



EYOUTH BOOTCAMP DATA ANALYSIS

AZURE CLOUD MIGRATION, ANALYTICS & ADVANCED INSIGHTS PROJECT

Team 5



Team Members

JANUARY 1, 2025

Table Of Content

1 Contents

2	Project Overview	4
3	Azure Cloud Infrastructure.....	4
3.1	Azure SQL Database Deployment.....	4
3.2	Security and Compliance	Error! Bookmark not defined.
3.3	Data Migration	Error! Bookmark not defined.
4	SQL Server and SSM	5
4.1	SSMS Connectivity	5
4.2	Data Cleaning	7
4.2.1	Category table	7
	7
4.2.2	Products table	8
4.2.3	Customers table	9
4.2.4	Orders table.....	10
4.2.5	Order details table.....	11
4.2.6	Payments table	11
4.2.7	Shipping table.....	12
4.2.8	Returns table.....	12
4.2.9	Reviews table	13
4.2.10	Suppliers table.....	13
4.2.11	Discounts table	14
4.2.12	Wishlist table.....	15
4.2.13	Inventory movement table	15
4.2.14	Customer sessions.....	15
4.3	Basic Queries.....	16
4.3.1	Total sales revenue from all orders.....	16
4.3.2	Top 5 best-selling products by quantity.....	17
4.3.3	Customers with the highest number of orders.....	17
4.3.4	Alert for low-stock products (under 20 units).....	18

4.3.5	Percentage of orders with discounts.....	18
4.3.6	Average product ratings.....	19
4.4	Advanced Queries.....	19
4.4.1	30-day customer retention post first purchase.	19
4.4.2	Product recommendations based on Wishlist co-purchases.	20
4.4.3	30-day moving average for inventory turnover trends.	20
4.4.4	Identify customers who purchased every product in a category.	21
4.4.5	Discover product pairs commonly bought together.	21
4.4.6	Calculate order delivery times in days.	22
4.5	Extra Queries	22
4.5.1	Customer's Table Exploration	22
4.5.2	Monthly total sales.....	23
4.5.3	# of new Customer per Month	23
4.5.4	Top 10 Categories by Revenue.....	23
4.5.5	Total Sales by Category/sub-category.....	24
4.5.6	Extract Region From address column in customer table	25
	25
4.5.7	Identify Military Regions	25
4.5.8	Create view Customer Segmentation.....	26
	26
	26
5	Power BI Integration and Dashboarding	28
4.1	DirectQuery Configuration	28
4.2	Dashboard Components	28
4.3	Data Cleaning and Modeling	33
6	Advanced Insights and Machine Learning	34
6.1	Automated Data Cleaning by Mito.....	34
6.1.1	Order Details Table	34
6.1.2	products Table.....	38
6.1.3	Customers Table.....	45
6.1.4	Wish Lists Table.....	49
6.1.5	Shipping Table.....	53
6.1.6	Orders Table	59

6.1.7	Payment Table.....	68
6.1.8	Review Table.....	73
6.1.9	Suppliers Table	75
6.1.10	Inventory movement Table.....	77
6.1.11	Customer Sessions Table.....	79
6.1.12	Discounts Table.....	80
6.1.13	Returns Table	83
6.1.14	Categories Table.....	84
5.2	Predictive Modeling	86
5.3	Natural Language Queries.....	89
7	Conclusion and recommendations.....	89
8	Submission and Evaluation.....	Error! Bookmark not defined.
9	Tools and Technologies Used.....	Error! Bookmark not defined.

2 Project Overview

This documentation outlines the design and implementation of a cloud migration and advanced data analytics project for the 'ecommerce_db' dataset. The goal is to securely migrate the data infrastructure to Microsoft Azure, enable scalable and real-time analytics using Power BI, and extract advanced insights using machine learning techniques.

3 Azure Cloud Infrastructure

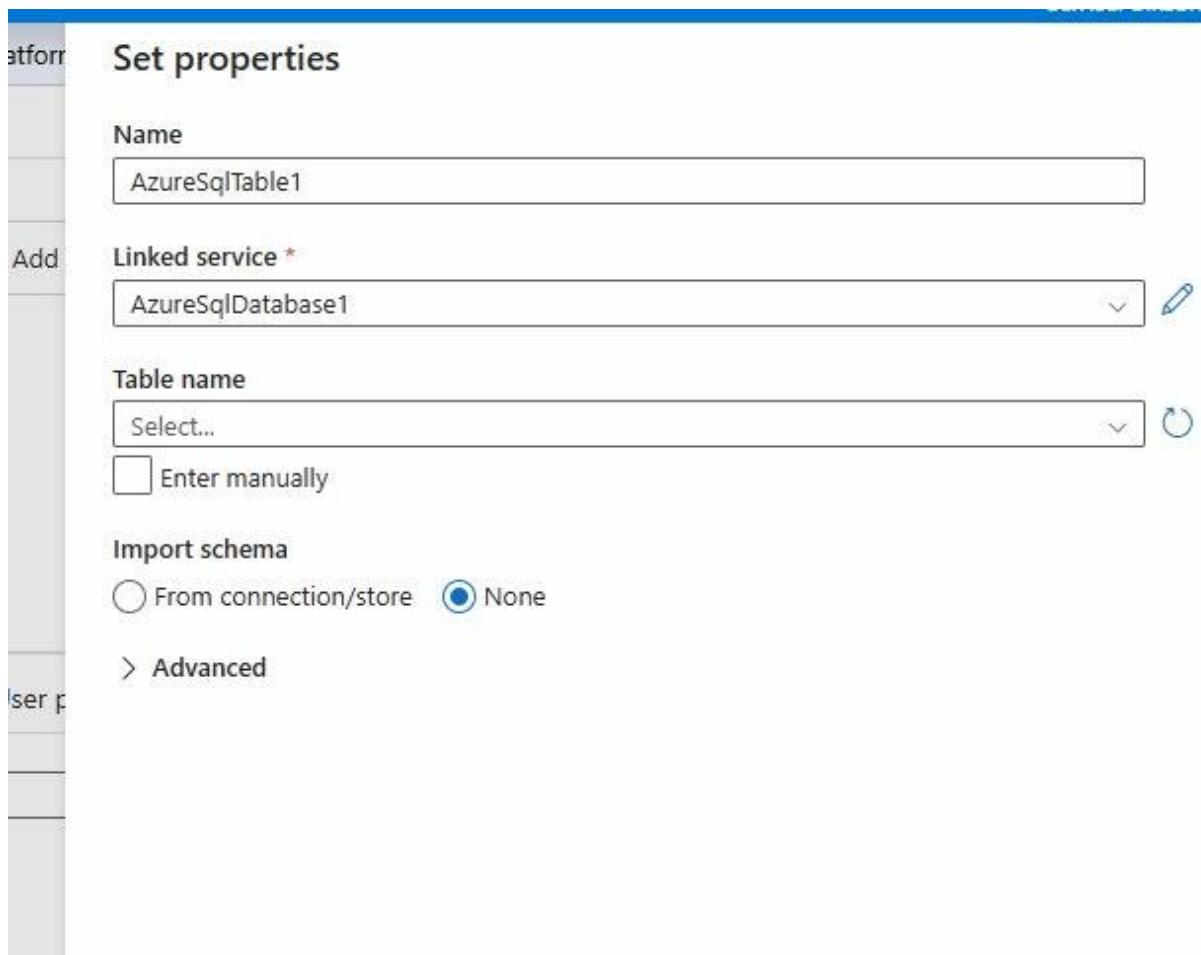
3.1 Azure SQL Database Deployment

Uploading SQL database to azure cloud

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar and a Copilot button. Below the header, there's a section for 'Azure services' with icons for Create a resource, SQL databases, Quickstart Center, Azure AI foundry, Kubernetes services, Virtual machines, App Services, Storage accounts, Azure Cosmos DB, and More services. The main area is titled 'Resources' with tabs for 'Recent' and 'Favorite'. It lists two items: 'e-commercedb' (SQL database) and 'database' (Resource group). Below this is a 'Navigate' section with links for Subscriptions, Resource groups, All resources, and Dashboard.

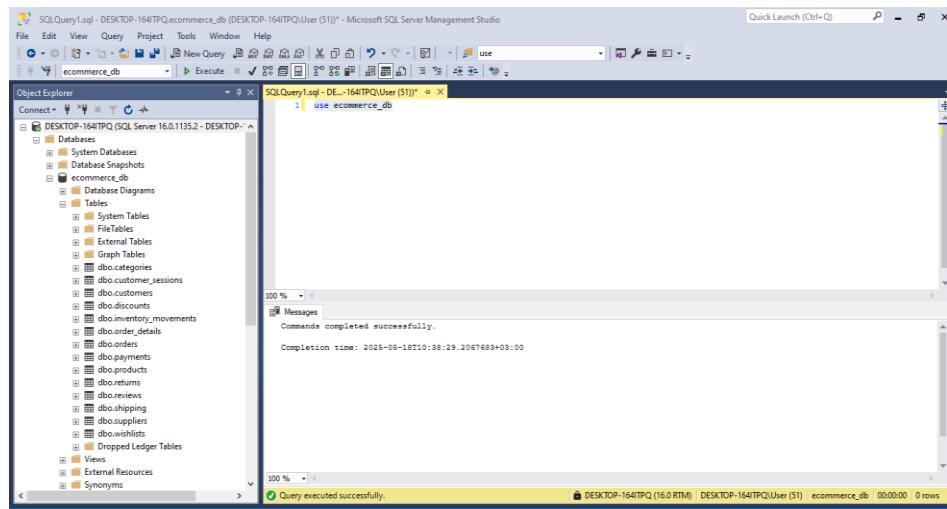
The screenshot shows the Microsoft Azure portal interface, specifically the 'database' resource group page. At the top, it shows the resource group name 'database' and a 'Search' bar. Below the header, there's a 'Overview' section with links for Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Cost Management, Monitoring, Automation, and Help. To the right of the overview, there's a 'Manage view' section with buttons for Create, Refresh, Export to CSV, Open query, Assign tags, Move, Delete, and JSON View. Below this, there's a 'Resources' section with a table showing three records:

Name	Type	Last Viewed
e-commerce-sales	SQL server	51 minutes ago
e-commercecb (e-commerce-sales/e-commercedb)	SQL database	51 minutes ago
ecommerce_cb (e-commerce-sales/ecommerce_db)	SQL database	51 minutes ago



4 SQL Server and SSM

4.1 SSMS Connectivity



Microsoft Azure

All services > database

Resource group

Search resources, services, and docs (G+)

Copilot

nadayousif2002@gmail.com

DEFAULT DIRECTORY

Search

Create Manage view Delete resource group Refresh Export to CSV Open query Assign tags Move Delete

Overview

Essentials

Activity log

Access control (IAM)

Tags

Resource visualizer

Events

Settings

Cost Management

Monitoring

Automation

Help

Resources Recommendations (1)

Filter for any field... Type equals all Location equals all Add filter

Showing 1 to 4 of 4 records. Show hidden types

No grouping List view

Name	Type	Location
data-factoryex	Data factory (V2)	Switzerland North
e-commerce-sales	SQL server	UAE North
e-commerceedb (e-commerce-sales/e-commerceedb)	SQL database	UAE North
ecommerce_db (e-commerce-sales/ecommerce_db)	SQL database	UAE North

https://portal.azure.com/#using Ctrl+Shift+F

4.2 Data Cleaning by SQL

4.2.1 Category table

```
--Categories table
select * from Categories
--there are nulls in parent_id because they are top parents
```

119 %

	id	name	description	parent_id
1	1	Part	Win color election her. Idea society understand a...	NULL
2	2	Program	Remain great line dog alone either level. Arm mo...	NULL
3	3	Large	Table people job civil here. View seat according ...	NULL
4	4	Hold	Yard call test investment state hundred. Scene h...	NULL
5	5	Speech	Forward upon provide. Live world bring if continu...	NULL
6	6	Bom	Study them region lose chance visit. Choose rath...	3
7	7	Recent	Day attorney total any. Require view safe cover ...	2
8	8	Window	Today whose improve guy cover. Success land ...	2
9	9	Half	Author drug air however capital. Street quickly sa...	4
10	1...	Adult	Address our whatever ago. War away remember ...	5
11	1...	Table	Although him again natural yeah maintain. Call up...	NULL
12	1...	Mission	Doctor hair create agency white protect front. Te...	NULL
13	1...	Share	Trouble yard state. Might and special deep owne...	NULL
14	1...	Free	Authority speak happen arrive table hour. Certainl...	NULL
15	1...	Letter	Heart offer trouble sport list hospital. Serve get ag...	NULL
16	1...	Maria...	Around watch better possible if standard. Military ...	NULL
17	1...	Who	Officer radio green meeting choice crime nation p...	NULL
18	1...	Game	Community card its some gun big increase. Soon ...	NULL
19	1...	Choose	Cell line ahead authority eight develop season. E...	NULL
20	2...	Energy	Especially near a either rich himself. Perform who...	NULL

Update null values

```
update categories
set parent_id = id
where parent_id is null
```

119 %

(15 rows affected)

Completion time: 2025-05-21T02:43:00.5629439+03:00

4.2.2 Products table

```
--check null values
select
    sum(case when name is null then 1 else 0 end) as missing_name,
    sum(case when price is null then 1 else 0 end) as missing_price,
    sum(case when category_id is null then 1 else 0 end) as missing_category,
    sum(case when supplier_id is null then 1 else 0 end) as missing_supplier,
    sum(case when sku is null then 1 else 0 end) as missing_sku,
    sum(case when stock_quantity is null then 1 else 0 end) as missing_stock
from products;
```

119 %

	missing_name	missing_price	missing_category	missing_supplier	missing_sku	missing_stock
1	0	0	0	0	0	0

```
--check for duplicated SKUs
select sku, count(*) as count
from products
group by sku
having count(*) > 1;
```

119 %

sku	count

```
--check for invalid prices and stock
select * from Products
where price < 0 OR stock_quantity < 0;
```

119 %

id	name	description	price	category_id	supplier_id	sku	stock_quantity

4.2.3 Customers table

```
--check null values
select
    sum(case when first_name is null then 1 else 0 end) as missing_first_name,
    sum(case when last_name is null then 1 else 0 end) as missing_last_name,
    sum(case when email is null then 1 else 0 end) as missing_email,
    sum(case when phone is null then 1 else 0 end) as missing_phone,
    sum(case when address is null then 1 else 0 end) as missing_address,
    sum(case when registration_date is null then 1 else 0 end) as missing_registration_date
from customers;
```

Results Messages

	missing_first_name	missing_last_name	missing_email	missing_phone	missing_address	missing_registration_date
1	0	0	0	0	0	0


```
--check for duplicated emails|
select email, count(*) as count
from customers
group by email
having count(*) > 1;
```

Results Messages

email	count

```
--check for duplicated phone
select phone, count(*) as count
from customers
group by phone
having count(*) > 1;
```

119 %

Results	Messages
phone count	


```
--remove extra spaces from emails
update customers
set email = ltrim(rtrim(email))
where email like ' %' or email like '% ';
```

119 %

Messages
(0 rows affected)

Completion time: 2025-05-21T02:54:44.8929396+03:00

4.2.4 Orders table

```
select * from Orders
--check null values
select
    sum(case when customer_id is null then 1 else 0 end) as missing_customer_id,
    sum(case when order_date is null then 1 else 0 end) as missing_order_date,
    sum(case when total_amount is null then 1 else 0 end) as missing_total_amount,
    sum(case when status is null then 1 else 0 end) as missing_status
from orders;
```

119 %

Results	Messages		
missing_customer_id missing_order_date missing_total_amount missing_status	0 0 0 0		
1			

4.2.5 Order details table

```
--check null values
select
    sum(case when order_id is null then 1 else 0 end) as missing_order_id,
    sum(case when product_id is null then 1 else 0 end) as missing_product_id,
    sum(case when quantity is null then 1 else 0 end) as missing_quantity,
    sum(case when unit_price is null then 1 else 0 end) as missing_unit_price
from order_details;
```

119 %

Results Messages

	missing_order_id	missing_product_id	missing_quantity	missing_unit_price
1	0	0	0	0

```
select order_id, count(*) as num_of_items
from order_details
group by order_id
having count(*) > 5
```

119 %

Results Messages

order_id	num_of_items

```
--check for invalid quantity or price
```

```
select * from order_details
where quantity <= 0 or unit_price < 0;
```

119 %

Results Messages

id	order_id	product_id	quantity	unit_price

4.2.6 Payments table

```
select
    sum(case when order_id is null then 1 else 0 end) as missing_order_id,
    sum(case when customer_id is null then 1 else 0 end) as missing_customer_id,
    sum(case when amount is null then 1 else 0 end) as missing_amount,
    sum(case when payment_date is null then 1 else 0 end) as missing_payment_date,
    sum(case when payment_method is null then 1 else 0 end) as missing_payment_method,
    sum(case when status is null then 1 else 0 end) as missing_status
from payments;
```

119 %

	missing_order_id	missing_customer_id	missing_amount	missing_payment_date	missing_payment_method	missing_status
1	0	0	0	0	0	0

4.2.7 Shipping table

```
--check for null values
select
    sum(case when order_id is null then 1 else 0 end) as missing_order_id,
    sum(case when shipping_date is null then 1 else 0 end) as missing_shipping_date,
    sum(case when tracking_number is null then 1 else 0 end) as missing_tracking_number,
    sum(case when carrier is null then 1 else 0 end) as missing_carrier,
    sum(case when status is null then 1 else 0 end) as missing_status
from shipping;
```

119 %

	missing_order_id	missing_shipping_date	missing_tracking_number	missing_carrier	missing_status
1	0	0	0	0	0

4.2.8 Returns table

```

select
    sum(case when order_id is null then 1 else 0 end) as missing_order_id,
    sum(case when return_date is null then 1 else 0 end) as missing_return_date,
    sum(case when reason is null then 1 else 0 end) as missing_reason,
    sum(case when status is null then 1 else 0 end) as missing_status
from returns;

```

Results

	missing_order_id	missing_return_date	missing_reason	missing_status
1	0	0	0	0

4.2.9 Reviews table

```

--check null values
select
    sum(case when product_id is null then 1 else 0 end) as missing_product_id,
    sum(case when customer_id is null then 1 else 0 end) as missing_customer_id,
    sum(case when rating is null then 1 else 0 end) as missing_rating,
    sum(case when review_date is null then 1 else 0 end) as missing_review_date
from reviews;

```

Results

	missing_product_id	missing_customer_id	missing_rating	missing_review_date
1	0	0	0	0

```

--check invalid rating values
select * from reviews
where rating < 1 or rating > 5;

```

Results

	id	product_id	customer_id	rating	comment	review_date

4.2.10 Suppliers table

```
--check null values
select
    sum(case when name is null then 1 else 0 end) as missing_name,
    sum(case when contact_person is null then 1 else 0 end) as missing_contact_person,
    sum(case when email is null then 1 else 0 end) as missing_email,
    sum(case when phone is null then 1 else 0 end) as missing_phone,
    sum(case when address is null then 1 else 0 end) as missing_address
from suppliers;
```

119 %

Results Messages

	missing_name	missing_contact_person	missing_email	missing_phone	missing_address
1	0	0	0	0	0

```
--check for duplicated supplier names
select name, count(*) as count
from suppliers
group by name
having count(*) > 1;
```

119 %

Results Messages

	name	count
--	------	-------

```
--check for duplicated supplier emails
select email, count(*) as count
from suppliers
group by email
having count(*) > 1;
```

119 %

Results Messages

	email	count
--	-------	-------

4.2.11 Discounts table

There are null values at that table but that because may be the discount not supported any product or category

4.2.12 Wishlist table

```
--check null values
select
    sum(case when customer_id is null then 1 else 0 end) as missing_customer_id,
    sum(case when product_id is null then 1 else 0 end) as missing_product_id,
    sum(case when added_date is null then 1 else 0 end) as missing_added_date
from wishlists
```

119 %

	missing_customer_id	missing_product_id	missing_added_date
1	0	0	0

4.2.13 Inventory movement table

```
--check null values
select
    sum(case when product_id is null then 1 else 0 end) as missing_product_id,
    sum(case when quantity is null then 1 else 0 end) as missing_quantity,
    sum(case when movement_type is null then 1 else 0 end) as missing_movement_type,
    sum(case when movement_date is null then 1 else 0 end) as missing_movement_date
from inventory_movements
```

119 %

	missing_product_id	missing_quantity	missing_movement_type	missing_movement_date
1	0	0	0	0

```
--check invalid movement type
select distinct movement_type
from inventory_movements
```

119 %

	movement_type
1	sale

4.2.14 Customer sessions

```
--check null values
select
    sum(case when customer_id is null then 1 else 0 end) as missing_customer_id,
    sum(case when session_start is null then 1 else 0 end) as missing_session_start,
    sum(case when session_end is null then 1 else 0 end) as missing_session_end,
    sum(case when ip_address is null then 1 else 0 end) as missing_ip_address
from customer_sessions
```

119 %

	missing_customer_id	missing_session_start	missing_session_end	missing_ip_address
1	0	0	0	0

```
--check sessions with invalid end time and start time
select *
from customer_sessions
where session_end < session_start
```

119 %

	id	customer_id	session_start	session_end	ip_address
--	----	-------------	---------------	-------------	------------

4.3 Basic Queries

4.3.1 Total sales revenue from all orders.

```
Basic And Advanced...164|TPQ\User (54)  ✘ × SQLQuery1.sql - DE...-164|TPQ\User (51))*
1 2 /********************************************************************* Basic And Advanced Queries *****/
3 3 [--1) total sales revenue from all orders
4 4
5 5 select sum(total_amount) as total_amounts
6 6 from orders
7 7
8 8 select sum(total_amount) as total_amounts
9 9 from orders
10 10 where status = 'delivered'
11 11
```

100 %

	total_amounts
1	44943966.02

	total_amounts
1	8873678.24

4.3.2 Top 5 best-selling products by quantity.

The screenshot shows a SQL query window with the following code:

```
12 --> List the top 5 best-selling products by quantity sold.
13
14 SELECT TOP 5
15     p.id AS product_id,
16     p.name AS product_name,
17     SUM(od.quantity) AS total_quantity_sold
18     FROM
19     order_details od
20     JOIN
21     Products p ON od.product_id = p.id
22     GROUP BY
23     p.id, p.name
24     ORDER BY
25     total_quantity_sold DESC;
26
27 select * from orders
```

The results table shows the top 5 products:

	product_id	product_name	total_quantity_sold
1	591	Vision-oriented scalable archive	253
2	499	Versatile holistic help-desk	252
3	216	Up-sized interactive time-frame	245
4	195	Vision-oriented 3rd-generation data-warehouse	242
5	348	Team-oriented discrete hierarchy	241

4.3.3 Customers with the highest number of orders.

The screenshot shows a SQL query window with the following code:

```
28
29
30 --> Identify customers with the highest number of orders.
31
32 SELECT
33     c.id + ' ' + c.last_name AS full_name,
34     COUNT(o.customer_id) AS num_of_orders
35     FROM customers c
36     JOIN orders o ON c.id = o.customer_id
37     GROUP BY c.id, c.first_name, c.last_name
38     ORDER BY num_of_orders DESC;
```

The results table shows the top 16 customers:

	full_name	num_of_orders
1	212 Ryan Chang	73
2	90 Suzanne Bennett	68
3	134 Robyn Reed	68
4	116 Zachary Aguine	66
5	189 Mark Spence	65
6	190 Leslie Alvarado	64
7	188 Tiffany Boone	64
8	91 Dana Farrell	64
9	55 John Perez	64
10	61 Emily Tornes	63
11	60 Emily Walker	62
12	71 Danielle Barton	62
13	78 David Rivas	62
14	206 Christopher Dunn	62
15	236 Thomas Jensen	62
16	207 Tiffany Harrison	61

4.3.4 Alert for low-stock products (under 20 units).

```

Basic And Advanced...164ITPQ\User (54) -> SQLQuery1.sql - DE...-164ITPQ\User (51)*
59
40 --4) Generate an alert for products with stock quantities below 20 units
41 SELECT id, name, stock_quantity , 'LOW STOCK ALERT' AS AlertMessage
42 FROM products
43 WHERE stock_quantity < 20;
44

```

Results				
	id	name	stock_quantity	AlertMessage
1	1	Enterprise-wide stable synergy	14	LOW STOCK ALERT
2	2	Synergized uniform contingency	17	LOW STOCK ALERT
3	3	Function-based well-modulated intranet	6	LOW STOCK ALERT
4	8	Business-focused zero tolerance functionalities	8	LOW STOCK ALERT
5	9	Networked bandwidth-monitored secured line	12	LOW STOCK ALERT
6	10	Secured reciprocal support	10	LOW STOCK ALERT
7	11	Visionary regional budgetary management	16	LOW STOCK ALERT
8	12	Advanced modular capacity	3	LOW STOCK ALERT
9	15	Focused radical protocol	6	LOW STOCK ALERT
10	16	Innovative context-sensitive matrices	19	LOW STOCK ALERT
11	19	Right-sized didactic function	7	LOW STOCK ALERT
12	20	Reverse-engineered fresh-thinking success	6	LOW STOCK ALERT
13	22	Vision-oriented solution-oriented hub	17	LOW STOCK ALERT
14	23	Cross-platform homogeneous ability	16	LOW STOCK ALERT
15	24	Universal context-sensitive hub	11	LOW STOCK ALERT
16	25	Exclusive actuating open system	6	LOW STOCK ALERT
17	27	Secured maximized structure	11	LOW STOCK ALERT
18	29	Synchronized multi-tasking hierarchy	19	LOW STOCK ALERT
19	30	Synergistic intangible product	7	LOW STOCK ALERT
20	31	Exclusive interactive instruction set	16	LOW STOCK ALERT

4.3.5 Percentage of orders with discounts.

```

--5)Determine the percentage of orders that used a discount
WITH
total_orders AS (
    SELECT COUNT(*) AS totalOrdersCnt
    FROM orders
    WHERE status <> 'cancelled'
),
orders_with_discount AS (
    SELECT COUNT(DISTINCT o.id) AS orderWithDiscnt
    FROM orders o
    JOIN order_details od ON od.order_id = o.id
    JOIN discounts d
        ON d.is_active = 1
        AND o.order_date BETWEEN d.start_date AND d.end_date
        AND (
            (d.product_id = od.product_id) OR
            (d.category_id = (SELECT category_id FROM products p WHERE p.id = od.product_id))|
            WHERE o.status <> 'cancelled')
    SELECT
        cast(ROUND(owd.orderWithDiscnt * 100.0 / to_cnt.totalOrdersCnt, 2) as float) AS percent_with_discount
    FROM orders_with_discount owd
    JOIN total_orders to_cnt ON 1 = 1

```

Results	
	percent_with_discount
	18.56

4.3.6 Average product ratings.

```
Basic And Advanced...164ITPQ(User (54)) => X [SQLQuery1.sql - DE...164ITPQ(User (51))]

101 -----+
102 |(-6) Avg rating Per product
103 |select p_id , p_name as product_name ,
104 |       round(avg(r.rating*1.0),2) as Rating
105 |from customers c
106 |join reviews r
107 |on c.id = r.customer_id
108 |join products p
109 |on p.id = r.product_id
110 |group by p_id , p_name
111 |order by round(avg(r.rating*1.0),2) desc
112

100 %
```

	product_name	Rating
1	69 Grassroots intangible superstructure	5.000000
2	64 Optimized content-based help-desk	5.000000
3	45 Balanced content-based conglomeration	4.670000
4	11 Visionary regional budgetary management	4.330000
5	37 Centralized dynamic monitoring	4.330000
6	85 De-engineered optimal initiative	4.250000
7	326 Userfriendly discrete time-frame	4.130000
8	303 Synergistic demand-driven conglomeration	4.000000
9	220 Expanded attitude-oriented algorithm	4.000000
10	28 Digitized real administration help-desk	4.000000
11	84 Upgradable fresh-thinking model	4.000000
12	55 Focused solution-oriented monitoring	4.000000
13	75 Operative heuristic infrastructure	3.910000
14	389 Open-source didactic circuit	3.890000
15	181 Assimilated disintermediate initiative	3.850000

4.4 Advanced Queries

4.4.1 30-day customer retention post first purchase.

```
Basic And Advanced...164ITPQ(User (54)) => X [SQLQuery1.sql - DE...164ITPQ(User (51))]

113 |(-7)the 30-day customer retention rate after their first purchase
114 |with orderDates as (
115 |    select
116 |        concat(c.first_name , ' ', c.last_name) CustName, o.order_date, o.total_amount, o.status, ROW_NUMBER()
117 |        over(partition by c.id order by o.order_date) rn,
118 |        lag(order_date) over(partition by c.id order by o.order_date) firstOrder
119 |    from customers c
120 |    join orders o on c.id = o.customer_id
121 |    and o.status <> 'cancelled'
122 |), with2ndDate as (
123 |    select o.order_date, firstOrder.fstOrdDate, order_date sndOrdDate, DATEDIFF(Day, firstOrder.order_date, noOfDays)
124 |    from orderDates
125 |    where firstOrder is not null
126 |    and order_date <= DATEADD(DAY, 30, firstOrder)
127 |    and rn = 2
128 |), TOTALCUST AS
129 |(
130 |    select count(distinct CustName) totalCust from orderDates
131 |), retainedCustomers as (
132 |    select
133 |        count(CustName) retainedCust from with2ndDate
134 |        select *, round(cast(retainedCust * 100 as float) / totalCust, 2) [customer%]
135 |    from retainedCustomers
136 |    JOIN TOTALCUST ON 1 = 1
137 |

100 %
```

	retainedCust	totalCust	customer%
1	227	248	91.53

4.4.2 Product recommendations based on Wishlist co-purchases.

```

138 -->-- products frequently bought together with items in customer wishlists
139
140 -->-- select
141 -->--     w1.product_id as product_1_id,
142 -->--     w1.name          as product_1_name,
143 -->--     w2.product_id   as product_2_id,
144 -->--     w2.name          as product_2_name,
145 -->--     COUNT(*)        as purchase_frequency
146 -->-- from wishlists w1
147 -->--     JOIN wishlists w2
148 -->--     ON w1.customer_id = w2.customer_id
149 -->--     AND w1.product_id <> w2.product_id -- to not duplicate eg. AAA BBB - BBB AAA count 1 time
150 -->--     JOIN products p1
151 -->--     ON w1.product_id = p1.id
152 -->--     JOIN products p2
153 -->--     ON w2.product_id = p2.id
154 -->--     GROUP BY w1.product_id, p1.name, w2.product_id, p2.name
155 -->--     ORDER BY COUNT(*) DESC
156
157

```

Results grid:

product_1_id	product_1_name	product_2_id	product_2_name	purchase_frequency
1	246	546	Switchable zero-defect productivity	4
2	127	287	Horizontal non-volatile installation	3
3	301	273	Reverse-engineered methodology	3
4	246	389	Open-source deduct circuit	3
5	319	390	Visionary regional methodology	2
6	322	417	Pre-emptive systematic middleware	2
7	147	423	Virtual systemic benchmark	2
8	276	406	Innovative methodical superstructure	2

4.4.3 30-day moving average for inventory turnover trends.

```

--9)Inventory turnover trends using a 30-day moving average.
SELECT
    s.order_date AS [date],
    s.total_sales AS [daily_sales],
    i.inventory_value AS [daily_inventory_value],
    ROUND(s.total_sales / NULLIF(i.inventory_value, 0), 2) AS inventory_turnover,
    ROUND(
        AVG(s.total_sales / NULLIF(i.inventory_value, 0)) OVER (
            ORDER BY s.order_date
            ROWS BETWEEN 29 PRECEDING AND CURRENT ROW), 2) AS ma_30_turnover
FROM (SELECT order_date, SUM(total_amount) AS total_sales
      FROM orders
      GROUP BY order_date) AS s
INNER JOIN (
    SELECT im.movement_date, SUM(im.quantity * p.price) AS inventory_value
    FROM inventory_movements AS im
    INNER JOIN products AS p ON im.product_id = p.id
    GROUP BY im.movement_date) AS i
ON s.order_date = i.movement_date
ORDER BY s.order_date;

```

Results grid:

date	daily_sales	daily_inventory_value	inventory_turnover	ma_30_turnover
2025-01-01 00:03:42.000	1274.37	-1274.37	-1.000000	-1.000000
2025-01-01 00:20:43.000	6220.98	-6220.98	-1.000000	-1.000000
2025-01-01 00:21:45.000	5840.58	-5840.58	-1.000000	-1.000000

4.4.4 Identify customers who purchased every product in a category.

The screenshot shows a SQL query in the 'Basic And Advanced...164|TPQ\User (54)*' window. The code uses common table expressions (CTEs) to calculate the number of products per category and per customer, then joins these with order details and products to find matches. The results are displayed in the 'Results' tab, showing columns for customer_id and category_id.

```
Basic And Advanced...164|TPQ\User (54)*  SQLQuery1.sql - DE...-164|TPQ\User (51)*
210 -->-->Customers who have purchased every product in a specific category
211 with num_products_per_category as
212 (select category_id, count(distinct id) as num_of_products
213  from products
214  group by category_id),
215 customer_purchased_products_per_category as
216 (select
217  o.customer_id,
218  p.category_id,
219  count(distinct od.product_id) as num_of_products
220  from orders o
221  join order_details od on o.id = od.order_id
222  join products p on od.product_id = p.id
223  group by o.customer_id, p.category_id),
224 filters as
225 (select
226  cpppc.customer_id,
227  cpppc.category_id,
228  cpppc.num_of_products
229  from num_products_per_category nppc
230  join customer_purchased_products_per_category cppc
231  on nppc.category_id = cppc.category_id
232  and nppc.num_of_products = cppc.num_of_products)
233 select
234  customer_id,
235  category_id
236  from filters
```

100 %

Results Messages

customer_id	category_id
-------------	-------------

4.4.5 Discover product pairs commonly bought together.

The screenshot shows a SQL query in the 'Extra Queries.sql ...P-164|TPQ\User (55)' window. The code uses a self-join on the order_products table to find pairs of products bought together in the same order. It then counts the number of times each pair appears and filters for pairs bought more than once. The results are displayed in the 'Results' tab, showing columns for FirstProduct_ID, FirstProduct_Name, SecondProduct_ID, SecondProduct_Name, and times_bought_together.

```
Extra Queries.sql ...P-164|TPQ\User (55)  Basic And Advanced...164|TPQ\User (54)*  SQLQuery1.sql - DE...-164|TPQ\User (51)*
240 |-11) Find pairs of products commonly bought together in the same order.
241 WITH order_products AS (
242     SELECT
243         order_id,
244         product_id,
245         p.name
246     FROM order_details od
247     JOIN products p ON od.product_id = p.id
248 )
249     SELECT
250         p1.product_id AS FirstProduct_ID,
251         p1.name AS FirstProduct_Name,
252         p2.product_id AS SecondProduct_ID,
253         p2.name AS SecondProduct_Name,
254         COUNT(*) AS times_bought_together
255     FROM
256     order_products p1
257     JOIN
258     order_products p2 ON p1.order_id = p2.order_id
259     WHERE
260         p1.product_id < p2.product_id -- Avoid duplicates and self-pairs
261     GROUP BY
262         p1.product_id, p1.name, p2.product_id, p2.name
263     HAVING
264         COUNT(*) > 1 -- Only show pairs bought together more than once
265     ORDER BY p1.product_id
```

100 %

Results Messages

FirstProduct_ID	FirstProduct_Name	SecondProduct_ID	SecondProduct_Name	times_bought_together
1	Enterprise-wide stable synergy	36	Total 24hour synergy	2
2	Monitored bifurcated database	28	Digitalized zero administration help-desk	2
3	Versatile modular info-mediaries	49	Profit-focused solution-oriented benchmark	2
4	Versatile modular info-mediaries	51	Diverse impactful encryption	2

4.4.6 Calculate order delivery times in days.

The screenshot shows a SQL Server Management Studio window with three tabs: 'Extra Queries.sql ...P-164ITPQ\User (55)', 'Basic And Advanced...164ITPQ\User (54)*', and 'SQLQuery1.sql - DE...-164ITPQ\User (51)*'. The code in the first tab is:

```
267 --12) Time taken to deliver orders in days
268 select o.id, DATEDIFF(DAY, o.order_date, s.shipping_date) AS "shipping duration days"
269 from shipping s
270 join orders o on s.order_id = o.id
271 where s.status = 'delivered'
272 order by "shipping duration days" desc
273
274 select order_id , count(1)from shipping
275 group by order_id
276 having count(1) > 1
```

The results tab displays a table with two columns: 'id' and 'shipping duration days'. The data is as follows:

id	shipping duration days
1	8
2	9
3	17
4	33
5	34
6	36
7	38
8	39
9	44
10	45
11	47
12	52
13	55
14	57
15	58
16	62

4.5 Extra Queries

4.5.1 Customer's Table Exploration

```
35 --Customers table
36 select * from customers
37
38 --check null values
39 select
40 sum(case when first_name is null then 1 else 0 end) as missing_first_name,
41 sum(case when last_name is null then 1 else 0 end) as missing_last_name,
42 sum(case when email is null then 1 else 0 end) as missing_email,
43 sum(case when phone is null then 1 else 0 end) as missing_phone,
44 sum(case when address is null then 1 else 0 end) as missing_address,
45 sum(case when registration_date is null then 1 else 0 end) as missing_registration_date
46 from customers;
47
48 --check for duplicated emails
49 select email, count(*) as count
50 from customers
51 group by email
52 having count(*) > 1;
53
54 --check for duplicated phone
55 select phone, count(*) as count
56 from customers
57 group by phone
58 having count(*) > 1;
59
60
61 --remove extra spaces from emails
62 update customers
63 set email = ltrim(rtrim(email))
64 where email like ' %' or email like '% ';
```

4.5.2 Monthly total sales

The screenshot shows a SQL query window titled "Basic And Advanced...164ITPQ\User (54)*". The query is as follows:

```
598 -----TOTAL AMOUNT OVER TIME-----
599 select
600     datename(month, order_date) as order_month,
601     sum(total_amount) as total_sales
602 from
603     orders
604 group by
605     --datepart(year, order_date),
606     --datepart(quarter, order_date),
607     datename(month, order_date),
608     datepart(month, order_date)
609 order by
610     --datepart(year, order_date),
611     --datepart(quarter, order_date),
612     datepart(month, order_date)
```

The results table shows the total sales for each month from January to May:

order_month	total_sales
January	11000631.35
February	10074306.56
March	10818113.43
April	10793091.56
May	2257823.12

4.5.3 # of new Customer per Month

The screenshot shows a SQL query window titled "Basic And Advanced...164ITPQ\User (54)*". The query is as follows:

```
814
815 use ecommerce_db
816 -- new customer per month
817 SELECT
818     FORMAT(registration_date, 'yyyy-MM') AS Month,
819     COUNT(*) AS NewCustomers
820 FROM
821     Customers
822 GROUP BY
823     FORMAT(registration_date, 'yyyy-MM')
824 ORDER BY
825     Month
```

The results table shows the number of new customers per month from January to May:

Month	NewCustomers
2025-01	78
2025-02	41
2025-03	64
2025-04	57
2025-05	10

4.5.4 Top 10 Categories by Revenue

```

470 -- Calculate Top Selling Categories by Revenue
471 SELECT TOP 10
472     c.name AS CategoryName,
473     SUM(od.quantity * od.unit_price) AS TotalRevenue
474     FROM Categories c
475     JOIN Products p ON c.id = p.category_id
476     JOIN Order_Details od ON p.id = od.product_id
477     GROUP BY c.name
478     ORDER BY TotalRevenue DESC;
479
480

```

100 %

Results Messages

	CategoryName	TotalRevenue
1	Hold	4112031.82
2	Recent	2821877.22
3	Marriage	2818829.80
4	Half	2505758.30
5	Speech	2422420.17
6	Window	2359321.89
7	Share	2347130.64
8	Energy	2330269.31
9	Program	2306954.76
10	Adult	2278867.09

Query executed successfully.

DESKTOP-164ITPQ (16.0 RTM) | DESKTOP-164ITPQ\User (55) | ecommerce_db |

4.5.5 Total Sales by Category/sub-category

```

Extra Queries.sql -...P-164ITPQ\User (55)* ⇨ Basic And Advanced...164ITPQ\User (54)* SQLQuery1.sql - DE...-164ITPQ\User (51)*
576 -----TOTAL AMOUNT (CATEGORY --> SUB-CATEGORY --> PRODUCTS)-----
577 select
578     parent_category.name as category_name,
579     child_category.name as sub_category_name,
580     products.name as product_name,
581     SUM(order_details.quantity * order_details.unit_price) as total_sales
582 from
583     order_details
584 join
585     products on order_details.product_id = products.id
586 join
587     categories as child_category on products.category_id = child_category.id
588 left join
589     categories as parent_category on child_category.parent_id = parent_category.id
590 group by
591     parent_category.name,
592     child_category.name,
593     products.name
594 order by
595     parent_category.name,
596     child_category.name,
597     total_sales desc
598
599 -----TOTAL AMOUNT OVER TIME-----

```

100 %

Results Messages

	category_name	sub_category_name	product_name	total_sales
1	NULL	Choose	Programmable regional installation	184411.26
2	NULL	Choose	Sharable non-volatile customer loyalty	160826.86
3	NULL	Choose	Optimized uniform strategy	130002.18
4	NULL	Choose	Self-enabling non-volatile standardiz...	125000.00
5	NULL	Choose	Inverse empowering approach	123696.32
6	NULL	Choose	Front-line real-time algorithm	122234.40

4.5.6 Extract Region From address column in customer table

The screenshot shows a SQL Server Management Studio window with three tabs: 'Extra Queries.sql' (P-164ITPQ\User (55)*), 'Basic And Advanced...164ITPQ\User (54)*', and 'SQLQuery1.sql - DE...-164ITPQ\User (51)*'. The 'Extra Queries.sql' tab contains the following script:

```
283 --Add Column Region
284 ALTER TABLE customers
285 ADD Region VARCHAR(10);
286
287 UPDATE customers
288 SET Region =
289     REVERSE(
290         SUBSTRING(
291             REVERSE(address),
292             CHARINDEX(' ', REVERSE(address)) + 1,
293             CHARINDEX(' ', REVERSE(address)), CHARINDEX(' ', REVERSE(address)) + 1
294             - CHARINDEX(' ', REVERSE(address)) - 1
295         )
296     );
297
298 --To convert Abbreviations to long fullname
299 ALTER TABLE customers
300 ALTER COLUMN region VARCHAR(100);
301
302 --Convert Abbreviations to long fullname
303 UPDATE customer...
304 select DISTINCT Region from customers
```

The 'Results' tab shows the output of the query:

	Region
1	AA
2	AE
3	AK
4	AL
5	AP
6	AR
7	AS

4.5.7 Identify Military Regions

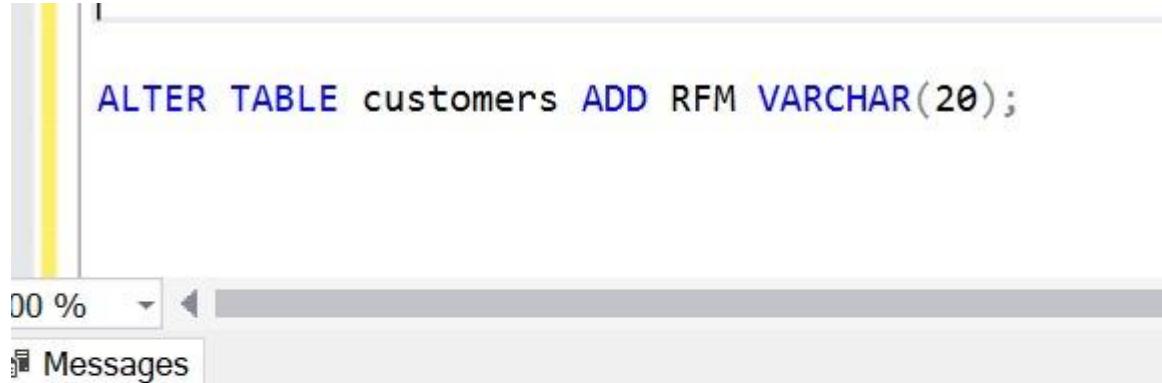
The screenshot shows a SQL Server Management Studio window with three tabs: 'Extra Queries.sql' (P-164ITPQ\User (55)*), 'Basic And Advanced...164ITPQ\User (54)*', and 'SQLQuery1.sql - DE...-164ITPQ\User (51)*'. The 'Extra Queries.sql' tab contains the following script:

```
--Identify Military Regions
SELECT distinct Region
FROM customers
WHERE Region LIKE 'Armed%'
```

The 'Results' tab shows the output of the query:

Region
Armed Forces Americas
Armed Forces Europe
Armed Forces Pacific

4.5.8 Create view Customer Segmentation

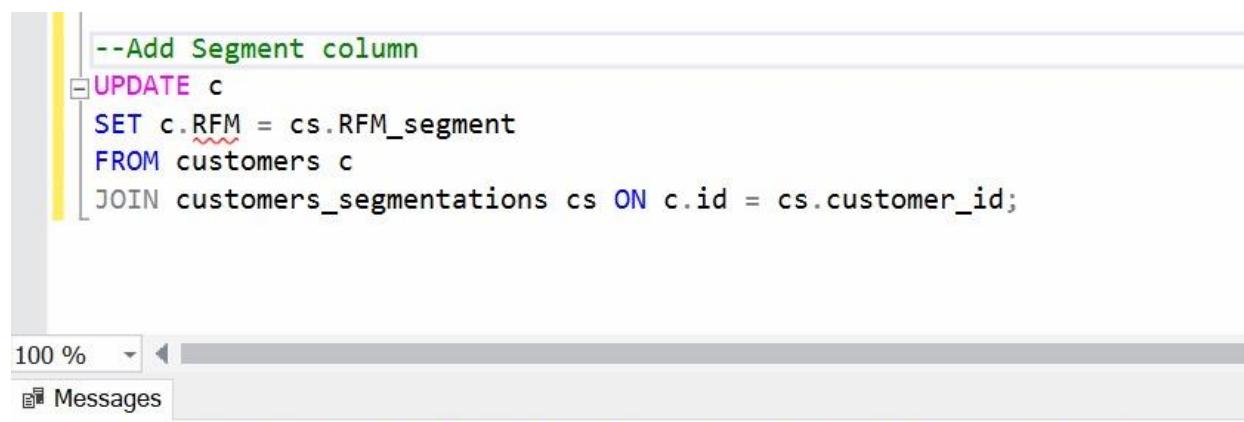


The screenshot shows a database interface with a code editor and a message log. The code editor contains the following SQL command:

```
ALTER TABLE customers ADD RFM VARCHAR(20);
```

The message log below it shows the command was completed successfully.

Completion time: 2025-05-18T15:27:12.0395003+03:00



The screenshot shows a database interface with a code editor and a message log. The code editor contains the following SQL query:

```
--Add Segment column
UPDATE c
SET c.RFM = cs.RFM_segment
FROM customers c
JOIN customers_segmentations cs ON c.id = cs.customer_id;
```

The message log below it shows a warning about null values and the completion time.

Warning: Null value is eliminated by an aggregate or other SET operation.

(250 rows affected)

Completion time: 2025-05-18T15:36:16.1736409+03:00

```

create view customers_segmentations
as
    with RFM as
    (
        select
            c.id as customer_id,
            c.first_name+' '+last_name as customer_name,
            datediff(day, max(o.order_date), getdate()) as recency_days,
            count(o.id) as frequency,
            sum(o.total_amount) as monetary
        from
            customers c
        left join
            orders o on c.id = o.customer_id
        group by
            c.id,
            c.first_name+' '+last_name
    ),
    scores as
    (
        select|
            customer_id,

```

100 % ▶ Messages

Commands completed successfully.

Completion time: 2025-05-18T15:05:07.2066745+03:00

```

--Add Segment column
UPDATE c
SET c.RFM = cs.RFM_segment
FROM customers c
JOIN customers_segmentations cs ON c.id = cs.customer_id;

select || from customers

```

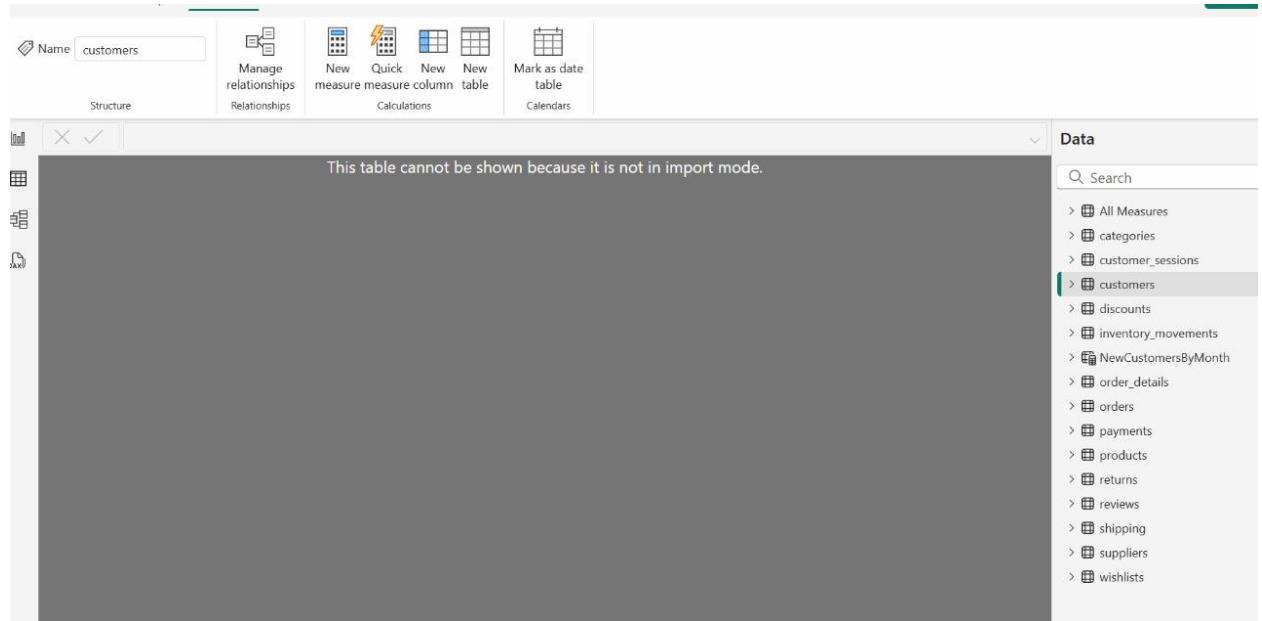
100 % ▶ Results ▶ Messages

	first_name	last_name	email	phone	address	registration_date	Region	RFM
1	Oretta	Weaver	matthewesstephen@example.net	+10000000001	8649 Gonzales Stream Jeffreyft, NV 63120	2025-01-15 18:12:50.000	Nevada	Normal
2	Jennifer	Kelley	xwilson@example.net	+10000000002	7838 Dustin Skyway Lake Robin, AS 24112	2025-03-17 03:46:07.000	American Samoa	Normal
3	Michael	Mcintosh	alisoncollins@example.com	+10000000003	68025 David Squares Apt. 363 Lake Josephstad, MA ...	2025-04-25 16:08:23.000	Massachusetts	Normal
4	James	Nguyen	pacesarah@example.com	+10000000004	PSC 9889, Box 2185 APO AA 28386	2025-02-19 09:51:26.000	Armed Forces Americas	Normal
5	Sheila	Cohen	natashaa33@example.com	+10000000005	03530 Howard Knolls Suite 638 Hickshaven, AS 71445	2025-04-24 00:30:40.000	American Samoa	Normal
6	Christina	Himes	lisa26@example.net	+1-679-559-8927	2380 Christopher Forges West Josephtown, VI 18531	2025-04-11 19:41:49.000	Virgin Islands	Normal
7	Susan	Barron	amythompson@example.net	+10000000007	45291 Harmon Bridge Suite 357 New Amymouth, ND ...	2025-01-16 16:21:58.000	North Dakota	Normal
8	Jason	Lewis	frankcollins@example.org	+10000000008	71391 Erica Trace Suite 798 Christopherberg, KS 51685	2025-05-03 02:18:12.000	Kansas	Normal
9	Jana	Ross	anthonyortega@example.net	736-395-1083	85571 Michele Vista Gomezfurt, ID 67434	2025-04-04 00:04:58.000	Idaho	Normal
10	Vanessa	Garcia	markellott@example.org	683-843-1063	30963 Lisa Dale Butlerfort, ME 06545	2025-01-16 19:22:43.000	Maine	Normal
11	Ammy	Johnson	ronaldo02@example.org	+1-978-496-5613	698 Farmer Junctions Suite 193 South Dusinburgh, D...	2025-01-20 22:14:13.000	District of Columbia	Normal
12	Jim	Lowery	harperjohn@example.com	+10000000012	634 Mason Harbor Apt. 165 West Jesse, WV 12562	2025-04-26 20:20:40.000	West Virginia	Normal
13	Undrea	Miller	jameskathryn@example.org	+10000000013	723 Patty Locks Apt. 286 Lake Kristinemouth, NC 50801	2025-03-25 19:49:53.000	North Carolina	Normal
14	Cathryn	Hart	jeremywallace@example.net	2989993268	058 Brianna Neck Warnerland, WV 92032	2025-03-25 16:37:31.000	Wyoming	Normal
15	Donna	Ward	charlene10@example.net	+10000000015	PSC 4359, Box 3660 APO AA 71109	2025-02-16 08:58:00.000	Armed Forces Americas	Normal
16	Matthew	Smith	tcurtis@example.net	(943)877-8436	53910 Matthew Fall Apt. 520 South Michaelton, HI 04...	2025-01-04 02:23:44.000	Hawaii	Normal

5 Power BI Integration and Dashboarding

5.1 Direct Query Configuration

The data relationships were entered manually (direct query without relation)



5.2 Dashboard Components

5.2.1 Dax functions

```
TotalOrders = COUNT(orders[id])
```

```
TotalOrders = COUNT(orders[id])
```

```
TotalCustomers =
DISTINCTCOUNT(Customers[id])
```

```
Total Revenue =
SUMX('order_details', 'order_details'[quantity] *
'order_details'[unit_price])
```

```
Total Products = COUNTROWS(products)
```

```
ReturnRate =
VAR TotalOrders = COUNTROWS(Orders)
VAR ReturnedOrders = DISTINCTCOUNT(Returns[order_id])
```

```

RETURN
IF(
    TotalOrders = 0,
    BLANK(),
    DIVIDE(ReturnedOrders, TotalOrders, 0) * 100)

RepeatPurchaseRate =
VAR CustomerOrderCounts =
ADDCOLUMNS(
    VALUES(Orders[customer_id]),
    "OrderCount", CALCULATE(COUNTROWS(Orders)))
)
VAR RepeatCustomers =
COUNTROWS(FILTER( CustomerOrderCounts , [OrderCount] > ))
VAR TotalCustomers =
COUNTROWS(CustomerOrderCounts)
RETURN DIVIDE(RepeatCustomers, TotalCustomers) * 100

```

```

RepeatPurchaseRate =
VAR CustomerOrderCounts =
ADDCOLUMNS( VALUES(Orders[customer_id]), "OrderCount",
CALCULATE(COUNTROWS(Orders)) )

VAR RepeatCustomers = COUNTROWS( FILTER( CustomerOrderCounts , [OrderCount]
> 1 ) )
) VAR TotalCustomers =
COUNTROWS(CustomerOrderCounts)
RETURN DIVIDE(RepeatCustomers, TotalCustomers) * 100

```

```

Recency =
VAR LastOrderDate =
CALCULATE(
    MAX(Orders[order_date]),
    FILTER(Orders, Orders[customer_id] = SELECTEDVALUE(Customers[id])))
)
RETURN
DATEDIFF(LastOrderDate, TODAY(), DAY)

```

```
Number of Products in Wishlist = COUNTROWS(wishlists)
```

```

NonOrderCustomerCount =
COUNTROWS( EXCEPT( VALUES(Customers[id]) , VALUES(Orders[Customer_ID])
)
)
```

```

NonOrderCustomerCount =
COUNTRWS ( EXCEPT ( VALUES(Customers[id]),
                      VALUES(Orders[Customer_ID]) ) )
)

LastMonthSales =
VAR CurrentMonth =
    DATE(YEAR(MAX(Orders[order_date])), MONTH(MAX(Orders[order_date])), 1)
RETURN CALCULATE( SUM(Orders[total_amount]), FILTER( ALL(Orders),
    DATE(YEAR(Orders[order_date]), MONTH(Orders[order_date]), 1) =
EDATE(CurrentMonth, -1)
)
)

Growth% =
IF( [LastMonthSales] = 0,
    BLANK(), DIVIDE( [CurrentMonthSales] - [LastMonthSales],
        [LastMonthSales]
    ) * 100
)

Frequency =
CALCULATE(
    COUNT(Orders[id]),
    FILTER(Orders, Orders[customer_id] = SELECTEDVALUE(Customers[id]))
)

CustomerSegment =
SWITCH(
    TRUE(),
    [Frequency] > 60, "gold",
    [Frequency] > 40, "silver",
    [Frequency] > 30, "bronze",
    "normal"
)

CurrentMonthSales =
VAR CurrentMonth =
    DATE(YEAR(MAX(Orders[order_date])), MONTH(MAX(Orders[order_date])), 1)
RETURN
    CALCULATE(
        SUM(Orders[total_amount]),
        FILTER(
            ALL(Orders),
            DATE(YEAR(Orders[order_date]), MONTH(Orders[order_date]), 1) =
CurrentMonth
        )
    )

```

```
AOV = DIVIDE(SUM(Orders[total_amount]), COUNTROWS(Orders))
```

```
#CustInDis =
CALCULATE(
    COUNT(orders[customer_id]),
    FILTER(
        order_details,
        VAR currentProduct = order_details[product_id]
        VAR currentOrderDate =
            LOOKUPVALUE(
                orders[order_date],
                orders[id], order_details[order_id] )
    RETURN
        CALCULATE(
            COUNTROWS(discounts),
            FILTER(
                discounts,
                discounts[product_id] = currentProduct &&
                currentOrderDate >= discounts[start_date] &&
                currentOrderDate <= discounts[end_date] ) ) ) > ))
```

5.2.2 Dashboards

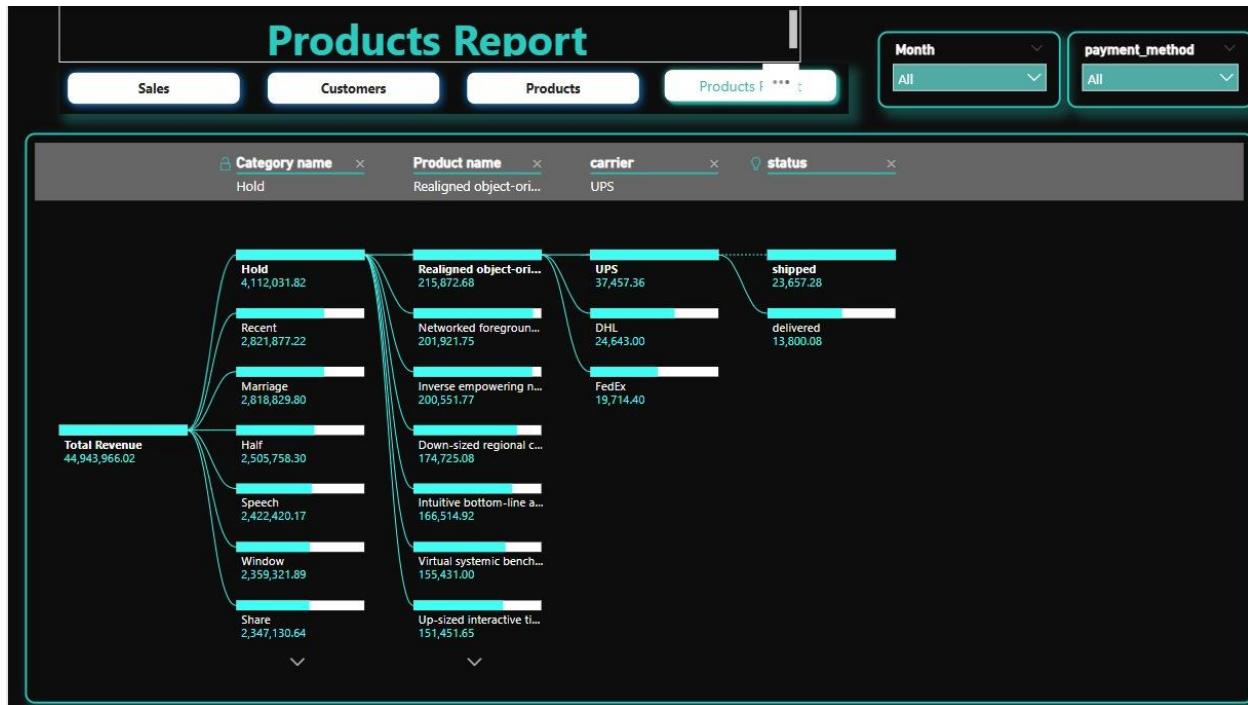
5.2.2.1 Sales performance



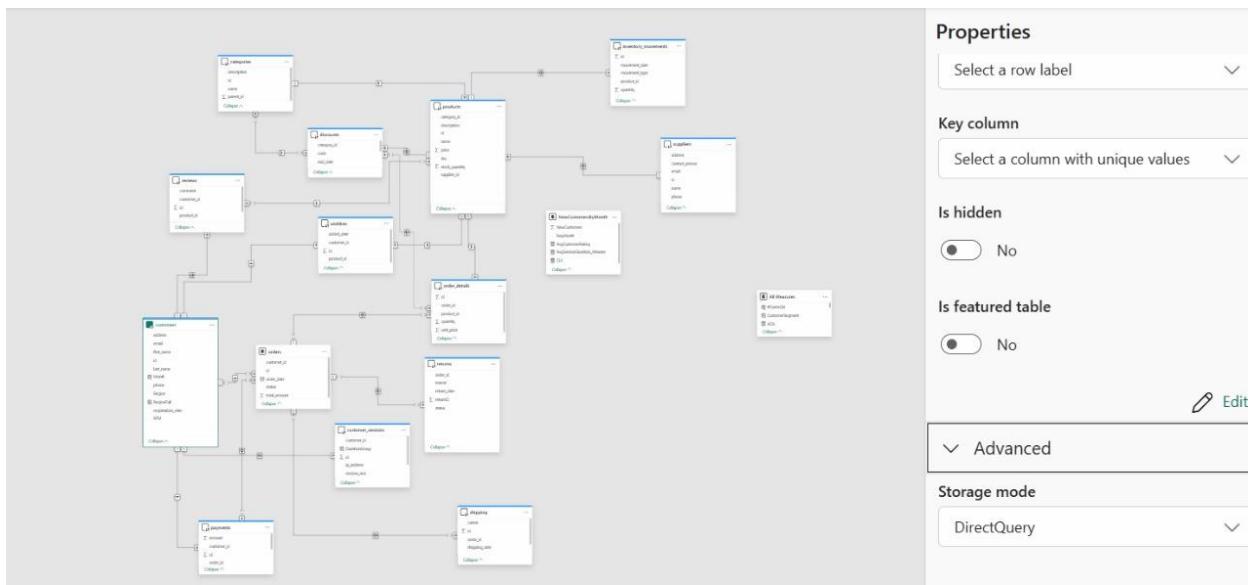
5.2.2.2 Customer Segmentation



5.2.2.3 Product analytics



5.3 Data Modeling



6 Advanced Insights and Machine Learning

6.1 Automated Data Cleaning by Mito

Mito is an interactive tool that simplifies data cleaning by offering a spreadsheet-like interface integrated with Python. In this project, it was used to streamline preprocessing tasks such as handling missing values, formatting, and column analysis without complex coding.

6.1.1 Order Details Table

The screenshot shows the Mito interface with a code editor. The code is as follows:

```

info

Let's start by displaying
general information
about the DataFrame
df using the .info()
and .describe()
methods. This will give us
insight into the columns,
their data types, and
basic statistics.

# Show DataFrame info
print('DataFrame Info:')
df.info()

# Show summary statistics for numeric columns
print('\nSummary Statistics:')
df.describe()

```

The output of the code is displayed below the code editor, showing the DataFrame Info and Summary Statistics.

Now let's print the first 10 rows of `df` with `.head(10)` to get a better idea of the example data in each column.

```
# Display the first 10 rows
df.head(10)
```

Here is a markdown summary of what we've learned so far about your `order_details` data: columns, data types, and a preview of the first

Ask agent to do anything

Would like to run more other data quality checks or begin analysis!

nulls

Let's check each column in `df` for missing (null) values by displaying the sum of nulls per column.

```
# Check for missing (null)
null_counts = df.isnull().sum()
print('Missing values per column:')
null_counts
```

None of the columns contain missing (null)

trim

Let's remove any leading or trailing whitespace in string columns. We'll use `.str.strip()` on all object (string) columns, which are: `id`, `order_id`, and `product_id`.

```
# Strip Leading/trailing whitespace
for col in ['id', 'order_id', 'product_id']:
    df[col] = df[col].str.strip()

df.head()
```

To finish the cleaning process, let's also trim

`df.head(10)`

Home Insert Data Formulas Code

id | id

	id	order_id	product_id	quantity	unit_price
	int	int	int	int	float
0	1	1	38	3	173.95
1	2	2	35	5	23.95
2	3	2	46	3	105.72
3	4	3	36	1	993.24
4	5	3	83	5	416.15
5	6	3	29	5	658.88
6	7	3	48	1	144.16
7	8	3	13	4	299.75
8	9	4	32	5	933.87
9	10	4	38	3	173.95

order_details.ipynb

order_details.ipynb

Code New MitoSheet Notebook

[10]: # Check for missing (null) values in
null_counts = df.isnull().sum()
print('Missing values per column:')
null_counts

Missing values per column:

```
[10]: id      0
order_id  0
product_id  0
quantity   0
unit_price  0
dtype: int64
```

Null Value Check Summary

No columns in the 'order_details' DataFrame contain missing (null) values; all entries are present.

[11]:

[]:

[]:

[]:

[]:

```
[13]: # Trim all string/object columns in df for maximum cleanliness (future-proofing)
for col in df.select_dtypes(include='object').columns:
    df[col] = df[col].str.strip()

df.head()
```

Home Insert Data Formulas Code

id | id

	id	order_id	product_id	quantity	unit_price
	str	str	str	int	float
0	1	1	38	3	173.95
1	2	2	35	5	23.95
2	3	2	46	3	105.72
3	4	3	36	1	993.24
4	5	3	83	5	416.15

processing or analysis

check outliers

Let's visually and statistically check for outliers in the two numeric columns, `quantity` and `unit_price`. We'll use boxplots for each column and print basic outlier stats using the IQR (Interquartile Range) method.

```
import matplotlib.pyplot as plt
import numpy as np
```

Visualize outliers with boxplots
`fig, axes = plt.subplots(1, 2, figsize=(12, 5))
df.boxplot(column='quantity', ax=axes[0])
axes[0].set_title('Quantity')
df.boxplot(column='unit_price', ax=axes[1])
axes[1].set_title('Unit Price')
plt.tight_layout()
plt.show()`

IQR method for both columns
`for col in ['quantity', 'unit_price']:
 q1 = df[col].quantile(0.25)
 q3 = df[col].quantile(0.75)
 iqr = q3 - q1
 lower_bound = q1 - 1.5 * iqr
 upper_bound = q3 + 1.5 * iqr
 outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
 print(f'({col}) outlier count: {outliers.shape[0]}')
if not outliers.empty:
 print(outliers[['id', col]].head())`

check outliers

Let's visually and statistically check for outliers in the two numeric columns, `quantity` and `unit_price`. We'll use boxplots for each column and print basic outlier stats using the IQR (Interquartile Range) method.

```
import matplotlib.pyplot as plt
import numpy as np
```

Visualize outliers with boxplots
`fig, axes = plt.subplots(1, 2)
df.boxplot(column='quantity', ax=axes[0])
df.boxplot(column='unit_price', ax=axes[1])`

Ask agent to do anything

Chat Agent Submit

Outlier Analysis Summary

No outliers were identified in the columns `quantity` or `unit_price` using the IQR (Interquartile Range) method. Both fields appear free of extreme values under the chosen criteria.

Media Player

Activate Windows. Go to Settings to activate Windows.

concert to string id ,
order_id , product_id

Let's convert the `id`,
`order_id`, and
`product_id` columns in
`df` from int64 to string
types using
`.astype(str)`.

```
# Convert columns to string
columns_to_convert = ['id']
df[columns_to_convert] = df[columns_to_convert].astype(str)

# Display new dtypes to confirm
print(df.dtypes)
```

The columns `id`,
`order_id`, and
`product_id` have been
successfully converted to

Ask agent to do anything

mito-data-copilot.ipynb x order_details.ipynb x + New MitoSheet

Let me know your next analysis goal (sum sales, group by order, find popular products, etc.)!

```
[8]: # Convert columns to string type
columns_to_convert = ['id', 'order_id', 'product_id']
df[columns_to_convert] = df[columns_to_convert].astype(str)

# Display new dtypes to confirm
print(df.dtypes)
```

	<code>id</code>	<code>order_id</code>	<code>product_id</code>	<code>quantity</code>	<code>unit_price</code>	<code>total_sales</code>
0	1	1	56	5	173.95	521.85
1	2	2	35	5	23.95	119.75
2	3	2	46	3	105.72	317.16
3	4	3	36	1	993.24	993.24
4	5	3	93	5	416.15	2,080.75

```
[9]: # Find the number of duplicate rows
num_duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {num_duplicates}")

# Show all duplicate rows, if any
if num_duplicates > 0:
    display(df[df.duplicated(keep=False)])
```

Number of duplicate rows: 0

Duplicate Check Summary

No duplicate rows were found in the `order_details` DataFrame based on all columns.

Some of your recent edits
now documented in a
summary markdown cell.

add total sales column

Let's add a column called
`total_sales` to your
DataFrame, which will
multiply `quantity` by
`unit_price` for each
row.

```
# Add a 'total_sales' column (quantity * unit_price for each row)
df['total_sales'] = df['quantity'] * df['unit_price']
df.head()
```

A new column,
`total_sales`, has been
added to your
DataFrame. It calculates

the data further!

data types

Let's explicitly display the data types of each column in `df` using the `.dtypes` attribute.

```
# Display the data types of each column
df.dtypes
```

We've displayed the data types of all columns in your `order_details.csv` DataFrame, confirming that `id`, `order_id`, `product_id`, and

Summary of `order_details.csv`

- Columns and Types:**
 - `id` : unique row identifier (int64)
 - `order_id` : identifier grouping rows by order (int64)
 - `product_id` : which product was ordered (int64)
 - `quantity` : how many units were ordered (int64)
 - `unit_price` : price per unit (float64)
- The dataset contains order-level details suitable for sales analysis (e.g., sales by product, order size, revenue calculations).

Let me know your next analysis goal (sum sales, group by order, find popular products, etc.)!

Ask agent to do anything

6.1.2 products Table

describe

I'll display summary statistics for the numeric columns in your DataFrame using the `describe()` method.

```
df.describe()
```

You've generated summary statistics for all numeric columns in your products DataFrame (e.g., count, mean, std, min, max, quartiles) using the

	<code>id</code>	<code>price</code>	<code>category_id</code>	<code>supplier_id</code>	<code>stock_quantit</code>
	float	float	float	float	float
<code>count</code>	600.00	600.00	600.00	600.00	600.00
<code>mean</code>	300.50	485.12	9.88	23.08	153.14
<code>std</code>	173.35	284.19	5.61	14.87	65.82
<code>min</code>	1.00	10.98	1.00	1.00	0.00
<code>25%</code>	150.75	235.66	5.00	10.00	144.00
<code>50%</code>	300.50	464.79	9.00	20.00	173.50
<code>75%</code>	450.25	732.50	14.25	37.00	196.25
<code>max</code>	600.00	996.57	20.00	50.00	253.00

This helps you quickly review central tendency, dispersion, and outliers in your numeric data.

data types

I'll display the data types of each column in your DataFrame using the dtypes attribute.

```
[8]: df.dtypes
```

[8]:	df.dtypes
	id int64 name object description object price float64 category_id int64 supplier_id int64 sku object stock_quantity int64 dtype: object

You've displayed the data types for each column in your DataFrame using the dtypes attribute [\[1\]](#). This confirms which columns are integers, floats, etc.

id to object

I'll change the data type of the id column to 'object', which is suitable for non-numeric identifiers, and then display the updated dtypes.

```
[9]: df['category_id'] = df['category_id'].astype('object')  
df.dtypes
```

[9]:	df['category_id'] = df['category_id'].astype('object') df.dtypes
	id int64 name object description object price float64 category_id object supplier_id int64 sku object stock_quantity int64 dtype: object

```
[10]: df['id'] = df['id'].astype('object')  
df.dtypes
```

[10]:	df['id'] = df['id'].astype('object') df.dtypes
	id object name object description object price float64 category_id object supplier_id int64 sku object stock_quantity int64 dtype: object

duplicates

```
[11]: # Check for duplicate rows  
num_duplicates = df.duplicated().sum()  
num_duplicates
```

[11]:	# Check for duplicate rows num_duplicates = df.duplicated().sum() num_duplicates
	[11]: 0

info

I'll display basic information about the DataFrame, including column types and null counts, using the info() method.

```
df.info()
```

The basic info about your DataFrame has been displayed, including the

empty cells

I'll check for empty cells (i.e., cells that are blank but not necessarily NaN/null) by looking for empty strings in object-type columns and summing them per

trim

I'll trim leading and trailing whitespace from all object-type columns to ensure clean text fields. This is a common data cleaning step before analysis.

```
# Trim whitespace for all
for col in df.select_dtypes(include='object').columns:
    df[col] = df[col].apply(lambda x: x.strip() if isinstance(x, str) else x)
df.head()
```

Hmm, looks like my first attempt didn't work. Let's try again.

Untitled.ipynb

[6]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          600 non-null    int64  
 1   name         600 non-null    object  
 2   description  600 non-null    object  
 3   price        600 non-null    float64 
 4   category_id 600 non-null    int64  
 5   supplier_id 600 non-null    int64  
 6   sku          600 non-null    object  
 7   stock_quantity 600 non-null    int64  
dtypes: float64(1), int64(4), object(3)
memory usage: 37.6+ KB
```

[]:

[14]: # Check for empty strings in object-type columns
empty_counts = (df.select_dtypes(include='object') == '').sum()
empty_counts

	id	name	description	price	category_id	supplier_id
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

[16]: for col in df.select_dtypes(include='object').columns:
 df[col] = df[col].apply(lambda x: x.strip() if isinstance(x, str) else x)
df.head()

Home Insert Data Formulas Code

Untitled.ipynb

[]:

	id	name	description	price	category_id	supplier_id
0	1	Enterprise-wide stabilizer	Trial eight magazine	605.06	8	1
1	2	Synergized uniform culture	Cold cultural trade bin	678.15	10	2
2	3	Function-based well-being	Identify particular belief	712.42	9	3
3	4	Monitored bifurcated interface	Century note kind frequent	110.51	10	4
4	5	Versatile modular infrastructure	Interesting section even	285.49	9	5

outliers

I'll use the IQR (Interquartile Range) method to detect outliers in the numeric columns of your DataFrame. The result will show rows considered outliers for each numeric column.

```
[17]: # Detect outliers using the IQR method for all numeric columns
outlier_rows = pd.DataFrame()
for col in df.select_dtypes(include=['number']).columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    filtered = df[(df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR)]
    if not filtered.empty:
        filtered = filtered.copy()
        filtered['outlier_in'] = col
        outlier_rows = pd.concat([outlier_rows, filtered])
outlier_rows.reset_index(drop=True, inplace=True)
outlier_rows
```

Ask agent to do anything

id	id	name	description	price	category_id
8	9	Networked bandwidth	Statement where last	330.37	
9	10	Secured reciprocal su	Involve perhaps him i	751.34	
10	11	Visionary regional bu	Political describe sup	598.45	
11	12	Advanced modular ca	Their who want intere	716.47	
12	13	Horizontal didactic s	Every result explain. C	299.75	
13	14	Customizable object-	Son production site n	281.12	
14	15	Focused radical prot	Room mouth challeng	228.16	1
15	16	Innovative context-se	Power then song. Exp	248.85	1
16	17	Streamlined multi-sta	Enough dark young p	548.72	

+ outlier_rows ▾ (100 rows, 9 columns)

```
[18]: # Print IQR bounds for each numeric column to debug outlier detection
for col in df.select_dtypes(include=['number']).columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    print(f"(col): lower bound = {lower}, upper bound = {upper}")

outlier_rows.head()
```

id	id	name	description	price	category_id	supplier_id	sku	stock_quantity
0	1	Enterprise-wide stabl	Trial eight magazine	605.06	8	8	99ef99ca-2d89-4bfd-	
1	2	Synergized uniform c	Cold cultural trade bil	678.15	10	14	8dcbc611-5f75-41d5-	
2	3	Function-based well-t	Identify particular be	712.42	9	18	8f29473d-20e5-446c-	
3	4	Monitored bifurcated	Century note kind fre	110.51	10	2	eac97ea6-1bc9-4066	Windows

ideal for data analysis.

nulls

I'll count the number of null values in each column of the DataFrame to help identify missing data.

```
# Count null values in each column
null_counts = df.isnull().sum()
null_counts
```

You checked for null

upper = Q3 + 1.5 * IQR
print(f'{col}: lower {lower}\noutlier_rows.head()')
outlier_rows.head()

boxplot for outliers

I'll plot boxplots for the columns used in outlier detection ("price" and "stock_quantity") to visualize potential outliers. This will help you see the distribution and spot outliers graphically for each numeric feature.

```
import matplotlib.pyplot as plt
# Boxplots for columns high outlier columns = outlier_columns
```

Ask agent to do anything

Chat Agent Submit

Code

num_duplicates

[11]: 0

[13]: # Count null values in each column
null_counts = df.isnull().sum()
null_counts

[13]:

	id	name	description	price	category_id	supplier_id	sku	stock_quantity
	0	0	0	0	0	0	0	0

[]:

[14]: # Check for empty strings in object-type columns
empty_counts = (df.select_dtypes(include='object') == '').sum()

Boxplots Highlighting Outlier Detection Columns

stock_quantity

[25]:

```
# Remove rows flagged as outliers in any numeric column
if not outlier_rows.empty:
    outlier_indices = outlier_rows.index
    df_no_outliers = df.drop(index=outlier_indices).reset_index(drop=True)
else:
    df_no_outliers = df.copy()

# Show head of cleaned DataFrame
print('Rows remaining after outlier removal:', len(df_no_outliers))
df_no_outliers.head()
```

Rows remaining after outlier removal: 500

	id	name	description	price	category_id	supplier_id	sku	stock_quantity
0	101	Progressive human-re	Traditional fear add b	617.38	8	23	8639f5f3-0e1d-4458-	148
1	102	Versatile homogeneo	Production style ahead	233.00	17	28	67c01675-f506-4b85	155
2	103	Virtual executive flex	Story not plant increa	626.71	9	46	5d7e84a1-dab6-4593	221
3	104	Profound non-volatile	Respond campaign c	344.86	16	20	66faef3-55ec-40fa-8	235
4	105	Persistent user-facing	Cost suffer prove. Ca	532.62	4	39	ece441e4-4f05-47bc-	197

remainder of your data for further analysis. If you'd prefer another method (e.g., capping or imputing), let me know!

use clip for outliers

I'll use the "clip" method to cap the outliers in the "stock_quantity" column using the calculated IQR bounds (lower and upper). This keeps all rows in the DataFrame and limits extreme values without removing any data.

[26]:

```
# Cap outliers in 'stock_quantity' using the IQR bounds
clipped_df = df.copy()
lower, upper = 65.625, 274.625 # Previously calculated IQR bounds
clipped_df['stock_quantity'] = clipped_df['stock_quantity'].clip(lower, upper)

clipped_df.head()
```

	id	name	description	price	category_id	supplier_id
0	1	Enterprise-wide stabl	Trial eight magazine	605.06	8	8
1	2	Synergized uniform c	Cold cultural trade bil	678.15	10	14
2	3	Function-based well-	Identify particular bef	712.42	9	18
3	4	Monitored bifurcated	Century note kind fre	110.51	10	2
4	5	Versatile modular inf	Interesting section ev	285.49	9	6

Ask agent to do anything

new DataFrame, 'clipped_df'. Your data is now robust to outlier effects without losing any rows.

check outliers again

I'll check for outliers again in the clipped DataFrame ('clipped_df') using the IQR method for 'stock_quantity'. This will confirm whether capping using clip has removed all outliers.

Re-check for outliers after applying clipping
re_outlier_rows = pd.DataFrame()
for col in ['stock_quantity']:
 Q1 = clipped_df[col].quantile(0.25)
 Q3 = clipped_df[col].quantile(0.75)
 IQR = Q3 - Q1
 filtered = clipped_df[(clipped_df[col] < Q1 - 1.5 * IQR) | (clipped_df[col] > Q3 + 1.5 * IQR)]
 if not filtered.empty:
 filtered = filtered.copy()
 filtered['outlier_in'] = col
 re_outlier_rows = pd.concat([re_outlier_rows, filtered])
re_outlier_rows.reset_index(drop=True, inplace=True)
re_outlier_rows

No data in dataframe.

Ask agent to do anything

boxplot for outliers

I'll plot boxplots for the columns used in outlier detection ("price" and "stock_quantity") to visualize potential outliers. This will help you.

import matplotlib.pyplot as plt

Boxplots for columns highlighted in outlier detection
outlier_columns = outlier_rows['outlier_in'].unique().tolist() if not outlier_rows.empty else []
if len(outlier_columns) > 0:
 plt.figure(figsize=(max(6, len(outlier_columns) * 2), 6))
 df.boxplot(column=outlier_columns)
 plt.title('Boxplots Highlighting Outlier Detection Columns')
 plt.xticks(rotation=45)
 plt.tight_layout()
 plt.show()
else:
 print("No columns identified as having outliers.")

1. Remove duplicated SKUs if any (SKU should be unique for products)
if 'sku' in df.columns:
 df = df.drop_duplicates(subset=['sku'])

2. Ensure numerical columns are proper type (price, stock_quantity)
df['price'] = pd.to_numeric(df['price'], errors='coerce')
df['stock_quantity'] = pd.to_numeric(df['stock_quantity'], errors='coerce')

3. Remove products with negative price or stock_quantity
df = df[(df['price'] >= 0) & (df['stock_quantity'] >= 0)]

4. Confirm correct type for categorical ids (already mostly done)
for col in ['category_id', 'supplier_id', 'id']:
 if col in df.columns:
 df[col] = df[col].astype('object')

5. Trim whitespace everywhere for string columns
for col in df.select_dtypes(include='object').columns:
 df[col] = df[col].apply(lambda x: x.strip() if isinstance(x, str) else x)

6. Reset index after drops
df = df.reset_index(drop=True)

df.head()

	id	name	description	price	category_id	supplier_id	sku	stock_quantity
0	1	Enterprise-wide stabil	Trial eight magazine	605.06	8	8	99ef99ca-2d89-4bf1-8	to activate Win14

confirms the effectiveness of the clipping process for handling outliers while retaining all the data.

check outliers in boxplot

I'll plot a boxplot for the 'stock_quantity' column from the clipped_df DataFrame (where clipping was applied) to visually confirm that no outliers remain after capping.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 6))
clipped_df.boxplot(column=['stock_quantity'])
plt.title('Boxplot for Stock Quantity after Outlier Clipping')
plt.ylabel('stock_quantity')
plt.show()
```

Boxplot for Stock Quantity after Outlier Clipping

Ask agent to do anything

Chat Agent Submit

6.1.3 Customers Table

info

I'll display general information about the dataframe, including data types, number of non-null values, and memory usage using the .info() method.

```
df.info()
```

#	Column	Non-Null Count	Dtype
0	id	250 non-null	int64
1	first_name	250 non-null	object
2	last_name	250 non-null	object
3	email	250 non-null	object
4	phone	250 non-null	object
5	address	250 non-null	object
6	registration_date	250 non-null	object

dtypes: int64(1), object(6)
memory usage: 13.8+ KB

```
df.head()
```

will help summarize data such as counts, means, stddev, min, max, etc.

```
df.describe()
```

To provide descriptive statistics for object (categorical/text) columns, I'll use df.describe(include='object'). This will show metrics like unique counts and top frequency for those columns.

customer registration details for analysis [1] [2].

describe

I'll add a cell to generate descriptive statistics for the numerical columns using df.describe(). This will help summarize data such as counts, means, stddev, min, max, etc.

```
df.describe()
```

To provide descriptive statistics for object (categorical/text) columns, I'll use df.describe(include='object').

[7]: df.describe(include='object')

	first_name	last_name	email	phone	address	registration_da
count	250	250	250	250	250	250
unique	163	182	250	250	250	250
top	Michael	Smith	matthewsstephen@e+10000000001	+10000000001	8649 Gonzales Street	1/15/2025 18:12
freq	6	8	1	1	1	1

[6]: df.describe()

	id
count	250.00
mean	125.50
std	72.31
min	1.00
25%	63.25
50%	125.50
75%	187.75
max	250.00

[2]: import pandas as pd

[3]: df = pd.read_csv("customers.csv")
df

Home Insert Data Formulas Code Help

id | id

	id	first_name	last_name	email	phone	address	registration_da
	int	str	str	str	str	str	str
9	10	Vanessa	Garcia	markelliott@example.com	683-843-1063	30963 Lisa Dale Butk	1/16/2025 19:22
10	11	Tammy	Johnson	ronald02@example.com	+1-978-496-5613	698 Farmer Junction	1/20/2025 22:14
11	12	Kim	Lowery	harperjohn@example.com	+10000000012	634 Mason Harbor A	4/26/2025 20:20
12	13	Andrea	Miller	jameskathryn@example.com	+10000000013	723 Patty Locks Apt.	3/25/2025 19:49
13	14	Kathryn	Hart	jeremywallace@example.com	2989993268	058 Brianna Neck Wa	3/25/2025 16:37
14	15	Donna	Ward	charlene10@example.com	+10000000015	PSC 4359, Box 3660	2/16/2025 8:58
15	16	Matthew	Smith	tcurtis@example.net	(943)877-8436	53910 Matthew Fall A	1/4/2025 2:23
16	17	Angela	Roberts	charlesvelasquez@example.com	375-640-8582	042 Armstrong Freev	1/29/2025 9:14
17	18	Maria	Blanchard	hardy nicholas@example.com	(813)622-7540	PSC 7126, Box 3742	4/26/2025 18:25
18	19	Matthew	Foster	johnathanwood@example.com	4865074525	7805 Terrell Bypass F	3/9/2025 2:46
19	20	Michael	Perez	martinezjohn@example.com	+10000000020	669 Corey Fords Apt.	2/18/2025 12:37
20	21	Miranda	Thompson	jamesmcdonald@example.com	379.310.6395	175 Richard Knoll Ne	3/4/2025 23:46
21	22	Sean	Johnson	xhenderson@example.com	515.344.4489	43653 Miller Crossro	3/5/2025 0:50
22	23	Robin	Mayer	sherylgrant@example.com	+1-733-497-5974	63092 Julie Ford Wes	1/4/2025 11:43

you need any conversions or further exploration!

convert id to object

I'll convert the 'id' column from int64 to an object (string) type using .astype(str), so it will

registration_date to date

I'll convert the 'registration_date' column from an object (string) to datetime using pd.to_datetime, so it can be used in time-based

registration_date object
dtype: object

```
[9]: df['id'] = df['id'].astype(str)  
df.dtypes
```

id	object
first_name	object
last_name	object
email	object
phone	object
address	object
registration_date	object
dtype: object	
registration_date	object
dtype: object	

```
[10]: df['registration_date'] = pd.to_datetime(df['registration_date'])  
df.dtypes
```

id	object
first_name	object
last_name	object
email	object
phone	object
address	object
registration_date	datetime64[ns]
dtype: object	

need any further checks:

nulls

I'll show how many null (missing) values exist in each column using df.isnull().sum(). This will help identify if any columns need to be

```
[12]: 0
```

```
[13]: # Check the number of null values in each column  
df.isnull().sum()
```

```
[13]: id          0  
first_name    0  
last_name     0  
email         0  
phone         0  
address       0  
registration_date 0  
dtype: int64
```

duplicates

I'll check for duplicate rows in the dataframe using df.duplicated().sum(), so you know how many (if

```
[11]:
```

```
[ ]:
```

```
[12]: # Check the number of duplicate rows  
num_duplicates = df.duplicated().sum()  
num_duplicates
```

```
[12]: 0
```

DataFrame [2]. Let me know if you want more detailed analysis!

datatypes

I'll display the datatypes of all columns in the dataframe for you using the .dtypes attribute.

more data integrity checks!

unknown values

I'll count occurrences of common placeholder values representing 'unknown' or missing data (like 'unknown', 'n/a', 'none', etc.) in all object columns, as these sometimes indicate missing info even if not null or empty.

Ask agent to do anything

```
mito-data-copilot.ipynb  X  customers.ipynb  X  +  New MitoSheet
```

```
[7]:
```

```
[ ]:
```

```
[8]: df.dtypes
```

```
[8]: id          int64  
first_name    object  
last_name     object  
email         object  
phone         object  
address       object  
registration_date  object  
dtype: object
```

```
phone        0  
address      0  
dtype: int64
```

```
[15]: # Common "unknown" placeholder values to check
```

```
unknown_strings = [  
    'unknown', 'Unknown', 'UNKNOWN',  
    'n/a', 'N/A', 'na', 'NA', 'NaN',  
    'none', 'None', 'NONE',  
    '--', '--', '?', '???'
```

```
unknown_counts = df.select_dtypes(include='object').apply(  
    lambda col: col.str.strip().isin(unknown_strings).sum()  
)  
unknown_counts
```

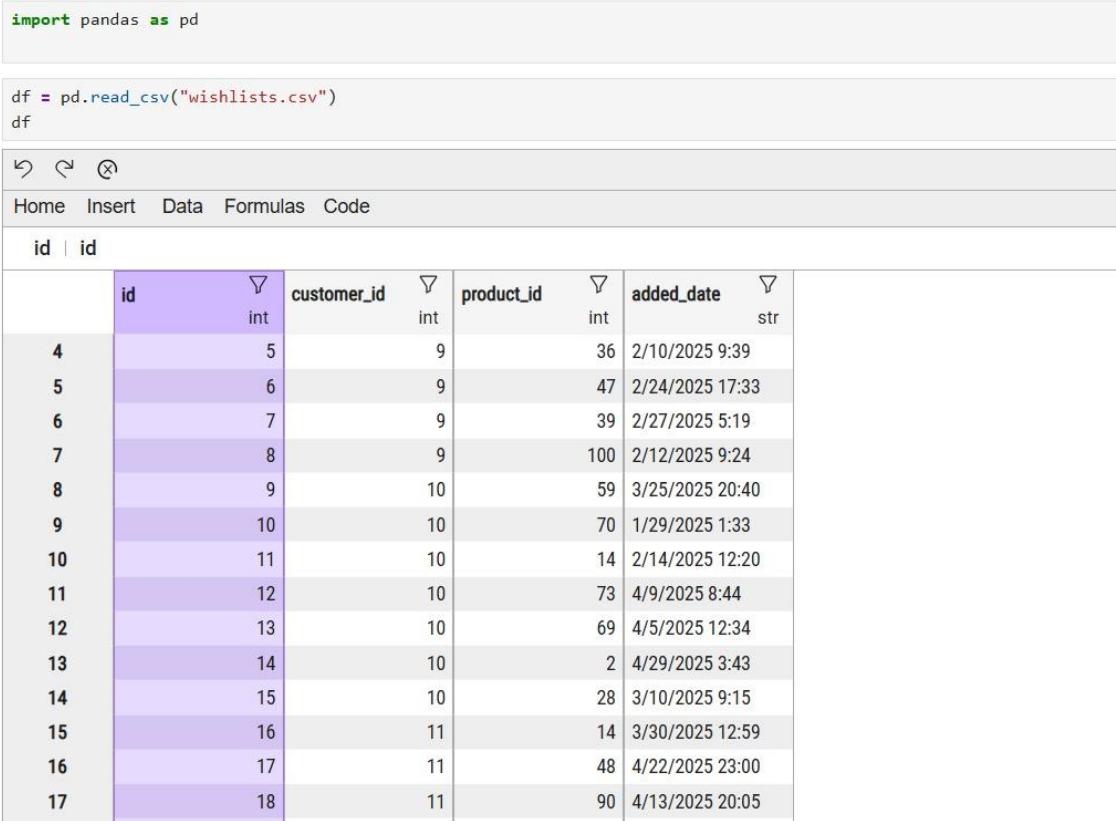
```
[15]: id          0  
first_name    0  
last_name     0  
email         0  
phone         0  
address       0  
dtype: int64
```

```
[16]: df
```

Security Checkup - Google

want to visualize nulls!	registration_date 0 dtype: int64
empty values	[14]: # Check for empty strings or whitespace-only entries in object columns empty_counts = df.select_dtypes(include='object').apply(lambda col: (col == '') (col.str.strip() == '')).sum() empty_counts
I'll display the number of empty strings or whitespace-only values in each object (string) column. While nulls are	[14]: id 0 first_name 0 last_name 0 email 0 phone 0 address 0 dtype: int64

6.1.4 Wish Lists Table

[]:	import pandas as pd																																																																																																																
[2]:	df = pd.read_csv("wishlist.csv") df																																																																																																																
																																																																																																																	
	Home Insert Data Formulas Code																																																																																																																
	<table border="1"> <thead> <tr> <th></th><th>id</th><th></th><th></th><th></th><th></th><th></th></tr> <tr> <th></th><th>id</th><th>int</th><th>customer_id</th><th>int</th><th>product_id</th><th>str</th></tr> </thead> <tbody> <tr><td>4</td><td></td><td>5</td><td></td><td>9</td><td>36</td><td>2/10/2025 9:39</td></tr> <tr><td>5</td><td></td><td>6</td><td></td><td>9</td><td>47</td><td>2/24/2025 17:33</td></tr> <tr><td>6</td><td></td><td>7</td><td></td><td>9</td><td>39</td><td>2/27/2025 5:19</td></tr> <tr><td>7</td><td></td><td>8</td><td></td><td>9</td><td>100</td><td>2/12/2025 9:24</td></tr> <tr><td>8</td><td></td><td>9</td><td></td><td>10</td><td>59</td><td>3/25/2025 20:40</td></tr> <tr><td>9</td><td></td><td>10</td><td></td><td>10</td><td>70</td><td>1/29/2025 1:33</td></tr> <tr><td>10</td><td></td><td>11</td><td></td><td>10</td><td>14</td><td>2/14/2025 12:20</td></tr> <tr><td>11</td><td></td><td>12</td><td></td><td>10</td><td>73</td><td>4/9/2025 8:44</td></tr> <tr><td>12</td><td></td><td>13</td><td></td><td>10</td><td>69</td><td>4/5/2025 12:34</td></tr> <tr><td>13</td><td></td><td>14</td><td></td><td>10</td><td>2</td><td>4/29/2025 3:43</td></tr> <tr><td>14</td><td></td><td>15</td><td></td><td>10</td><td>28</td><td>3/10/2025 9:15</td></tr> <tr><td>15</td><td></td><td>16</td><td></td><td>11</td><td>14</td><td>3/30/2025 12:59</td></tr> <tr><td>16</td><td></td><td>17</td><td></td><td>11</td><td>48</td><td>4/22/2025 23:00</td></tr> <tr><td>17</td><td></td><td>18</td><td></td><td>11</td><td>90</td><td>4/13/2025 20:05</td></tr> </tbody> </table>		id							id	int	customer_id	int	product_id	str	4		5		9	36	2/10/2025 9:39	5		6		9	47	2/24/2025 17:33	6		7		9	39	2/27/2025 5:19	7		8		9	100	2/12/2025 9:24	8		9		10	59	3/25/2025 20:40	9		10		10	70	1/29/2025 1:33	10		11		10	14	2/14/2025 12:20	11		12		10	73	4/9/2025 8:44	12		13		10	69	4/5/2025 12:34	13		14		10	2	4/29/2025 3:43	14		15		10	28	3/10/2025 9:15	15		16		11	14	3/30/2025 12:59	16		17		11	48	4/22/2025 23:00	17		18		11	90	4/13/2025 20:05
	id																																																																																																																
	id	int	customer_id	int	product_id	str																																																																																																											
4		5		9	36	2/10/2025 9:39																																																																																																											
5		6		9	47	2/24/2025 17:33																																																																																																											
6		7		9	39	2/27/2025 5:19																																																																																																											
7		8		9	100	2/12/2025 9:24																																																																																																											
8		9		10	59	3/25/2025 20:40																																																																																																											
9		10		10	70	1/29/2025 1:33																																																																																																											
10		11		10	14	2/14/2025 12:20																																																																																																											
11		12		10	73	4/9/2025 8:44																																																																																																											
12		13		10	69	4/5/2025 12:34																																																																																																											
13		14		10	2	4/29/2025 3:43																																																																																																											
14		15		10	28	3/10/2025 9:15																																																																																																											
15		16		11	14	3/30/2025 12:59																																																																																																											
16		17		11	48	4/22/2025 23:00																																																																																																											
17		18		11	90	4/13/2025 20:05																																																																																																											

(object) types . If you need further data cleaning, analysis, or exploration, let me know!

duplicates

Let's check for any duplicated rows in  and display them for review. This step does not delete, only shows duplicates.

```
# Find duplicated rows
duplicated_rows = df[df.duplicated()]
duplicated_rows
```

. if you need further analysis or cleaning, feel free to ask!

null

Let's check for any null (missing) values in the dataframe to identify columns that may need further cleaning.

```
# Check for null values in
df.isnull().sum()
```

Since we have now checked for null values,

let's display the rows that

Ask agent to do anything

```
[8]: # Remove duplicates based on customer_id and product_id (keep first occurrence)
df = df.drop_duplicates(subset=['customer_id', 'product_id'], keep='first')
df.reset_index(drop=True, inplace=True)
df.head()
```

Home Insert Data Formulas Code

id | id

	id	customer_id	product_id	added_date
0	1	9	59	2025-04-03 15:31:00
1	2	9	93	2025-03-28 12:58:00
2	3	9	92	2025-04-02 12:56:00
3	4	9	71	2025-02-25 15:23:00
4	5	9	36	2025-02-10 09:39:00

```
[8]:
```

```
[ ]:
```

```
[9]: # Check for null values in each column
df.isnull().sum()
```

```
[9]: id      0
customer_id  0
product_id   0
added_date   0
dtype: int64
```

```
[10]: # Display all rows that have any null value (if any)
df[df.isnull().any(axis=1)]
```

Home Insert Data Formulas Code

id | id

	id	customer_id	product_id	added_date

data types

Let's inspect the data types of the columns in `df` using the `dtypes` attribute.

```
[3]: df.dtypes
```

[3]:	<code>df.dtypes</code>	
		<pre>id int64 customer_id int64 product_id int64 added_date object dtype: object</pre>

Now let's convert the 'added_date' column from object to datetime to ensure accurate date operations.

```
[4]: df['added_date'] = pd.to_datetime(df['added_date'])
df.dtypes
```

[4]:	<code>df['added_date'] = pd.to_datetime(df['added_date'])</code>	
	<code>df.dtypes</code>	
		<pre>id int64 customer_id int64 product_id int64 added_date datetime64[ns] dtype: object</pre>

```
[5]: df['id'] = df['id'].astype(str)
df['customer_id'] = df['customer_id'].astype(str)
df['product_id'] = df['product_id'].astype(str)
df.dtypes
```

[5]:	<code>df['id'] = df['id'].astype(str)</code>	
	<code>df['customer_id'] = df['customer_id'].astype(str)</code>	
	<code>df['product_id'] = df['product_id'].astype(str)</code>	
	<code>df.dtypes</code>	
		<pre>id object customer_id object product_id object added_date datetime64[ns] dtype: object</pre>

We have inspected the structure, including column types and non-null counts.

```
[14]: df.info()
```

[14]:	<code>df.info()</code>	
		<pre><class 'pandas.core.frame.DataFrame'> RangeIndex: 999 entries, 0 to 998 Data columns (total 4 columns): # Column Non-Null Count Dtype --- -- -- -- 0 id 999 non-null object 1 customer_id 999 non-null object 2 product_id 999 non-null object 3 added_date 999 non-null datetime64[ns] dtypes: datetime64[ns](1), object(3) memory usage: 31.3+ KB</pre>

We used the `.info()` method to get a summary of the dataframe structure, showing the number of non-null entries, data types, and memory

empty cells

Let's check for empty strings (which are not the same as NaN/null) in our dataframe, as these can sometimes represent missing data but are not detected by isnull(). We'll count the number of empty strings for each column.

```
# Count empty strings in each column
(df == '').sum()
```

Show summary statistics for each column
summary = df.describe(include='all')
unique_counts = df.nunique()
summary, unique_counts

```
(,
 id 999
 customer_id 101
 product_id 489
 added_date 998
 dtype: int64)
```

describ()

Let's use the `describe()` method to get summary statistics (such as count, unique, top, freq for objects and mean, std, min, max for numerics/timestamps) for each column in your dataframe. We'll use `include='all'` to Ask agent to do anything

```
df.describe(include='all')
```

	id	customer_id	product_id	added_date
count	999	999	999	999
unique	999	101	489	NaN
top	1	149	535	NaN
freq	1	20	7	NaN
mean	NaN	NaN	NaN	1,741,393,245,285
min	NaN	NaN	NaN	1,735,702,920,000
25%	NaN	NaN	NaN	1,738,869,450,000
50%	NaN	NaN	NaN	1,741,535,340,000
75%	NaN	NaN	NaN	1,743,993,210,000
max	NaN	NaN	NaN	1,746,653,820,000

trim

Let's remove any leading or trailing whitespace in the string columns ('id', 'customer_id', 'product_id') to ensure consistency and cleanliness of your data.

```
# Trim whitespace from str
df['id'] = df['id'].str.strip()
df['customer_id'] = df['customer_id'].str.strip()
df['product_id'] = df['product_id'].str.strip()
```

```
df.head()
```

	id	customer_id	product_id	added_date
0	1	9	59	2025-04-03 15:31:00
1	2	9	93	2025-03-28 12:58:00
2	3	9	92	2025-04-02 12:56:00
3	4	9	71	2025-02-25 15:23:00
4	5	9	36	2025-02-10 09:39:00

6.1.5 Shipping Table

The screenshot shows a Jupyter Notebook interface with two tabs: "mito-data-copilot.ipynb" and "Untitled.ipynb". The "Untitled.ipynb" tab is active, displaying code and its output.

Code and Output:

```
[3]:  
[4]: # Show basic info about the dataframe  
df.info()  
  
# Display the first few rows for a sample  
df.head()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 4069 entries, 0 to 4068  
Data columns (total 6 columns):  
 #   Column           Non-Null Count  Dtype     
---  --  
 0   id               4069 non-null    int64    
 1   order_id         4069 non-null    int64    
 2   shipping_date    4069 non-null    object    
 3   tracking_number  4069 non-null    object    
 4   carrier          4069 non-null    object    
 5   status            4069 non-null    object    
dtypes: int64(2), object(4)  
memory usage: 190.9+ KB
```

Left Panel (Info Cell):

Let's get information about the dataframe, including types, nulls, and sample rows for context.

Code Cell (Info):

```
# Show basic info about the dataframe  
df.info()  
  
# Display the first few rows for a sample  
df.head()
```

Text Cell (Summary):

Here's a markdown cell summarizing the structure and key columns in `df` based on

The screenshot shows a Jupyter Notebook interface with two tabs: "mito-data-copilot.ipynb" and "Untitled.ipynb". The "Untitled.ipynb" tab is active, displaying code and its output.

Code and Output:

```
[5]: # Show statistics for numerical columns  
print(df.describe())  
print("\n")  
# Show value counts for each object (categorical) column  
for col in df.select_dtypes('object').columns:  
    print(f"--- {col} ---")  
    print(df[col].value_counts().head(10)) # Show up to top 10  
    print()
```

Output:

	id	order_id
count	4069.000000	4069.000000
mean	2035.000000	5157.767265
std	1174.763451	2937.443661
min	1.000000	1.000000
25%	1018.000000	2613.000000
50%	2035.000000	5196.000000
75%	3052.000000	7688.000000
max	4069.000000	10200.000000

--- shipping_date ---

shipping_date
2/17/2025 18:55
2/28/2025 14:23
2/11/2025 5:14
3/12/2025 3:35
2/6/2025 0:06
2/25/2025 2:10
1/15/2025 0:10
1/21/2025 1:51
2/15/2025 16:54
1/27/2025 6:53

Name: count, dtype: int64

Left Panel (Summary Cell):

gives you an overview of what kind of data is in each column. Let me know if you want to analyze any aspect further or focus on a specific column or metric!

Text Cell (Description):

description.T

Text Cell (Statistics):

Let's display summary statistics and basic descriptive information about all columns in `df` to give an overview. For categorical columns, we'll also see unique value counts.

Code Cell (Description):

```
# Show statistics for numerical columns  
print(df.describe())
```

Ask agent to do anything

shipping dates are parsed, there are no duplicates, no suspicious or missing categorical values, and no outlier dates present. For good measure, let's trim whitespace from all string columns and standardize categorical values to remove any invisible formatting issues.

```
# Enhancement: strip whitespace
def strip_strings(s):
    if s.dtype == 'object':
        return s.str.strip()
    return s

df = df.apply(strip_string)
df.head()
```

As a final clearing

Ask agent to do anything

	id	order_id	shipping_date	tracking_number	carrier	status
0	1	1	4/6/2025 9:06	ab562f03-0539-4407	FedEx	shipped
1	2	6	1/18/2025 21:10	b2b706fd-4c02-4780	UPS	shipped
2	3	8	3/13/2025 4:09	351b3cc7-cb1d-4204	UPS	delivered
3	4	9	1/4/2025 14:50	edd1a093-5860-4ef5	DHL	delivered
4	5	11	3/17/2025 21:32	a7151ffa-d439-45e8	UPS	shipped
5	6	13	2/26/2025 1:29	37176ec6-31af-4139	DHL	shipped
6	7	17	1/25/2025 0:10	41170d94-b80b-44fc	UPS	delivered
7	8	19	4/18/2025 19:18	23c45950-4b02-413f	UPS	shipped
8	9	24	1/16/2025 15:34	b9b988b8-395d-432d	FedEx	shipped
9	10	25	2/21/2025 0:39	7236cb0a-1163-490a	DHL	shipped
10	11	27	4/23/2025 21:33	f90d79b1-a719-4f4c	FedEx	shipped
11	12	28	4/16/2025 6:02	1453d31a-0c4e-46d6	DHL	shipped
12	13	32	1/31/2025 11:22	40212906-8314-448c	DHL	shipped
13	22	29	2/21/2025 16:20	b2d9b515-ab0a-4776	UPS	delivered

DataFrame Summary

- **id**: int64 — Unique identifier for each row
- **order_id**: int64 — Identifier for an order
- **shipping_date**: object — String of the date and time of shipping (e.g., '4/6/2025 9:06')
- **tracking_number**: object — Unique string identifier (UUID-like)
- **carrier**: object — Name of shipping carrier (e.g., 'FedEx', 'UPS', 'DHL')
- **status**: object — Shipping status (e.g., 'shipped', 'delivered')

df.describe(include='all')						
	id	order_id	shipping_date	tracking_number	carrier	status
count	4,069	4,069	4,069	4,069	4,069	4,069
unique	4,069	4,069	NaN	4,069	3	2
top	1	1	NaN	ab562f03-0539-4407	UPS	shipped
freq	1	1	NaN	1	1,405	2,086
mean	NaN	NaN	1,741,304,559,680	NaN	NaN	NaN
min	NaN	NaN	1,735,777,260,000	NaN	NaN	NaN
25%	NaN	NaN	1,738,641,840,000	NaN	NaN	NaN
50%	NaN	NaN	1,741,249,140,000	NaN	NaN	NaN
75%	NaN	NaN	1,744,075,860,000	NaN	NaN	NaN
max	NaN	NaN	1,746,746,280,000	NaN	NaN	NaN

in one transposed, concise format. If you have any specific questions or would like to analyze a particular aspect of your dataset, let me know! 

data types

Let's add a cell that shows the data types for each column in the dataframe `df`. This is useful for quickly checking how pandas interprets your data.

```
# Show the data types of each column
df.dtypes
```

markdown summary data gives you an overview of what kind of data is in each column. Let me know if you want to analyze any aspect further or focus on a specific column or metric!

description.T

Let's display summary statistics and basic descriptive information about all columns in `df` to give an overview. For categorical columns, we'll also see unique value counts.

```
# Show statistics for numerical columns
print(df.describe())
```

[7]: # Show the data types of each column
df.dtypes

[7]:	id	int64
	order_id	int64
	shipping_date	object
	tracking_number	object
	carrier	object
	status	object
	dtype:	object

[8]: # Convert 'id' and 'order_id' columns to string
df['id'] = df['id'].astype(str)
df['order_id'] = df['order_id'].astype(str)
Optional: check the new dtypes
df.dtypes

[8]:	id	object
	order_id	object
	shipping_date	object
	tracking_number	object
	carrier	object
	status	object
	dtype:	object

--- tracking_number ---
tracking_number

ab562f03-0539-4407-8f67-21f8a9b405bb	1	
01aaab24-13f4-4d34-a133-88d60287eaa2	1	
aae3e22b-2f2f-47cc-b576-3abc9e76118b	1	
26bb3c46-0e4c-4a02-95d0-3ae927963df7	1	
deecc025-c673-482b-a1d3-b00f263786ef	1	
e7464bbf-2c3b-4267-9abc-72ba019a4cc0	1	
2df90561-1630-44c0-92da-872833afc6fd	1	
f8ff355a-89e3-415e-a0d3-cefff466dc3f	1	
4568f821-d081-4540-af1c-81277652c025	1	
c0462088-e481-45c3-984a-1b14fe1e7017	1	

Name: count, dtype: int64

--- carrier ---
carrier

UPS	1405	
FedEx	1340	
DHL	1324	

Name: count, dtype: int64

--- status ---
status

shipped	2086	
delivered	1983	

Name: count, dtype: int64

DataFrame Summary

manipulations. Let me know if you need more transformations or analysis!

status to shipping status

Let's rename the `status` column to `shipping_status` for clarity and future reference. This step will not affect data, only the column name.

```
# Rename the 'status' column
df = df.rename(columns={'status': 'shipping_status'})
df.head()
```

Ask agent to do anything

clearer and improves your dataset's semantics . If you'd like to update any other columns, adjust values, or further explore your data, let me know!

duplications

Let's check for duplicated rows in the dataframe. We'll show how many duplicate rows exist and, if there are any, display the first few for inspection.

```
# Check for duplicated rows
duplicates = df[df.duplicated()]
print(f"Number of duplicates: {len(duplicates)}")
```

empty . This confirms the dataset is free of row-level duplications and ready for further analysis or cleaning if needed.

nulls

Let's check for null (missing) values in the dataframe, both by column counts and as a percentage of total rows. This helps identify columns needing possible cleanup.

```
# Check for nulls by column
print("Null values per column:")
print(df.isnull().sum())
print("\nPercentage of nulls per column:")
print(df.isnull().mean() * 100)
```

```
carrier          object
status           object
dtype: object

[11]: # Rename the 'status' column to 'shipping_status'
df = df.rename(columns={'status': 'shipping_status'})
df.head()

id | id
---|---
0  | 1
1  | 2
2  | 3
3  | 4
4  | 5

id      str   order_id    str   shipping_date  date   tracking_number  str   carrier     str   shipping_status  str
0      1      1   2025-04-06 09:06:00 ab562f03-0539-4407 FedEx      shipped
1      2      6   2025-01-18 21:10:00 b2b706fd-4c02-4780 UPS       shipped
2      3      8   2025-03-13 04:09:00 351b3cc7-cb1d-4204 UPS       delivered
3      4      9   2025-01-04 14:50:00 edd1a093-5860-4ef5 DHL       delivered
4      5      11  2025-03-17 21:32:00 a715ffa-d439-45e8 UPS       shipped
```

```
[12]: # Check for duplicated rows in the dataframe
duplicates = df[df.duplicated()]
print(f"Number of duplicated rows: {len(duplicates)}")
duplicates.head()

id | id
---|---
0  | 1
1  | 2
2  | 3
3  | 4
4  | 5

id      str   order_id    str   shipping_date  date   tracking_number  str   carrier     str   shipping_status  str
0      1      1   2025-04-06 09:06:00 ab562f03-0539-4407 FedEx      shipped
1      2      6   2025-01-18 21:10:00 b2b706fd-4c02-4780 UPS       shipped
2      3      8   2025-03-13 04:09:00 351b3cc7-cb1d-4204 UPS       delivered
3      4      9   2025-01-04 14:50:00 edd1a093-5860-4ef5 DHL       delivered
4      5      11  2025-03-17 21:32:00 a715ffa-d439-45e8 UPS       shipped
```

No rows in dataframe.

```
[12]: [ ]:

[13]: # Check for nulls by column
print("Null values per column:")
print(df.isnull().sum())
print("\nPercentage of nulls per column:")
print(df.isnull().mean() * 100)

Null values per column:
id          0
order_id    0
shipping_date 0
tracking_number 0
carrier     0
shipping_status 0
dtype: int64

Percentage of nulls per column:
id          0.0
order_id    0.0
shipping_date 0.0
tracking_number 0.0
carrier     0.0
shipping_status 0.0
dtype: float64
```

convert id and order id to string

Let's convert the columns `id` and `order_id` from int64 to string (object) type using the `astype(str)` method, so these identifiers are treated as strings.

```
# Convert 'id' and 'order_id'
df['id'] = df['id'].astype(str)
df['order_id'] = df['order_id'].astype(str)
# Optional: check the new
```

missing cells values

Let's display the actual rows that contain any missing (NaN) values in your dataframe. This helps you inspect problematic cells directly.

```
# Show all rows where at least one column has a missing value
df[df.isnull().any(axis=1)]
```

You've now identified every row in your dataframe that contains missing values. Ask agent to do anything

dtype: object

```
[9]: # Convert 'shipping_date' from string to datetime
df['shipping_date'] = pd.to_datetime(df['shipping_date'])
# Optional: check the new dtype
df.dtypes['shipping_date']
```

```
[9]: dtype('datetime64[ns]')
```

```
[10]: # Check data types after conversions
df.dtypes
```

[10]:	<code>id</code>	object
	<code>order_id</code>	object
	<code>shipping_date</code>	datetime64[ns]
	<code>tracking_number</code>	object
	<code>carrier</code>	object
	<code>status</code>	object
	dtype:	object

```
511450eb-0c9d-448d-981b-781b215a1014
'4aa125a2-7749-4fa5-9ce7-c11d95ced36a'
'86b6fb99-33d0-4e8e-809c-742663dbebec'
'30e26898-7421-4964-aaf5-7af2446325b9'
'7aed7ff0-89bb-4662-9b40-6d37350dec57'
```

```
--- carrier unique values (max 20 shown) ---
['FedEx' 'UPS' 'DHL']
```

```
--- shipping_status unique values (max 20 shown) ---
['shipped' 'delivered']
```

Interpreting Unique Values & Detecting Wrong Entries

- Review the unique values printed for each column.
- Look for possible issues such as:
 - Typos ("deliveerd" instead of "delivered"),
 - Numerals in the `carrier` column,
 - Abnormally formatted or short `tracking_number`s,
 - Odd dates in `shipping_date` (e.g., future years or missing months/days).
- If you spot any, share the odd entries (or column/values) and we can target them for fixing or filtering!

```
[16]: # Inspect possible wrong entries in 'carrier' and 'shipping_status'
```

data, such as dropping or filling nulls. Let me know if you want to investigate or address missing values further!

missing cells values

Let's display the actual rows that contain any missing (NaN) values in your dataframe. This helps you inspect problematic cells directly.

```
# Show all rows where at least one column has a missing value
df[df.isnull().any(axis=1)]
```

You've now identified every row in your dataframe that contains missing values.

Ask agent to do anything

Chat Agent Submit

or unusual values [?](#) . If you want to check for outliers in other features or perform more in-depth analysis, just let me know!

check outliers in boxplot

Let's plot a boxplot for `shipping_date` by converting it to numerical values (using its ordinal representation) to check for temporal outliers visually. We'll display the corresponding date values on the y-axis labels as well.

```
import matplotlib.dates as mdates
```

import agent to do anything

[15]:

```
# Show unique values for each column (up to 20, for inspection)
for col in df.columns:
    print(f"--- {col} unique values (max 20 shown) ---")
    print(df[col].unique()[:20])
    print()

--- id unique values (max 20 shown) ---
['1' '2' '3' '4' '5' '6' '7' '8' '9' '10' '11' '12' '13' '14' '15' '16'
 '17' '18' '19' '20']

--- order_id unique values (max 20 shown) ---
['1' '6' '8' '9' '11' '13' '17' '19' '24' '25' '27' '28' '32' '33' '34'
 '36' '38' '39' '44' '45']

--- shipping_date unique values (max 20 shown) ---
<DatetimeArray>
[2025-04-06 09:06:00, '2025-01-18 21:10:00', '2025-03-13 04:09:00',
 '2025-01-04 14:50:00', '2025-03-17 21:32:00', '2025-02-26 01:29:00',
 '2025-01-25 00:10:00', '2025-04-18 19:18:00', '2025-01-16 15:34:00',
 '2025-02-21 00:39:00', '2025-04-23 21:33:00', '2025-04-16 06:02:00',
 '2025-01-31 11:22:00', '2025-03-21 16:32:00', '2025-05-04 01:18:00',
 '2025-03-16 02:35:00', '2025-01-03 21:19:00', '2025-03-27 07:52:00',
 '2025-04-29 20:26:00', '2025-02-05 11:26:00']
Length: 20, dtype: datetime64[ns]

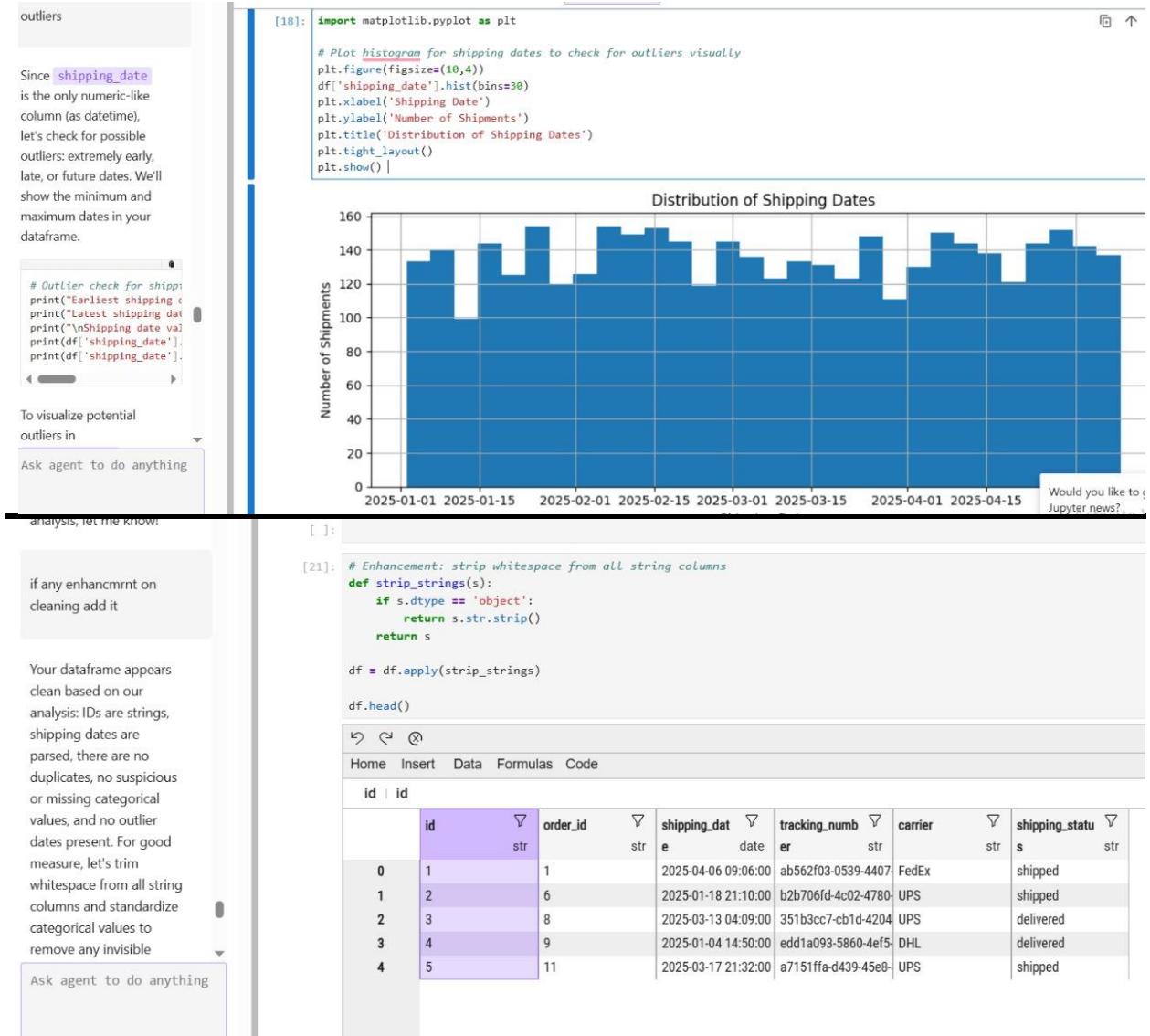
--- tracking_number unique values (max 20 shown) ---
['ab562f03-0539-4407-8f67-21f8a9b405bb'
 'b2b706fd-4c02-4780-84de-e2f606b88b16'
 '351b3cc7-cb1d-4204-a2cf-ea496a0a3636'
 'edd1a093-5860-4ef5-ab5a-7447ed86bc28'
 'a7151ffa-d439-45e8-8800-999d69ad8baa'
 '37176ec6-31af-4139-8367-1ef4bbdd1cd0'
 '41170d94-b80b-44fc-8818-cc43989c4d11'
 '23c45950-4b02-413f-b7a1-820006376312'
```

[19]:

```
import matplotlib.dates as mdates

# Convert dates to their numerical representation for boxplot
shipping_date_ordinal = df['shipping_date'].map(lambda x: x.toordinal())
plt.figure(figsize=(8, 5))
plt.boxplot(shipping_date_ordinal, vert=False, patch_artist=True)
plt.xlabel('Shipping Date (ordinal)')
plt.title('Boxplot of Shipping Dates')
# Add ticks for start/end dates
plt.yticks([1], ['Shipping Dates'])
plt.show()
```

Boxplot of Shipping Dates



6.1.6 Orders Table

Here's a short summary of your dataframe and its columns in plain English, for quick reference.

```
# Data summary in plain English
summary = ''\n- id: Unique
print(summary)
```

You now have a complete overview of your orders dataset, including column types, missing values, statistical summaries, and human-readable column descriptions. If you have a specific analysis or data transformation in mind, feel free to specify your next task!

shape of data

I'll display the shape of the dataframe to show you the number of rows and columns in your dataset.

```
df.shape
```

You've now displayed the shape of your dataset.

Ask agent to do anything

df.info()
df.describe(include='all')

	id	customer_id	order_date	total_amount	status
count	10,200.00	10,200.00	10,200	10,200.00	10,200
unique	Nan	Nan	9,974	Nan	5
top	Nan	Nan	4/25/2025 12:55	Nan	pending
freq	Nan	Nan	3	Nan	2,091
mean	5,100.50	147.63	Nan	4,406.27	Nan
std	2,944.63	59.67	Nan	2,901.80	Nan
min	1.00	1.00	Nan	11.41	Nan
25%	2,550.75	97.00	Nan	2,080.81	Nan
50%	5,100.50	148.00	Nan	4,024.12	Nan

Activate Windows
Go to Settings to activate Windows... ▾

File Edit View Run Kernel Tabs Settings Help mito-data-copilot.ipynb Untitled.ipynb + New MitoSheet Notebook Python 3 (ipykernel) Home Insert Data Formulas Code Help Upgrade to Mito Pro

df.info()
df.describe(include='all')

	id	customer_id	order_date	total_amount	status
0	1	13	4/5/2025 9:06	521.85	shipped
1	2	15	3/5/2025 8:27	436.91	processing
2	3	9	4/21/2025 21:19	7,711.55	cancelled
3	4	19	4/15/2025 2:16	7,952.82	cancelled
4	5	31	2/11/2025 17:30	2,785.65	cancelled
5	6	8	1/17/2025 21:10	6,179.44	shipped
6	7	45	2/28/2025 18:02	1,918.38	cancelled
7	8	8	3/12/2025 4:09	2,140.72	delivered
8	9	45	1/3/2025 14:50	8,104.69	delivered
9	10	46	4/4/2025 7:15	5,014.92	pending

Activate Windows
Go to Settings to activate Windows... ▾

+ df.head(10) ▾

File Edit View Run Kernel Tabs Settings Help mito-data-copilot.ipynb Untitled.ipynb + New MitoSheet Notebook Python 3 (ipykernel) Home Insert Data Formulas Code Help Upgrade to Mito Pro

mito-data-copilot.ipynb **Untitled.ipynb**

File Edit View Run Kernel Tabs Settings Help

info

I'll display summary information about the dataframe, including column types and missing values, to help you quickly understand its structure.

```
df.info()
df.describe(include='all')
```

I'll show the first 10 rows of the dataframe so you can see some example data for each column.

```
df.head(10)
```

Ask agent to do anything

Activate Windows
Get the Settings for this notebook

id	id	customer_id	order_date	total_amount	status
0	1	13	4/5/2025 9:06	521.85	shipped
1	2	15	3/5/2025 8:27	436.91	processing
2	3	9	4/21/2025 21:19	7,711.55	cancelled
3	4	19	4/15/2025 2:16	7,952.82	cancelled
4	5	31	2/11/2025 17:30	2,785.65	cancelled
5	6	8	1/17/2025 21:10	6,179.44	shipped
6	7	45	2/28/2025 18:02	1,918.38	cancelled
7	8	8	3/12/2025 4:09	2,140.72	delivered
8	9	45	1/3/2025 14:50	8,104.69	delivered
9	10	46	4/4/2025 7:15	5,014.92	pending
10	11	1	3/16/2025 21:32	1,038.29	shipped
11	12	9	2/11/2025 21:16	3,967.71	cancelled
12	13	15	2/25/2025 1:29	7,488.18	shipped
13	14	50	4/18/2025 19:38	856.47	processing
14	15	33	1/26/2025 1:29	6,848.16	cancelled

File Edit View Run Kernel Tabs Settings Help

print(f"Total duplicate : {df.duplicated().sum()}"

There are no duplicate rows across all columns and all values in the 'id' column are unique, so your orders dataset does not have any duplicates to be concerned about.

nulls

```
df.isnull().sum()
```

```
[9]: id      0
      customer_id  0
      order_date   0
      total_amount  0
      status       0
      dtype: int64
```

```
[10]: # Display only columns with missing values
missing = df.isnull().sum()
missing_cols = missing[missing > 0]
missing_cols
```

```
[10]: Series([], dtype: int64)
```

File Edit View Run Kernel Tabs Settings Help

address any missing values, analyze where they appear, or move on to the next step.

missing values

I'll show you only the columns with missing values, along with their null counts, to help focus only on what may need attention.

```
# Display only columns with missing
missing = df.isnull().sum()
missing_cols = missing[missing > 0]
missing_cols
```

No columns in your dataset have missing values; you have

Ask agent to do anything

Boxplot of Total Order Amount

You've now displayed the shape of your dataset, which tells you how many rows and columns there are in your orders data. If you need further summary or analysis, just specify the next step!

check duplicates

I'll check for duplicate rows in the data frame, both across all columns and just by the 'id' column, to see if any order records are repeated.

```
# Check for any duplicates
df.duplicated().sum()
```

Ask agent to do anything

such orders, with IDs shown in your results and their total_amount values exceeding this upper limit. If you'd like more detailed information on these orders or want to address them, let me know.

handle outliers in most suitable way

A common and suitable way to handle outliers in transaction data is to cap (winsorize) them at the upper bound. This keeps all records but reduces the impact of extreme values. Let's cap 'total_amount' values.

```
# Cap 'total_amount' at the upper outlier threshold
upper_bound = 12578.385 # From previous calculation

df['total_amount_capped'] = df['total_amount'].clip(lower=lower_bound, upper=upper_bound)
df[['total_amount', 'total_amount_capped']].head(10)
```

Ask agent to do anything

The screenshot shows a Jupyter Notebook interface with two tabs: 'mito-data-copilot.ipynb' and 'Untitled.ipynb'. The 'Untitled.ipynb' tab is active, displaying Python code for generating a data summary and checking for duplicates. The code includes imports for pandas, defines a summary string, prints the summary, and then uses the 'duplicated()' method on the DataFrame 'df' to find any duplicate rows. It also checks for duplicate 'id' values and prints the total number of duplicate rows and 'id's. The output shows that there are no duplicate rows or 'id's.

The screenshot shows a Jupyter Notebook interface with two tabs: 'mito-data-copilot.ipynb' and 'Untitled.ipynb'. The 'Untitled.ipynb' tab is active, displaying Python code for capping 'total_amount' values at an upper bound of 12578.385. The code uses the 'clip()' method on the 'total_amount' column of the DataFrame 'df'. Below the code, a preview of the DataFrame is shown with columns 'total_amount' and 'total_amount_capped'. The preview shows 10 rows of data, with the 'total_amount_capped' column showing values that have been capped at the upper bound of 12578.385.

allowing you to retain all order records while significantly reducing the influence of unusually high totals on your analysis. Your cleaned data is now in the new column 'total_amount_capped' for further analysis or modeling.

check outliers again

I'll check for outliers in the new capped 'total_amount_capped' column, using the same IQR approach, to confirm that no outliers remain after the earlier capping step.

Ask agent to do anything

```
[14]: Q1_capped = df['total_amount_capped'].quantile(0.25)
Q3_capped = df['total_amount_capped'].quantile(0.75)
IQR_capped = Q3_capped - Q1_capped
lower_bound_capped = Q1_capped - 1.5 * IQR_capped
upper_bound_capped = Q3_capped + 1.5 * IQR_capped

outliers_capped = df[(df['total_amount_capped'] < lower_bound_capped) | (df['total_amount_capped'] > upper_bound_capped)]
print(f"Number of outlier orders after capping: {outliers_capped.shape[0]}")
outliers_capped[['id', 'total_amount_capped']].head()
```

Number of outlier orders after capping: 0

No rows in dataframe.

outliers

I'll use a boxplot to visualize outliers in the numerical column `total_amount`, which is typically where outliers are most relevant in order data.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,4))
sns.boxplot(x=df['total_amount'])
plt.title('Boxplot of Total Amount')
plt.xlabel('Total Amount')
plt.show()
```

Let's identify the actual rows that are likely outliers

Ask agent to do anything

```
[12]: Q1 = df['total_amount'].quantile(0.25)
Q3 = df['total_amount'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df[(df['total_amount'] < lower_bound) | (df['total_amount'] > upper_bound)]
print(f"Number of outlier orders: {outliers.shape[0]}")
outliers[['id', 'total_amount']].head()
```

Number of outlier orders: 80

	<code>id</code>	<code>total_amount</code>
197	198	12,956.53
341	342	12,805.37
389	390	15,450.34
534	535	13,322.29
679	680	12,811.05

comparison or a deeper dive into another column, just specify your next question!

dtypes

I'll display the data types for each column in your dataframe so you can see how pandas has interpreted your schema.

```
df.dtypes
```

You've now displayed the data types for all columns in your orders dataframe. Both numeric and categorical fields are correctly typed—the only

fall within the calculated IQR bounds. Your order data is now robust for further analysis or modeling without undue influence from extreme transaction values.

box plot to visual that

I'll create a new boxplot for the 'total_amount_capped' column. This visualization will confirm the absence of high-end outliers and show how the distribution changed after capping.

```
plt.figure(figsize=(8,4))
sns.boxplot(x=df['total_amount_capped'])
```

Ask agent to do anything

[16]: df.dtypes

```
id          int64
customer_id  int64
order_date   object
total_amount float64
status       object
total_amount_capped float64
dtype: object
```

[17]: df['id'] = df['id'].astype(str)
df['customer_id'] = df['customer_id'].astype(str)
df.dtypes

```
id          object
customer_id object
order_date   object
total_amount float64
status       object
total_amount_capped float64
dtype: object
```

[18]: df = df.rename(columns={'status': 'order_status'})
df.head()

Notebook ▾ Python

[14]:

[15]: plt.figure(figsize=(8,4))
sns.boxplot(x=df['total_amount_capped'])
plt.title('Boxplot of Total Order Amount (Capped)')
plt.xlabel('Total Amount (Capped)')
plt.show()

Boxplot of Total Order Amount (Capped)

File Edit View Run Kernel Tabs Settings Help

mito-data-copilot.ipynb Untitled.ipynb

numbers for analysis. If you have further data cleaning or need new features/columns, let me know your next step!

rename status to order status

I'll rename the 'status' column to 'order_status' so your column naming is clearer. This won't affect your data and will improve code readability and usability.

```
df = df.rename(columns={'status': 'order_status'})  
df.head()
```

The column 'status' has

date and time without manual string parsing. Your dataset is now fully prepped for time-based analysis! Let me know your next task or question.

remove total_amount_capped

I'll remove the 'total_amount_capped' column from your dataframe since it is no longer needed.

```
df = df.drop(columns=['total_amount_capped'])  
df.head()
```

Notebook Python Help Up

Home Insert Data Formulas Code

id | id

	id	customer_id	order_date	total_amount	order_status	total_amount_capped
0	1	13	4/5/2025 9:06	521.85	shipped	521.85
1	2	15	3/5/2025 8:27	436.91	processing	436.91
2	3	9	4/21/2025 21:19	7,711.55	cancelled	7,711.55
3	4	19	4/15/2025 2:16	7,952.82	cancelled	7,952.82
4	5	31	2/11/2025 17:30	2,785.65	cancelled	2,785.65

[18]: df = df.rename(columns={'status': 'order_status'})
df.head()

dtype: object

Home Insert Data Formulas Code

id | id

	id	customer_id	order_date	total_amount	order_status	total_amount_capped
0	1	13	4/5/2025 9:06	521.85	shipped	521.85
1	2	15	3/5/2025 8:27	436.91	processing	436.91
2	3	9	4/21/2025 21:19	7,711.55	cancelled	7,711.55
3	4	19	4/15/2025 2:16	7,952.82	cancelled	7,952.82
4	5	31	2/11/2025 17:30	2,785.65	cancelled	2,785.65

Handling, and so you can see that this change is all compatible with this change. Let me know if you want to update other column names or continue with further transformations!

order date to timedate

I'll convert the 'order_date' column from string/object to datetime format using pandas to_datetime. This will allow you to perform date-based filtering and time series analysis.

```
[18]: df['order_date'] = pd.to_datetime(df['order_date'])
df.dtypes
```

	id	customer_id	order_date	total_amount	order_status
0	1	13	2025-04-05 09:06:00	521.85	shipped
1	2	15	2025-03-05 08:27:00	436.91	processing
2	3	9	2025-04-21 21:19:00	7,711.55	cancelled

Your data now contains only the cleaned, necessary columns for further analysis or export. If you'd like to export the cleaned data, dive into analysis, or perform further processing, let me know your next step!

amount per status

I'll group your data by 'order_status' and calculate the total and average 'total_amount' for each status, so you can see sales performance by order state.

```
# Aggregate total and mean
grouped = df.groupby('order_status')[['total_amount']].agg(['sum', 'mean', 'count']).reset_index()
grouped.columns = ['order_status', 'total_amount_sum', 'total_amount_mean', 'order_count']
grouped
```

	order_status	total_amount	total_amount	order_count
0	cancelled	8,867,722.70	4,407.42	2,012
1	delivered	8,873,678.24	4,474.88	1,983
2	pending	9,308,025.58	4,451.47	2,091
3	processing	8,976,126.91	4,426.10	2,028
4	shipped	8,918,412.59	4,275.37	2,086

pending, etc.) account for the most revenue, providing at-a-glance insights into where your sales are concentrated. If you'd like additional visualizations (e.g., mean order value by status or a count plot), just let me know your next analysis step!

check each column if there is wrong value in any cell

I'll start by checking for any unexpected or invalid values in the 'id' and 'customer_id' columns. For identifiers, the most common issues are

Ask agent to do anything

```
[24]: # Check for Leading/trailing whitespace in ID columns
id_whitespace = df['id'].str.match(r'^\s|\s$').sum()
custid_whitespace = df['customer_id'].str.match(r'^\s|\s$').sum()

# Check for non-digit characters (IDs should be digit strings)
id_non_digit = (~df['id'].str.isdigit()).sum()
custid_non_digit = (~df['customer_id'].str.isdigit()).sum()

print(f"Leading/trailing whitespace in 'id': {id_whitespace}")
print(f"Non-digit strings in 'id': {id_non_digit}")
print(f"Leading/trailing whitespace in 'customer_id': {custid_whitespace}")
print(f"Non-digit strings in 'customer_id': {custid_non_digit}")

Leading/trailing whitespace in 'id': 0
Non-digit strings in 'id': 0
Leading/trailing whitespace in 'customer_id': 0
Non-digit strings in 'customer_id': 0

[25]: # Check total_amount for negative values or zeros
neg_total = (df['total_amount'] < 0).sum()
zero_total = (df['total_amount'] == 0).sum()

# Check order_date for implausible range (e.g., before 2020 or in the far future)
earliest = df['order_date'].min()
latest = df['order_date'].max()

print(f"Negative total_amount cells: {neg_total}")
print(f"Zero total_amount cells: {zero_total}")
print(f"Earliest order_date: {earliest}")
print(f"Latest order_date: {latest}")

Negative total_amount cells: 0
Zero total_amount cells: 0
Earliest order_date: 2025-01-01 00:03:00
```

cancelled, delivered, pending—all reasonable, correct values.

Your data is clean and ready for analysis or export!

add any suitable cleaning step you suggest

Your data is already very clean, but a best practice is to ensure category columns (like 'order_status') use pandas' Categorical dtype. This saves memory and prevents typos in downstream processing. I'll convert 'order_status' to the categorical type.

Ask agent to do anything

```
[25]: # Check total_amount for negative values or zeros
neg_total = (df['total_amount'] < 0).sum()
zero_total = (df['total_amount'] == 0).sum()

# Check order_date for implausible range (e.g., before 2020 or in the far future)
earliest = df['order_date'].min()
latest = df['order_date'].max()

print(f"Negative total_amount cells: {neg_total}")
print(f"Zero total_amount cells: {zero_total}")
print(f"Earliest order_date: {earliest}")
print(f"Latest order_date: {latest}")

Negative total_amount cells: 0
Zero total_amount cells: 0
Earliest order_date: 2025-01-01 00:03:00
Latest order_date: 2025-05-07 23:18:00

[26]: # List all unique order_status values to check for typos/unexpected values
unique_statuses = df['order_status'].unique()
unique_statuses

[26]: array(['shipped', 'processing', 'cancelled', 'delivered', 'pending'],
           dtype=object)

[27]: df['order_status'] = pd.Categorical(df['order_status'])
df.dtypes

[27]: id          object
customer_id    object
order_date     datetime64[ns]
total_amount   float64
order_status   category
dtype: object
```

6.1.7 Payment Table

The screenshot shows a Jupyter Notebook interface with two panes. The left pane is a chat window with a message from 'mito' about the shape of the DataFrame and its columns. The right pane shows code cells and their outputs.

Chat Window (Left):

- df.shape
- The shape of `df` gives you the number of rows and columns in the `payments.csv` dataset (rows, columns). If you need to further explore, clean, or visualize this data, please let me know your next question!
- types
- Ask agent to do anything
- Chat Agent Submit

Notebook (Right):

- [7]: `df.shape`
[7]: `(8188, 7)`
- [8]: `df.dtypes`
[8]:

<code>id</code>	<code>int64</code>
<code>order_id</code>	<code>int64</code>
<code>customer_id</code>	<code>int64</code>
<code>amount</code>	<code>float64</code>
<code>payment_date</code>	<code>object</code>
<code>payment_method</code>	<code>object</code>
<code>status</code>	<code>object</code>
<code>dtype</code>	<code>object</code>
- [9]: `df['customer_id'] = df['customer_id'].astype(str)`
[9]:

<code>id</code>	<code>int64</code>
<code>order_id</code>	<code>int64</code>
<code>customer_id</code>	<code>object</code>
<code>amount</code>	<code>float64</code>
<code>payment_date</code>	<code>object</code>
<code>payment_method</code>	<code>object</code>
<code>status</code>	<code>object</code>
<code>dtype</code>	<code>object</code>
- [22]:
[23]:
[24]:
Boxplot of Payment Amount (cleaned)

... more advanced analysis!

check for any wrong entered values

Let's systematically check for wrongly entered values in all columns in the cleaned (filtered_df) DataFrame, such as:

- Non-numeric entries in 'amount'
- Negative or zero values (if not)

Ask agent to do anything

```
[27]: # Non-numeric in amount (should be float only, so this checks if type casting is needed)
non_numeric_amount = filtered_df[~filtered_df['amount'].apply(lambda x: isinstance(x, (int, float)))]

# Amount <= 0 (assuming it should be positive payments)
negative_or_zero_amount = filtered_df[filtered_df['amount'] <= 0]

# Duplicated IDs (should be unique if 'id' is a unique row identifier)
dup_ids = filtered_df[filtered_df['id'].duplicated(keep=False)]

# Unique payment methods and statuses
unique_methods = filtered_df['payment_method'].unique()
unique_statuses = filtered_df['payment_status'].unique()

# Output summary
dict(
    non_numeric_amount_rows=non_numeric_amount.shape[0],
    negative_or_zero_amount_rows=negative_or_zero_amount.shape[0],
    duplicated_id_rows=dup_ids.shape[0],
    unique_payment_methods=unique_methods,
    unique_payment_statuses=unique_statuses
)

[27]: {'non_numeric_amount_rows': 0,
       'negative_or_zero_amount_rows': 0,
       'duplicated_id_rows': 0,
       'unique_payment_methods': array(['paypal', 'credit_card', 'bank_transfer'], dtype=object),
       'unique_payment_statuses': array(['completed'], dtype=object)}
```

Name: amount, dtype: float64

```
[19]: # 1.5 IQR method
top = df['amount'].quantile(0.75) + 1.5 * (df['amount'].quantile(0.75) - df['amount'].quantile(0.25))
bottom = df['amount'].quantile(0.25) - 1.5 * (df['amount'].quantile(0.75) - df['amount'].quantile(0.25))
outliers_count = df[(df['amount'] > top) | (df['amount'] < bottom)].shape[0]
outliers_count

[19]: 64

[20]: # Remove outliers based on 1.5*IQR rule
filtered_df = df[(df['amount'] <= top) & (df['amount'] >= bottom)]
filtered_df.shape

[20]: (8124, 7)
```

check outliers again

Let's check if there are any remaining outliers in 'amount' in the cleaned DataFrame ('filtered_df') using the same 1.5*IQR rule.

```
[21]: # Recalculate IQR bounds on filtered_df
top_new = filtered_df['amount'].quantile(0.75) + 1.5 * (filtered_df['amount'].quantile(0.75) - filtered_df['amount'].quantile(0.25))
bottom_new = filtered_df['amount'].quantile(0.25) - 1.5 * (filtered_df['amount'].quantile(0.75) - filtered_df['amount'].quantile(0.25))
outliers_count_new = filtered_df[(filtered_df['amount'] > top_new) | (filtered_df['amount'] < bottom_new)].shape[0]
outliers_count_new

[21]: 10

[22]: filtered_df['amount'] = filtered_df['amount'].clip(lower=bottom_new, upper=top_new)
filtered_df.describe()
```

Warning (1)

	amount	payment_date
count	8,124.00	8,124
mean	4,331.45	1,741,165,332,747

to further analysis or modeling. Let me know if you'd like to take another step!

wrong entered values

To start checking for wrong or inconsistent values, let's display the unique values in each categorical column such as 'payment_method' and 'payment_status'. This will

Ask agent to do anything

check duplicates

I'll check for duplicate rows in the DataFrame and display how many there are. This helps ensure data quality before further analysis.

```
[17]: duplicate_count = df.duplicated().sum()
duplicate_count
```

There are no duplicate rows in your DF

Ask agent to do anything

columns. If you'd like to check for other types of quality issues or have another request, let me know!

outliers

To visually and statistically check for outliers in the 'amount' column, I'll show a boxplot and also display some basic quantile values. This helps to spot

Ask agent to do anything

[16]: print('Unique payment methods:', df['payment_method'].unique())
print('Unique payment statuses:', df['payment_status'].unique())

Unique payment methods: ['paypal' 'credit_card' 'bank_transfer']
Unique payment statuses: ['completed']

[17]: df[df['amount'] <= 0]

[12]:

```
[13]: duplicate_count = df.duplicated().sum()
duplicate_count
```

[13]: 0

```
[14]: df.isnull().sum()
```

[14]:

	0
id	0
order_id	0
customer_id	0
amount	0
payment_date	0
payment_method	0
payment_status	0

[15]: df[df.isnull().any(axis=1)]

[18]: import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 1))
sns.boxplot(x=df['amount'])
plt.title('Boxplot of Payment Amounts')
plt.show()

Display quantiles
quantiles = df['amount'].quantile([0.01, 0.25, 0.5, 0.75, 0.99])
quantiles

Boxplot of Payment Amounts

[18]: 0.01 107.7409
0.25 2078.6750
0.50 4031.5400
0.75 6227.8775

converted to string/object type. All IDs in your dataset ('id', 'order_id', and 'customer_id') are now treated as categorical/text variables. Let me know the next operation you'd like to perform!

status to payment status

I'll rename the column

Ask agent to do anything

convert other columns, or want to analyze, clean, or visualize this data!

payment_date to datetime

I'll convert the 'payment_date' column to datetime type so it's easier to work with timestamps and perform date-based operations.

Ask agent to do anything

some summary statistics, which will help understand numerical columns such as 'amount'.

df.describe()

Let's also examine the first few rows of the dataset to get a sense of its structure and possible data issues.

df.head()

Ask agent to do anything

status

dtype: object

```
[12]: df = df.rename(columns={'status': 'payment_status'})
df.head()
```

	id	order_id	customer_id	amount	payment_dat	payment_meth	payment_statu
0	1	1	13	521.85	2025-04-05 10:06:00	paypal	completed
1	2	2	15	436.91	2025-03-05 09:27:00	credit_card	completed
2	3	6	8	6,179.44	2025-01-17 22:10:00	credit_card	completed
3	4	8	8	2,140.72	2025-03-12 05:09:00	credit_card	completed
4	5	45		8,104.69	2025-01-03 15:50:00	bank_transfer	completed

dtype: object

```
[10]: df['payment_date'] = pd.to_datetime(df['payment_date'])
df.dtypes
```

	id	order_id	customer_id	amount	payment_date	payment_method	status	dtype
0	int64	int64	object	float64	datetime64[ns]	object	object	object

```
[11]: df['id'] = df['id'].astype(str)
df['order_id'] = df['order_id'].astype(str)
df.dtypes
```

	id	order_id	customer_id	amount	payment_date	payment_method	status	dtype
0	object	object	object	float64	datetime64[ns]	object	object	object

```
[11]: df = df.rename(columns={'status': 'payment_status'}) •••
```

memory usage: 447.9+ KB

```
[5]: df.describe()
```

	id	order_id	customer_id	amount
count	8,188.00	8,188.00	8,188.00	8,188.00
mean	4,094.50	5,098.49	147.88	4,405.99
std	2,363.82	2,943.55	59.76	2,894.59
min	1.00	1.00	1.00	11.41
25%	2,047.75	2,558.50	97.00	2,078.68
50%	4,094.50	5,086.50	148.00	4,031.54
75%	6,141.25	7,646.75	199.00	6,277.08
max	8,188.00	10,200.00	250.00	16,910.19

Let's also examine the first few rows of the dataset to get a sense of its structure and possible data issues.

```
df.describe()
```

We've completed an initial inspection of the `payments.csv` dataset.

Ask agent to do anything

`df.head()`

id	order_id	customer_id	amount	payment_date	payment_method	status
0	1	13	521.85	4/5/2025 10:06	paypal	completed
1	2	15	436.91	3/5/2025 9:27	credit_card	completed
2	3	8	6,179.44	1/17/2025 22:10	credit_card	completed
3	4	8	2,140.72	3/12/2025 5:09	credit_card	completed
4	5	45	8,104.69	1/3/2025 15:50	bank_transfer	completed

read

Let's start by reading one of the datasets into a DataFrame. I'll read `'meta_stock_prices.csv'` as an example. If you want to read a different file next, let me know!

```
meta_stock_prices_df = pd.read_csv("meta_stock_prices.csv")
```

Ask agent to do anything

Chat Agent Submit

meta_stock_prices_df.head()

Agent execution stopped. You can continue the conversation or start a new one.

info

I'll display general info about the `df`

`df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8188 entries, 0 to 8187
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          8188 non-null    int64  
 1   order_id    8188 non-null    int64  
 2   customer_id 8188 non-null    int64  
 3   amount       8188 non-null    float64
 4   payment_date 8188 non-null    object 
 5   payment_method 8188 non-null    object 
 6   status       8188 non-null    object 
dtypes: float64(1), int64(3), object(3)
memory usage: 447.9+ KB
```

6.1.8 Review Table

The 'reviews.csv' file has been successfully loaded into the DataFrame `review_df` and its contents are now available for further analysis or inspection.

```
[3]: import pandas as pd
review_df = pd.read_csv('reviews.csv')
review_df
```

	<code>id</code>	<code>product_id</code>	<code>Customer_id</code>	<code>rating</code>	<code>comment</code>	<code>review_date</code>
0	1	1	19	3	Travel information le...	44:43.0
1	2	1	2	1	Whose happen recent...	51:21.0
2	3	1	20	5	Charge movie girl car...	01:41.0
3	4	1	2	4	Way our discover sig...	06:02.0
4	5	1	28	3	Health democratic tri...	58:45.0
5	6	1	1	5	Find cover garden ea...	17:14.0
6	7	1	23	2	Value woman billion ...	33:09.0


```
[4]: print("Column names:", review_df.columns)
print("\nDataFrame Info:")
review_df.info()
print("\nSample rows:")
review_df.head()
```

Column names: Index(['id', 'product_id', 'Customer_id', 'rating', 'comment', 'review_date'], dtype='object')

DataFrame Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14196 entries, 0 to 14195
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  --          --          --          --    
 0   id          14196 non-null   int64  
 1   product_id  14196 non-null   int64  
 2   Customer_id 14196 non-null   int64  
 3   rating       14196 non-null   int64  
 4   comment      14196 non-null   object 
 5   review_date  14196 non-null   object 
dtypes: int64(4), object(2)
memory usage: 665.6+ KB
```

Sample rows:

	<code>id</code>	<code>product_id</code>	<code>Customer_id</code>	<code>rating</code>	<code>comment</code>	<code>review_date</code>
0	1	1	19	3	Travel information le...	44:43.0
1	2	1	2	1	Whose happen recent...	51:21.0
2	3	1	20	5	Charge movie girl car...	01:41.0
3	4	1	2	4	Way our discover sig...	06:02.0
4	5	1	28	3	Health democratic tri...	58:45.0

```
[4]: 
[5]: review_df.columns = [col.strip() for col in review_df.columns]
review_df.head()
```

	id	product_id	Customer_id	rating	comment	review_date
0	1	1	19	3	Travel information le...	44:43.0
1	2	1	2	1	Whose happen recent...	51:21.0
2	3	1	20	5	Charge movie girl car...	01:41.0
3	4	1	2	4	Way our discover sig...	06:02.0
4	5	1	28	3	Health democratic tri...	58:45.0

```
[7]: # Drop duplicates
review_df = review_df.drop_duplicates()

# Convert 'review_date' to pandas Timedelta for standardization since these are times (format appears like 'MM:SS.s')
# If this is not meaningful, you can leave as string instead
review_df['review_time'] = pd.to_timedelta('00:' + review_df['review_date'])

# Optionally, drop the original column
review_df = review_df.drop(columns=['review_date'])

# Show cleaned data stats
print("Cleaned DataFrame shape:", review_df.shape)
print("Columns:", review_df.columns)
review_df.head()
```

Cleaned DataFrame shape: (14196, 6)
Columns: Index(['id', 'product_id', 'Customer_id', 'rating', 'comment', 'review_time'], dtype='object')

```
Cleaned DataFrame shape: (14196, 6)
Columns: Index(['id', 'product_id', 'Customer_id', 'rating', 'comment', 'review_time'], dtype='object')
```

	id	product_id	Customer_id	rating	comment	review_time
0	1	1	19	3	Travel information le...	0 days 00:44:43
1	2	1	2	1	Whose happen recent...	0 days 00:51:21
2	3	1	20	5	Charge movie girl car...	0 days 00:01:41
3	4	1	2	4	Way our discover sig...	0 days 00:06:02
4	5	1	28	3	Health democratic tri...	0 days 00:58:45

6.1.9 Suppliers Table

```

File Edit View Run Kernel Tabs Settings Help
called suppliers_df
and display the first few
rows to verify import.

suppliers_df = pd.read_csv('suppliers.csv')
suppliers_df.head()

The 'suppliers.csv' file has
been successfully loaded
into the DataFrame
suppliers_df. The first
few rows display sample
supplier IDs, names,
contact information, and
addresses for further
analysis or joining with
other datasets if needed.

Ask agent to do anything

mito-data-copilot.ipynb + 
File Edit View Run Kernel Tabs Settings Help
+ X C Code New Mitosheet Notebook Python 3 (ipykernel)
[7]: 
[8]: suppliers_df = pd.read_csv('suppliers.csv')
suppliers_df.head()

Home Insert Data Formulas Code
id | id
id | int | name | str | contact_person | str | email | str | phone | str | address | str
0 | 1 | Sanders Ltd | Jasmine Walsh | phelpsmark@example.com | 815-907-4514 | 6251 Scott Road New
1 | 2 | Morrison and Sons | Ryan Hall | julie44@example.com | (781)629-2294x7633 | 43339 Berry Creek Sc
2 | 3 | Freeman-Valenzuela | Stephanie Carter | jacksonanne@example.com | 001-621-791-1635x22 | USS Wood FPO AE 43
3 | 4 | Wright-Davis | Kevin Bennett | wallerkurt@example.com | +1-747-674-4399x954 | 060 Patricia Extension
4 | 5 | Osborn-Stewart | Carmen Caldwell | csantos@example.com | -11018 | 9300 Davis Spurs Au

mito-data-copilot.ipynb + 
File Edit View Run Kernel Tabs Settings Help
+ X C Code New Mitosheet Notebook Python 3 (ipykernel)
[8]: 
[9]: print("Column names:", suppliers_df.columns)
print("\nDataFrame Info:")
suppliers_df.info()
print("\nSample rows:")
suppliers_df.head()

Column names: Index(['id', 'name', 'contact_person', 'email', 'phone', 'address'], dtype='object')

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70 entries, 0 to 69
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          70 non-null    int64  
 1   name        70 non-null    object  
 2   contact_person 70 non-null  object  
 3   email       70 non-null    object  
 4   phone       70 non-null    object  
 5   address     70 non-null    object  
dtypes: int64(1), object(5)
memory usage: 3.4+ KB

mito-data-copilot.ipynb + 
File Edit View Run Kernel Tabs Settings Help
+ X C Code New Mitosheet Notebook Python 3 (ipykernel)
Sample rows:
Home Insert Data Formulas Code
id | id
id | int | name | str | contact_person | str | email | str | phone | str | address | str
0 | 1 | Sanders Ltd | Jasmine Walsh | phelpsmark@example.com | 815-907-4514 | 6251 Scott Road New
1 | 2 | Morrison and Sons | Ryan Hall | julie44@example.com | (781)629-2294x7633 | 43339 Berry Creek Sc
2 | 3 | Freeman-Valenzuela | Stephanie Carter | jacksonanne@example.com | 001-621-791-1635x22 | USS Wood FPO AE 43
3 | 4 | Wright-Davis | Kevin Bennett | wallerkurt@example.com | +1-747-674-4399x954 | 060 Patricia Extension
4 | 5 | Osborn-Stewart | Carmen Caldwell | csantos@example.com | -11018 | 9300 Davis Spurs Au

```

[9]:

```
[10]: suppliers_df.columns = [col.strip() for col in suppliers_df.columns]
suppliers_df = suppliers_df.drop_duplicates()

print("Missing values per column:\n", suppliers_df.isnull().sum())
suppliers_df.head()
```

Missing values per column:

	id	name	contact_person	email	phone	address
0	Sanders Ltd	Jasmine Walsh	phelpsmark@example.com	815-907-4514	6251 Scott Road New	
1	Morrison and Sons	Ryan Hall	julie44@example.com	(781)629-2294x7633	43339 Berry Creek Sc	
2	Freeman-Valenzuela	Stephanie Carter	jacksonanne@example.com	001-621-791-1635x22	USS Wood FPO AE 43	
3	Wright-Davis	Kevin Bennett	wallerkurt@example.com	+1-747-674-4399x954	060 Patricia Extension	

[10]:

```
[11]: # Review unique phone number formats to evaluate normalization
print("Unique phone number examples:", suppliers_df['phone'].unique()[:10])
suppliers_df.head()
```

Unique phone number examples: ['815-907-4514' '(781)629-2294x76331' '001-621-791-1635x225' '+1-747-674-4399x95476' '-11018' '776.236.5945' '668-221-4144x9803' '651.584.5443' '752-610-2847' '001-230-232-9302x04324']

	id	name	contact_person	email	phone	address
0	Sanders Ltd	Jasmine Walsh	phelpsmark@example.com	815-907-4514	6251 Scott Road New	
1	Morrison and Sons	Ryan Hall	julie44@example.com	(781)629-2294x7633	43339 Berry Creek Sc	
2	Freeman-Valenzuela	Stephanie Carter	jacksonanne@example.com	001-621-791-1635x22	USS Wood FPO AE 43	
3	Wright-Davis	Kevin Bennett	wallerkurt@example.com	+1-747-674-4399x954	060 Patricia Extension	
4	Osborn-Stewart	Carmen Caldwell	csantos@example.com	-11018	9300 Davis Spurs Au	

6.1.10 Inventory movement Table

The screenshot shows two tabs of a Jupyter Notebook interface titled 'mito-data-copilot.ipynb'. The top tab displays code and a data preview, while the bottom tab displays descriptive statistics and another data preview.

Top Notebook (Cell 10):

```
[10]: inv_mov_df = pd.read_csv('inventory_movement.csv')
inv_mov_df.head()
```

The data preview shows the following table:

	id	product_id	quantity	movement_type	movement_date
0	1	38	-3	sale	06:50.0
1	2	35	-5	sale	27:50.0
2	3	46	-3	sale	27:50.0
3	4	36	-1	sale	19:19.0
4	5	83	-5	sale	19:19.0

Bottom Notebook (Cell 11):

```
[11]: inv_mov_df.info()
inv_mov_df.describe(include='all')
```

The data preview shows the following table:

	id	product_id	quantity	movement_type	movement_date
count	30,668.00	30,668.00	30,668.00	30,668	30,668
unique	NaN	NaN	NaN	1	3,399
top	NaN	NaN	NaN	sale	20:13.0
freq	NaN	NaN	NaN	NaN	22

mito-data-copilot.ipynb

New Mitosheet

Notebook Python 3 (ipykernel)

Home Insert Data Formulas Code Help Upgrade to Mito Pro

id | id

	id	product_id	quantity	movement_typ	movement_dat
	float	float	float	str	str
count	30,668.00	30,668.00	30,668.00	30,668	30,668
unique	NaN	NaN	NaN	1	3,399
top	NaN	NaN	NaN	sale	20:13.0
freq	NaN	NaN	NaN	30,668	32
mean	15,334.50	344.99	-3.00	NaN	NaN
std	8,853.23	148.43	1.42	NaN	NaN
min	1.00	1.00	-5.00	NaN	NaN
25%	7,667.75	219.00	-4.00	NaN	NaN
50%	15,334.50	346.00	-3.00	NaN	NaN
75%	23,001.25	473.00	-2.00	NaN	NaN
max	30,668.00	600.00	-1.00	NaN	NaN

mito-data-copilot.ipynb

New Mitosheet

Notebook Python 3 (ipykernel)

```
[12]: import re

# Count missing values in movement_date
movement_date_nans = inv_mov_df['movement_date'].isna().sum()

# Check if values match mm:ss.0 pattern
pattern = re.compile(r'^\d{2}:\d{2}\.\d{3}$')
movement_nonstandard = inv_mov_df[~inv_mov_df['movement_date'].astype(str).apply(lambda x: bool(pattern.fullmatch(x)))]

print(f"Missing values in movement_date: {movement_date_nans}")
print(f"Rows with non-standard movement_date: {len(movement_nonstandard)}")
if len(movement_nonstandard) > 0:
    print(movement_nonstandard['movement_date'].unique()[:10])

Missing values in movement_date: 0
Rows with non-standard movement_date: 0
```

```
[13]: def mmss_to_seconds(val):
    if pd.isna(val):
        return None
    mm, ss = val[:-2].split(':')
    return int(mm) * 60 + int(ss)

inv_mov_df['movement_seconds'] = inv_mov_df['movement_date'].apply(mmss_to_seconds)
inv_mov_df[['movement_date', 'movement_seconds']].head()
```

The screenshot shows two data tables in the Mito interface:

Top Table:

	movement_date	movement_seconds
0	06:50.0	410
1	27:50.0	1,670
2	27:50.0	1,670
3	19:19.0	1,159
4	19:19.0	1,159

Bottom Table:

	id	product_id	quantity	movement_type	movement_date	movement_seconds
0	1	38	-3	sale	06:50.0	410
1	2	35	-5	sale	27:50.0	1,670
2	3	46	-3	sale	27:50.0	1,670
3	4	36	-1	sale	19:19.0	1,159
4	5	83	-5	sale	19:19.0	1,159

6.1.11 Customer Sessions Table

The screenshot shows a data table in the Mito interface, representing a DataFrame named `customer_sessions_df`:

Code Area:

```
customer_sessions_df = pd.read_csv('customer_sessions.csv')
customer_sessions_df
```

Table Preview:

	id	customer_id	session_start	session_end	ip_address
0	1	1	19:52.0	17:52.0	180.254.129.172
1	2	1	17:21.0	24:21.0	19.62.206.54
2	3	1	52:42.0	46:42.0	193.28.97.70
3	4	1	30:35.0	50:35.0	24.110.203.249
4	5	1	42:49.0	49:49.0	5.208.17.212
5	6	1	55:36.0	48:36.0	207.136.203.221
6	7	2	20:18.0	01:18.0	175.196.47.211
7	8	2	44:08.0	25:08.0	119.16.10.55
8	9	2	28:32.0	13:32.0	59.232.64.253
9	10	2	47:59.0	37:59.0	88.243.152.114
10	11	2	33:17.0	42:17.0	67.145.122.152
11	12	2	19:52.0	07:52.0	134.50.159.187
12	13	2	26:25.0	59:25.0	181.60.57.220

```

The customer session data is now cleaned: duplicates are removed. and time columns (session_start, session_end) are properly parsed into timedelta format, making them ready for analysis. Column names have been standardized and unneeded columns dropped. You can now use customer_sessions_df for reliable downstream analysis.

[2]:
[3]: # Check for missing values and duplicates
print('Missing values by column:')
print(customer_sessions_df.isnull().sum())

print('\nDuplicate rows:')
print(customer_sessions_df.duplicated().sum())

# Show random sample to inspect data types
customer_sessions_df.sample(5)

Missing values by column:
id          0
customer_id  0
session_start 0
session_end   0
ip_address    0
dtype: int64

Duplicate rows:
0

```



```

[2]:
[3]: # Remove the original time columns and rename the cleaned columns
customer_sessions_df = customer_sessions_df.drop(['session_start', 'session_end'], axis=1)
customer_sessions_df = customer_sessions_df.rename(columns={
    'session_start_clean': 'session_start',
    'session_end_clean': 'session_end'
})

# Preview the cleaned dataframe
customer_sessions_df.head()

```

id	customer_id	ip_address	session_start	session_end
0	1	180.254.129.172	0 days 00:19:52	0 days 00:17:52
1	2	19.62.206.54	0 days 00:17:21	0 days 00:24:21
2	3	193.28.97.70	0 days 00:52:42	0 days 00:46:21
3	4	24.110.203.249	0 days 00:30:35	0 days 00:50:35
4	5	5.208.17.212	0 days 00:42:49	0 days 00:49:49

6.1.12 Discounts Table

```

is now loaded into the DataFrame

discounts_df and inspected. We confirmed there are no missing or non-standard values in the time columns, and both 'start_date' and 'end_date' have been converted into total seconds (as 'start_seconds' and 'end_seconds') for analysis. Let me know if you want to analyze, visualize, or further process this discount data!

Ask agent to do anything

```



```

[2]: discounts_df = pd.read_csv('discounts.csv')
discounts_df.head()

```

id	code	percentage	start_date	end_date	is_active	product_id
0	F3178C34	44.16	36.44.0	36.44.0	0	N
1	F92CDCB	8.06	09.02.0	09.02.0	0	N
2	CD0B53F2	36.60	39.57.0	56.52.0	1	N
3	B15B73F9	12.38	26.42.0	35.43.0	1	88
4	00A8B505	22.99	50.56.0	47.43.0	1	N

File Edit View Run Kernel Tabs Settings Help

mito-data-copilot.ipynb [3]: discounts_ur.head(10)

```
def mms_to_seconds(val):
    if pd.isna(val):
        return None
    mm, ss = val[-2].split(':')
    return int(mm) * 60 + int(ss)

discounts_df['start_seconds'] = discounts_df['end_seconds'].apply(mms_to_seconds)
discounts_df[['start_date', 'end_date']] = discounts_df[['start_date', 'end_date']].apply(pd.to_datetime)
```

The `discounts.csv` file is now loaded into the DataFrame `discounts_df` and inspected. We confirmed there are no missing or non-standard values in the time columns, and both 'start_date' and 'end_date' are now datetime objects.

Ask agent to do anything

Chat Agent Submit

File Edit View Run Kernel Tabs Settings Help

mito-data-copilot.ipynb [4]: discounts_df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          60 non-null     int64  
 1   code         60 non-null     object  
 2   percentage   60 non-null     float64 
 3   start_date   60 non-null     object  
 4   end_date     60 non-null     object  
 5   is_active    60 non-null     int64  
 6   product_id   26 non-null     float64 
 7   category_id  24 non-null     float64 
dtypes: float64(3), int64(2), object(3)
memory usage: 3.9+ KB
```

The `discounts.csv` file is now loaded into the DataFrame `discounts_df` and inspected. We confirmed there are no missing or non-standard values in the time columns, and both 'start_date' and 'end_date' are now datetime objects.

File Edit View Run Kernel Tabs Settings Help

mito-data-copilot.ipynb [4]: discounts_df.describe(include='all')

	id	code	percentage	start_date	end_date	is_active	product_id
count	60.00	60	60.00	60	60	60.00	26.00
unique	Nan	60	Nan	60	59	Nan	Nan
top	Nan	F3178C34	Nan	36:44:0	47:43:0	Nan	Nan
freq	Nan	1	Nan	1	2	Nan	Nan
mean	30.50	Nan	28.90	Nan	Nan	0.45	338.46
std	17.46	Nan	13.29	Nan	Nan	0.50	184.32
min	1.00	Nan	6.43	Nan	Nan	0.00	9.00
25%	15.75	Nan	18.48	Nan	Nan	0.00	184.50
50%	30.50	Nan	30.16	Nan	Nan	0.00	386.00
75%	45.25	Nan	39.59	Nan	Nan	1.00	473.50
max	60.00	Nan	49.74	Nan	Nan	1.00	579.00

(11 rows, 8 cols)

The image displays three vertically stacked screenshots of a Jupyter Notebook interface, specifically for the file "mito-data-copilot.ipynb".

Screenshot 1: This screenshot shows the execution of cell [5]. The code prints unique values from the 'start_date' and 'end_date' columns of the 'discounts_df' DataFrame.

```
[5]: print("Unique values in start_date column:")
print(discounts_df['start_date'].unique()[:15])
print("\nUnique values in end_date column:")
print(discounts_df['end_date'].unique()[:15])

Unique values in start_date column:
[ '36:44.0' '09:02.0' '39:57.0' '26:42.0' '50:56.0' '53:23.0' '24:15.0'
 '03:05.0' '16:28.0' '51:55.0' '14:12.0' '18:24.0' '42:01.0' '13:30.0'
 '43:30.0']

Unique values in end_date column:
[ '36:44.0' '09:02.0' '56:52.0' '35:43.0' '47:43.0' '51:54.0' '51:39.0'
 '03:05.0' '44:53.0' '18:45.0' '14:12.0' '18:24.0' '42:01.0' '26:43.0'
 '13:12.0']
```

Screenshot 2: This screenshot shows the execution of cell [6]. The code performs several checks on the 'start_date' and 'end_date' columns, including counting missing values and checking if values match a mm:ss.0 pattern.

```
[6]: import re

# Count missing values
start_date_nans = discounts_df['start_date'].isna().sum()
end_date_nans = discounts_df['end_date'].isna().sum()

# Check if values match mm:ss.0 pattern
pattern = re.compile(r'^\d{2}:\d{2}\.\d$')
start_nonstandard = discounts_df[~discounts_df['start_date'].astype(str).apply(lambda x: bool(pattern.fullmatch(x)))]
end_nonstandard = discounts_df[~discounts_df['end_date'].astype(str).apply(lambda x: bool(pattern.fullmatch(x)))]

print(f"Missing values in start_date: {start_date_nans}")
print(f"Missing values in end_date: {end_date_nans}")
print(f"Rows with non-standard start_date: {len(start_nonstandard)}")
print(f"Rows with non-standard end_date: {len(end_nonstandard)}")

Missing values in start_date: 0
Missing values in end_date: 0
Rows with non-standard start_date: 0
Rows with non-standard end_date: 0
```

Screenshot 3: This screenshot shows the execution of cell [7] and includes a screenshot of the Mito sheet interface.

The code defines a function to convert time strings to seconds and applies it to the 'start_date' and 'end_date' columns of the 'discounts_df' DataFrame.

```
[7]: def mmss_to_seconds(val):
    if pd.isna(val):
        return None
    mm, ss = val[:-2].split(':')
    return int(mm) * 60 + int(ss)

discounts_df['start_seconds'] = discounts_df['start_date'].apply(mmss_to_seconds)
discounts_df['end_seconds'] = discounts_df['end_date'].apply(mmss_to_seconds)
discounts_df[['start_date', 'start_seconds', 'end_date', 'end_seconds']].head()
```

The Mito sheet interface shows the DataFrame with the added 'start_seconds' and 'end_seconds' columns.

	start_date	start_seconds	end_date	end_seconds
0	36:44.0	2,204	36:44.0	2,204
1	09:02.0	542	09:02.0	542
2	39:57.0	2,397	56:52.0	3,412
3	26:42.0	1,602	35:43.0	2,143
4	50:56.0	3,056	47:43.0	2,863

6.1.13 Returns Table

The screenshot shows three Jupyter Notebook tabs:

- mito-data-copilot.ipynb**:
 - Cell [14]: `returns_df = pd.read_csv('returns.csv')`
 - Text: "Let's start by loading the 'returns.csv' file to inspect its contents and structure. This will help decide what cleaning steps might be necessary."
 - Cell [15]: `returns_df.head()`
 - Output: A Mito sheet showing the first 5 rows of the 'returns_df' DataFrame. The columns are id, order_id, return_date, reason, and status. The data includes rows for defective items, wrong sizes, and changed minds.
- mito-data-copilot.ipynb**:
 - Cell [16]:

```
returns_df.info()
returns_df.head(10)
returns_df.describe(include='all')
# Check for missing values
print(returns_df.isna().sum())
# Check for duplicates
print(f"Duplicate rows: {returns_df.duplicated().sum()}")
# Show unique reason values for cleaning
print(returns_df.columns)
print(returns_df['reason'].unique())
```
 - Cell [17]:

```
# Show unique reason values for cleaning
print(returns_df.columns)
print(returns_df['reason'].unique())
```
 - Output: A detailed DataFrame info output showing column types (int64 for id, order_id, return_date, status; object for reason), non-null counts, and dtypes. It also lists unique values for each column.

```
[17]: # Remove trailing spaces from column names
def clean_column_names(df):
    df.columns = df.columns.str.strip()
    return df

returns_df = clean_column_names(returns_df)

def mmss_to_seconds(val):
    if pd.isna(val):
        return None
    mm, ss = val[:-2].split(":")
    return int(mm)*60 + int(ss)

returns_df['return_seconds'] = returns_df['return_date'].apply(mmss_to_seconds)
returns_df.head()
```

Home Insert Data Formulas Code Help Upgrade to Mito Pro

	id	order_id	return_date	reason	status	return_seconds
0	1	47	37:24.0	Wrong size	approved	2,244
1	2	224	09:13.0	Defective	approved	553
2	3	421	34:57.0	Not as described	rejected	2,097
3	4	544	51:11.0	Changed mind	rejected	3,071

6.1.14 Categories Table

This code reads the 'categories.csv' file into a pandas DataFrame named `categories_df`.

open categories.csv

```
[5]: categories_df = pd.read_csv('categories.csv')
[6]: categories_df
```

Home Insert Data Formulas Code Help Upgrade to Mito Pro

	id	name	description	parent_id
0	1	Part	Win color election he	NaN
1	2	Program	Remain great line doo	NaN
2	3	Large	Table people job civil	NaN
3	4	Hold	Yard call test investm	NaN
4	5	Speech	Forward upon provide	NaN
5	6	Born	Study them region loc	3.00
6	7	Recent	Day attorney total any	2.00
7	8	Window	Today whose improve	2.00
8	9	Half	Author drug air howe	4.00
9	10	Adult	Address our whateve	5.00
10	11	Table	Although him again n	NaN
11	12	Mission	Doctor hair create ag	NaN

Let's check the data types of each column in `categories_df` and display them. This will help us identify if any columns need conversion to appropriate types for cleaning.

```
# Display the data types of categories_df.dtypes
```

Agent execution stopped. You can continue the conversation or start a new one.

Ask agent to do anything

File Edit View Run Kernel Tabs Settings Help

done. Let's check for missing values per column to confirm the completeness of your cleaned dataframe.

```
# Display the count of missing values
def missing_summary(df):
    return df.isnull().sum()

missing_summary(categories_df)
```

The dataframe `categories_df` has been cleaned: whitespace was stripped from all columns and values, column types were aligned (notably `parent_id` to `Int64`), duplicates were removed, and missing values have been summarized. This ensures your data is ready for further analysis or processing [1][2][3][4].

Ask agent to do anything

Chat Agent Submit ↗

mito-data-copilot.ipynb x Untitled1.ipynb + New MitoSheet Notebook Python 3 (ipykernel) Code

28	29	Onto	Any environmental te	4.00
29	30	Almost	Soldier upon others r	8.00

(30 rows, 4 cols)

[2]:

```
categories_df.head(), categories_df.info()
```

[3]:

```
# Show the first few rows
display(categories_df.head())
# Show information about columns, types, and missing values
categories_df.info()
```

[3]:

```
# Display the data types of each column
categories_df.dtypes
```

[3]:

```
id      int64
name    object
description    object
parent_id   float64
dtype: object
```

mito-data-copilot.ipynb x Untitled.ipynb + New MitoSheet Notebook Python 3 (ipykernel) Code

[8]: # Remove Leading/trailing whitespace from column names
categories_df.columns = categories_df.columns.str.strip()

[9]:

```
for col in categories_df.select_dtypes(include='object').columns:
    categories_df[col] = categories_df[col].str.strip()
categories_df.head()
```

Home Insert Data Formulas Code

id | id

	id	name	description	parent_id
0	1	Part	Win color election he	NaN
1	2	Program	Remain great line doc	NaN
2	3	Large	Table people job civil	NaN
3	4	Hold	Yard call test investm	NaN
4	5	Speech	Forward upon provide	NaN

mito-data-copilot.ipynb x Untitled.ipynb + New MitoSheet Notebook Python 3 (ipykernel) Code

[10]: # Convert parent_id to Int64 (nullable integer) and fill Nans if desired
categories_df['parent_id'] = categories_df['parent_id'].astype('Int64')
categories_df.head()

Home Insert Data Formulas Code

id | id

	id	name	description	parent_id
0	1	Part	Win color election he	NaN
1	2	Program	Remain great line doc	NaN
2	3	Large	Table people job civil	NaN
3	4	Hold	Yard call test investm	NaN
4	5	Speech	Forward upon provide	NaN

The screenshot shows a Jupyter Notebook interface with two code cells and an AI-generated note.

```

n_duplicates = categories_df.duplicated().sum()
print(f"Number of duplicate rows: {n_duplicates}")
# Optionally display duplicate rows themselves if any
if n_duplicates > 0:
    display(categories_df[categories_df.duplicated()])

```

You have verified for duplicate rows in `categories_df`, and based on previous cleaning and checks, your dataframe contains only unique entries. If you need to delve deeper and check for duplicates on specific columns or composite keys, just let me know!

Number of duplicate rows: 0

6.2 Predictive Modeling

6.2.1 Prepare data to ML

```

SQLQuery1.sql - DESKTOP-164ITPQ\user (55)* | x
1 --Prepare Data For Machine Learning
2
3 SELECT
4     o.id AS order_id,
5     o.order_date,
6     o.customer_id,
7
8     -- ملخصات رئيسية من تفاصيل الطلب
9     ISNULL(sh.address, 'Unknown Address') AS address,
10    ISNULL(sh.carrier, 'Unknown Carrier') AS carrier,
11    ISNULL(pay.payment_method, 'Unknown Payment') AS payment_method,
12
13    -- ملخصات أساسية من تفاصيل الطلب
14    COUNT(DISTINCT od.product_id) AS num_products,
15    SUM(od.quantity) AS total_quantity,
16    AVG(od.unit_price) AS avg_unit_price,
17
18    -- متوسط تقييم المنتجات (مع معالجة NULL)
19    ISNULL(re.avg_rating, 0) AS avg_rating,
20
21    -- معرفة الهدف
22    ISNULL(o.total_amount, 0) AS sales_target,
23
24
25
26
27
28
29
30
31
32

```

100 %

Results Messages

Query executed successfully.

```

SQLQuery1.sql - DE...-164ITPQ\User (55)* - X
29   -- معايير SQL Server
30   YEAR(o.order_date) AS order_year,
31   MONTH(o.order_date) AS order_month,
32   DATEPART(WEEKDAY, o.order_date) AS order_day_of_week, -- 1 = السبت ... 7 = الأحد
33   DAY(o.order_date) AS order_day,
34   DATEPART(DAYOFYEAR, o.order_date) AS order_day_of_year,
35
36   -- معايير جدول العميل (تم حفظها من الأستعلامات القراءة)
37   ISNULL(cust_sessions.avg_session_duration, 0) AS avg_session_duration,
38   ISNULL(cust_sessions.num_sessions, 0) AS num_sessions,
39   ISNULL(cust_sessions.days_since_last_session, 0) AS days_since_last_session,
40   ISNULL(cust_sessions.total_session_duration, 0) AS total_session_duration,
41
42
43   -- معايير المنتج والفلترة (أ لأن معايير على مستوى الطلبات)
44   ISNULL(order_product_category_info.min_category_id_in_order, 0) AS min_category_id_in_order,
45   ISNULL(order_product_category_info.min_supplier_id_in_order, 0) AS min_supplier_id_in_order,
46   ISNULL(order_product_category_info.min_category_parent_id_in_order, 0) AS min_category_parent_id_in_order,
47   ISNULL(order_product_category_info.num_unique_categories_in_order, 0) AS num_unique_categories_in_order,
48   ISNULL(order_product_category_info.num_unique_suppliers_in_order, 0) AS num_unique_suppliers_in_order,
49
50
51   -- معايير الخصومات (أ لأن معايير التفاصيل الصناعية)
52   ISNULL(discount_info.has_active_discount, 0) AS has_active_discount,
53   ISNULL(discount_info.avg_discount_percentage, 0) AS avg_discount_percentage,
54
55
56   -- مجموعات على مستوى الطلبات (أ لأن معايير القراءة)
57   ISNULL(order_aggregations.avg_product_price_in_order, 0) AS avg_product_price_in_order,
58   ISNULL(order_aggregations.total_products_in_order, 0) AS total_products_in_order
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
100 %

```

```

SQLQuery1.sql - DE...-164ITPQ\User (55)* - X
58 FROM orders o
59 JOIN order_details od ON o.id = od.order_id
60 LEFT JOIN customers c ON o.customer_id = c.id
61 LEFT JOIN shipping sh ON o.id = sh.order_id
62 LEFT JOIN payments pay ON o.id = pay.order_id
63 LEFT JOIN returns r ON o.id = r.order_id
64 -- هنا، سنتم إدارتها في الاستعلام القرائي الجديد ونلبيها لـ JOINs
65 -- LEFT JOIN products p ON od.product_id = p.id
66 -- LEFT JOIN categories cat ON p.category_id = cat.id
67
68 -- معايير فرعية لتوسيع تقييم المنتجات في الطلبات
69 LEFT JOIN (
70     SELECT
71         od_re.order_id,
72         AVG(r_re.rating) AS avg_rating
73     FROM order_details od_re
74     JOIN reviews r_re ON od_re.product_id = r_re.product_id
75     GROUP BY od_re.order_id
76 ) r ON o.id = re.order_id
77
78 -- معايير لخواص جدول العميل (لتجميع على مستوى العميل قبل بث المهم)
79 LEFT JOIN (
80     SELECT
81         cs_sub.customer_id,
82         AVG(DATEDIFF(SECOND, cs_sub.session_start, cs_sub.session_end)) AS avg_session_duration,
83         COUNT(DISTINCT cs_sub.id) AS num_sessions,
84         MAX(DATEDIFF(DAY, cs_sub.session_end, GETDATE())) AS days_since_last_session,
85         SUM(DATEDIFF(SECOND, cs_sub.session_start, cs_sub.session_end)) AS total_session_duration
86     FROM customer_sessions cs_sub
87     GROUP BY cs_sub.customer_id
88 ) cust_sessions ON o.customer_id = cust_sessions.customer_id
89

```

```

SQLQuery1.sql - DE...-164ITPQ\User (55)* - X
89
90   -- معايير لخواص الخصومات
91   LEFT JOIN (
92       SELECT
93           od_disc.order_id,
94           MAX(CASE
95               WHEN d_disc.id IS NOT NULL AND d_disc.is_active = 1 AND o_disc.order_date BETWEEN d_disc.start_date AND d_disc.end_date
96                   THEN 1
97                   ELSE 0
98               END) AS has_active_discount,
99               -- متوسط نسبة الخصم لجميع الخصومات المرتبطة بمنتجات في الطلب
100              AVG(d_disc.percentage) AS avg_discount_percentage
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
100 %

```

```

SQLQuery1.sql - DE...164ITPQ\User (55)* ✘
113
114
115
116 -- استعلام فرعى جديد لخواص المنتج والفلنة على مستوى الطلب
117 LEFT JOIN (
118     SELECT
119         od_prod.order_id,
120         MIN(p_prod.category_id) AS min_category_id_in_order,
121         MIN(p_prod.supplier_id) AS min_supplier_id_in_order,
122         MIN(cat_prod.parent_id) AS min_category_parent_id_in_order,
123         COUNT(DISTINCT p_prod.category_id) AS num_unique_categories_in_order,
124         COUNT(DISTINCT p_prod.supplier_id) AS num_unique_suppliers_in_order
125     FROM
126     order_details od_prod
127     JOIN
128     products p_prod ON od_prod.product_id = p_prod.id
129     LEFT JOIN
130     categories cat_prod ON p_prod.category_id = cat_prod.id
131     GROUP BY
132         od_prod.order_id
133 ) order_product_category_info ON o.id = order_product_category_info.order_id
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
100 % ✘
-- استعلام فرعى للتجهيزات على مستوى تفاصيل الطلب
Results Messages
Query executed successfully. | DESKTOP-164ITPQ (16.0 RTM) | DESKTOP-164ITPQ\User (55) | ecommerce_db | 00:00:22 | 10,200

```

```

SQLQuery1.sql - DE...164ITPQ\User (55)* ✘
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
100 % ✘
-- استعلام فرعى للتجهيزات على مستوى تفاصيل الطلب
LEFT JOIN (
    SELECT
        od_agg.order_id,
        AVG(od_agg.unit_price) AS avg_product_price_in_order,
        SUM(od_agg.quantity) AS total_products_in_order
    FROM order_details od_agg
    GROUP BY od_agg.order_id
) order_aggregations ON o.id = order_aggregations.order_id
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
-- الأспектات المختلفة في الطلب
ISNULL(sh.carrier, 'Unknown Carrier'),
ISNULL(pay.payment_method, 'Unknown Payment'),
r.return_date,
ISNULL(re.avg_rating, 0),
ISNULL(o.total_amount, 0),
YEAR(o.order_date), MONTH(o.order_date), DATEPART(WEEKDAY, o.order_date), DAY(o.order_date), DATEPART(DAYOFYEAR, o.order_date),
ISNULL(cust_sessions.avg_session_duration, 0), ISNULL(cust_sessions.num_sessions, 0), ISNULL(cust_sessions.days_since_last_session, 0),
ISNULL(cust_sessions.total_session_duration, 0),
ISNULL(order_product_category_info.min_category_id_in_order, 0),
ISNULL(order_product_category_info.min_supplier_id_in_order, 0),
ISNULL(order_product_category_info.min_category_parent_id_in_order, 0),
ISNULL(order_product_category_info.num_unique_categories_in_order, 0),
ISNULL(order_product_category_info.num_unique_suppliers_in_order, 0),
ISNULL(discount_info.has_active_discount, 0),
ISNULL(discount_info.avg_discount_percentage, 0),
ISNULL(order_aggregations.avg_product_price_in_order, 0),
ISNULL(order_aggregations.total_products_in_order, 0);
Results Messages
Query executed successfully. | DESKTOP-164ITPQ (16.0 RTM) | DESKTOP-164ITPQ\User (55) | ecommerce_db | 00:00:22 | 10,200

```

Query results

SQLQuery1.sql - DE...-164ITPQ\User (55)*

100 %

Results Messages

	order_id	order_date	customer_id	address	carrier	payment_method	return_flag	num_products	total_quantity	avg_unit_price	avg_rating	sales_target	order_year	order_month
5...	5757	2025-04-29 08:05:45.000	113	NM	FedEx	paypal	0	4	9	597.75000	3	5282.82	2025	4
5...	5758	2025-01-14 23:18:19.000	140	PR	FedEx	credit_card	0	4	11	699.77500	3	6003.94	2025	1
5...	5759	2025-02-27 21:44:03.000	115	MD	Unknown Carrier	Unknown Payment	0	2	4	640.77500	3	2563.10	2025	2
5...	5760	2025-02-01 18:21:11.000	125	VI	Unknown Carrier	Unknown Payment	0	3	12	734.44000	3	8647.77	2025	2
5...	5761	2025-03-12 19:05:46.000	229	WI	FedEx	credit_card	0	5	16	378.98600	3	5067.33	2025	3
5...	5762	2025-02-15 19:29:53.000	79	PA	UPS	credit_card	0	5	11	431.66600	2	5271.12	2025	2
5...	5763	2025-01-24 22:07:53.000	109	IL	Unknown Carrier	Unknown Payment	0	5	10	335.71400	3	3956.35	2025	1
5...	5764	2025-05-05 05:59:09.000	200	SC	Unknown Carrier	Unknown Payment	0	1	2	173.06000	2	346.12	2025	5
5...	5765	2025-02-05 16:31:57.000	191	NM	Unknown Carrier	Unknown Payment	0	2	9	701.77500	3	6046.12	2025	2
5...	5766	2025-04-02 06:10:21.000	180	MN	UPS	paypal	0	5	13	312.73200	3	3017.68	2025	4
5...	5767	2025-05-02 03:50:50.000	124	PR	Unknown Carrier	credit_card	0	1	3	535.24000	3	1605.72	2025	5
5...	5768	2025-04-04 05:24:10.000	145	FL	FedEx	paypal	0	3	7	432.56333	3	2724.81	2025	4
5...	5769	2025-01-08 09:10:47.000	172	HI	Unknown Carrier	Unknown Payment	0	1	5	334.77000	2	1673.85	2025	1
5...	5770	2025-01-08 00:30:22.000	182	SD	DHL	bank_transfer	0	4	9	537.11750	2	6093.06	2025	1
5...	5771	2025-03-05 04:12:26.000	241	OH	DHL	paypal	0	5	9	413.92800	2	3001.52	2025	3
5...	5772	2025-05-02 15:34:54.000	142	NM	Unknown Carrier	bank_transfer	0	2	6	886.73500	2	5184.96	2025	5
5...	5773	2025-03-07 22:06:26.000	209	GU	FedEx	credit_card	1	1	3	981.10000	2	2943.30	2025	3
5...	5774	2025-01-13 03:16:17.000	180	MN	Unknown Carrier	Unknown Payment	0	4	9	707.43000	2	6429.90	2025	1
5...	5775	2025-04-09 07:03:53.000	77	GA	Unknown Carrier	Unknown Payment	0	2	6	953.85500	2	5671.11	2025	4
5...	5776	2025-03-20 17:41:46.000	51	AK	Unknown Carrier	credit_card	0	1	5	428.91000	3	2144.55	2025	3
5...	5777	2025-02-03 00:32:08.000	55	MT	Unknown Carrier	Unknown Payment	0	1	1	636.41000	2	636.41	2025	2
5...	5778	2025-01-01 15:24:35.000	187	AA	Unknown Carrier	paypal	0	3	8	515.57333	3	4020.10	2025	1
5...	5779	2025-02-15 23:55:47.000	160	OK	UPS	paypal	0	3	8	256.22666	2	2467.21	2025	2
5...	5780	2025-03-19 00:15:40.000	106	NJ	DHL	credit_card	0	3	10	837.79666	2	8173.70	2025	3

[Click here to see full query result](#)

6.3 Natural Language Queries

7 Conclusion and recommendations