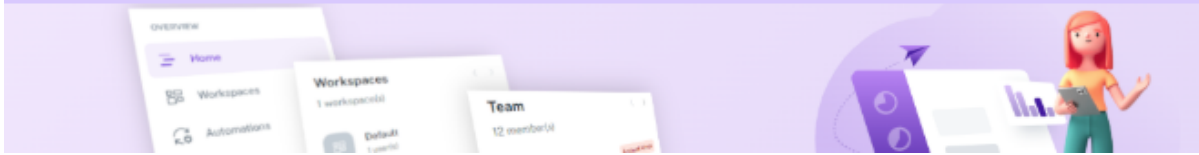




DataGOL



DataGOL 2025 in Review

In 2025, DataGOL redefined how teams access and act on data with AI. Every team can now build **context that powers smarter AI Agents and delivers insights in minutes**, not days.

Most teams still struggle with scattered data, analyst bottlenecks, and dashboards that explain what happened, but not why.

The Big Picture

DataGOL in 2025 redefined an AI Native data stack. It became:

- A smart **conversation layer** to interact with your data
- A **decision engine** for your business
- A **centralized place** to analyze, enrich, visualize, and act

AI and Data for Everyone

Why this matters:

Conversational interfaces are replacing dashboards. Business users expect to ask questions and receive answers instantly based on AI ready data and context.

What's new:

DataGOL gives every team such as Product, Finance, Ops, Marketing, Customer Success **out-of-the-box AI agents** that understand your data and answer questions instantly.

AI Agents

- **Data Science Agent:** Turn 3-day of manual dashboard-ing to 3-minutes of conversation. Click [here](#) to learn more.



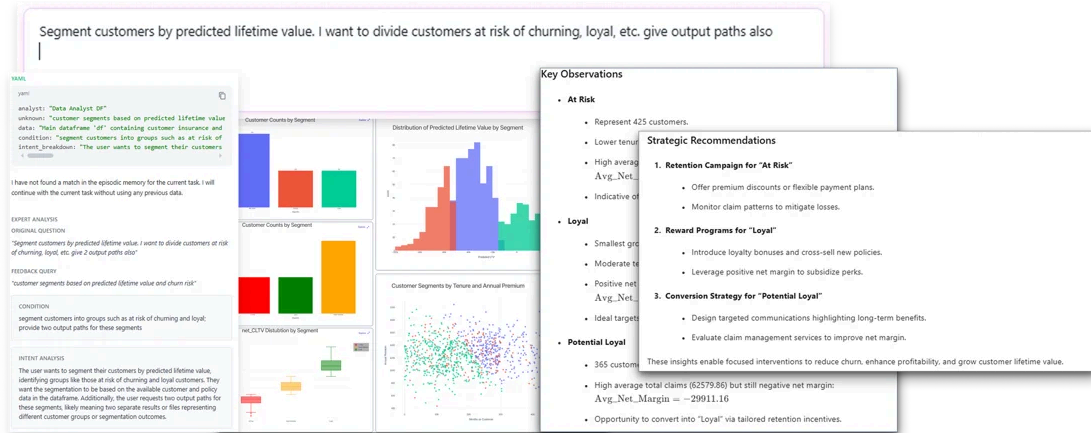
```
python
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Rename for convenience
df = df.rename(columns={'sub-category': 'sub_category'})

# Step A: Convert date columns to datetime with error handling
try:
    df['order_date'] = pd.to_datetime(df['order_date'])
    df['ship_date'] = pd.to_datetime(df['ship_date'])
except Exception:
    df['order_date'] = pd.to_datetime(df['order_date'], format='%ISO')
    df['ship_date'] = pd.to_datetime(df['ship_date'], format='%ISO')

# Step B: Compute key metrics
df['delivery_days'] = (df['ship_date'] - df['order_date']).dt.days
df['discount_cost'] = df['sales'] * (df['discount'] / (1 - df['discount']))
df['net_profit_shipdisc'] = df['net_profit'] - df['shipping_cost'] - df['discount_cost']
df['margin_shipdisc_pct'] = df['net_profit_shipdisc'] / df['sales']
df['fulfillment_speed'] = pd.qcut(df['delivery_days'], q=3, labels=['At Risk', 'Loyal', 'Potential Loyal'])
```

- **Data Conversation (DC) Agent:** Ask plain-English questions and get accurate, SQL-powered answers in seconds.
- **BI (Chart) Agent:** Turn questions into charts instantly and pin your widgets created by AI. No dashboard building required. Click [here](#) to learn more.



- **SQL Agent:** Generate, optimize, and debug complex SQL across data sources. Easily build pipelines and orchestrations.

SQL Agent

Give me all customer transactions by department with their budgets

This query filters `ns_pg_finance.transactions` to only customer-related rows (using `customer_id IS NOT NULL` or `customer_vendor` indicating 'customer'), then aggregates those transactions by `department_id` to compute total, count, and average amounts per department. It then joins these per-department aggregates to `ns_pg_finance.department_budgets` on the shared `department_id`, automatically restricting budgets to only departments that appear in the filtered transaction result set. The final `SELECT` returns the requested department identifier and name, aggregated customer transaction metrics, and the corresponding budget details (amount, month, and category), limited to 100 rows.

Run

JBDC

```
sql
WITH customer_transactions AS (
  SELECT
    t."department_id",
    t."amount"
  FROM ns_pg_finance."transactions" t
  WHERE
    -- Assuming customer-related transactions are identified via customer_id being not null
    -- and/or customer_vendor indicating 'customer'. Adjust as needed.
    (
      t."customer_id" IS NOT NULL
      OR (t."customer_vendor" ILIKE 'customer')
    )
),
aggregated_by_department AS (
  SELECT
    d."department_id",
    d."department_name",
    SUM(ct."amount") AS total_amount,
    COUNT(ct."amount") AS total_count,
    AVG(ct."amount") AS avg_amount
  FROM ns_pg_finance."departments" d
  JOIN customer_transactions ct ON d."department_id" = ct."department_id"
  GROUP BY d."department_id", d."department_name"
),
department_budgets AS (
  SELECT * FROM ns_pg_finance."department_budgets"
)
SELECT
  ag."department_id",
  ag."department_name",
  ag."total_amount",
  ag."total_count",
  ag."avg_amount",
  bu."amount",
  bu."month",
  bu."category"
FROM aggregated_by_department ag
JOIN department_budgets bu ON ag."department_id" = bu."department_id"
LIMIT 100
```

View Data

ns_pg_finance

Execution

Data Explorer

Analysis session started

AI THINKING

Intent: The user is asking to retrieve and view information: customer transactions grouped or organized by department along with their budgets. There is no indication of changing data, schema, or modifying an existing SQL query—only a read/reporting request.

Show 3 previous steps

DATA SOURCE TABLES (3)

REASONING

User explicitly mentions `ns_pg_finance.transactions` and `ns_pg_finance.department_budgets`, and needs department context; `transactions` holds customer-related activity by department, `department_budgets` holds budgets keyed by department and account_code, and departments can provide readable department names if needed.

ns_pg_finance.transactions (10 cols)

ns_pg_finance.department_budgets (6 cols)

ns_pg_finance.departments (2 cols)

SQL PLAN

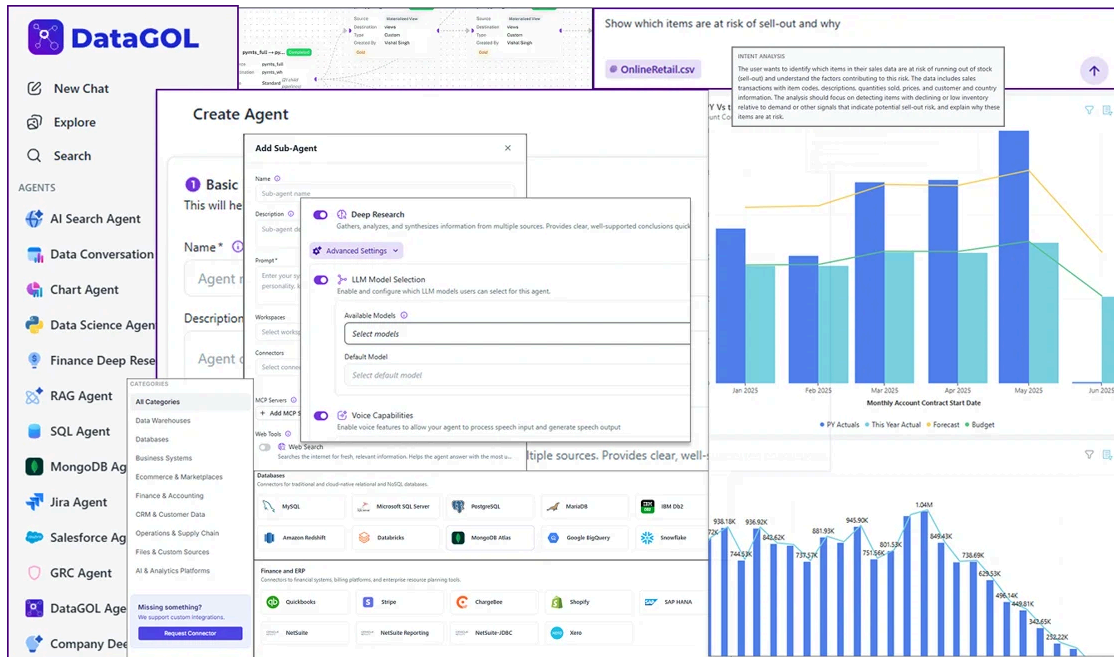
Plan Generated Successfully

REASONING

The projections include department identifiers and names for clear grouping, aggregated customer transaction metrics from `ns_pg_finance.transactions`, and corresponding budget details from `ns_pg_finance.department_budgets`, which together satisfy the requirement to organize customer-related transactions by

Powered by DataGOL

- Learn more about [Custom Agents](#), [RAG Agent](#), [HubSpot Agent](#), [Jira Agent](#)



BI That Moves at the Speed of the Business

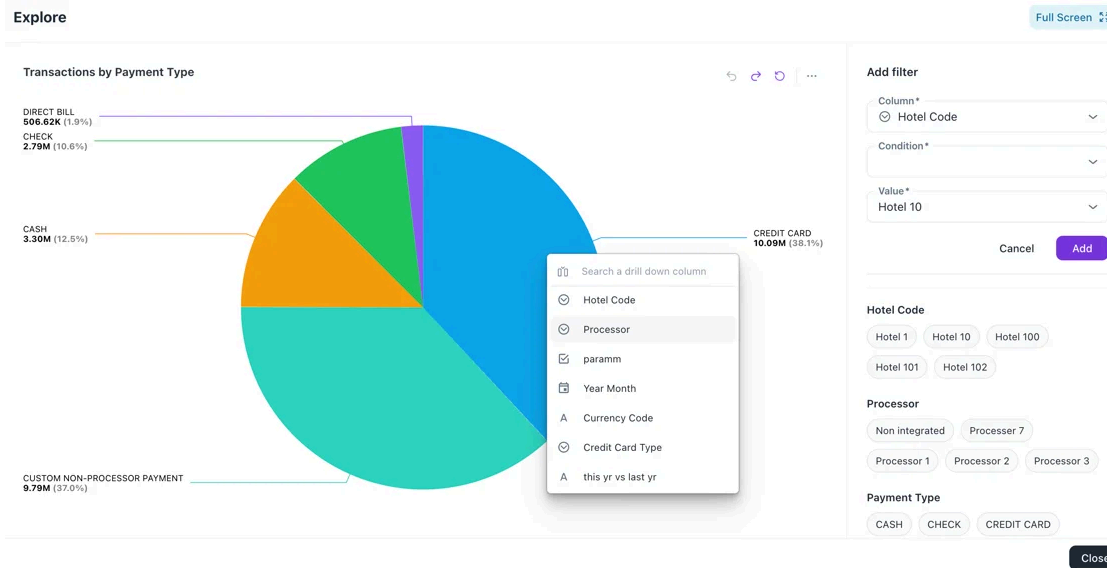
Why this matters:

Static dashboards don't keep up with fast-changing businesses. Teams need alerts, flexibility and clarity, not just charts.

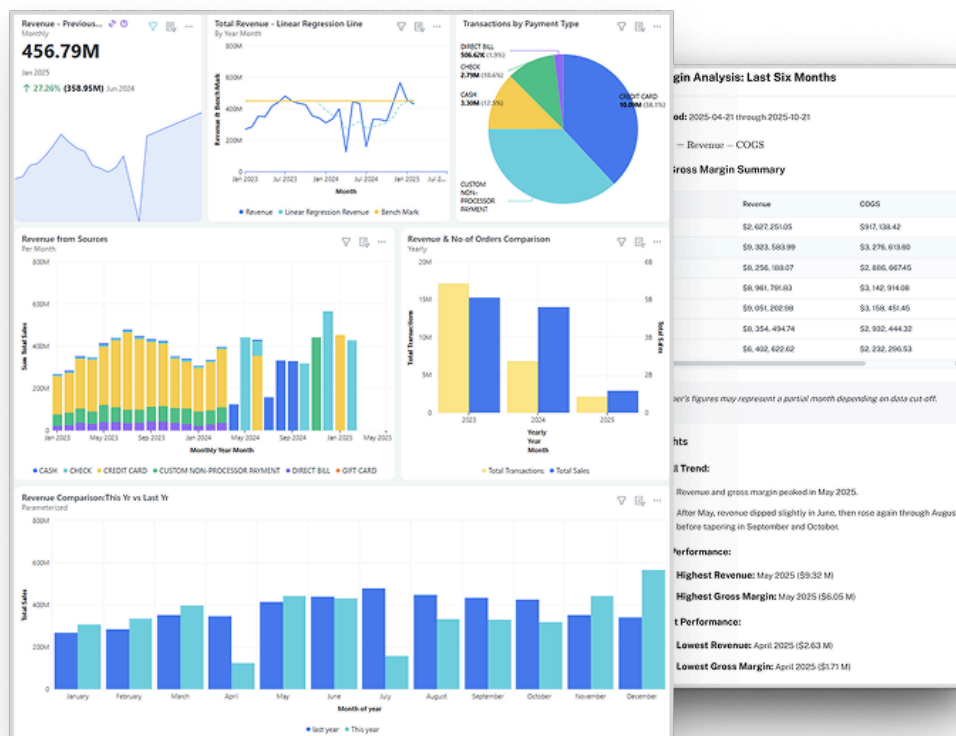
What's new:

DataGOL BI introduces powerful new capabilities designed to make dashboards faster to build, easier to customize, and more intuitive to use.

- Dependent & dynamic dashboard filters: control what data users see and [drill down](#) seamlessly



- Alerts & signals directly on BI widgets: get notified immediately when key metrics change More details [here](#).



- Advanced conditional formatting with AND/OR logic
- Inline editing, KPI alignment, heatmaps, N-day averages
- One-click dashboard duplication and external theming
- End-of-month projections and automatic time bucketing

Workbooks That Do More Than Store Data

Why this matters:

Data usually needs enrichment, fixes, and business logic after ingestion and that work often lives outside the data platform.

What's new:

Workbooks are now living, intelligent layers:

The screenshot displays a data table with columns: contact_reason, customer_id, customer_segment, language, nps_score, ces_score, and nps_category. The table lists various customer interactions and their segments. To the right, an 'AI Generate' panel is open, showing a prompt: 'use the (contact_reason) column to classify :'. Below the prompt, several classification rules are listed, such as 'High Business Value', 'New Booking Inquiry - High', 'Upgrade Request - High', 'Payment Failed - High', and 'Medium Business Value'.

contact_reason	customer_id	customer_segment
Meal Preference	CUS91122	Standard
Invoice Request	CUS55775	6E Rewards Silver
Meal Preference	CUS26591	Standard
Baggage Claim	CUS22672	Standard
Loyalty Points Query	CUS71991	6E Rewards Gold
Feedback	CUS76823	Standard
Invoice Request	CUS17385	6E Rewards Silver
Cancel Booking	CUS84333	6E Rewards Silver
Web/App Issue	CUS44268	6E Rewards Silver
Flight Status	CUS84341	Standard
Payment Failed	CUS98136	Standard
Complaint	CUS42783	Standard
Upgrade Request	CUS91377	Standard
Upgrade Request	CUS10653	6E Rewards Platinum
Cancel Booking	CUS77133	Standard

- Tables with advanced AI generate formulas (classify, summarize, extract) & manual inputs on top of your cross functional data
- Reverse ETL back into Salesforce, HubSpot, Attio, Jira
- No-code left joins across workbooks

AI Native Lakehouse Built for Scale (and Ease)

Why this matters:

As data grows, teams need control over performance, cost, governance, and reliability without complexity.

What's new:

- Native support for Snowflake, Redshift, BigQuery

The screenshot shows the 'New Data Source' interface. It features a header with 'Data Sources' and a 'New' button. Below the header, there are three main categories of integrations: Databases, CRM / Sales / Product / Support, and Finance and ERP. Each category contains a grid of icons representing different data sources.

Databases
Connectors for traditional and cloud-native relational and NoSQL databases.

- MySQL
- Microsoft SQL Server
- PostgreSQL
- MariaDB
- IBM Db2
- Amazon Redshift
- Databricks
- MongoDB Atlas
- Google BigQuery
- Snowflake

CRM / Sales / Product / Support
Integrations with customer relationship, support, and product analytics platforms.

- Salesforce
- HubSpot
- Pipedrive
- Monday
- Jira
- Zendesk Support
- GitHub (Beta)
- Amplitude
- Posthog
- Mixpanel
- Attio
- Asana

Finance and ERP
Connectors to financial systems, billing platforms, and enterprise resource planning tools.

- Quickbooks
- Stripe
- ChargeBee
- Shopify
- SAP HANA

- Live JDBC querying for real-time access
- **New connectors:** Zendesk, Mixpanel, Chargebee, Jira, AppsFlyer, Cvent, Stax, Apple Ads, Meta Ads, Snapchat Ads, TikTok Ads, Mixpanel, Posthog, Mailchimp, Xero, Google BigQuery. More details [here](#).
- Built-in database ingestion (Postgres, MySQL & more) with incremental, append-only updates. More details [here](#).
- Granular compute scaling & node group controls
- Stronger Lakehouse permission model

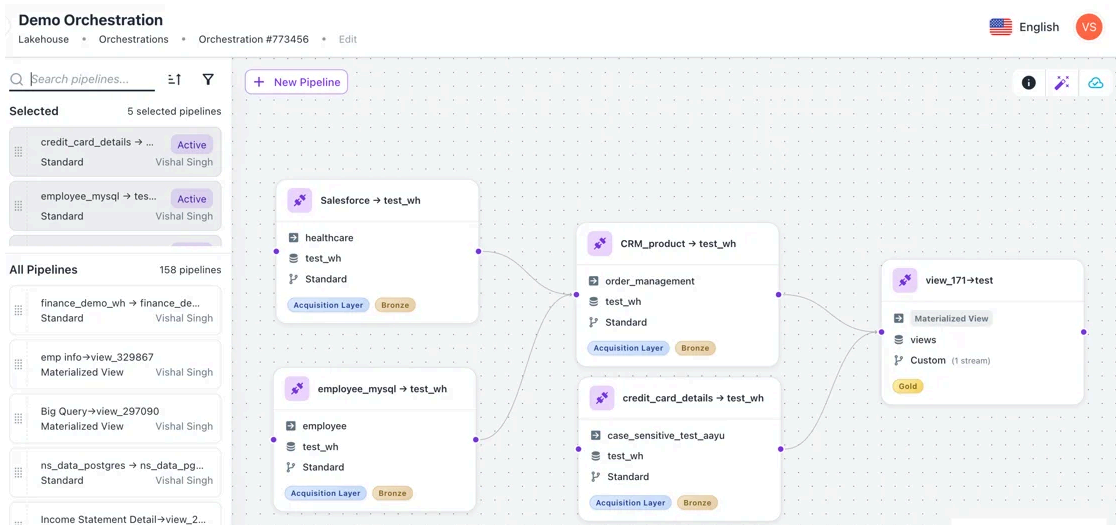
Pipelines & Orchestration (No More Infra Headaches)

Why this matters:

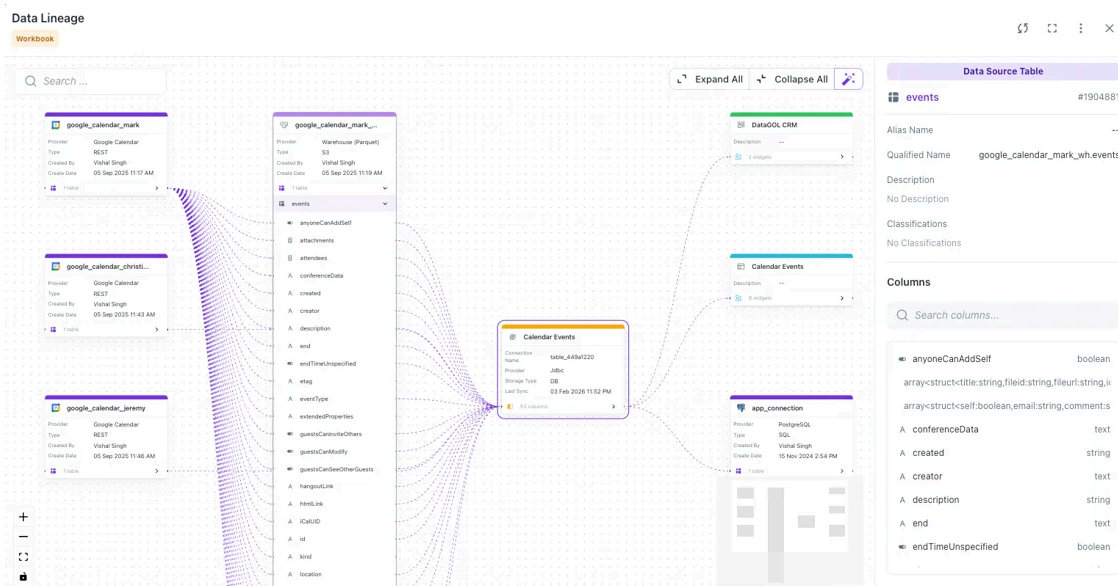
Data is only valuable when it's reliable. Strong orchestration ensures pipelines run predictably, failures are easy to fix, and teams can depend on data without second-guessing it.

What's new:

- Unified pipeline monitoring & logs
- Smarter alerts focused on failures
- Visual Orchestration Graphs that clearly show how data flows across pipelines, jobs, and dependencies.



- **End-to-end Data Lineage** across sources, pipelines, workbooks, and dashboards, giving teams instant visibility into how data is produced, transformed, and consumed



- Retry last job with one click
- Clearer error messages from third-party tools
- Standard pipeline SQL scoped per table
- UI-driven cron configuration (no cron anxiety)

The impact:

- Reliable and trustworthy data.
- No more firefighting.
- Lower costs.

DataGOL is built for teams that want answers, not bottlenecks.

We're excited to keep building the future of AI-native analytics with you.

Get a free trial now! Contact sales@datagol.ai

Support us on [Linkedin](#)

Learn to stay ahead in AI on [DataGOL University](#)

Join our [Next Event](#)

Context Plus Guardrails: The Only Way AI Ships to Production

You've built AI pilots. They work. But they're stuck.

Join us for a 45-minute moderated conversation with Founders: Dr. Rishabh Bhardwaj (Walled AI), and Vishal Singh (DataGOL) on what it actually takes to run AI safely and effectively in real production environments.

Feb 25th, 2026
Wednesday, 10:00am EST

Vishal Singh
Founder - DataGOL

Rishabh Bhardwaj
Founder - Walled AI

Jeremy Hamerman
Director of Product and Growth

DataGOL www.datagol.ai



DataGOL



DataGOL, 103 Carnegie Center, Princeton, New Jersey 08540, United States

[Unsubscribe](#) [Manage preferences](#)