditions. We'll assign weights as discussed.

```
[ ] def compute_leakage_score(row):
    score = 0
    if row['is_micro_spend']: # Micro spend (< ₹300)
        score += 1
    if row['Is_Necessary'] == 'No': # Non-necessary
        score += 2
    if row['context'] in ['Unplanned', 'Emergency', 'Festive']: # Impulsive contexts
        score += 2
    if row['is_recurring'] == 'Yes' and row['context'] != 'Planned': # Recurring but unplanned
        score += 2
    if row['time_block'] in ['Evening', 'Night']: # Impulse-prone time
        score += 1
    if row['day_type'] == 'Weekend': # Impulse-prone day
        score += 1
    return score

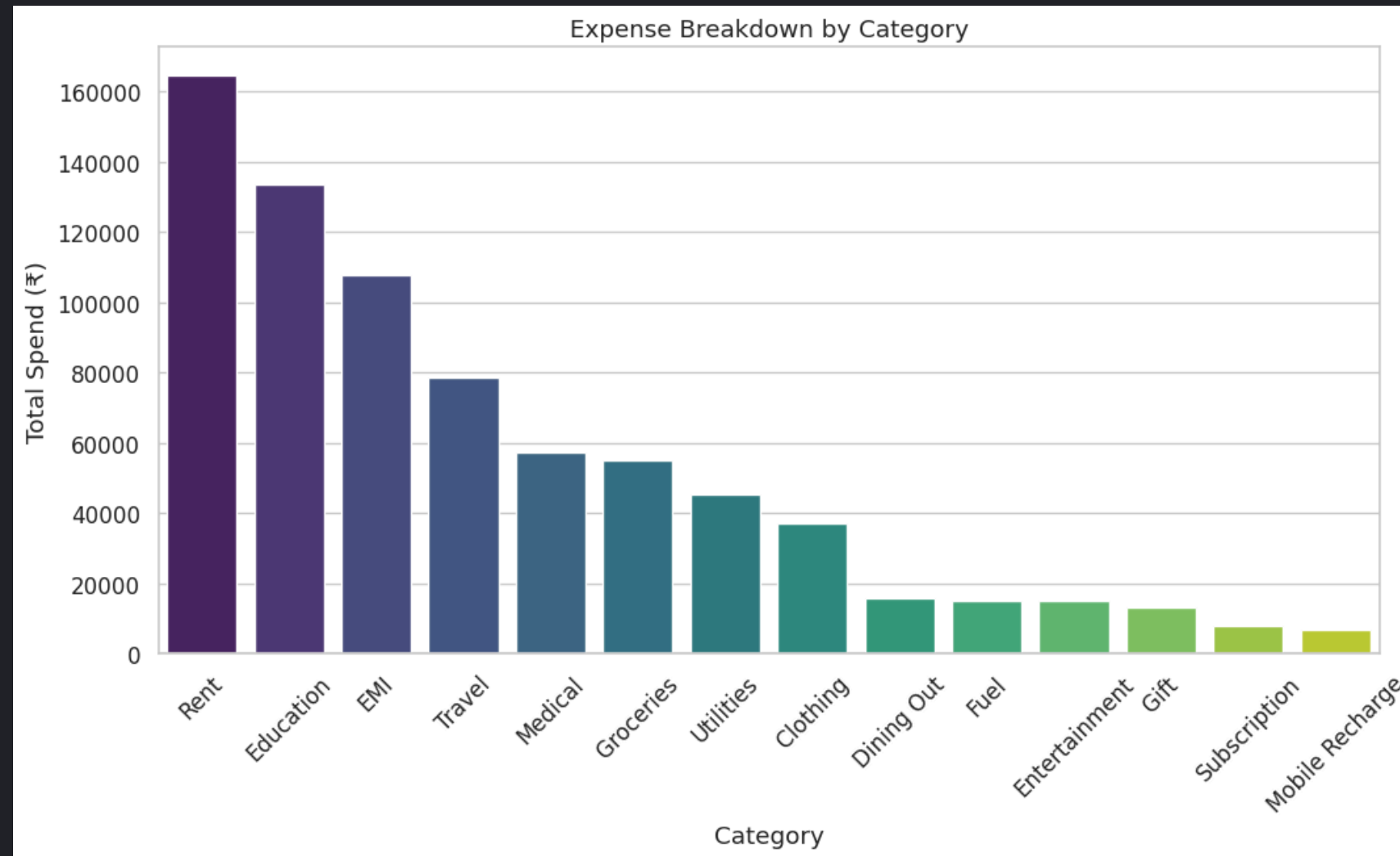
# Apply to dataset
df['leakage_score'] = df.apply(compute_leakage_score, axis=1)
```

This system defines a `compute_leakage_score` function to quantify a transaction's leakage potential based on six weighted conditions:

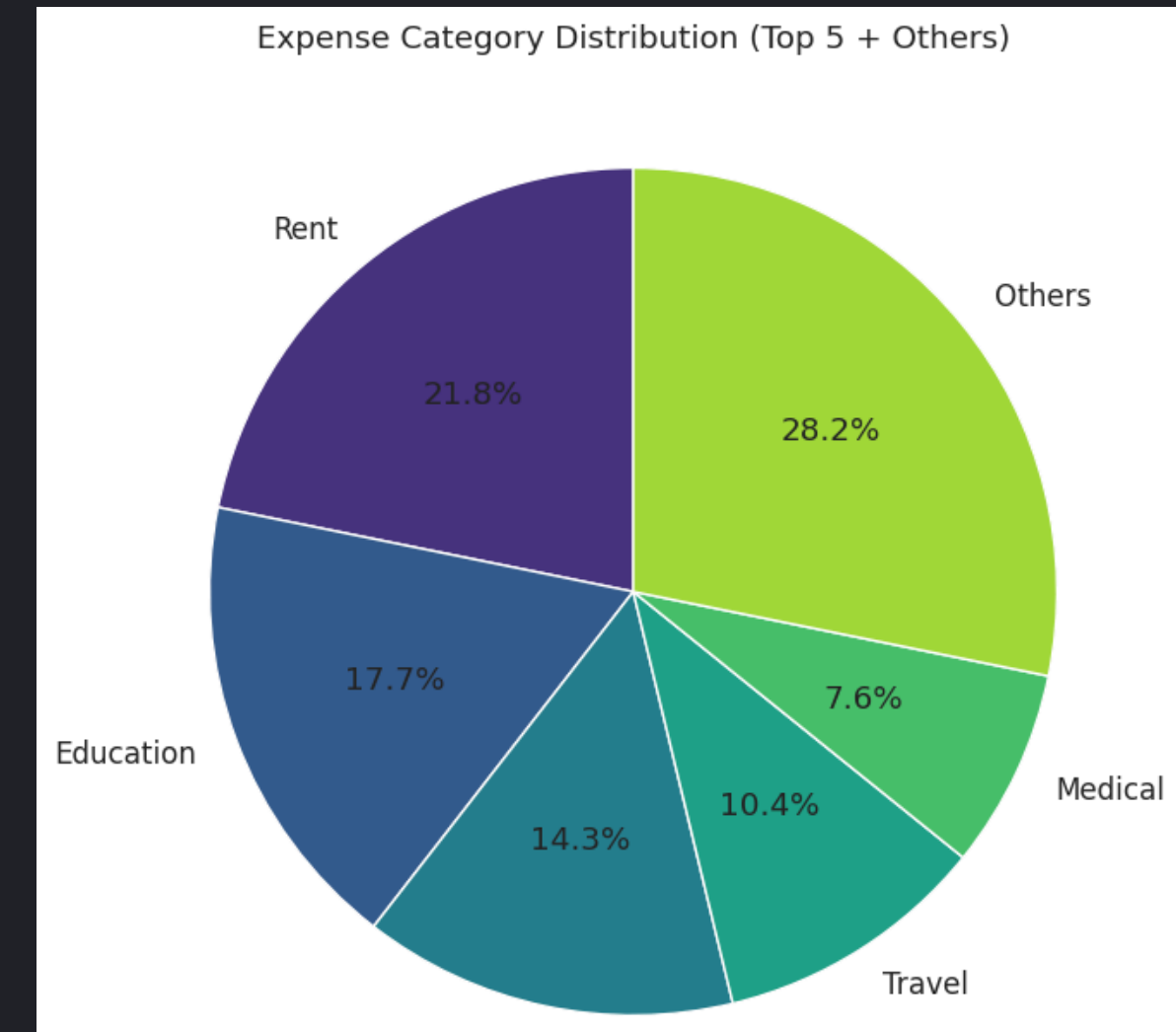
- Micro spend (< ₹300): +1 point
  - Non-necessary: +2 points
  - Impulsive contexts (Unplanned, Emergency, Festive): +2 points
  - Recurring & Unplanned: +2 points
  - Impulse-prone time (Evening, Night): +1 point
  - Impulse-prone day (Weekend): +1 point
- The function then applies this score to each transaction in the dataset to determine its "leakage potential."

## EDA ANALYSIS - P1

Understanding our dataset more briefly using visuals.



We observed Rent, Education, and EMI as the top three expense categories, indicating the largest portions of spending. While these major expenses are expected, the breakdown also highlights smaller categories like Subscription and Mobile Recharge, which, due to their recurring nature, warrant closer inspection for potential micro leakages.

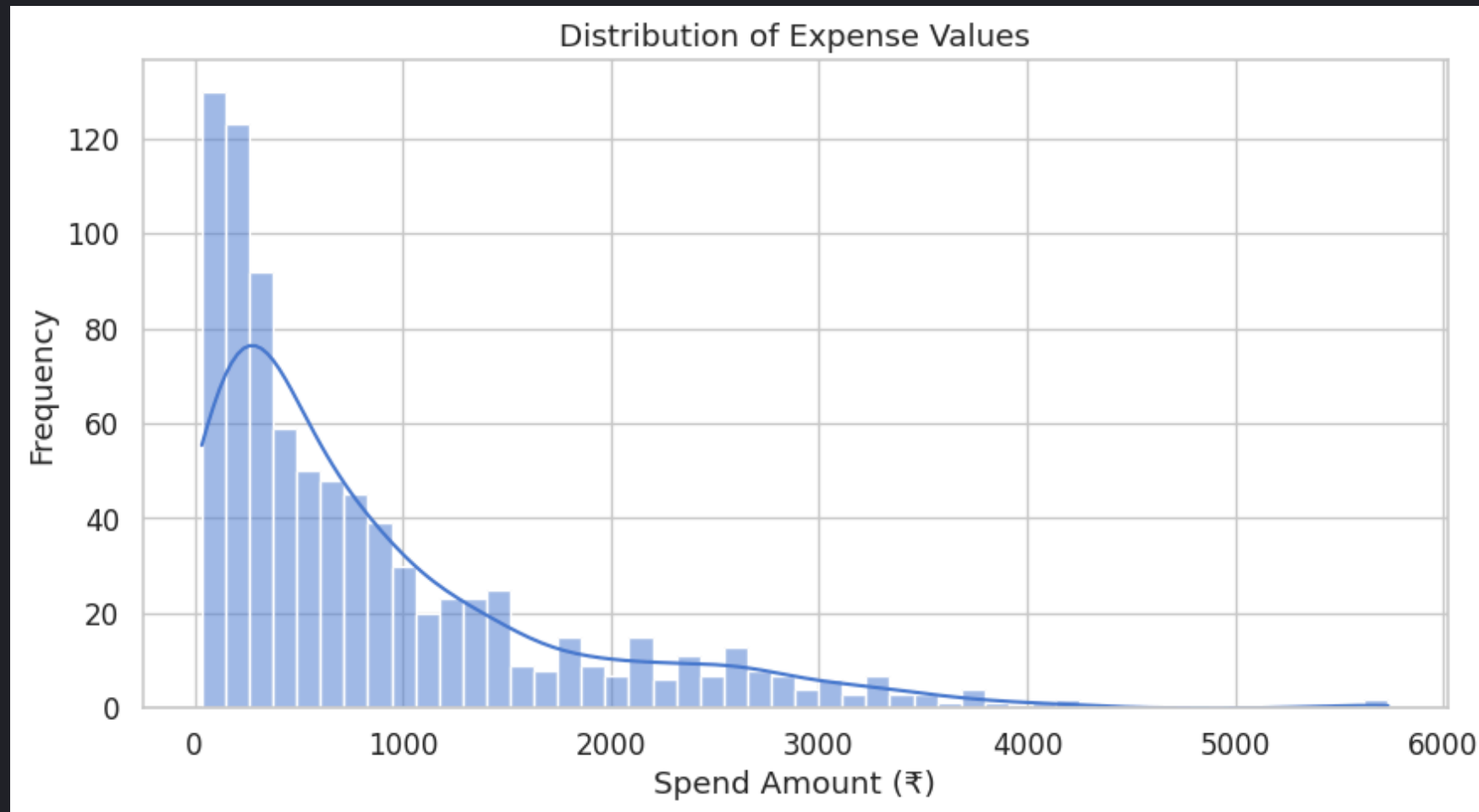


We observed that the "Others" category constitutes the largest proportion of spending at 28.2%, surpassing even individual top expenses like Rent (21.8%) and Education (17.7%). This significant "Others" slice suggests a substantial aggregation of smaller, potentially unidentified or unclassified expenses, indicating a prime area for deeper investigation to detect micro leakages.

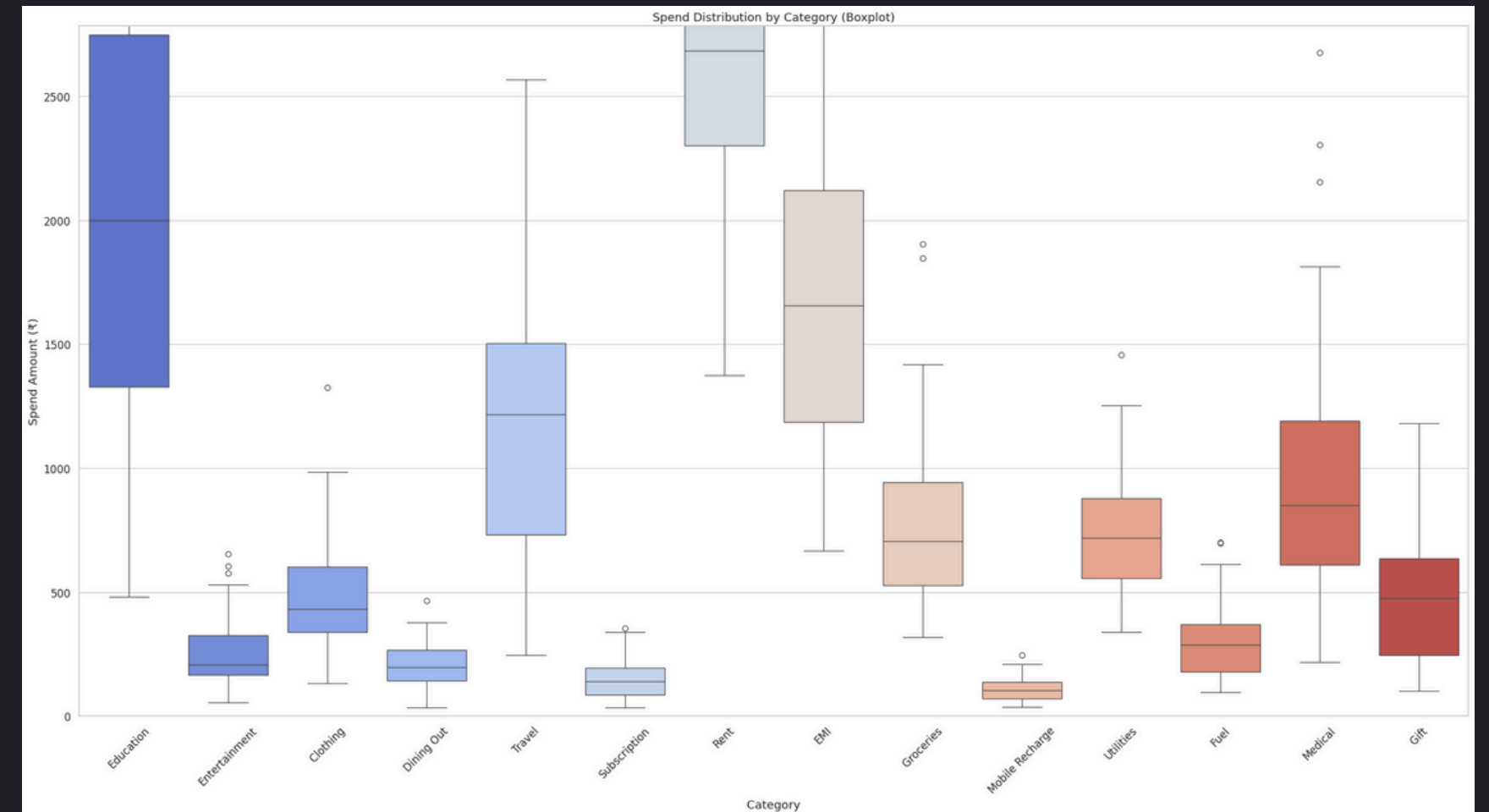


## EDA ANALYSIS - P2

Understanding our dataset more briefly using visuals.



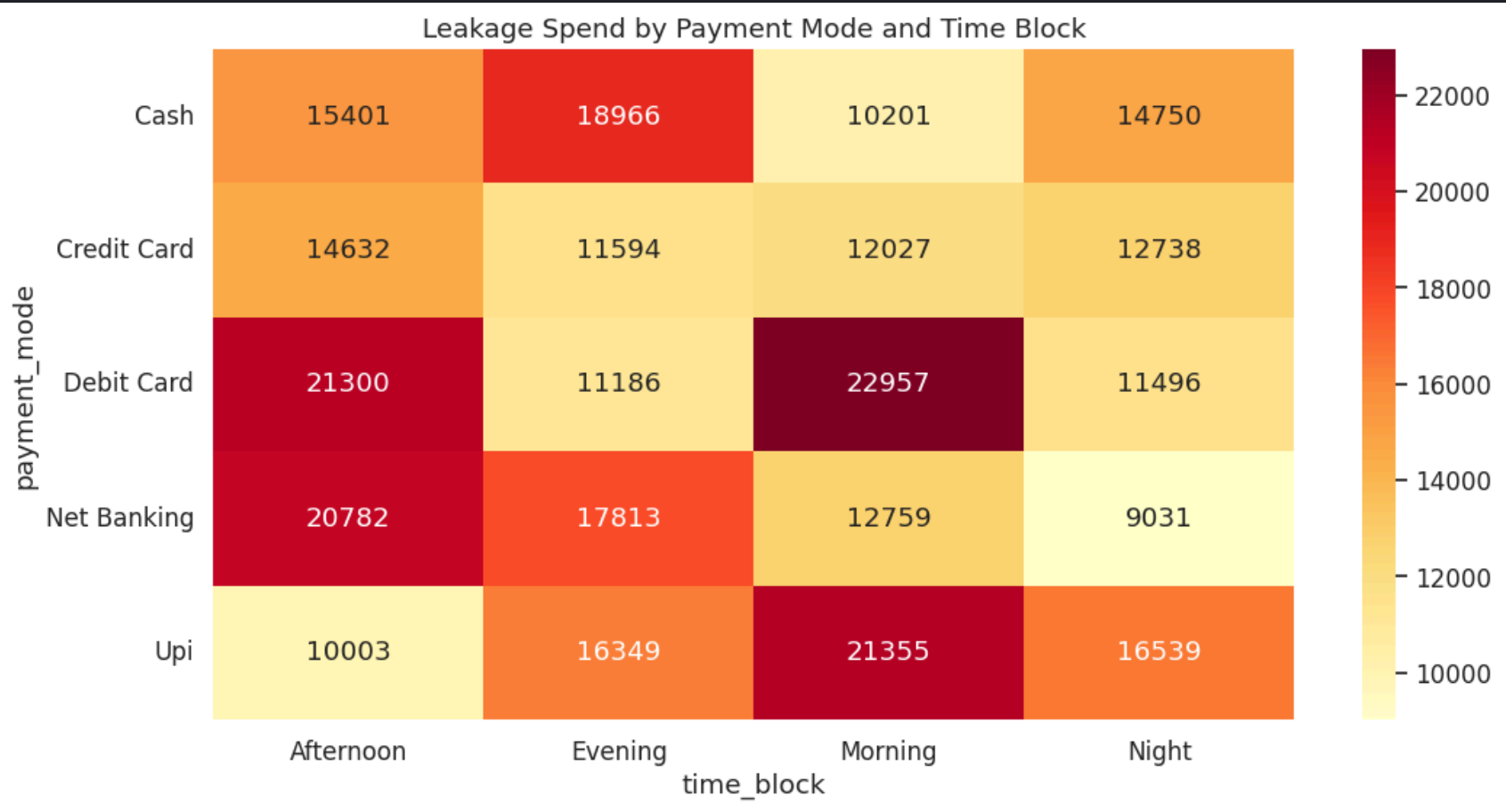
We observed that the "Distribution of Expense Values" shows a high frequency of smaller transactions, predominantly below ₹1000, with a peak around the ₹200-₹300 mark. This distribution strongly suggests that micro leakages are likely to manifest as numerous small, frequent expenditures, making them harder to notice without detailed analysis.



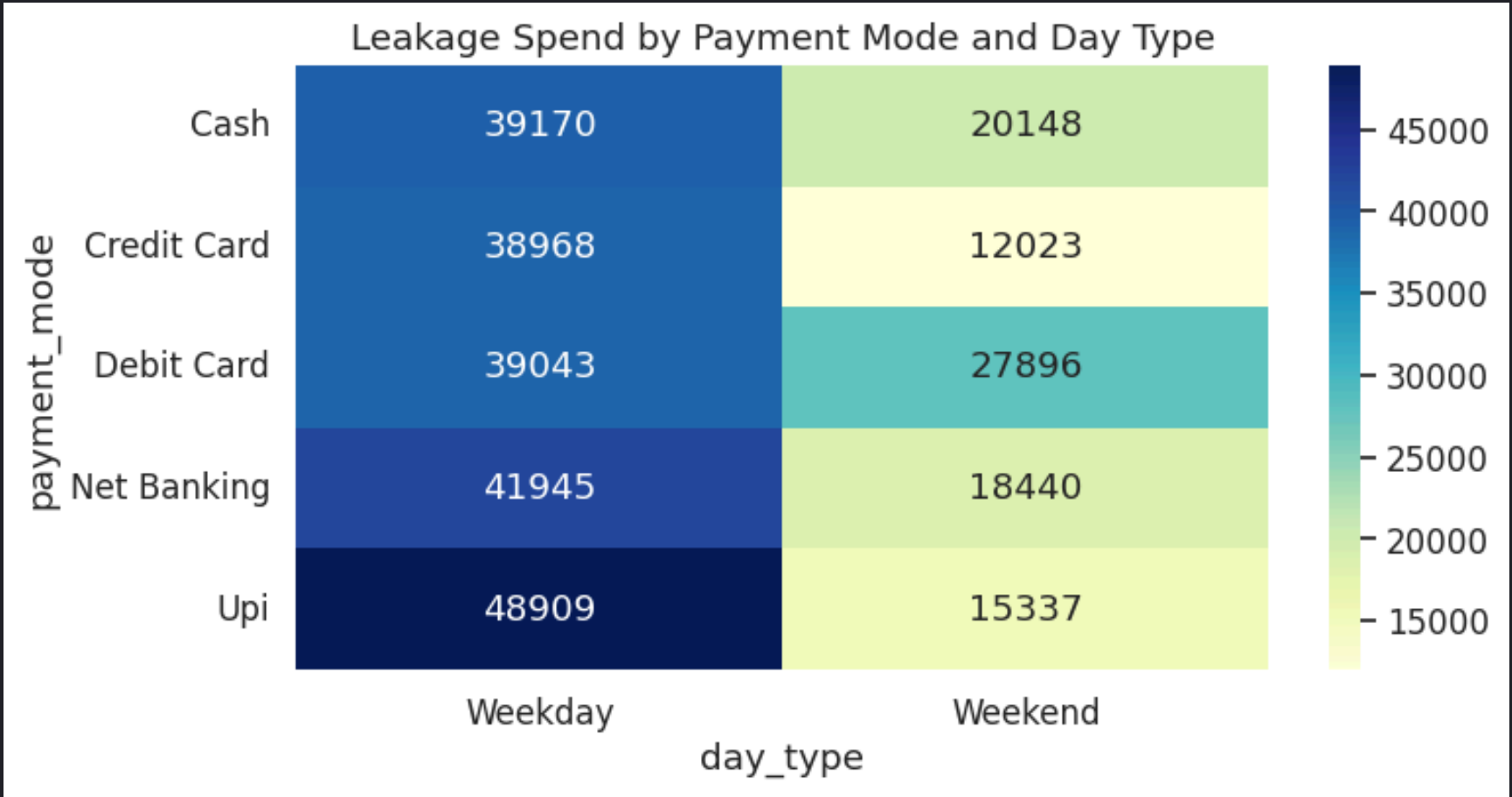
We observed significant variability in spending across different categories, with "Education" and "Rent" showing large median expenses and wider ranges, indicating substantial individual transactions. Conversely, categories like "Subscription" and "Mobile Recharge" exhibit much smaller medians and tighter interquartile ranges, confirming they are primary candidates for identifying recurring, small-value micro leakages that accumulate over time.

# EDA ANALYSIS - P3

Understanding our dataset more briefly using visuals.



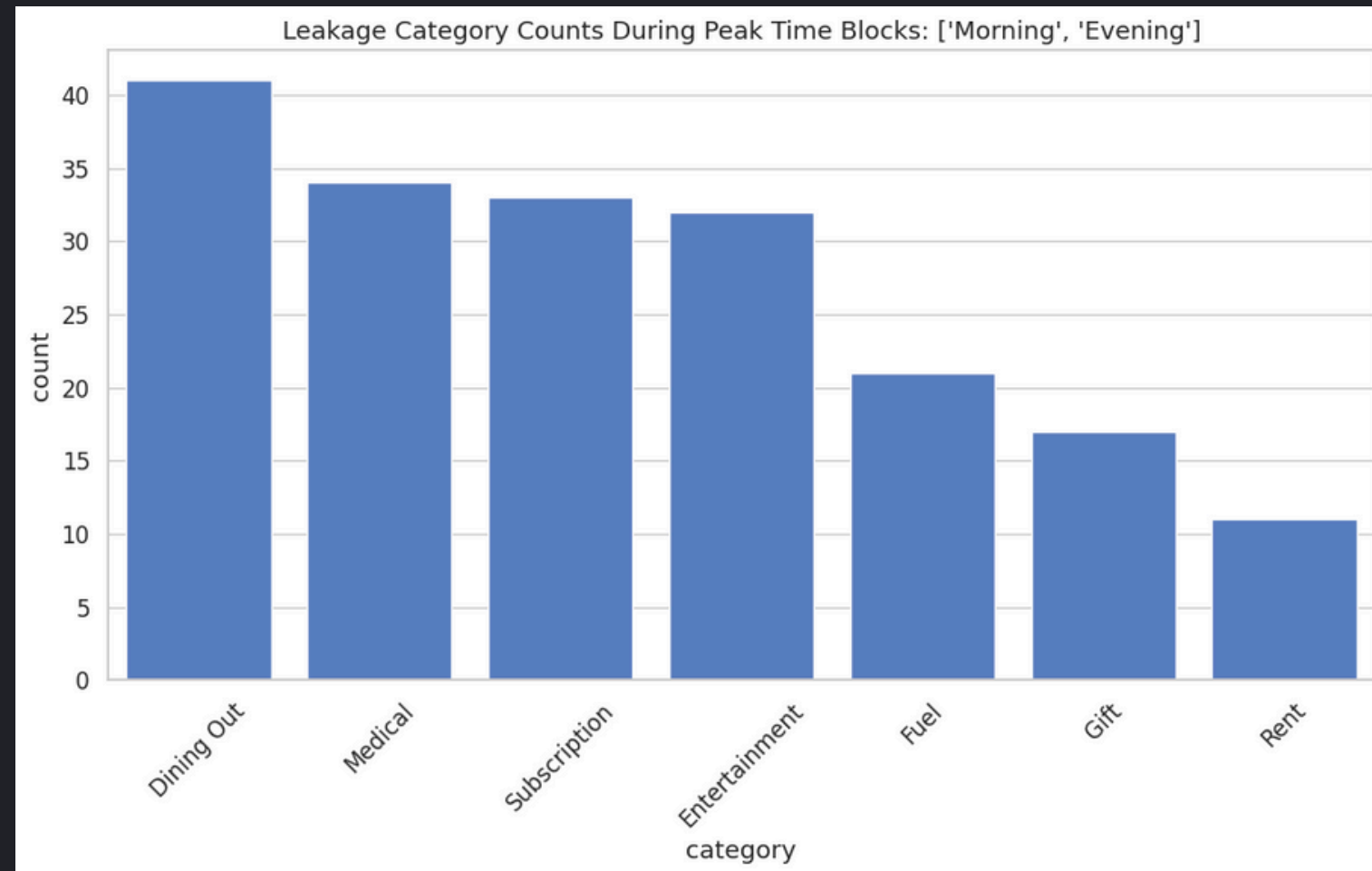
We observed that "Leakage Spend by Payment Mode and Time Block" highlights higher leakage amounts particularly in "Morning" through "Debit Card" (₹22,957) and "UPI" (₹21,355), and in "Evening" via "Cash" (₹18,966). This indicates specific patterns where micro leakages are more prevalent based on both the payment method used and the time of day, offering targeted areas for intervention.



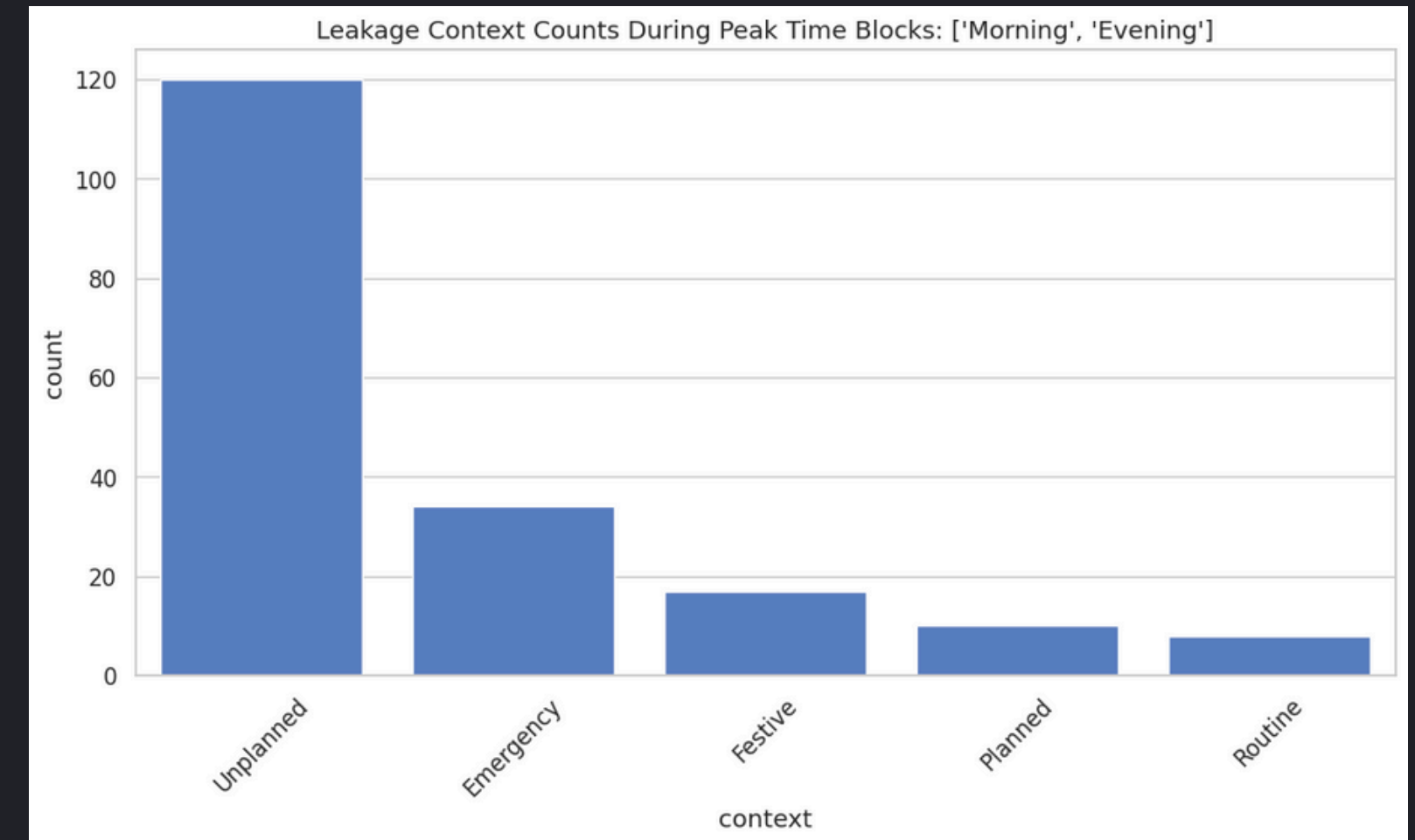
We observed that "Leakage Spend by Payment Mode and Day Type" reveals significantly higher micro leakages during "Weekdays" across all payment modes compared to "Weekends." Notably, UPI (₹48,909) and Net Banking (₹41,945) show the highest leakage amounts on weekdays, suggesting routine, often digital, small expenses are more prevalent during the work week.

## EDA ANALYSIS - P4

Understanding our dataset more briefly using visuals.



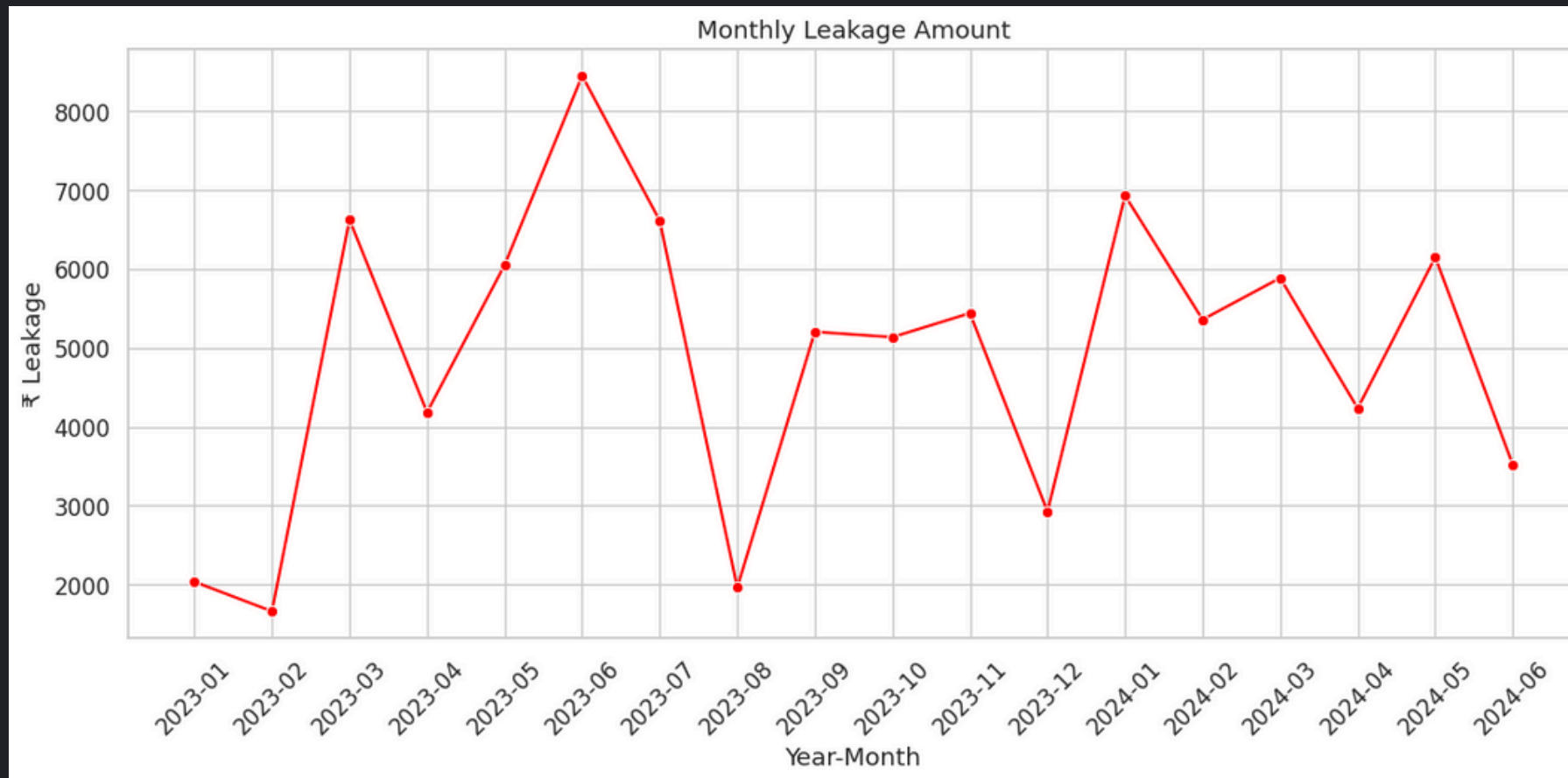
We observed that "Leakage Category Counts During Peak Time Blocks" highlights "Dining Out" as the most frequent source of micro leakages during morning and evening hours. Categories like "Medical" and "Subscription" also show significant counts, indicating that these specific areas are common recurring small expenses contributing to financial leaks during these peak times



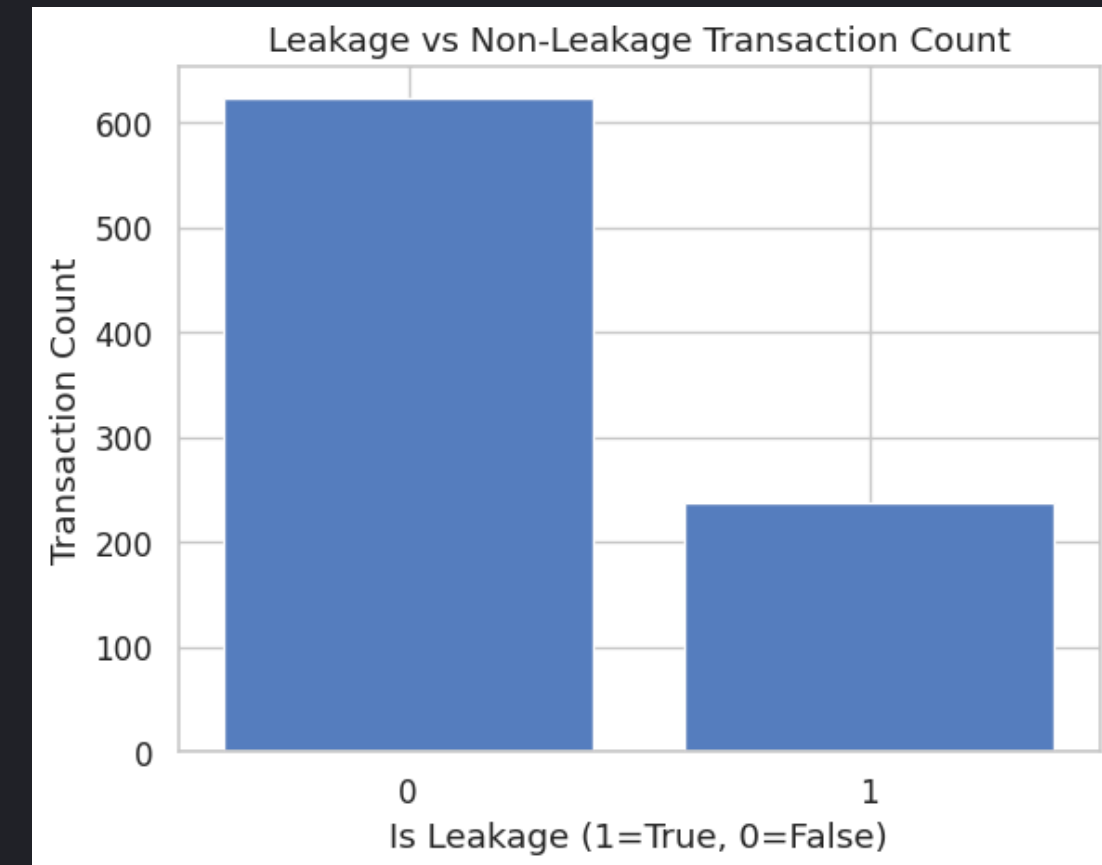
We observed that "Leakage Context Counts During Peak Time Blocks" overwhelmingly identifies "Unplanned" expenses as the most frequent context for micro leakages. This suggests that impulsive or unforeseen small expenditures are a significant contributor to financial leaks, far more than planned or routine small spends during morning and evening hours.

## EDA ANALYSIS - P5

Understanding our dataset more briefly using visuals.



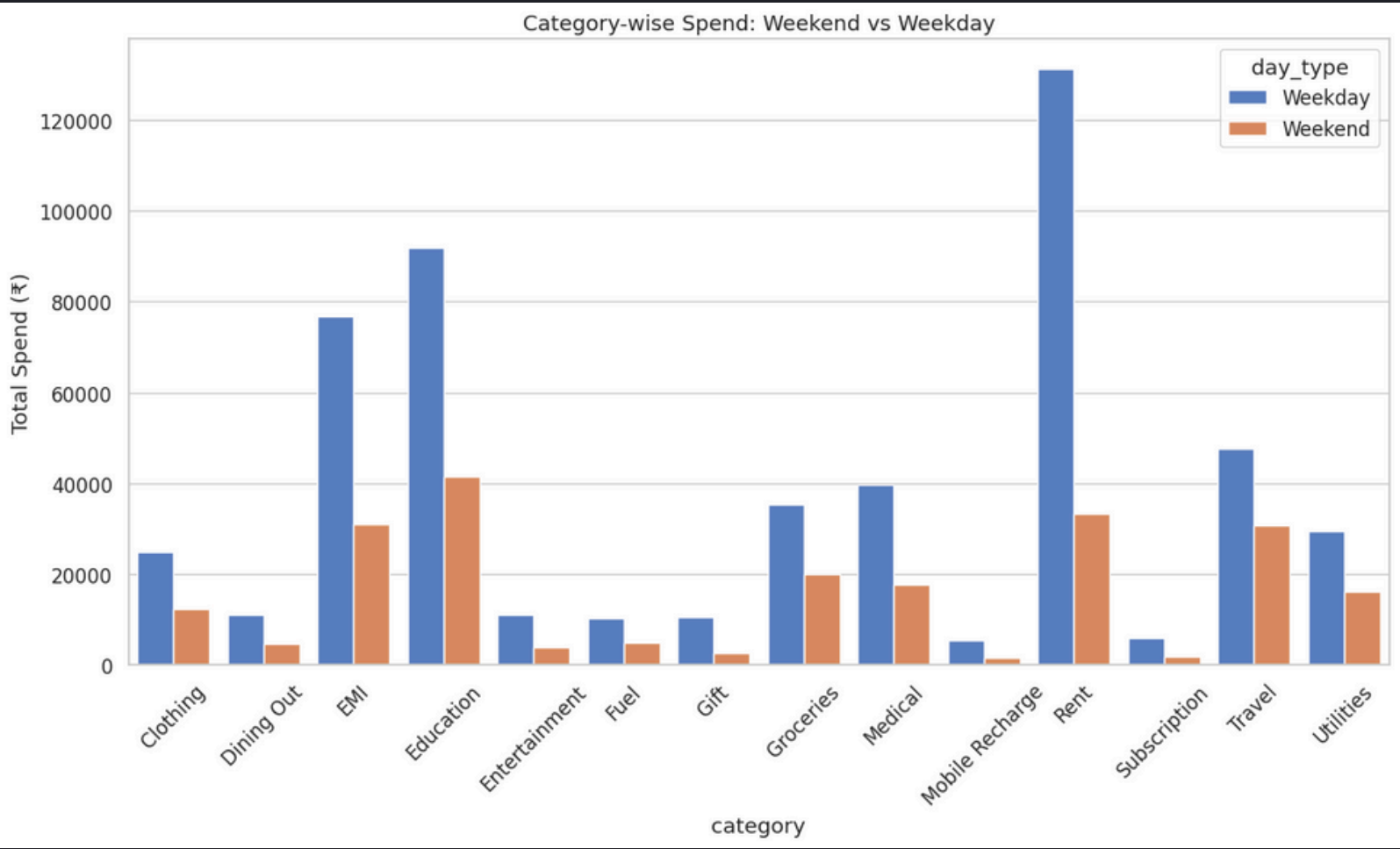
We observed that the "Monthly Leakage Amount" exhibits significant fluctuations, with notable peaks in May and July 2023, and again in January and May 2024. This trend indicates that micro leakages are not consistent but rather surge during specific months, suggesting potential seasonal or event-driven patterns that could be further investigated for root causes.



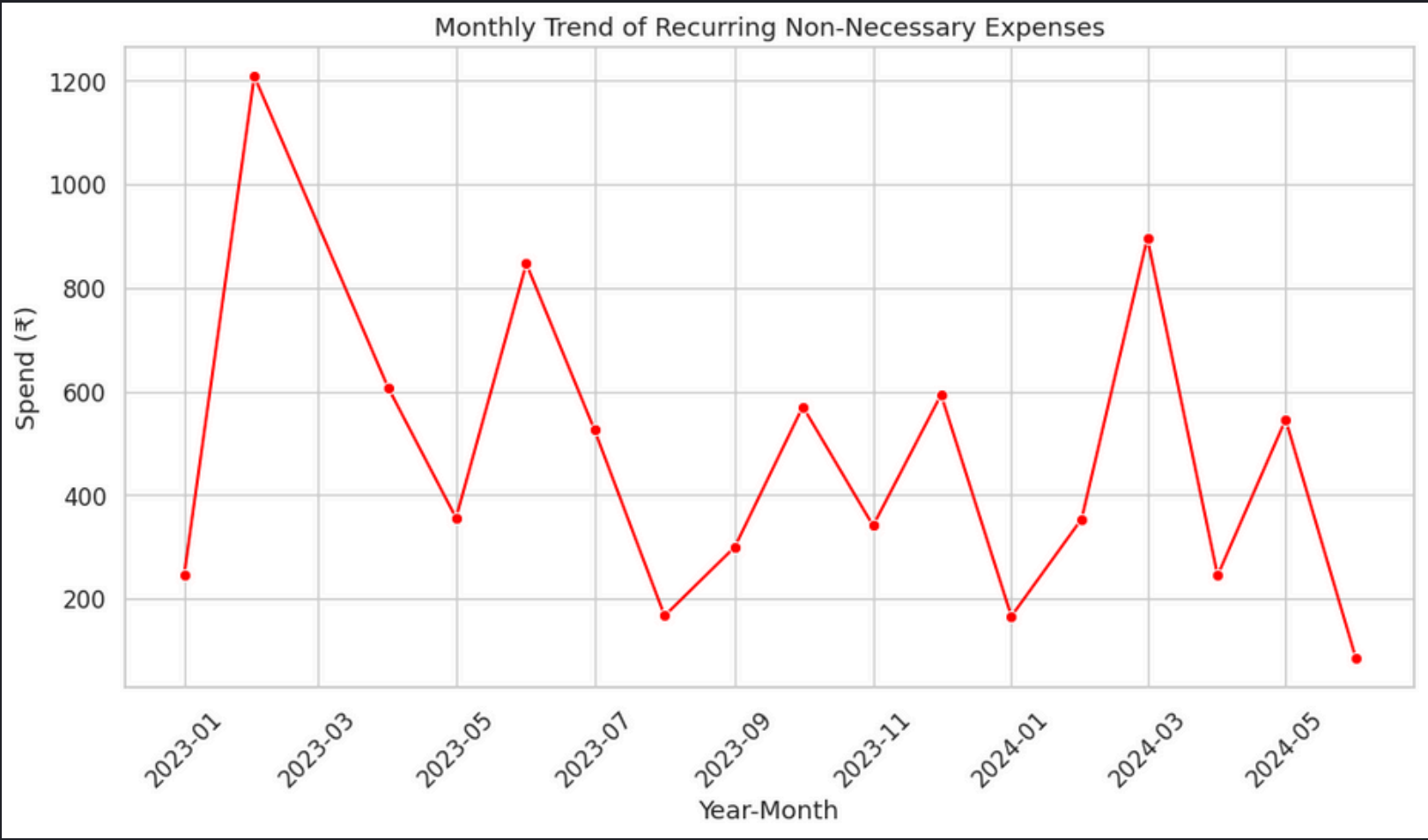
We observed that the "Leakage vs Non-Leakage Transaction Count" clearly shows a much higher count of non-leakage transactions (represented by 0) compared to leakage transactions (represented by 1). This class imbalance indicates that true micro leakages, while significant in their cumulative financial impact, are a smaller subset of the total transaction volume.

# EDA ANALYSIS - P6

Understanding our dataset more briefly using visuals.



We observed that "Category-wise Spend: Weekend vs Weekday" shows significantly higher spending during weekdays for major categories like Rent, EMI, Education, and Groceries. Conversely, "Dining Out" and "Entertainment" show higher spending on weekdays, which contradicts intuitive weekend spikes, suggesting that even leisure-related micro leakages might be more prevalent during the week.



We observed that the "Monthly Trend of Recurring Non-Necessary Expenses" shows significant volatility, with sharp peaks in February 2023 and March 2024. This highlights that recurring non-essential spending, a key component of micro leakages, is not constant but experiences considerable surges in specific months, indicating periods where attention to discretionary spending is crucial.



# ML Workflow

1. Data Preparation and Preprocessing - The data was cleaned, split 80/20 using stratification, and processed via a pipeline that standardized numerical features and one-hot encoded categorical ones.

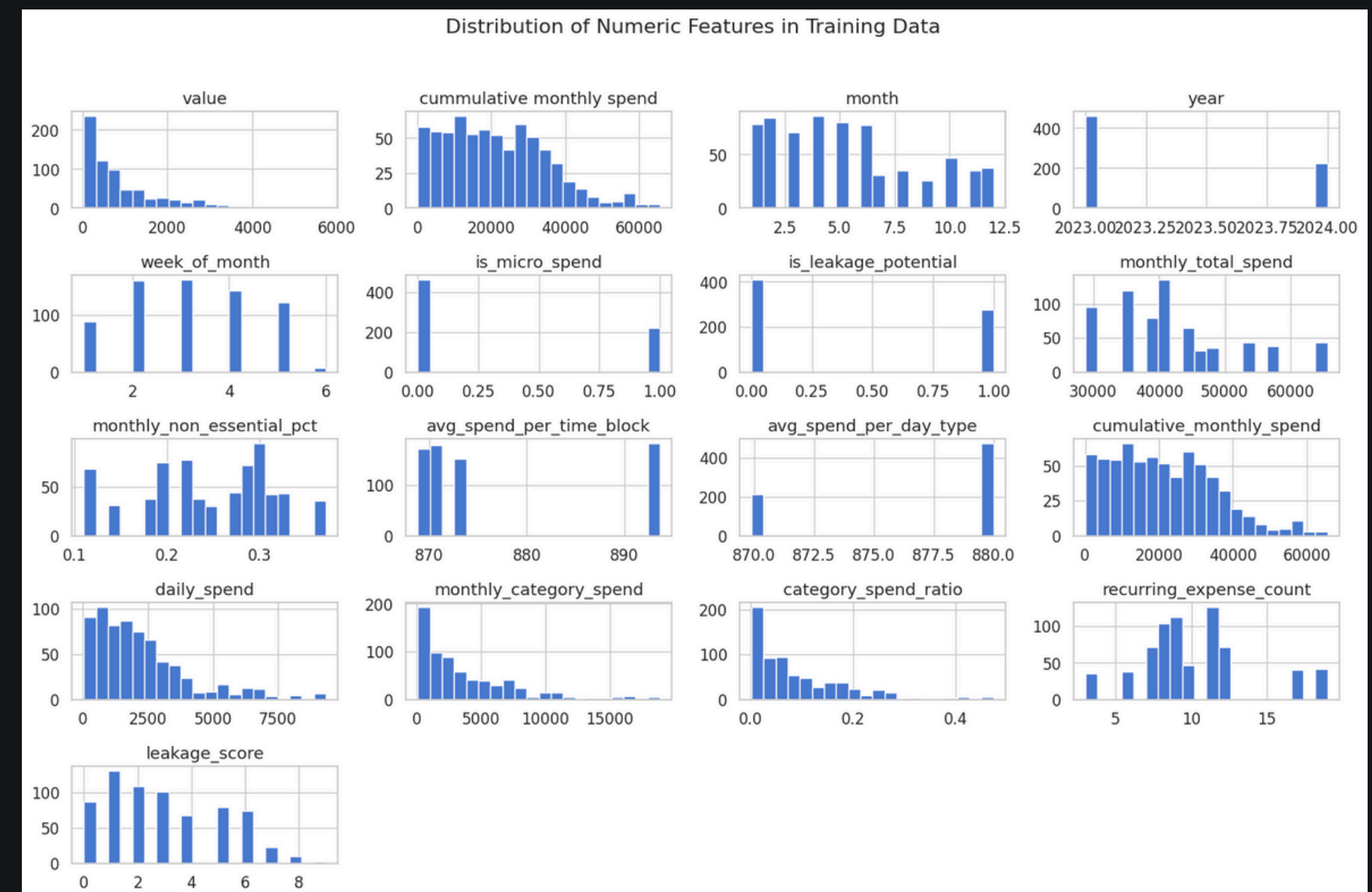
2. Modeling and Training Three model - Logistic Regression, Random Forest, and XGBoost—were integrated into individual pipelines with the preprocessor and trained on the dataset.

3. Performance Evaluation and Overfitting Analysis - Initial evaluation showed exceptionally high accuracy, but further analysis revealed slight overfitting, which can be resolved with a larger dataset in the future.

4. Hyperparameter Tuning and Final Model Selection - After tuning for the best F1-score using GridSearchCV, the Random Forest classifier was selected as the final, most robust model for the task.

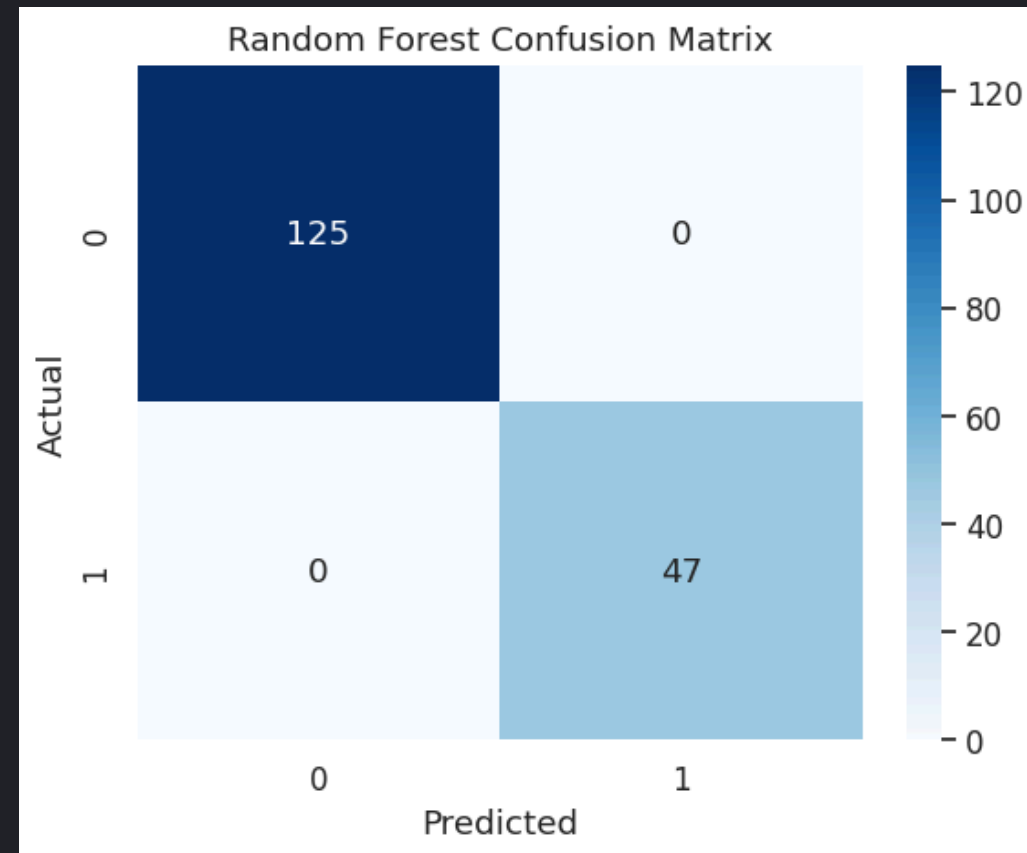
```
df['is_leakage'] = df['leakage_score'].apply(lambda x: 1 if x >= 5 else 0)
```

Target Column Creation based on Leakage Scoring Function

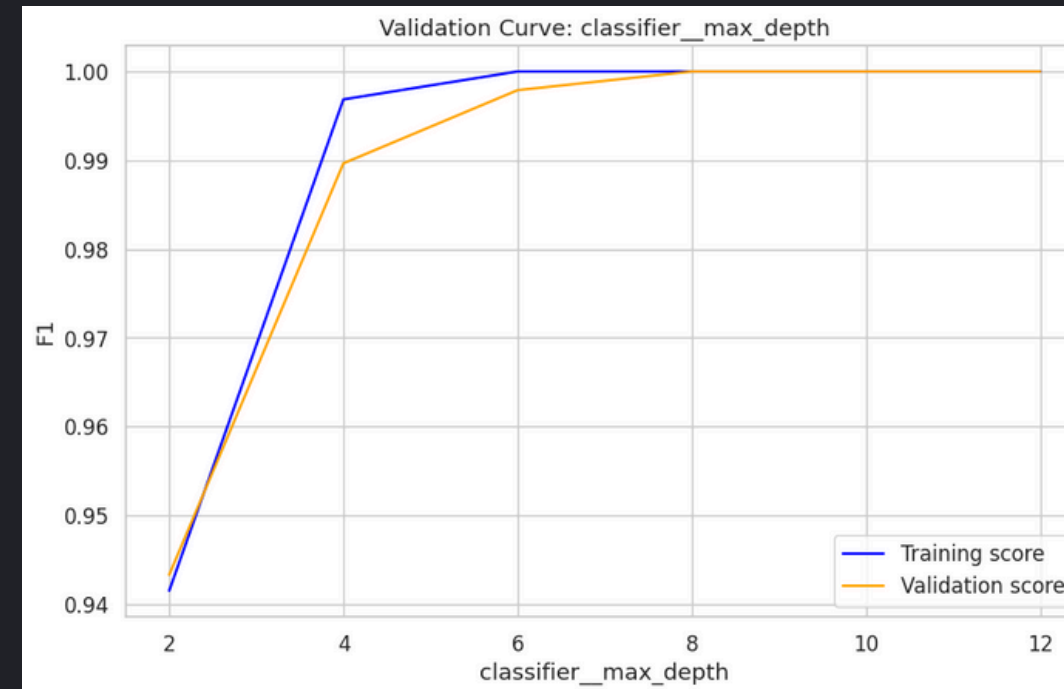


Numeric Value Distribution of the training data

# Confusion Matrix and Overfitting Analysis

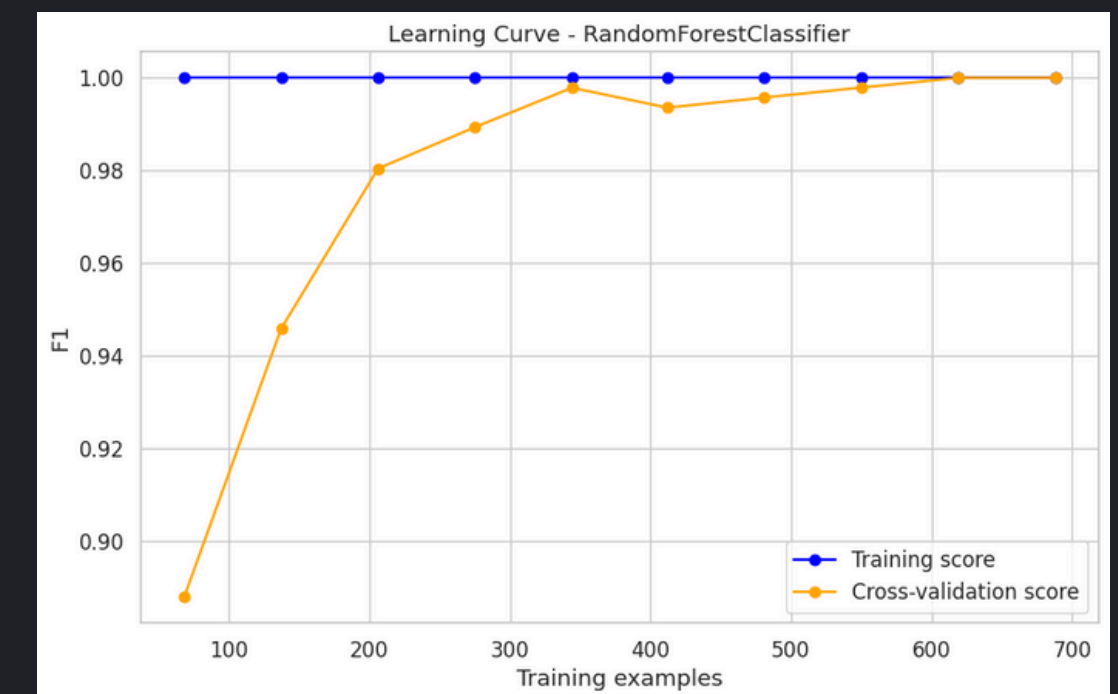


We observed a perfect classification from the Random Forest Confusion Matrix, accurately identifying 125 non-leakage and 47 leakage transactions with zero false positives or negatives. To validate its robustness, checks with validation and learning curves showed only slight overfitting. The perfect classification might have been due to less number of entries in the testing dataset - which may vary based on the size of the testing data.



We observed the Validation Curve shows training score quickly reaching 1.0, while the validation score lags slightly, indicating minimal overfitting. This small gap suggests the model perfectly fits training data but could generalize marginally less. Increased data or minor hyperparameter tuning can further reduce this slight overfitting.

We observed the Learning Curve shows the training score consistently at 1.0, while the cross-validation score improves but exhibits a slight, persistent gap. This indicates minor overfitting, which can be further reduced and model generalization improved by increasing the training dataset size.



# SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queires

```
-- 1. What is the total amount spent each month in 2023?
SELECT year, month, SUM(value) AS total_monthly_spend
FROM personal_spending
WHERE type = 'Expense' AND year = 2023
GROUP BY year, month
ORDER BY year, month;
```

year integer	month integer	total_monthly_spend bigint
2023	1	59720
2023	2	57772
2023	3	58248
2023	4	54674
2023	5	42533
2023	6	57655
2023	7	39359
2023	8	43006
2023	9	35463
2023	10	54198
2023	11	53227
2023	12	53336



```
-- 2. How many transactions are marked as 'Non-Essential' each month in 2024
-- for the first 6 months?
SELECT month, COUNT(*) AS non_essential_count
FROM personal_spending
WHERE Is_Necessary = 'No' AND year = 2023
GROUP BY year, month
ORDER BY year, month
LIMIT 6;
```

month integer	non_essential_count bigint
1	10
2	18
3	17
4	13
5	15
6	12



## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries

```
-- 3. What are the top 5 most frequent expense categories?  
SELECT category, COUNT(*) AS transaction_count  
FROM personal_spending  
GROUP BY category  
ORDER BY transaction_count DESC  
LIMIT 5;
```

category 	transaction_count 
text	bigint
Dining Out	76
Clothing	76
Groceries	71
Education	67
Mobile Recharge	65



```
-- 4. Which payment mode is most commonly used for micro spends?  
SELECT payment_mode, COUNT(*) AS micro_spend_count  
FROM personal_spending  
WHERE is_micro_spend = 1  
GROUP BY payment_mode  
ORDER BY micro_spend_count DESC;
```

payment_mode 	micro_spend_count 
text	bigint
Debit Card	22
Net Banking	16
Upi	12
Credit Card	11
Cash	11



## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries

```
-- 5. Compare average expense value for recurring vs non-recurring transactions.  
SELECT is_recurring, AVG(value) AS avg_spend  
FROM personal_spending  
WHERE type = 'Expense'  
GROUP BY is_recurring;
```

is_recurring 	avg_spend 
text	numeric
No	1073.6611694152923538
Yes	1059.1696969696969697

```
-- 6. Which day of the week has the highest average daily spend?  
SELECT day_of_week, AVG(daily_spend) AS avg_spend  
FROM personal_spending  
GROUP BY day_of_week  
ORDER BY avg_spend DESC;
```



day_of_week 	avg_spend 
text	numeric
Sunday	3322.9548872180451128
Thursday	2895.2222222222222222
Tuesday	2746.1615384615384615
Monday	2728.1487603305785124
Friday	2681.1111111111111111
Saturday	2653.8034188034188034
Wednesday	2285.7422680412371134





## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries

```
--7. Which three time blocks see the most leakage-prone transactions?  
SELECT time_block, COUNT(*) AS leakage_count  
FROM personal_spending  
WHERE is_leakage = 1  
GROUP BY time_block  
ORDER BY leakage_count DESC  
LIMIT 3;
```

time_block 	leakage_count 
text	bigint
Evening	50
Night	48
Morning	25



```
-- 10. Find users' top 3 non-essential categories  
--      that most frequently appear in salary weeks.  
SELECT category, COUNT(*) AS count  
FROM personal_spending  
WHERE is_salary_week = TRUE AND Is_Necessary = 'No'  
GROUP BY category  
ORDER BY count DESC  
LIMIT 3;
```

category 	count 
text	bigint
Education	8
Subscription	7
Clothing	7

## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries

```
-- 8. Which contexts contribute most to high leakage scores
-- (above 75th percentile)?
WITH threshold AS (
    SELECT PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY leakage_score)
    AS cutoff FROM personal_spending
),
high_leaks AS (
    SELECT * FROM personal_spending, threshold
    WHERE leakage_score > threshold.cutoff
)
SELECT context, COUNT(*) AS high_leak_count
FROM high_leaks
GROUP BY context
ORDER BY high_leak_count DESC;
```

context 	high_leak_count 
Emergency	51
Unplanned	43
Festive	36
Routine	10
Seasonal	5

## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries



```
-- 9. Estimate monthly savings by cutting top 10% leakage-scored
--     expenses in half.
WITH ranked AS (
    SELECT *, NTILE(10) OVER (ORDER BY leakage_score DESC) AS score_decile
    FROM personal_spending
)
SELECT month, ROUND(SUM(value) * 0.5) AS potential_savings
FROM ranked
WHERE score_decile = 1
GROUP BY month
ORDER BY month;
```

month integer	potential_savings numeric
1	6091
2	5061
3	9845
4	4022
5	3785
6	4138
7	2987
8	2283
9	126
10	1315
11	695
12	2586

## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries

```
-- 11. Track average leakage score by payment mode to assess risky channels.  
SELECT payment_mode, AVG(leakage_score) AS avg_leakage_score  
FROM personal_spending  
GROUP BY payment_mode  
ORDER BY avg_leakage_score DESC;
```

payment_mode 	avg_leakage_score 
text	numeric
Debit Card	2.8324324324324324
Cash	2.8031914893617021
Net Banking	2.7947019867549669
Credit Card	2.7708333333333333
Upi	2.7195121951219512

## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries

```
-- 12. Identify Top Leakage Contexts by Month with Running Rank
WITH monthly_leakage AS (
    SELECT
        month,
        context,
        SUM(value) AS leakage_sum
    FROM personal_spending
    WHERE is_leakage = 1
    GROUP BY month, context
),
ranked_leaks AS (
    SELECT
        month,
        context,
        leakage_sum,
        RANK() OVER (
            PARTITION BY month
            ORDER BY leakage_sum DESC
        ) AS leakage_rank
    FROM monthly_leakage
)
SELECT *
FROM ranked_leaks
WHERE leakage_rank <= 3
ORDER BY month, leakage_rank
LIMIT 18 -- for just first 6 months;
```

month integer	context text	leakage_sum bigint	leakage_rank bigint
1	Emergency	9990	1
1	Festive	4241	2
1	Routine	1542	3
2	Unplanned	8920	1
2	Festive	5911	2
2	Emergency	2073	3
3	Emergency	11245	1
3	Unplanned	9001	2
3	Routine	4167	3
4	Unplanned	6409	1
4	Emergency	4860	2
4	Festive	4284	3
5	Emergency	7907	1
5	Festive	6908	2
5	Unplanned	3514	3
6	Emergency	7063	1
6	Unplanned	5196	2
6	Festive	4332	3



## SQL Queries

Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries





```
-- 13.Flag months where leakage significantly jumps right after salary arrives.
WITH monthly_stats AS (
    SELECT
        month,
        is_salary_week,
        SUM(value) FILTER (WHERE is_leakage = 1) AS total_leakage
    FROM personal_spending
    GROUP BY month, is_salary_week
),
leak_jump AS (
    SELECT
        month,
        total_leakage,
        LAG(total_leakage) OVER (ORDER BY month) AS prev_leakage
    FROM monthly_stats
    WHERE is_salary_week = TRUE
)
SELECT
    month,
    total_leakage,
    prev_leakage,
    ROUND((total_leakage - prev_leakage) * 100.0 / prev_leakage, 2) AS pct_jump
FROM leak_jump
WHERE prev_leakage IS NOT NULL
    AND total_leakage > prev_leakage;
```

month integer	total_leakage bigint	prev_leakage bigint	pct_jump numeric
2	3557	3204	11.02
3	4182	3557	17.57
5	1318	231	470.56
6	5078	1318	285.28
8	2666	1976	34.92

## SQL Queries

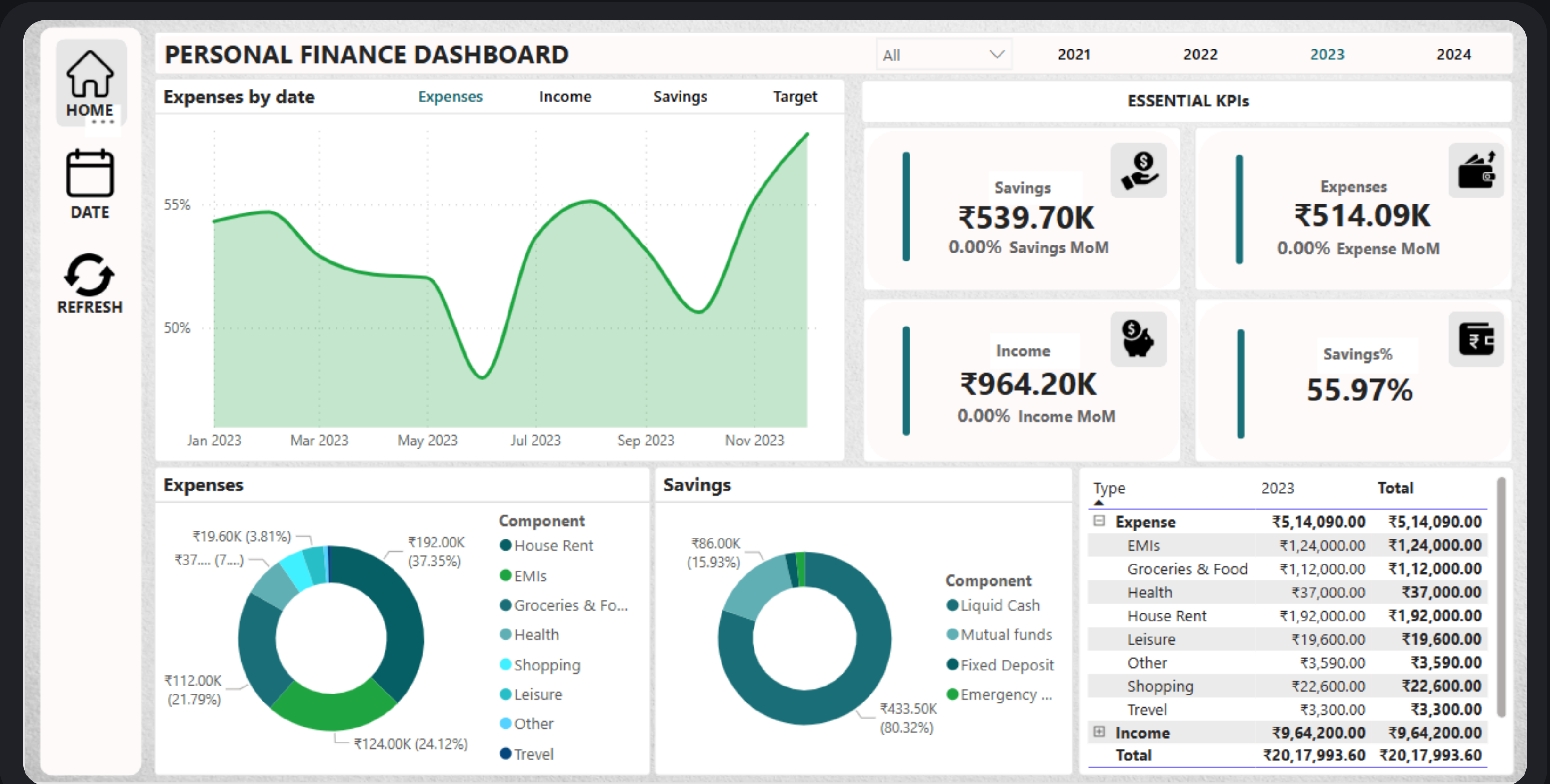
Understanding our final data that we extracted after model training and feature engineering from python and now query analysis of the data is done using SQL queries

```
--14. Find which payment channels have highest leakage rate per transaction.
WITH mode_counts AS (
  SELECT
    payment_mode,
    COUNT(*) AS total_txn,
    SUM(CASE WHEN is_leakage = 1 THEN 1 ELSE 0 END) AS leakage_txn
  FROM personal_spending
  GROUP BY payment_mode
),
mode_rate AS (
  SELECT
    payment_mode,
    leakage_txn,
    total_txn,
    ROUND(leakage_txn * 100.0 / total_txn, 2) AS leakage_rate_pct
  FROM mode_counts
)
SELECT *
FROM mode_rate
ORDER BY leakage_rate_pct DESC;
```

payment_mode 	leakage_txn 	total_txn 	leakage_rate_pct 
text	bigint	bigint	numeric
Cash	36	188	19.15
Net Banking	28	151	18.54
Credit Card	25	144	17.36
Debit Card	32	185	17.30
Upi	24	164	14.63

# Personal Finance Tracking Dashboard

The dataset was synthetically generated to simulate real-world personal spending behavior across multiple categories and time blocks. It includes 1000+ transaction records with attributes relevant to financial analysis and leak detection.



## Final Business Insights

- Key outcomes include identifying that 4%-5% of total transactions are attributed to micro leakage.
- Primary sources of micro leakage were highlighted as dining out, medical emergencies, and fuel.
- Additional insights revealed temporal and behavioral trends such as "Evening based" or "Weekday based" spending patterns.
- SQL analysis revealed that "Dining Out" and "Clothing" were the most frequent expense categories.
- "Debit Card" and "Net Banking" were identified as frequently used payment modes for micro spends.
- "Emergency," "Unplanned," and "Festive" contexts contributed most to high leakage scores.
- Cash, Net Banking, and Credit Card showed the highest leakage rates per transaction.

## Challenges & Learnings

- Defining "Micro Leakage": Establishing clear criteria for what constitutes micro leakage was a key challenge, addressed by a leakage scoring system based on six weighted conditions (e.g., micro spend, non-necessary, impulsive contexts, recurring & unplanned, impulse-prone time/day).
- Class Imbalance: The dataset showed a much higher count of non-leakage transactions compared to leakage transactions.
- Overfitting: Initial analysis showed slight overfitting, which can be resolved with a larger dataset or minor hyperparameter tuning.

## Future Scope

- Add automation for leakage alerts.
- Incorporate real-time dashboards for live transaction data.
- Extend the model to corporate finance data for broader application.

**Thank You**