**A blue and white logo

AI-generated content may be incorrect.SQL PRETRAINING SESSION 5**

**V1: PORTFOLIO TRENDS & TIME INTELLIGENCE FOUNDATIONS**

**Subtitle**: Temporal Analysis, Forecasting & Predictive Business Intelligence

**Registered: Krishnav Tech**

**Initiative: Skill AI**

**Context:** This pretraining document is part of the **Skill AI Data Analyst Track**.  
It focuses on mastering time intelligence and temporal analytics for portfolio monitoring, crisis detection, and future trend prediction.  
You’ll work with advanced date/time functions, perform period-over-period comparisons, conduct cohort analysis, and recognize seasonal or cyclical patterns in business performance.  
By the end, you’ll have the skills to uncover when portfolio issues began, identify trends in risk or growth, and create actionable, time-driven business intelligence strategies.

**Organization:** Krishnav Tech | Skill AI Path

# TABLE OF CONTENTS

1. Session Overview
2. Learning Objectives
3. Part 1: ADVANCED DATE/TIME FUNCTIONS
4. Part 2: PERIOD-OVER-PERIOD ANALYSIS
5. Part 3: COHORT ANALYSIS & CUSTOMER LIFECYCLE
6. Part 4: TREND ANALYSIS & FORECASTING
7. Conclusion & Next Steps

# 1. SESSION OVERVIEW

**Session Focus:**  
Portfolio Trends & Time Intelligence — “When did problems start, and where are we heading?”

**Time Investment:**  
8–9 hours of focused learning

**Business Context:**  
EduFin’s leadership wants a clear picture of portfolio health over time to detect crises early and anticipate future performance.  
In this session, you’ll apply advanced temporal analytics to track loan portfolio trends, identify critical turning points, forecast outcomes, and reveal seasonal risk patterns.  
By the end, you’ll know exactly when portfolio risks began increasing, how current trends compare to historical benchmarks, and what the data predicts for the next quarter.

**2. LEARNING OBJECTIVES**

You will master:

* ✅ Advanced date/time functions and calculations
* ✅ Month-over-month (MoM), quarter-over-quarter (QoQ), and year-over-year (YoY) comparisons
* ✅ Cohort analysis to track customer groups over time
* ✅ Time-based window functions for rolling and cumulative metrics
* ✅ Seasonal trend detection and cyclical pattern analysis
* ✅ Forecasting and projection techniques for portfolio performance
* ✅ Translating time-based insights into proactive business strategies

# PART 1: ADVANCED DATE/TIME FUNCTIONS

## 1.1 Date Manipulation and Extraction

**Purpose:**To demonstrate how SQL Server’s date and time functions can be used to extract, format, and calculate meaningful time-based metrics from transactional data. This includes breaking down dates into components (year, month, day, quarter, weekday), creating human-readable period labels, calculating start/end of periods, and measuring time durations. The goal is to enable accurate time-series reporting, trend analysis, and loan lifecycle categorization.

**Learning Goal:**

* Learn to extract date parts such as year, month, day, quarter, weekday name, and day of year for reporting.
* Understand how to generate standard and custom period labels for dashboards.
* Learn to calculate period boundaries using functions like DATEFROMPARTS and EOMONTH.
* Gain skills in measuring durations in days, months, and years with DATEDIFF.
* Apply business logic to classify loans into lifecycle stages based on their age.

SELECT

loan\_id,

customer\_id,

disbursement\_date,

-- Basic date extraction

YEAR(disbursement\_date) as loan\_year,

MONTH(disbursement\_date) as loan\_month,

DAY(disbursement\_date) as loan\_day,

DATENAME(MONTH, disbursement\_date) as month\_name,

DATENAME(WEEKDAY, disbursement\_date) as weekday\_name,

-- Advanced period calculations

DATEPART(QUARTER, disbursement\_date) as loan\_quarter,

DATEPART(WEEK, disbursement\_date) as week\_of\_year,

DATEPART(DAYOFYEAR, disbursement\_date) as day\_of\_year,

-- Period grouping

CONCAT('Q', DATEPART(QUARTER, disbursement\_date), '-', YEAR(disbursement\_date)) as quarter\_label,

CONCAT(DATENAME(MONTH, disbursement\_date), ' ', YEAR(disbursement\_date)) as month\_year\_label,

-- First/last day calculations

DATEFROMPARTS(YEAR(disbursement\_date), MONTH(disbursement\_date), 1) as month\_start,

EOMONTH(disbursement\_date) as month\_end,

DATEFROMPARTS(YEAR(disbursement\_date), 1, 1) as year\_start,

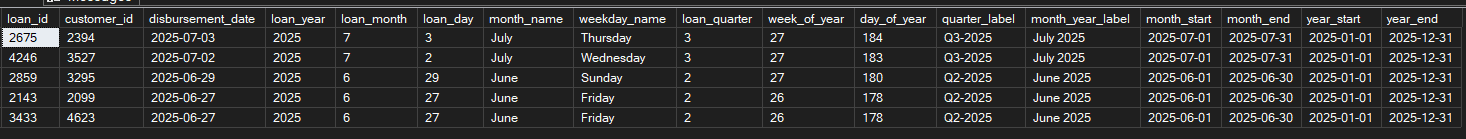
DATEFROMPARTS(YEAR(disbursement\_date), 12, 31) as year\_end

FROM loans

WHERE disbursement\_date IS NOT NULL

ORDER BY disbursement\_date DESC;

**Output:**

****

**-- Loan aging and maturity analysis**

SELECT

loan\_id,

customer\_id,

disbursement\_date,

loan\_amount,

loan\_status,

-- Age calculations

DATEDIFF(DAY, disbursement\_date, GETDATE()) as loan\_age\_days,

DATEDIFF(MONTH, disbursement\_date, GETDATE()) as loan\_age\_months,

DATEDIFF(YEAR, disbursement\_date, GETDATE()) as loan\_age\_years,

-- Business day calculations

DATEDIFF(WEEKDAY, disbursement\_date, GETDATE()) as business\_days\_elapsed,

-- Maturity analysis (assuming 5-year term)

DATEADD(YEAR, 5, disbursement\_date) as expected\_maturity\_date,

DATEDIFF(MONTH, GETDATE(), DATEADD(YEAR, 5, disbursement\_date)) as months\_to\_maturity,

-- Lifecycle categorization

CASE

WHEN DATEDIFF(MONTH, disbursement\_date, GETDATE()) <= 6 THEN 'New Loan (0-6M)'

WHEN DATEDIFF(MONTH, disbursement\_date, GETDATE()) <= 18 THEN 'Early Stage (6M-18M)'

WHEN DATEDIFF(MONTH, disbursement\_date, GETDATE()) <= 36 THEN 'Mid Stage (18M-3Y)'

WHEN DATEDIFF(MONTH, disbursement\_date, GETDATE()) <= 60 THEN 'Mature (3Y-5Y)'

ELSE 'Extended Term (>5Y)'

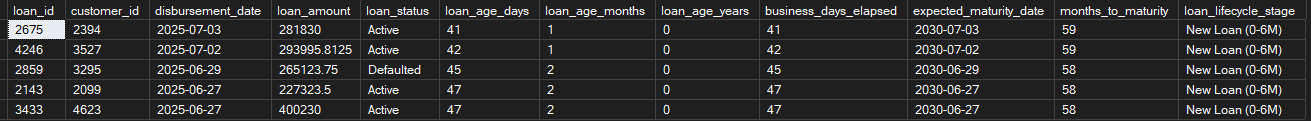
END as loan\_lifecycle\_stage

FROM loans

WHERE disbursement\_date IS NOT NULL

ORDER BY disbursement\_date DESC;

**Output:**

****

## 1.2 Period Boundary and Range Calculations

**Purpose:**To build reusable, dynamic date ranges for reporting and perform time-based comparisons in loan metrics. This section focuses on defining period boundaries (e.g., current month, previous month, current quarter, previous quarter, last 6/12 months) and applying them in conditional aggregations to track performance changes over time.

**Learning Goal:**

* Learn to dynamically calculate start and end dates for standard reporting periods.
* Understand how to define rolling periods like “last 12 months” and “last 6 months” for trend monitoring.
* Learn to join period definitions with loan data to calculate period-specific metrics such as count, volume, and growth.
* Develop skills in calculating both absolute and percentage growth between two periods.
* Gain experience using conditional aggregation (CASE inside COUNT/SUM) for time-based KPI calculations.

**-- Dynamic period range creation**

DECLARE @analysis\_date DATE = GETDATE();

SELECT

'Current Analysis Date' as period\_type,

@analysis\_date as period\_date

UNION ALL

SELECT 'Current Month Start', DATEFROMPARTS(YEAR(@analysis\_date), MONTH(@analysis\_date), 1)

UNION ALL

SELECT 'Current Month End', EOMONTH(@analysis\_date)

UNION ALL

SELECT 'Previous Month Start', DATEFROMPARTS(YEAR(DATEADD(MONTH, -1, @analysis\_date)), MONTH(DATEADD(MONTH, -1, @analysis\_date)), 1)

UNION ALL

SELECT 'Previous Month End', EOMONTH(DATEADD(MONTH, -1, @analysis\_date))

UNION ALL

SELECT 'Current Quarter Start', DATEFROMPARTS(YEAR(@analysis\_date), (DATEPART(QUARTER, @analysis\_date) - 1) \* 3 + 1, 1)

UNION ALL

SELECT 'Previous Quarter Start', DATEFROMPARTS(YEAR(DATEADD(QUARTER, -1, @analysis\_date)), (DATEPART(QUARTER, DATEADD(QUARTER, -1, @analysis\_date)) - 1) \* 3 + 1, 1)

UNION ALL

SELECT 'Year Start', DATEFROMPARTS(YEAR(@analysis\_date), 1, 1)

UNION ALL

SELECT 'Previous Year Start', DATEFROMPARTS(YEAR(@analysis\_date) - 1, 1, 1)

UNION ALL

SELECT 'Last 12 Months Start', DATEADD(MONTH, -12, @analysis\_date)

UNION ALL

SELECT 'Last 6 Months Start', DATEADD(MONTH, -6, @analysis\_date);

A screenshot of a black screen

AI-generated content may be incorrect.

**-- Period-based loan analysis**

WITH period\_definitions AS (

SELECT

DATEFROMPARTS(YEAR(GETDATE()), MONTH(GETDATE()), 1) as current\_month\_start,

EOMONTH(GETDATE()) as current\_month\_end,

DATEFROMPARTS(YEAR(DATEADD(MONTH, -1, GETDATE())), MONTH(DATEADD(MONTH, -1, GETDATE())), 1) as prev\_month\_start,

EOMONTH(DATEADD(MONTH, -1, GETDATE())) as prev\_month\_end,

DATEFROMPARTS(YEAR(GETDATE()), (DATEPART(QUARTER, GETDATE()) - 1) \* 3 + 1, 1) as current\_quarter\_start,

DATEFROMPARTS(YEAR(DATEADD(QUARTER, -1, GETDATE())), (DATEPART(QUARTER, DATEADD(QUARTER, -1, GETDATE())) - 1) \* 3 + 1, 1) as prev\_quarter\_start,

DATEADD(MONTH, -12, GETDATE()) as last\_12\_months\_start

)

SELECT

-- Current month metrics

COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.current\_month\_start AND pd.current\_month\_end THEN 1 END) as current\_month\_loans,

SUM(CASE WHEN l.disbursement\_date BETWEEN pd.current\_month\_start AND pd.current\_month\_end THEN l.loan\_amount ELSE 0 END) as current\_month\_volume,

-- Previous month metrics

COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.prev\_month\_start AND pd.prev\_month\_end THEN 1 END) as prev\_month\_loans,

SUM(CASE WHEN l.disbursement\_date BETWEEN pd.prev\_month\_start AND pd.prev\_month\_end THEN l.loan\_amount ELSE 0 END) as prev\_month\_volume,

-- Last 12 months metrics

COUNT(CASE WHEN l.disbursement\_date >= pd.last\_12\_months\_start THEN 1 END) as last\_12m\_loans,

SUM(CASE WHEN l.disbursement\_date >= pd.last\_12\_months\_start THEN l.loan\_amount ELSE 0 END) as last\_12m\_volume,

-- Growth calculations

COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.current\_month\_start AND pd.current\_month\_end THEN 1 END) -

COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.prev\_month\_start AND pd.prev\_month\_end THEN 1 END) as month\_over\_month\_loan\_growth,

-- Growth percentages

CASE

WHEN COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.prev\_month\_start AND pd.prev\_month\_end THEN 1 END) > 0

THEN ROUND(

(COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.current\_month\_start AND pd.current\_month\_end THEN 1 END) -

COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.prev\_month\_start AND pd.prev\_month\_end THEN 1 END)) \* 100.0 /

COUNT(CASE WHEN l.disbursement\_date BETWEEN pd.prev\_month\_start AND pd.prev\_month\_end THEN 1 END),

2

)

ELSE NULL

END as month\_over\_month\_growth\_percent

FROM loans l

CROSS JOIN period\_definitions pd

WHERE l.disbursement\_date IS NOT NULL;

**Output:**



# PART 2: PERIOD-OVER-PERIOD ANALYSIS

## 2.1 Month-over-Month and Quarter-over-Quarter Analysis

**Purpose:**  
To provide a structured approach for analyzing performance changes between consecutive periods (month-over-month, quarter-over-quarter) and across the same period in previous years (year-over-year). This allows businesses to track growth trends, detect slowdowns, and benchmark current performance against historical patterns. It also introduces trend indicators for quick interpretation and rolling annual totals for a broader context.

**Learning Goal:**

* Learn to build period-based aggregations for months and quarters using date functions like DATEFROMPARTS and DATEPART.
* Apply LAG() to fetch prior-period values for calculating growth rates.
* Develop both absolute and percentage change calculations to measure performance shifts.
* Implement YoY comparisons for the same month/quarter in the previous year.
* Use CASE logic to generate trend indicators (e.g., Growth, Decline, Stable).
* Create rolling annual totals for long-term performance tracking.

WITH monthly\_metrics AS (

SELECT

YEAR(disbursement\_date) AS year,

MONTH(disbursement\_date) AS month,

CONCAT(DATENAME(MONTH, disbursement\_date), ' ', YEAR(disbursement\_date)) AS month\_label,

COUNT(loan\_id) AS loans\_originated,

SUM(loan\_amount) AS monthly\_volume,

AVG(loan\_amount) AS avg\_loan\_size,

COUNT(CASE WHEN loan\_status = 'Defaulted' THEN 1 END) AS current\_defaults,

-- Create date for proper ordering

DATEFROMPARTS(YEAR(disbursement\_date), MONTH(disbursement\_date), 1) AS month\_date

FROM loans

WHERE disbursement\_date IS NOT NULL

AND disbursement\_date >= '2023-01-01'

GROUP BY

YEAR(disbursement\_date),

MONTH(disbursement\_date),

CONCAT(DATENAME(MONTH, disbursement\_date), ' ', YEAR(disbursement\_date)),

DATEFROMPARTS(YEAR(disbursement\_date), MONTH(disbursement\_date), 1)

),

monthly\_comparisons AS (

SELECT

year,

month,

month\_label,

month\_date,

loans\_originated,

monthly\_volume,

avg\_loan\_size,

current\_defaults,

-- Previous month metrics using LAG

LAG(loans\_originated, 1) OVER (ORDER BY month\_date) AS prev\_month\_loans,

LAG(monthly\_volume, 1) OVER (ORDER BY month\_date) AS prev\_month\_volume,

LAG(avg\_loan\_size, 1) OVER (ORDER BY month\_date) AS prev\_month\_avg\_size,

-- Same month previous year

LAG(loans\_originated, 12) OVER (ORDER BY month\_date) AS same\_month\_prev\_year\_loans,

LAG(monthly\_volume, 12) OVER (ORDER BY month\_date) AS same\_month\_prev\_year\_volume

FROM monthly\_metrics

)

SELECT TOP 5

month\_label,

loans\_originated,

FORMAT(monthly\_volume, 'C0', 'en-IN') AS volume\_formatted,

FORMAT(avg\_loan\_size, 'C0', 'en-IN') AS avg\_loan\_formatted,

current\_defaults,

-- Month-over-month growth

loans\_originated - prev\_month\_loans AS mom\_loan\_growth,

CASE

WHEN prev\_month\_loans > 0

THEN ROUND((loans\_originated - prev\_month\_loans) \* 100.0 / prev\_month\_loans, 2)

ELSE NULL

END AS mom\_loan\_growth\_percent,

CASE

WHEN prev\_month\_volume > 0

THEN ROUND((monthly\_volume - prev\_month\_volume) \* 100.0 / prev\_month\_volume, 2)

ELSE NULL

END AS mom\_volume\_growth\_percent,

-- Year-over-year comparison

loans\_originated - same\_month\_prev\_year\_loans AS yoy\_loan\_growth,

CASE

WHEN same\_month\_prev\_year\_loans > 0

THEN ROUND((loans\_originated - same\_month\_prev\_year\_loans) \* 100.0 / same\_month\_prev\_year\_loans, 2)

ELSE NULL

END AS yoy\_loan\_growth\_percent,

-- Trend indicators

CASE

WHEN loans\_originated > prev\_month\_loans THEN '↗ Growing'

WHEN loans\_originated < prev\_month\_loans THEN '↘ Declining'

ELSE '→ Stable'

END AS mom\_trend,

CASE

WHEN loans\_originated > same\_month\_prev\_year\_loans THEN '↗ YoY Growth'

WHEN loans\_originated < same\_month\_prev\_year\_loans THEN '↘ YoY Decline'

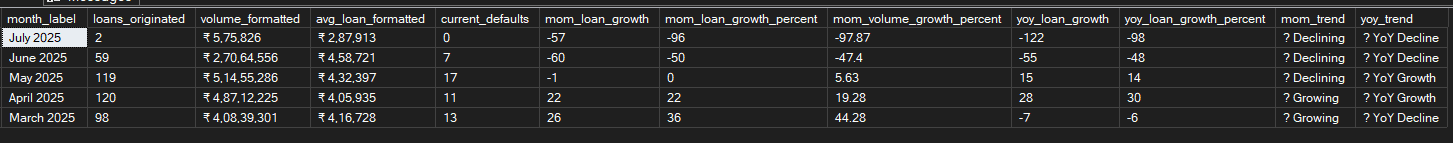
ELSE '→ YoY Stable'

END AS yoy\_trend

FROM monthly\_comparisons

ORDER BY month\_date DESC;

Output:



**-- Quarterly business performance analysis**

WITH quarterly\_metrics AS (

SELECT

YEAR(disbursement\_date) as year,

DATEPART(QUARTER, disbursement\_date) as quarter,

CONCAT('Q', DATEPART(QUARTER, disbursement\_date), '-', YEAR(disbursement\_date)) as quarter\_label,

COUNT(loan\_id) as quarterly\_loans,

SUM(loan\_amount) as quarterly\_volume,

COUNT(DISTINCT customer\_id) as unique\_customers,

COUNT(CASE WHEN loan\_status = 'Defaulted' THEN 1 END) as quarterly\_defaults,

-- Create proper date for ordering

DATEFROMPARTS(YEAR(disbursement\_date), (DATEPART(QUARTER, disbursement\_date) - 1) \* 3 + 1, 1) as quarter\_start\_date

FROM loans

WHERE disbursement\_date IS NOT NULL

GROUP BY YEAR(disbursement\_date), DATEPART(QUARTER, disbursement\_date)

)

SELECT

quarter\_label,

quarterly\_loans,

FORMAT(quarterly\_volume, 'C0', 'en-IN') as volume\_formatted,

unique\_customers,

quarterly\_defaults,

ROUND(quarterly\_defaults \* 100.0 / quarterly\_loans, 2) as quarterly\_default\_rate,

-- Quarter-over-quarter comparisons

LAG(quarterly\_loans, 1) OVER (ORDER BY quarter\_start\_date) as prev\_quarter\_loans,

CASE

WHEN LAG(quarterly\_loans, 1) OVER (ORDER BY quarter\_start\_date) > 0

THEN ROUND(

(quarterly\_loans - LAG(quarterly\_loans, 1) OVER (ORDER BY quarter\_start\_date)) \* 100.0 /

LAG(quarterly\_loans, 1) OVER (ORDER BY quarter\_start\_date),

2

)

ELSE NULL

END as qoq\_growth\_percent,

-- Year-over-year quarterly comparison

LAG(quarterly\_loans, 4) OVER (ORDER BY quarter\_start\_date) as same\_quarter\_prev\_year,

CASE

WHEN LAG(quarterly\_loans, 4) OVER (ORDER BY quarter\_start\_date) > 0

THEN ROUND(

(quarterly\_loans - LAG(quarterly\_loans, 4) OVER (ORDER BY quarter\_start\_date)) \* 100.0 /

LAG(quarterly\_loans, 4) OVER (ORDER BY quarter\_start\_date),

2

)

ELSE NULL

END as yoy\_quarterly\_growth\_percent,

-- Running annual total

SUM(quarterly\_loans) OVER (

ORDER BY quarter\_start\_date

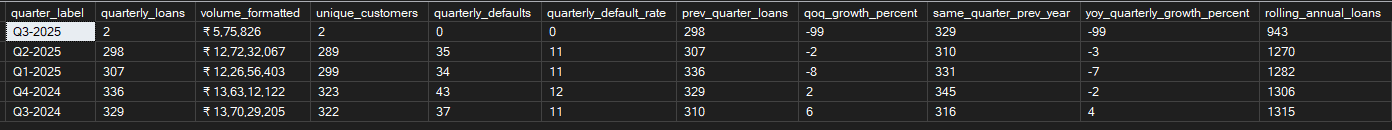
ROWS 3 PRECEDING

) as rolling\_annual\_loans

FROM quarterly\_metrics

ORDER BY quarter\_start\_date DESC;

Output:



## 2.2 Rolling and Cumulative Analysis

**Purpose:**  
To measure performance trends over moving time windows and cumulative periods, providing a smoother and more meaningful view of business activity than raw daily or monthly counts. This helps in identifying medium-term patterns, seasonal effects, and volatility in key metrics.

**Learning Goal:**

* Understand how to calculate moving averages (7-day, 30-day, 90-day) to smooth short-term fluctuations.
* Learn to compute rolling totals for specific time windows using ROWS PRECEDING.
* Apply cumulative (year-to-date) calculations with ROWS UNBOUNDED PRECEDING for progressive tracking within a year.
* Incorporate statistical measures like STDEV() to quantify volatility.
* Enhance skills in window functions for advanced time-series analytics.

**-- Rolling performance analysis with multiple time windows**

WITH daily\_metrics AS (

SELECT

CAST(disbursement\_date AS DATE) as loan\_date,

COUNT(loan\_id) as daily\_loans,

SUM(loan\_amount) as daily\_volume,

AVG(loan\_amount) as daily\_avg\_loan

FROM loans

WHERE disbursement\_date IS NOT NULL

AND disbursement\_date >= '2024-01-01'

GROUP BY CAST(disbursement\_date AS DATE)

)

SELECT

loan\_date,

daily\_loans,

FORMAT(daily\_volume, 'C0', 'en-IN') as daily\_volume\_formatted,

-- 7-day rolling averages

AVG(daily\_loans) OVER (

ORDER BY loan\_date

ROWS 6 PRECEDING

) as rolling\_7d\_avg\_loans,

AVG(daily\_volume) OVER (

ORDER BY loan\_date

ROWS 6 PRECEDING

) as rolling\_7d\_avg\_volume,

-- 30-day rolling metrics

SUM(daily\_loans) OVER (

ORDER BY loan\_date

ROWS 29 PRECEDING

) as rolling\_30d\_total\_loans,

AVG(daily\_volume) OVER (

ORDER BY loan\_date

ROWS 29 PRECEDING

) as rolling\_30d\_avg\_volume,

-- 90-day rolling metrics

AVG(daily\_loans) OVER (

ORDER BY loan\_date

ROWS 89 PRECEDING

) as rolling\_90d\_avg\_loans,

-- Cumulative metrics (year-to-date)

SUM(daily\_loans) OVER (

PARTITION BY YEAR(loan\_date)

ORDER BY loan\_date

ROWS UNBOUNDED PRECEDING

) as ytd\_cumulative\_loans,

SUM(daily\_volume) OVER (

PARTITION BY YEAR(loan\_date)

ORDER BY loan\_date

ROWS UNBOUNDED PRECEDING

) as ytd\_cumulative\_volume,

-- Volatility measures

STDEV(daily\_loans) OVER (

ORDER BY loan\_date

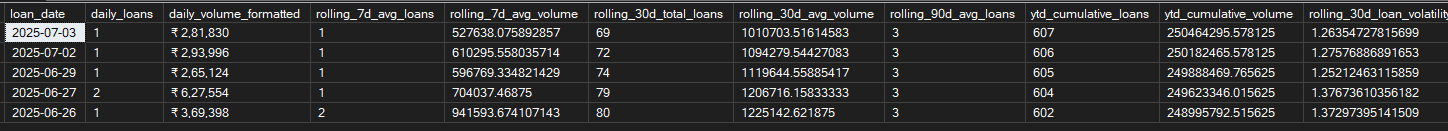
ROWS 29 PRECEDING

) as rolling\_30d\_loan\_volatility

FROM daily\_metrics

ORDER BY loan\_date DESC;

**Output:**

****

# PART 3: COHORT ANALYSIS & CUSTOMER LIFECYCLE

## 3.1 Registration Cohort Analysis

**Purpose:**  
To segment customers into cohorts based on their registration (first loan disbursement) date and analyze how each group performs over time. This approach helps identify patterns in customer engagement, loan activity, repayment behavior, and default rates, allowing the business to evaluate the long-term value and quality of different customer batches. It also supports vintage loan analysis to understand the performance of loans originated in the same time frame.

**Learning Goal:**

* Learn to define cohorts using date functions such as DATEFROMPARTS() and label them for reporting.
* Calculate key engagement metrics like loan penetration rate, average loans per active customer, and revenue per customer.
* Measure loan quality by tracking default and completion rates at the cohort level.
* Apply maturity classifications to cohorts (e.g., New, Early Stage, Developing, Mature).
* Perform loan vintage analysis to assess risk and performance trends for loans based on origination month.
* Use aggregated metrics to classify cohorts into performance categories (Excellent, Good, Average, Problem).

**-- Customer cohort analysis using first loan disbursement date**

WITH customer\_first\_loan AS (

SELECT

c.customer\_id,

c.full\_name,

c.employment\_type,

c.annual\_income,

MIN(l.disbursement\_date) AS first\_loan\_date

FROM customers c

JOIN loans l

ON c.customer\_id = l.customer\_id

AND l.disbursement\_date IS NOT NULL

GROUP BY

c.customer\_id,

c.full\_name,

c.employment\_type,

c.annual\_income

),

customer\_cohorts AS (

SELECT

customer\_id,

full\_name,

employment\_type,

annual\_income,

first\_loan\_date AS registration\_date,

DATEFROMPARTS(YEAR(first\_loan\_date), MONTH(first\_loan\_date), 1) AS cohort\_month,

CONCAT(DATENAME(MONTH, first\_loan\_date), ' ', YEAR(first\_loan\_date)) AS cohort\_label,

DATEDIFF(MONTH, first\_loan\_date, GETDATE()) AS customer\_age\_months

FROM customer\_first\_loan

),

cohort\_loan\_behavior AS (

SELECT

cc.cohort\_month,

cc.cohort\_label,

cc.customer\_age\_months,

COUNT(DISTINCT cc.customer\_id) AS cohort\_size,

COUNT(l.loan\_id) AS total\_loans,

COUNT(DISTINCT CASE WHEN l.loan\_id IS NOT NULL THEN cc.customer\_id END) AS customers\_with\_loans,

SUM(l.loan\_amount) AS total\_loan\_volume,

AVG(l.loan\_amount) AS avg\_loan\_amount,

COUNT(CASE WHEN l.loan\_status = 'Active' THEN 1 END) AS active\_loans,

COUNT(CASE WHEN l.loan\_status = 'Closed' THEN 1 END) AS completed\_loans,

COUNT(CASE WHEN l.loan\_status = 'Defaulted' THEN 1 END) AS defaulted\_loans,

AVG(DATEDIFF(DAY, cc.registration\_date, l.disbursement\_date)) AS avg\_days\_to\_first\_loan

FROM customer\_cohorts cc

LEFT JOIN loans l

ON cc.customer\_id = l.customer\_id

AND l.disbursement\_date IS NOT NULL

GROUP BY

cc.cohort\_month,

cc.cohort\_label,

cc.customer\_age\_months

),

cohort\_metrics AS (

SELECT

cohort\_month,

cohort\_label,

customer\_age\_months,

cohort\_size,

customers\_with\_loans,

total\_loans,

FORMAT(total\_loan\_volume, 'C0', 'en-IN') AS volume\_formatted,

ROUND(customers\_with\_loans \* 100 / cohort\_size, 2) AS loan\_penetration\_rate,

ROUND(total\_loans \* 1 / NULLIF(customers\_with\_loans, 0), 2) AS avg\_loans\_per\_active\_customer,

ROUND(total\_loan\_volume \* 1 / cohort\_size, 0) AS revenue\_per\_customer,

ROUND(defaulted\_loans \* 100 / NULLIF(total\_loans, 0), 2) AS cohort\_default\_rate,

ROUND(completed\_loans \* 100 / NULLIF(total\_loans, 0), 2) AS cohort\_completion\_rate,

ROUND(avg\_days\_to\_first\_loan, 0) AS avg\_days\_to\_first\_loan

FROM cohort\_loan\_behavior

)

SELECT TOP 5

cohort\_label,

customer\_age\_months,

cohort\_size,

customers\_with\_loans,

total\_loans,

volume\_formatted,

loan\_penetration\_rate,

avg\_loans\_per\_active\_customer,

revenue\_per\_customer,

cohort\_default\_rate,

cohort\_completion\_rate,

avg\_days\_to\_first\_loan,

CASE

WHEN loan\_penetration\_rate > 80 AND cohort\_default\_rate < 5 THEN 'EXCELLENT COHORT'

WHEN loan\_penetration\_rate > 60 AND cohort\_default\_rate < 10 THEN 'GOOD COHORT'

WHEN loan\_penetration\_rate > 40 THEN 'AVERAGE COHORT'

WHEN cohort\_default\_rate > 20 THEN 'PROBLEM COHORT'

ELSE 'LOW ENGAGEMENT COHORT'

END AS cohort\_classification,

CASE

WHEN customer\_age\_months >= 36 THEN 'MATURE COHORT'

WHEN customer\_age\_months >= 18 THEN 'DEVELOPING COHORT'

WHEN customer\_age\_months >= 6 THEN 'EARLY STAGE COHORT'

ELSE 'NEW COHORT'

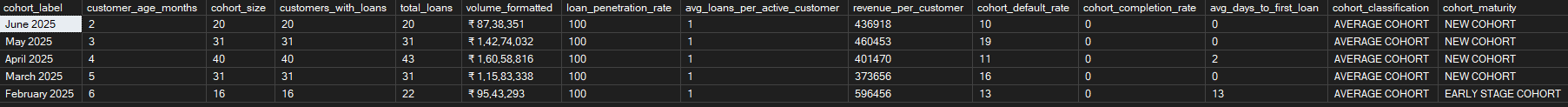
END AS cohort\_maturity

FROM cohort\_metrics

WHERE cohort\_size >= 10

ORDER BY cohort\_month DESC;

Output:



**-- Loan vintage analysis (cohorts based on loan origination)**

WITH loan\_vintages AS (

SELECT

loan\_id,

customer\_id,

disbursement\_date,

loan\_amount,

loan\_status,

DATEFROMPARTS(YEAR(disbursement\_date), MONTH(disbursement\_date), 1) AS vintage\_month,

CONCAT(DATENAME(MONTH, disbursement\_date), ' ', YEAR(disbursement\_date)) AS vintage\_label,

DATEDIFF(MONTH, disbursement\_date, GETDATE()) AS loan\_age\_months

FROM loans

WHERE disbursement\_date IS NOT NULL

AND disbursement\_date >= '2023-01-01'

)

SELECT TOP 5

vintage\_month,

vintage\_label,

loan\_age\_months,

COUNT(\*) AS vintage\_size,

FORMAT(SUM(loan\_amount), 'C0', 'en-IN') AS vintage\_volume,

FORMAT(AVG(loan\_amount), 'C0', 'en-IN') AS avg\_loan\_size,

COUNT(CASE WHEN loan\_status = 'Active' THEN 1 END) AS still\_active,

COUNT(CASE WHEN loan\_status = 'Closed' THEN 1 END) AS successfully\_closed,

COUNT(CASE WHEN loan\_status = 'Defaulted' THEN 1 END) AS defaulted,

COUNT(CASE WHEN loan\_status = 'Overdue' THEN 1 END) AS currently\_overdue,

ROUND(COUNT(CASE WHEN loan\_status = 'Defaulted' THEN 1 END) \* 100 / COUNT(\*), 2) AS vintage\_default\_rate,

ROUND(COUNT(CASE WHEN loan\_status = 'Closed' THEN 1 END) \* 100 / COUNT(\*), 2) AS vintage\_completion\_rate,

CASE

WHEN loan\_age\_months >= 24 AND COUNT(CASE WHEN loan\_status = 'Defaulted' THEN 1 END) \* 100 / COUNT(\*) < 3

THEN 'EXCELLENT VINTAGE'

WHEN loan\_age\_months >= 12 AND COUNT(CASE WHEN loan\_status = 'Defaulted' THEN 1 END) \* 100 / COUNT(\*) < 8

THEN 'GOOD VINTAGE'

WHEN COUNT(CASE WHEN loan\_status = 'Defaulted' THEN 1 END) \* 100 / COUNT(\*) > 15

THEN 'PROBLEM VINTAGE'

ELSE 'MONITORING REQUIRED'

END AS vintage\_quality

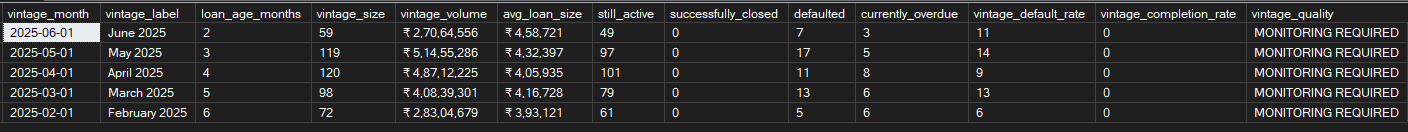
FROM loan\_vintages

GROUP BY vintage\_month, vintage\_label, loan\_age\_months

HAVING COUNT(\*) >= 20

ORDER BY vintage\_month DESC;

Output:



## 3.2 Customer Lifecycle Analysis

**Purpose:**  
To track customers’ complete journey from their first loan onwards, evaluating activity, engagement, and risk profiles over time. By mapping customers into lifecycle stages, value tiers, and engagement levels, this analysis supports personalized marketing, retention strategies, and risk management.

**Learning Goal:**

* Understand how to derive lifecycle metrics such as tenure, frequency of borrowing, and average monthly loan value.
* Learn to segment customers into lifecycle stages (e.g., New Prospect, Loyal Customer, Dormant Customer) using business rules.
* Build value tiers based on total loan value to identify high-value or VIP customers.
* Measure engagement through loans-per-month and classify customers accordingly.
* Detect risk levels by combining default history and active loan exposure.
* Develop actionable classifications that support targeted customer management strategies.

**-- Comprehensive customer lifecycle analysis (using first loan date as registration date)**

WITH customer\_journey AS (

SELECT

c.customer\_id,

c.full\_name,

c.employment\_type,

c.annual\_income,

MIN(l.disbursement\_date) AS registration\_date, -- First loan date used as registration

COUNT(l.loan\_id) AS total\_loans,

SUM(l.loan\_amount) AS lifetime\_loan\_value,

MAX(l.disbursement\_date) AS most\_recent\_loan\_date,

COUNT(p.payment\_id) AS total\_payments,

SUM(p.payment\_amount) AS total\_payments\_made,

COUNT(CASE WHEN l.loan\_status = 'Active' THEN 1 END) AS active\_loans,

COUNT(CASE WHEN l.loan\_status = 'Defaulted' THEN 1 END) AS defaulted\_loans,

MAX(CASE WHEN l.loan\_status = 'Defaulted' THEN 1 ELSE 0 END) AS has\_ever\_defaulted

FROM customers c

LEFT JOIN loans l

ON c.customer\_id = l.customer\_id

AND l.disbursement\_date IS NOT NULL

LEFT JOIN payments p

ON l.loan\_id = p.loan\_id

GROUP BY c.customer\_id, c.full\_name, c.employment\_type, c.annual\_income

),

lifecycle\_metrics AS (

SELECT

\*,

DATEDIFF(MONTH, registration\_date, GETDATE()) AS customer\_tenure\_months,

DATEDIFF(DAY, registration\_date, registration\_date) AS days\_to\_first\_loan, -- always 0 since registration=first loan

DATEDIFF(DAY, most\_recent\_loan\_date, GETDATE()) AS days\_since\_last\_loan,

CASE

WHEN total\_loans = 0 THEN 0

ELSE ROUND(total\_loans \* 1.0 / NULLIF(DATEDIFF(MONTH, registration\_date, GETDATE()), 0), 2)

END AS loans\_per\_month,

CASE

WHEN total\_loans = 0 THEN 0

ELSE ROUND(lifetime\_loan\_value \* 1.0 / NULLIF(DATEDIFF(MONTH, registration\_date, GETDATE()), 0), 0)

END AS average\_monthly\_loan\_value

FROM customer\_journey

)

SELECT

full\_name,

employment\_type,

FORMAT(annual\_income, 'C0', 'en-IN') AS income\_formatted,

customer\_tenure\_months,

total\_loans,

FORMAT(lifetime\_loan\_value, 'C0', 'en-IN') AS lifetime\_value\_formatted,

days\_to\_first\_loan,

days\_since\_last\_loan,

loans\_per\_month,

active\_loans,

defaulted\_loans,

CASE

WHEN total\_loans = 0 AND customer\_tenure\_months <= 3 THEN 'NEW PROSPECT'

WHEN total\_loans = 0 AND customer\_tenure\_months > 3 THEN 'INACTIVE CUSTOMER'

WHEN total\_loans = 1 AND active\_loans = 1 THEN 'FIRST-TIME BORROWER'

WHEN total\_loans BETWEEN 2 AND 4 AND active\_loans > 0 THEN 'DEVELOPING CUSTOMER'

WHEN total\_loans >= 5 AND active\_loans > 0 THEN 'LOYAL CUSTOMER'

WHEN total\_loans > 0 AND active\_loans = 0 AND days\_since\_last\_loan <= 180 THEN 'RECENT COMPLETER'

WHEN total\_loans > 0 AND active\_loans = 0 AND days\_since\_last\_loan > 180 THEN 'DORMANT CUSTOMER'

WHEN has\_ever\_defaulted = 1 THEN 'PROBLEM CUSTOMER'

ELSE 'UNDEFINED'

END AS lifecycle\_stage,

CASE

WHEN lifetime\_loan\_value >= 5000000 THEN 'VIP CUSTOMER'

WHEN lifetime\_loan\_value >= 2000000 THEN 'HIGH VALUE CUSTOMER'

WHEN lifetime\_loan\_value >= 500000 THEN 'STANDARD CUSTOMER'

WHEN lifetime\_loan\_value > 0 THEN 'ENTRY LEVEL CUSTOMER'

ELSE 'PROSPECT'

END AS value\_tier,

CASE

WHEN loans\_per\_month >= 0.5 THEN 'HIGHLY ENGAGED'

WHEN loans\_per\_month >= 0.2 THEN 'MODERATELY ENGAGED'

WHEN loans\_per\_month > 0 THEN 'LIGHTLY ENGAGED'

ELSE 'NOT ENGAGED'

END AS engagement\_level,

CASE

WHEN has\_ever\_defaulted = 1 THEN 'HIGH RISK'

WHEN active\_loans > 2 THEN 'MEDIUM RISK - HIGH EXPOSURE'

WHEN total\_loans = 0 THEN 'UNKNOWN RISK'

ELSE 'LOW RISK'

END AS risk\_classification

FROM lifecycle\_metrics

ORDER BY lifetime\_loan\_value DESC;

Output:

# PART 4: TREND ANALYSIS & FORECASTING

## 4.1 Trend Identification and Pattern Recognition

**Purpose:**  
To detect long-term business trends and growth patterns in loan origination and disbursement volume. This section uses historical performance data to calculate linear trends and forecast future activity under different scenarios (historical average, trend-based, conservative, and optimistic). This helps businesses set realistic targets, prepare for potential demand changes, and allocate resources accordingly.

**Learning Goal:**

* Learn how to structure historical loan data by month and assign sequential period numbers for trend analysis.
* Calculate historical averages to establish baseline performance.
* Apply simple linear regression in SQL to determine slope (growth rate) and intercept (baseline).
* Forecast next-period performance using different forecasting models.
* Adjust forecasts for conservative and optimistic planning scenarios.
* Translate loan volume forecasts into monetary value projections for budgeting and decision-making.

WITH historical\_performance AS (

SELECT

DATEFROMPARTS(YEAR(disbursement\_date), MONTH(disbursement\_date), 1) AS month\_date,

COUNT(loan\_id) AS monthly\_loans,

SUM(loan\_amount) AS monthly\_volume,

ROW\_NUMBER() OVER (ORDER BY YEAR(disbursement\_date), MONTH(disbursement\_date)) AS period\_number

FROM loans

WHERE disbursement\_date IS NOT NULL

AND disbursement\_date >= '2023-01-01'

AND disbursement\_date < DATEADD(MONTH, -1, GETDATE())

GROUP BY YEAR(disbursement\_date), MONTH(disbursement\_date)

),

averages AS (

SELECT

COUNT(\*) AS n\_periods,

AVG(period\_number \* 1.0) AS avg\_x,

AVG(monthly\_loans \* 1.0) AS avg\_y

FROM historical\_performance

),

joined\_data AS (

SELECT

hp.period\_number,

hp.monthly\_loans,

a.n\_periods,

a.avg\_x,

a.avg\_y

FROM historical\_performance hp

CROSS JOIN averages a

),

slope\_intercept AS (

SELECT

n\_periods,

avg\_y,

SUM((period\_number - avg\_x) \* (monthly\_loans - avg\_y)) /

SUM(POWER(period\_number - avg\_x, 2)) AS slope,

avg\_y -

(SUM((period\_number - avg\_x) \* (monthly\_loans - avg\_y)) /

SUM(POWER(period\_number - avg\_x, 2))) \* avg\_x AS intercept

FROM joined\_data

GROUP BY n\_periods, avg\_y, avg\_x

)

SELECT

'Historical Average' AS forecast\_type,

ROUND(avg\_y, 0) AS predicted\_loans,

FORMAT(avg\_y \* (SELECT AVG(monthly\_volume \* 1.0 / NULLIF(monthly\_loans,0)) FROM historical\_performance), 'C0', 'en-IN') AS predicted\_volume

FROM slope\_intercept

UNION ALL

SELECT

'Linear Trend Forecast',

ROUND(intercept + slope \* (n\_periods + 1), 0),

FORMAT((intercept + slope \* (n\_periods + 1)) \* (SELECT AVG(monthly\_volume \* 1.0 / NULLIF(monthly\_loans,0)) FROM historical\_performance), 'C0', 'en-IN')

FROM slope\_intercept

UNION ALL

SELECT

'Conservative Forecast (10% below trend)',

ROUND((intercept + slope \* (n\_periods + 1)) \* 0.9, 0),

FORMAT((intercept + slope \* (n\_periods + 1)) \* 0.9 \* (SELECT AVG(monthly\_volume \* 1.0 / NULLIF(monthly\_loans,0)) FROM historical\_performance), 'C0', 'en-IN')

FROM slope\_intercept

UNION ALL

SELECT

'Optimistic Forecast (10% above trend)',

ROUND((intercept + slope \* (n\_periods + 1)) \* 1.1, 0),

FORMAT((intercept + slope \* (n\_periods + 1)) \* 1.1 \* (SELECT AVG(monthly\_volume \* 1.0 / NULLIF(monthly\_loans,0)) FROM historical\_performance), 'C0', 'en-IN')

FROM slope\_intercept;

**Output:**

A screenshot of a black and white screen

AI-generated content may be incorrect.

## 4.2 Seasonal Analysis and Cyclical Patterns

**Purpose:**  
To uncover recurring seasonal or cyclical patterns in loan origination and volume, enabling proactive business planning. Seasonal trends help in understanding peak and slow periods, aligning marketing campaigns, staffing, and funding requirements with predictable demand cycles.

**Learning Goal:**

* Aggregate and average monthly loan and volume data across multiple years to detect seasonal trends.
* Calculate **seasonal indexes** to measure how each month compares to the overall average.
* Classify months into **Peak**, **High**, **Normal**, **Slow**, or **Low** seasons based on performance thresholds.
* Map seasonal behavior to business events and contexts (e.g., festivals, financial year end, education season).
* Analyze **day-of-week patterns** to identify high-traffic loan days for operational optimization.

**-- Comprehensive seasonal analysis**

WITH monthly\_seasonality AS (

SELECT

MONTH(disbursement\_date) as month\_number,

DATENAME(MONTH, disbursement\_date) as month\_name,

COUNT(loan\_id) as total\_loans\_all\_years,

SUM(loan\_amount) as total\_volume\_all\_years,

AVG(loan\_amount) as avg\_loan\_size,

COUNT(DISTINCT YEAR(disbursement\_date)) as years\_of\_data,

-- Calculate averages across years

ROUND(COUNT(loan\_id) \* 1.0 / COUNT(DISTINCT YEAR(disbursement\_date)), 1) as avg\_monthly\_loans,

ROUND(SUM(loan\_amount) \* 1.0 / COUNT(DISTINCT YEAR(disbursement\_date)), 0) as avg\_monthly\_volume

FROM loans

WHERE disbursement\_date IS NOT NULL

AND disbursement\_date >= '2023-01-01'

GROUP BY MONTH(disbursement\_date), DATENAME(MONTH, disbursement\_date)

),

overall\_averages AS (

SELECT

AVG(avg\_monthly\_loans) as overall\_avg\_loans,

AVG(avg\_monthly\_volume) as overall\_avg\_volume

FROM monthly\_seasonality

)

SELECT

ms.month\_name,

ms.month\_number,

ms.years\_of\_data,

ROUND(ms.avg\_monthly\_loans, 1) as avg\_loans\_per\_month,

FORMAT(ms.avg\_monthly\_volume, 'C0', 'en-IN') as avg\_volume\_per\_month,

FORMAT(ms.avg\_loan\_size, 'C0', 'en-IN') as avg\_loan\_size\_formatted,

-- Seasonal index (vs overall average)

ROUND(ms.avg\_monthly\_loans / oa.overall\_avg\_loans \* 100, 1) as loan\_seasonal\_index,

ROUND(ms.avg\_monthly\_volume / oa.overall\_avg\_volume \* 100, 1) as volume\_seasonal\_index,

-- Seasonal classification

CASE

WHEN ms.avg\_monthly\_loans > oa.overall\_avg\_loans \* 1.2 THEN '🔥 PEAK SEASON'

WHEN ms.avg\_monthly\_loans > oa.overall\_avg\_loans \* 1.1 THEN '📈 HIGH SEASON'

WHEN ms.avg\_monthly\_loans < oa.overall\_avg\_loans \* 0.8 THEN '❄️ LOW SEASON'

WHEN ms.avg\_monthly\_loans < oa.overall\_avg\_loans \* 0.9 THEN '📉 SLOW SEASON'

ELSE '➡️ NORMAL SEASON'

END as seasonal\_classification,

-- Quarter context

CASE

WHEN ms.month\_number IN (1, 2, 3) THEN 'Q1'

WHEN ms.month\_number IN (4, 5, 6) THEN 'Q2'

WHEN ms.month\_number IN (7, 8, 9) THEN 'Q3'

ELSE 'Q4'

END as quarter,

-- Business insights

CASE

WHEN ms.month\_number IN (1, 2) THEN 'Post-holiday recovery period'

WHEN ms.month\_number IN (3, 4) THEN 'Financial year end activity'

WHEN ms.month\_number IN (5, 6) THEN 'Summer preparation loans'

WHEN ms.month\_number IN (7, 8) THEN 'Education season lending'

WHEN ms.month\_number IN (9, 10) THEN 'Festival season financing'

WHEN ms.month\_number IN (11, 12) THEN 'Year-end and holiday loans'

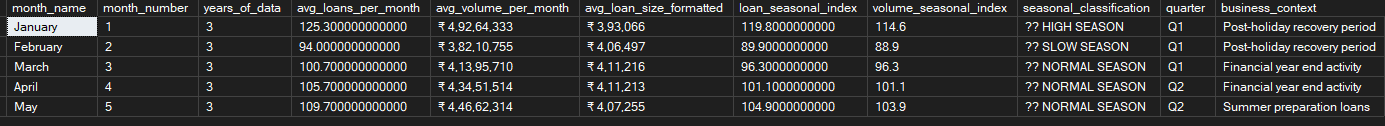
END as business\_context

FROM monthly\_seasonality ms

CROSS JOIN overall\_averages oa

ORDER BY ms.month\_number;

**Output:**



**-- Day of week and time-based patterns**

SELECT

DATENAME(WEEKDAY, disbursement\_date) as day\_name,

DATEPART(WEEKDAY, disbursement\_date) as day\_number,

COUNT(loan\_id) as total\_loans,

FORMAT(SUM(loan\_amount), 'C0', 'en-IN') as total\_volume,

ROUND(AVG(loan\_amount), 0) as avg\_loan\_amount,

-- Calculate percentage of weekly activity

ROUND(COUNT(loan\_id) \* 100.0 / SUM(COUNT(loan\_id)) OVER (), 2) as percent\_of\_weekly\_loans,

-- Business day classification

CASE

WHEN DATEPART(WEEKDAY, disbursement\_date) IN (1, 7) THEN 'WEEKEND'

WHEN DATEPART(WEEKDAY, disbursement\_date) = 2 THEN 'MONDAY'

WHEN DATEPART(WEEKDAY, disbursement\_date) = 6 THEN 'FRIDAY'

ELSE 'MID-WEEK'

END as day\_category

FROM loans

WHERE disbursement\_date IS NOT NULL

AND disbursement\_date >= '2023-01-01'

GROUP BY DATENAME(WEEKDAY, disbursement\_date), DATEPART(WEEKDAY, disbursement\_date)

ORDER BY day\_number;

**Output:**

A black screen with white text

AI-generated content may be incorrect.

**Enterprise Revenue Forecasting Blueprint**  
© 2024 Skill AI Path. All Rights Reserved.  
Created by: Viresh Gendle | tech@skillaipath.com | skillaipath.com

Licensed for educational and internal business use.  
Attribution required when sharing. Commercial redistribution prohibited.