WALMART

HTTPS://GITHUB.COM/NOORIFA/WALMART_PYTHON_SQL.GIT

project steps

Project Steps

1. Set Up the Environment

- **Tools Used**: Visual Studio Code (VS Code), Python, SQL (PostgreSQL)
- **Goal**: Create a structured workspace within VS Code and organize project folders for smooth development and data handling.

2. SET UP KAGGLE API

- **API Setup**: Obtain your Kaggle API token from Kaggle by navigating to your profile settings and downloading the JSON file.
- **Configure Kaggle**:
    - Place the downloaded kaggle.json file in your local .kaggle folder.
    - Use the command kaggle datasets download -d <dataset-path> to pull datasets directly into your project.

set Up Kaggle API

- Data Source: Use the Kaggle API to download the Walmart sales datasets from Kaggle.
- Dataset Link: Walmart Sales Dataset
- Storage: Save the data in the data/ folder for easy reference and access.

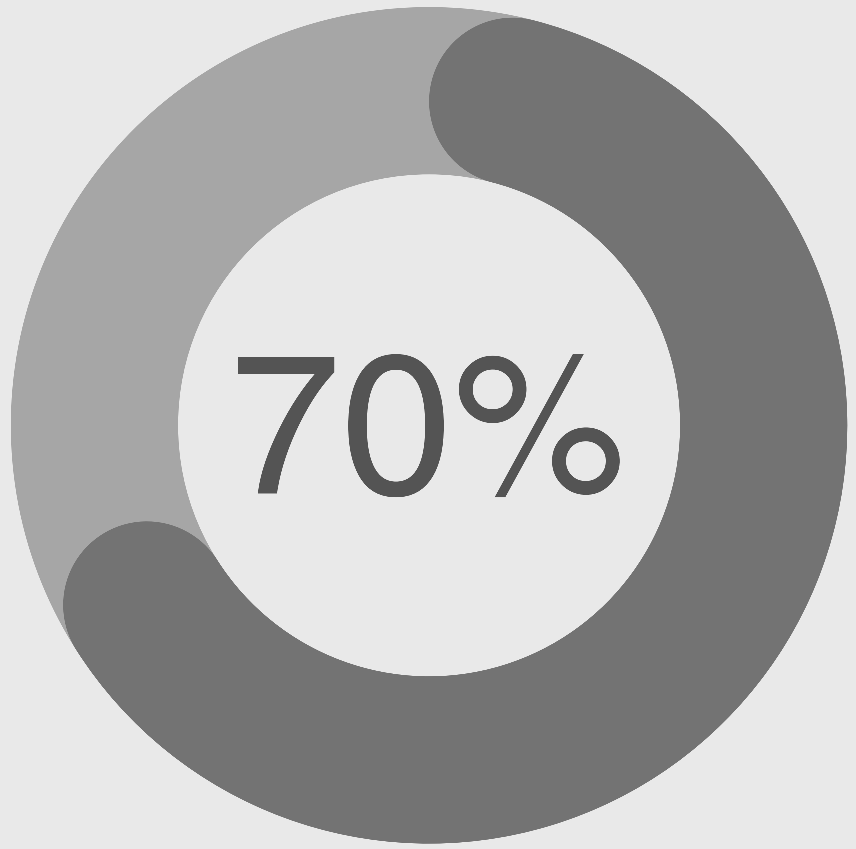
4. INSTALL REQUIRED LIBRARIES AND LOAD DATA

- **Libraries**: Install necessary Python libraries using:
- pip install pandas numpy sqlalchemy -connector-python psycopg2

- **Loading Data**: Read the data into a Pandas DataFrame for initial analysis and transformations.

5. EXPLORE THE DATA

- Goal: Conduct an initial data exploration to understand data distribution, check column names, types, and identify potential issues.
- Analysis: Use functions like .info(), .describe(), and .head() to get a quick overview of the data structure and statistics.



DATA CLEANING

- Remove Duplicates: Identify and remove duplicate entries to avoid skewed results.
- Handle Missing Values: Drop rows or columns with missing values if they are insignificant; fill values where essential.
- Fix Data Types: Ensure all columns have consistent data types (e.g., dates as datetime, prices as float).
- Currency Formatting: Use .replace() to handle and format currency values for analysis.
- Validation: Check for any remaining inconsistencies and verify the cleaned data.

7. FEATURE ENGINEERING

- Create New Columns: Calculate the Total Amount for each transaction by multiplying unit_price by quantity and adding this as a new column.
- Enhance Dataset: Adding this calculated field will streamline further SQL analysis and aggregation tasks.

.

Load Data into PostgreSQL

- **set Up Connections**: Connect to PostgreSQL using sqlalchemy and load the cleaned data into each database.
- **Table Creation**: Set up tables in PostgreSQL using Python SQLAlchemy to automate table creation and data insertion.
- **Verification**: Run initial SQL queries to confirm that the data has been loaded accurately.

9. SQL Analysis: Complex Queries and Business Problem Solving

- Business Problem-Solving: Write and execute complex SQL queries to answer critical business questions, such as:
  - Revenue trends across branches and categories.
  - Identifying best-selling product categories.
  - Sales performance by time, city, and payment method.
  - Analyzing peak sales periods and customer buying patterns.
  - Profit margin analysis by branch and category.
- Documentation: Keep clear notes of each query's objective, approach, and results.

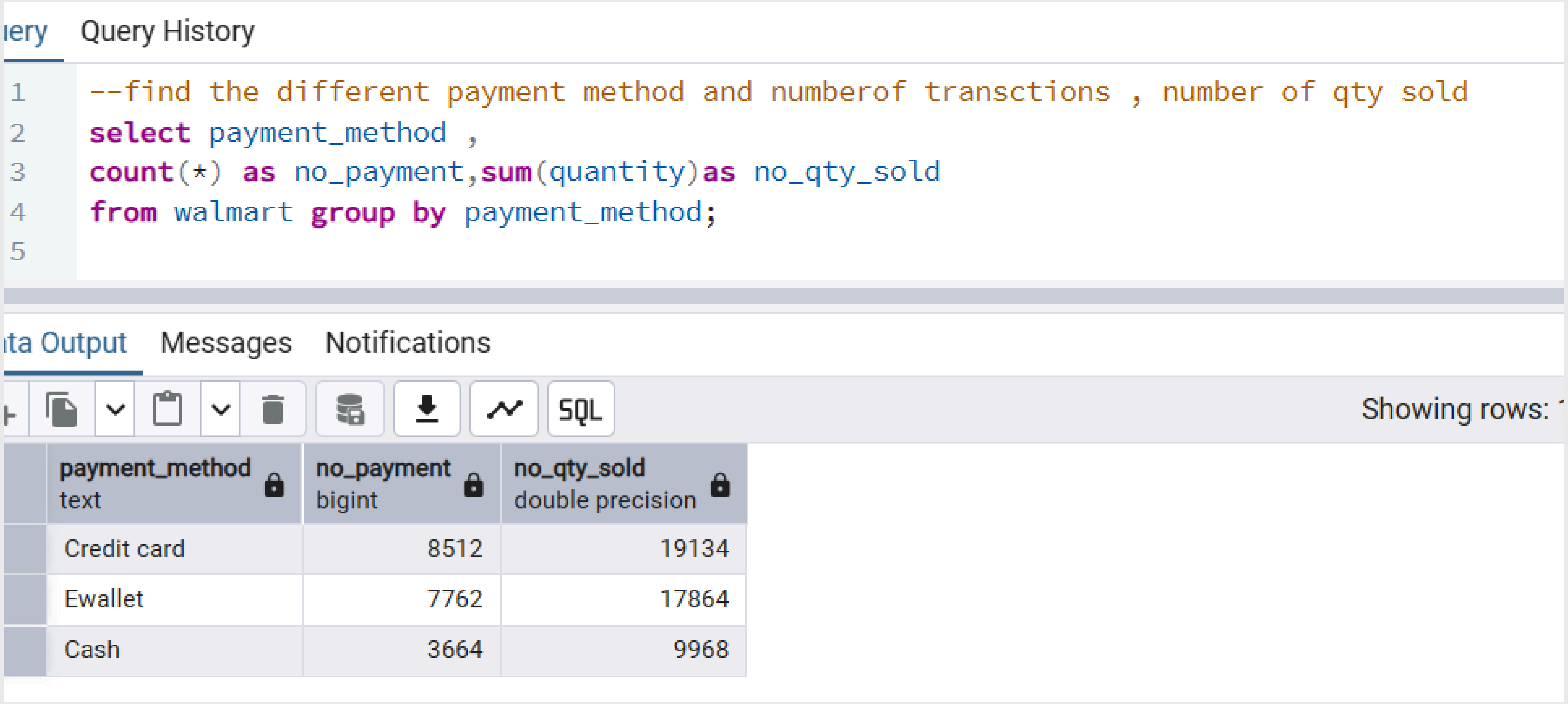
.

Documentation: Maintain well-structured documentation of the entire process in Markdown or a Jupyter Notebook.

- Project Publishing: Publish the completed project on GitHub or any other version control platform, including:
  - The README.md file (this document).
  - Jupyter Notebooks (if applicable).
  - SQL query scripts.
  - Data files (if possible) or steps to access them
-

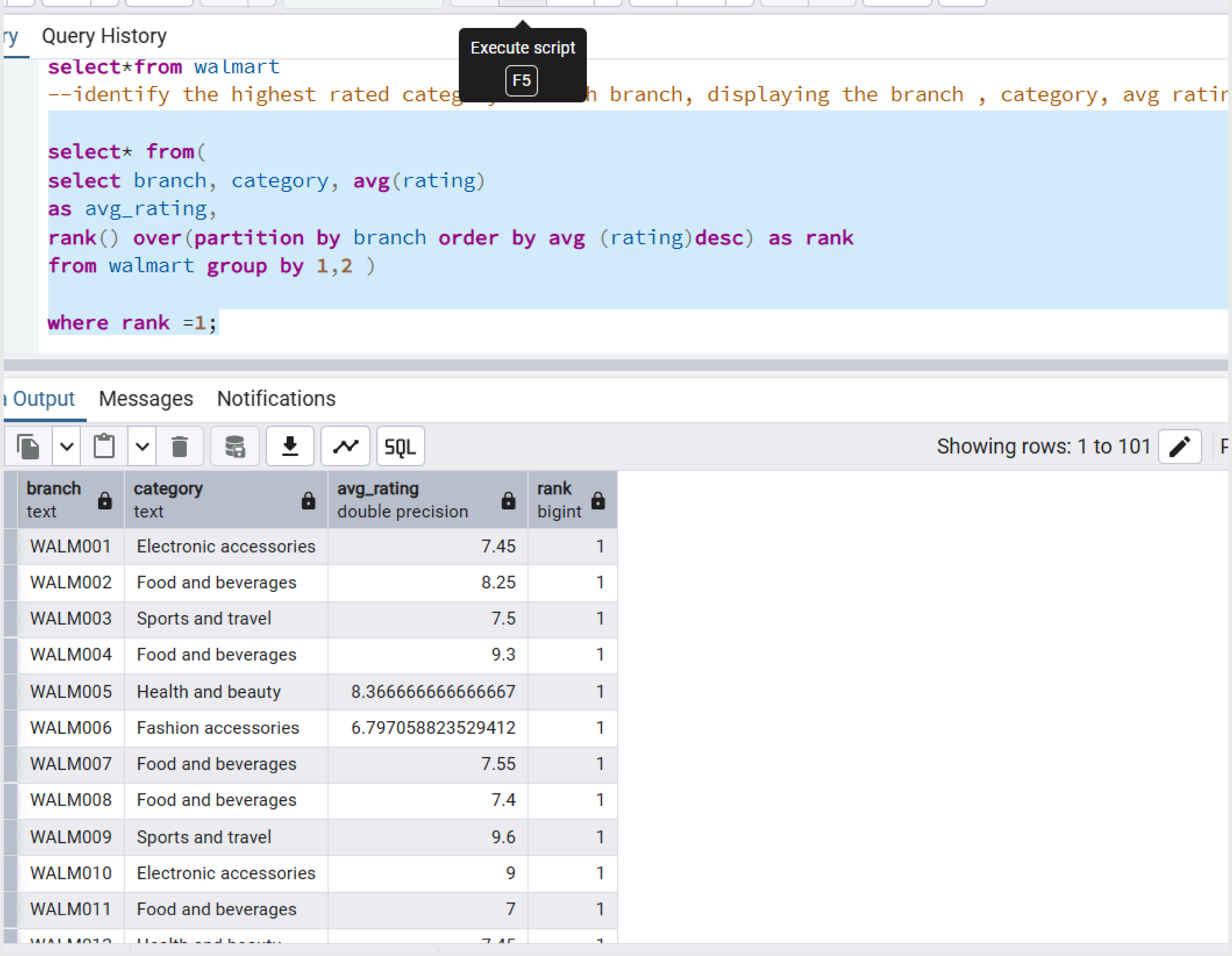
Documentation:

- **Documentation**: Maintain well-structured documentation of the entire process in Markdown or a Jupyter Notebook.
- **Project Publishing**: Publish the completed project on GitHub or any other version control platform, including:
    - The README.md file (this document).
    - Jupyter Notebooks (if applicable).
    - SQL query scripts.
    - Data files (if possible) or steps to access them



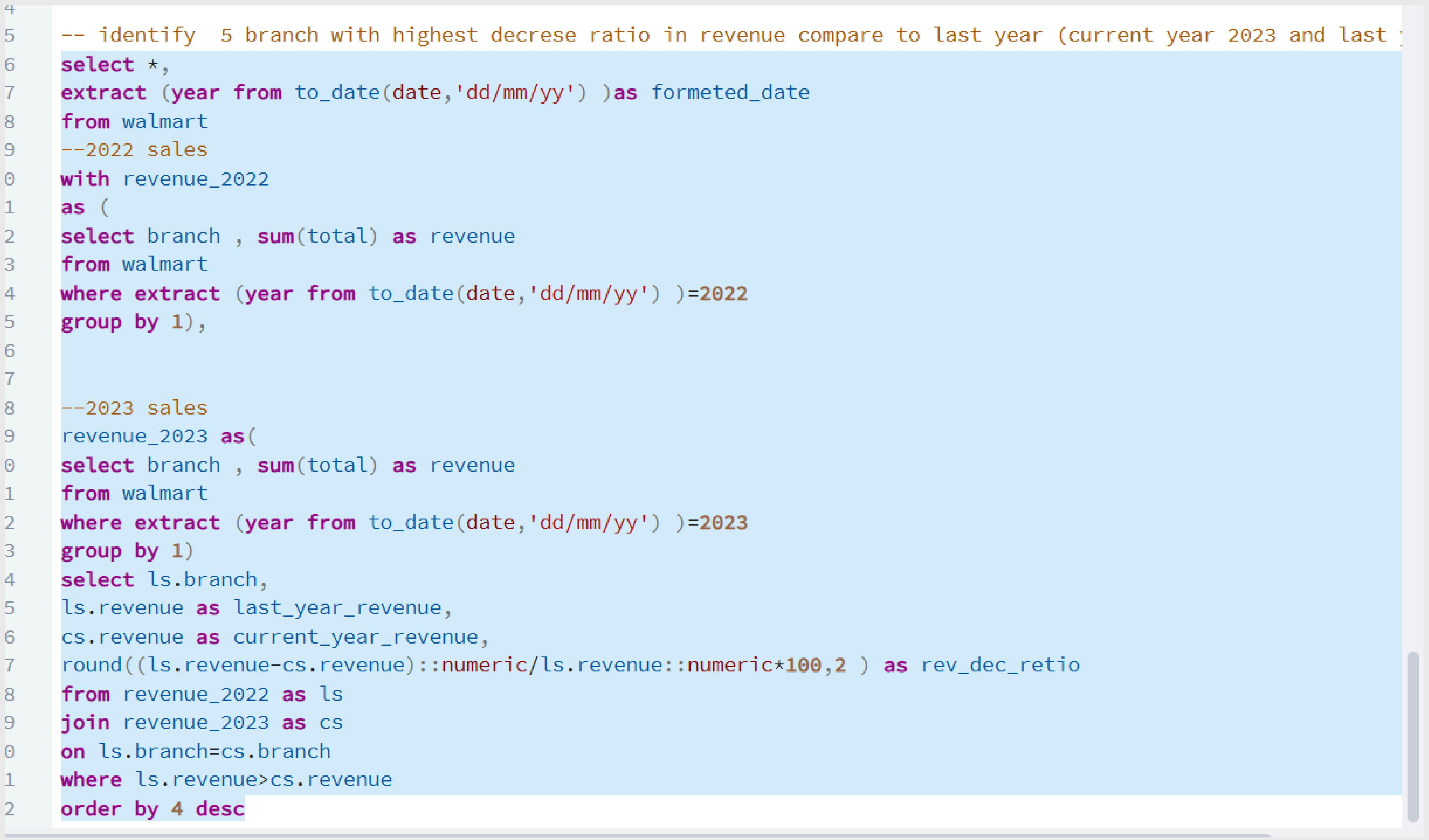
```sql
--find the different payment method and numberof transctions , number of qty sold
select payment_method ,
count(*) as no_payment,sum(quantity)as no_qty_sold
from walmart group by payment_method;

```

payment_method text	no_payment bigint	no_qty_sold double precision
Credit card	8512	19134
Ewallet	7762	17864
Cash	3664	9968

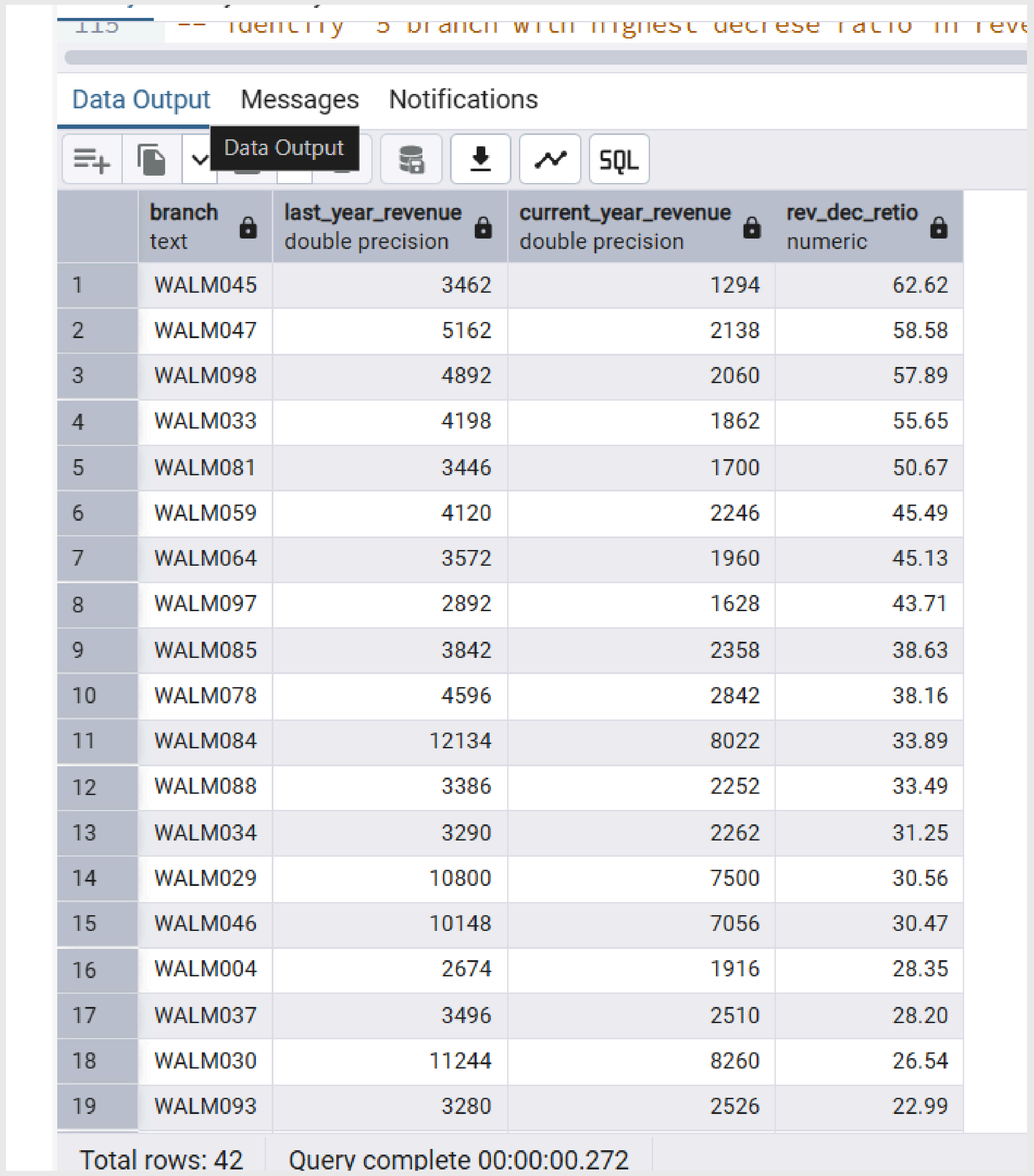


```
select*from walmart
--identify the highest rated categ                h branch, displaying the branch , category, avg ratir

select* from(
select branch, category, avg(rating)
as avg_rating,
rank() over(partition by branch order by avg (rating)desc) as rank
from walmart group by 1,2 )

where rank =1;
```

branch text	category text	avg_rating double precision	rank bigint
WALM001	Electronic accessories	7.45	1
WALM002	Food and beverages	8.25	1
WALM003	Sports and travel	7.5	1
WALM004	Food and beverages	9.3	1
WALM005	Health and beauty	8.366666666666667	1
WALM006	Fashion accessories	6.797058823529412	1
WALM007	Food and beverages	7.55	1
WALM008	Food and beverages	7.4	1
WALM009	Sports and travel	9.6	1
WALM010	Electronic accessories	9	1
WALM011	Food and beverages	7	1
WALM012	Health and beauty	7.45	1



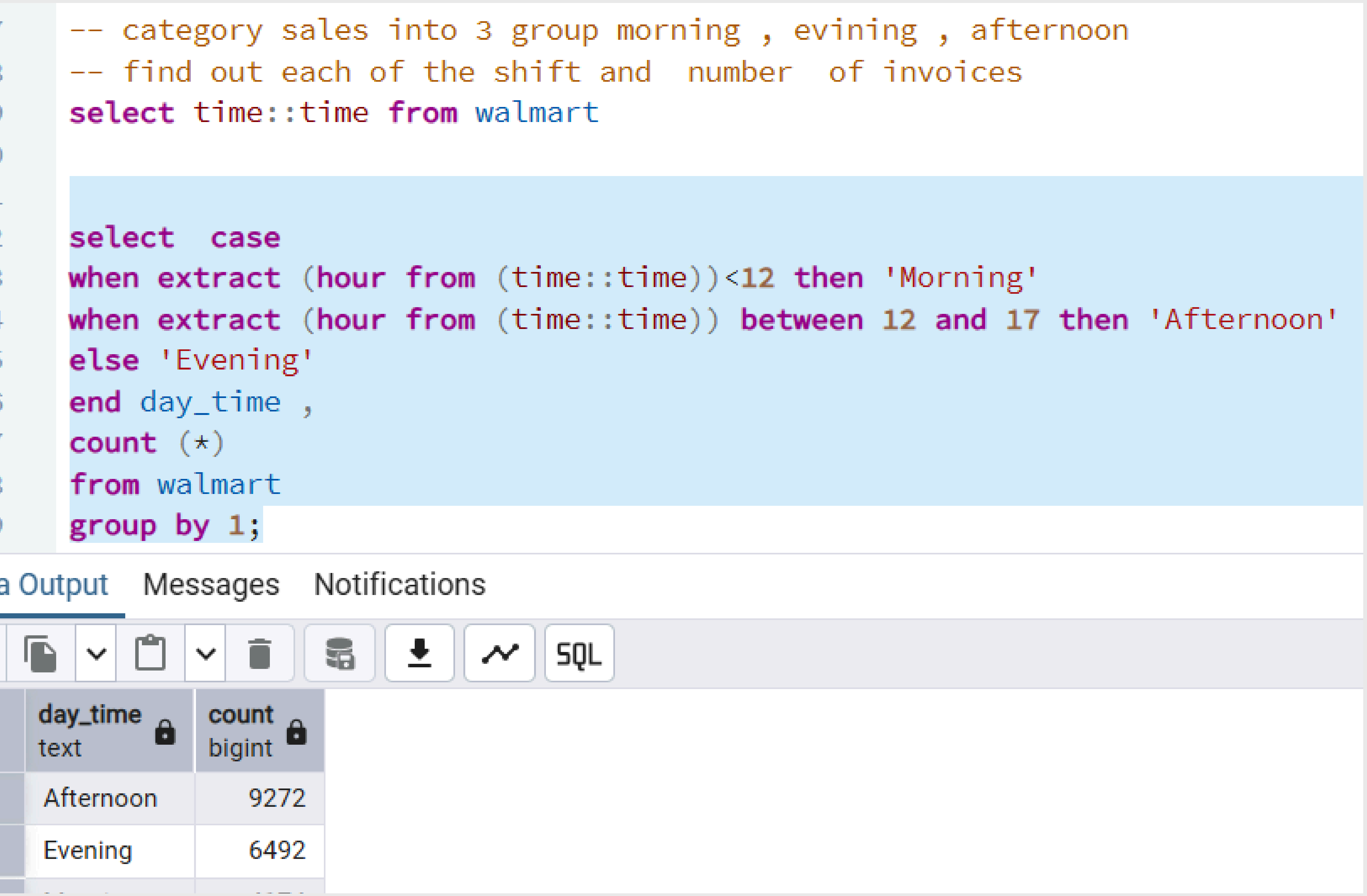
```
-- identify  5 branch with highest decrese ratio in revenue compare to last year (current year 2023 and last
select *,
extract (year from to_date(date,'dd/mm/yy') )as formeted_date
from walmart
--2022 sales
with revenue_2022
as (
select branch , sum(total) as revenue
from walmart
where extract (year from to_date(date,'dd/mm/yy') )=2022
group by 1),


--2023 sales
revenue_2023 as(
select branch , sum(total) as revenue
from walmart
where extract (year from to_date(date,'dd/mm/yy') )=2023
group by 1)
select ls.branch,
ls.revenue as last_year_revenue,
cs.revenue as current_year_revenue,
round((ls.revenue-cs.revenue)::numeric/ls.revenue::numeric*100,2 ) as rev_dec_retio
from revenue_2022 as ls
join revenue_2023 as cs
on ls.branch=cs.branch
where ls.revenue>cs.revenue
order by 4 desc
```



```
-- identify  5 branch with highest decrese ratio in reve
```

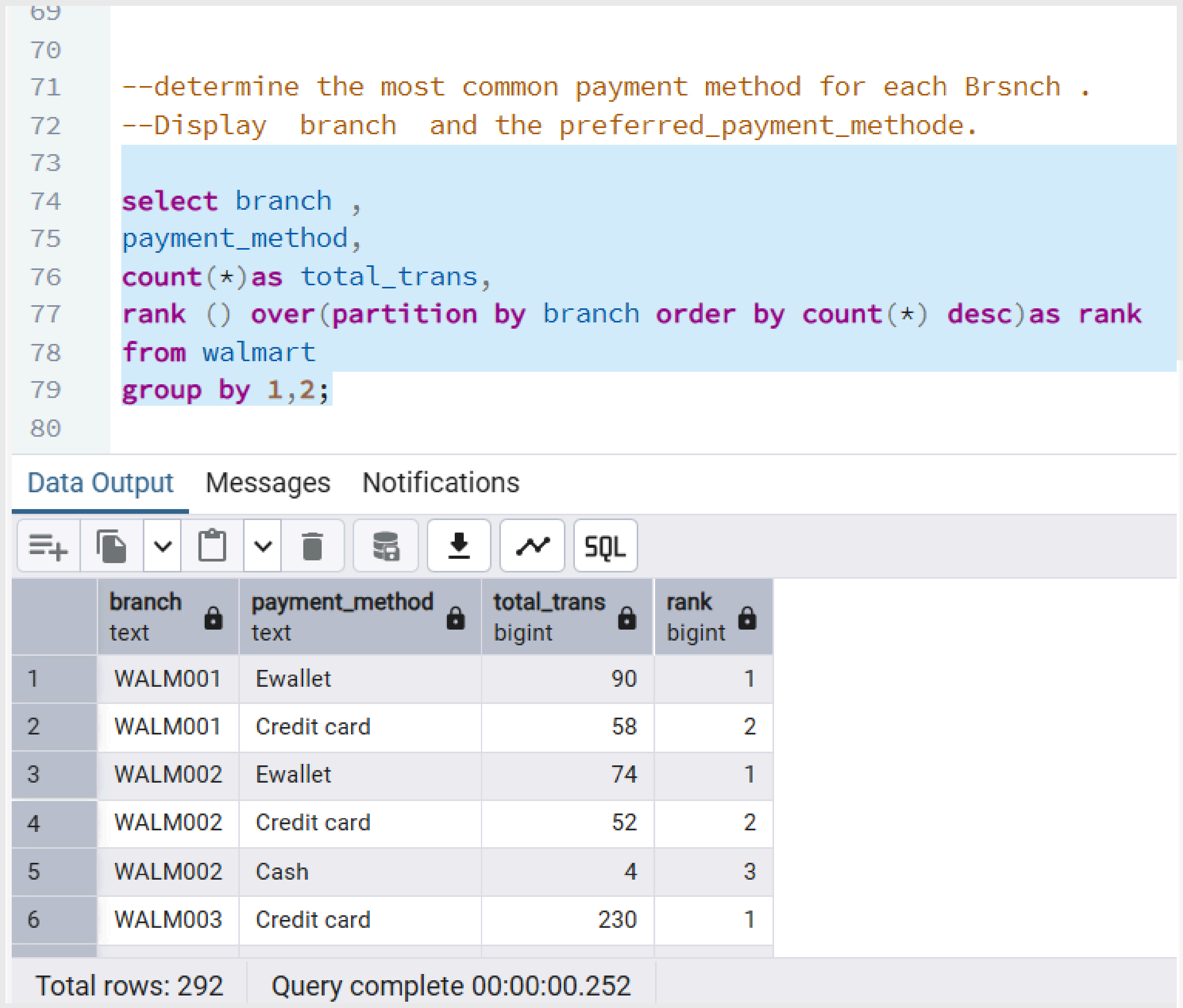
	branch text	last_year_revenue double precision	current_year_revenue double precision	rev_dec_retio numeric
1	WALM045	3462	1294	62.62
2	WALM047	5162	2138	58.58
3	WALM098	4892	2060	57.89
4	WALM033	4198	1862	55.65
5	WALM081	3446	1700	50.67
6	WALM059	4120	2246	45.49
7	WALM064	3572	1960	45.13
8	WALM097	2892	1628	43.71
9	WALM085	3842	2358	38.63
10	WALM078	4596	2842	38.16
11	WALM084	12134	8022	33.89
12	WALM088	3386	2252	33.49
13	WALM034	3290	2262	31.25
14	WALM029	10800	7500	30.56
15	WALM046	10148	7056	30.47
16	WALM004	2674	1916	28.35
17	WALM037	3496	2510	28.20
18	WALM030	11244	8260	26.54
19	WALM093	3280	2526	22.99

Total rows: 42   Query complete 00:00:00.272



```sql
-- category sales into 3 group morning , evining , afternoon
-- find out each of the shift and  number  of invoices
select time::time from walmart


select  case
when extract (hour from (time::time))<12 then 'Morning'
when extract (hour from (time::time)) between 12 and 17 then 'Afternoon'
else 'Evening'
end day_time ,
count (*)
from walmart
group by 1;
```

```
day_time    count
text        bigint
Afternoon   9272
Evening     6492
```

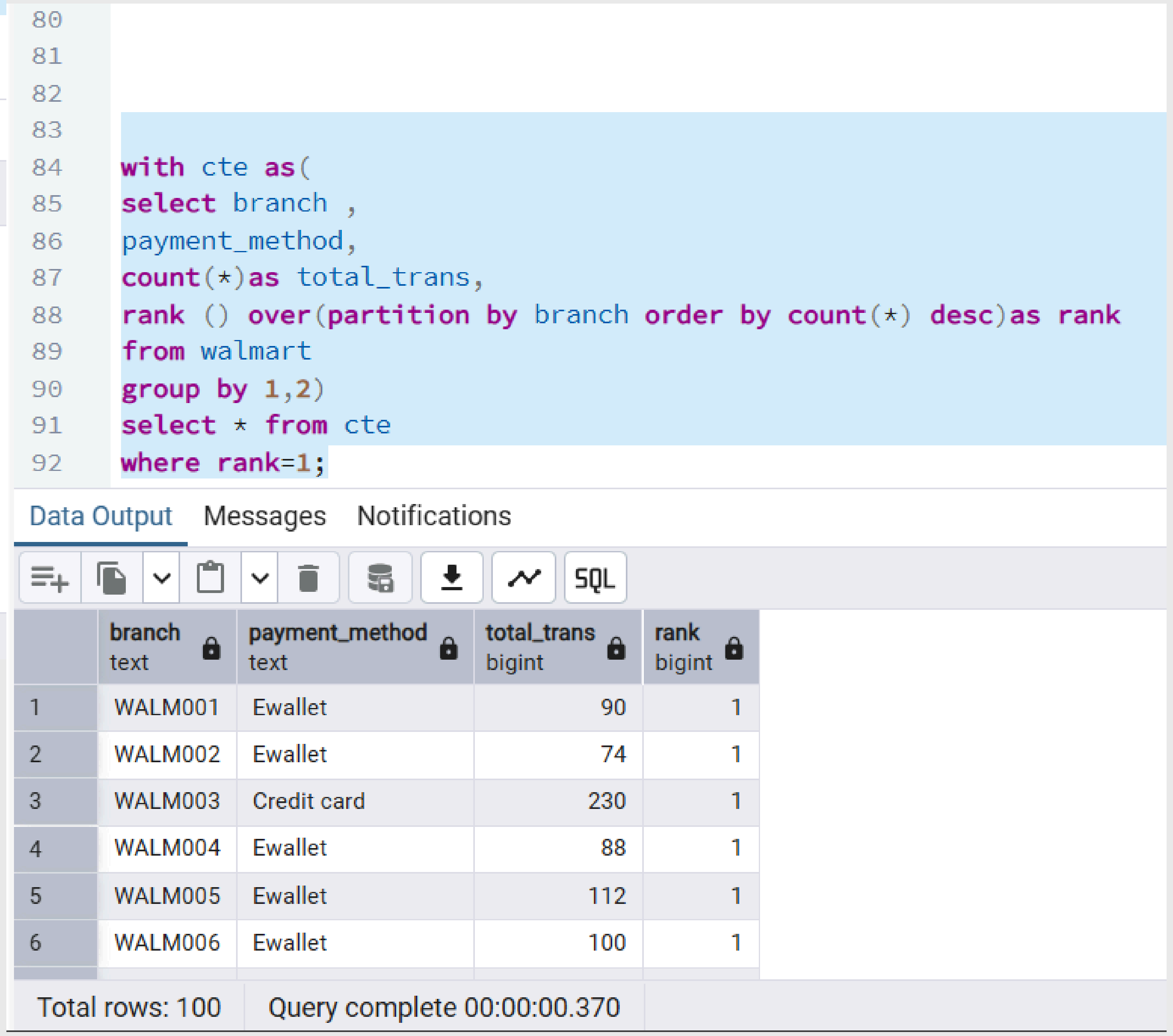


```
--determine the most common payment method for each Brsnch .
--Display  branch  and the preferred_payment_methode.

select branch ,
payment_method,
count(*)as total_trans,
rank () over(partition by branch order by count(*) desc)as rank
from walmart
group by 1,2;
```

Data Output Messages Notifications

	branch text	payment_method text	total_trans bigint	rank bigint
1	WALM001	Ewallet	90	1
2	WALM001	Credit card	58	2
3	WALM002	Ewallet	74	1
4	WALM002	Credit card	52	2
5	WALM002	Cash	4	3
6	WALM003	Credit card	230	1

Total rows: 292 Query complete 00:00:00.252

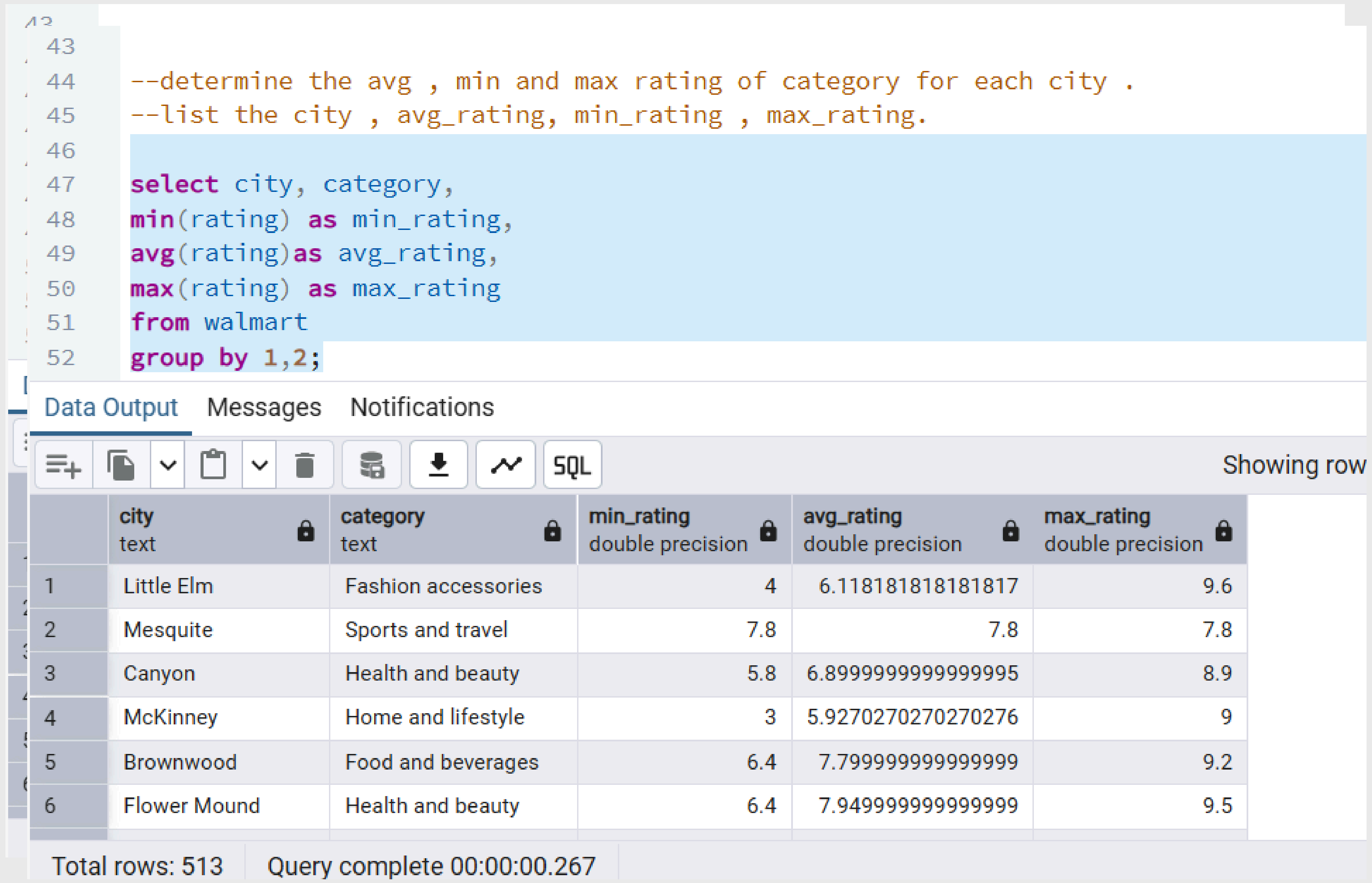


```
with cte as(
select branch ,
payment_method,
count(*)as total_trans,
rank () over(partition by branch order by count(*) desc)as rank
from walmart
group by 1,2)
select * from cte
where rank=1;
```

Data Output Messages Notifications

	branch text	payment_method text	total_trans bigint	rank bigint
1	WALM001	Ewallet	90	1
2	WALM002	Ewallet	74	1
3	WALM003	Credit card	230	1
4	WALM004	Ewallet	88	1
5	WALM005	Ewallet	112	1
6	WALM006	Ewallet	100	1

Total rows: 100 Query complete 00:00:00.370



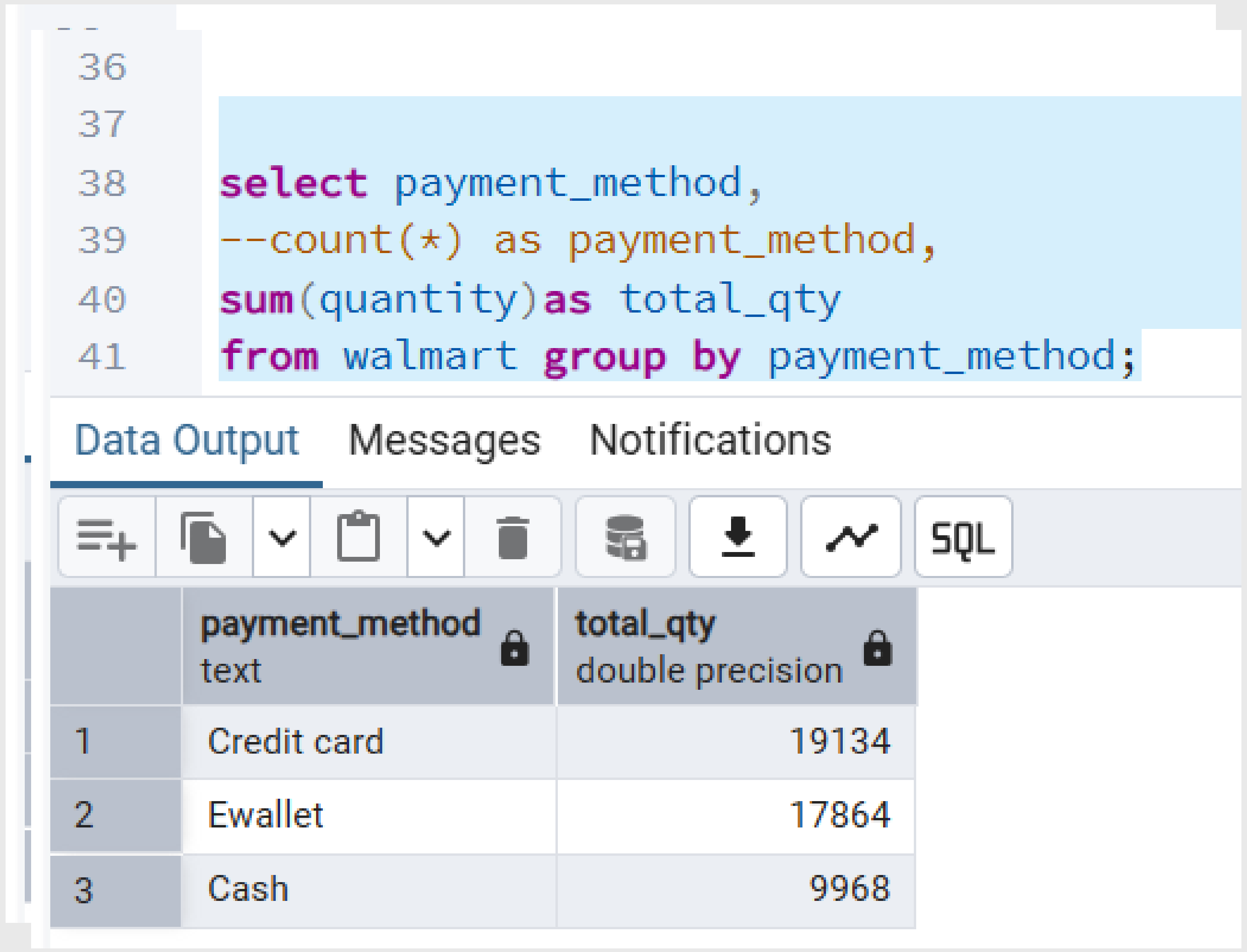
```sql
--determine the avg , min and max rating of category for each city .
--list the city , avg_rating, min_rating , max_rating.

select city, category,
min(rating) as min_rating,
avg(rating)as avg_rating,
max(rating) as max_rating
from walmart
group by 1,2;
```

Data Output Messages Notifications

Showing row

	city text	category text	min_rating double precision	avg_rating double precision	max_rating double precision
1	Little Elm	Fashion accessories	4	6.118181818181817	9.6
2	Mesquite	Sports and travel	7.8	7.8	7.8
3	Canyon	Health and beauty	5.8	6.8999999999999995	8.9
4	McKinney	Home and lifestyle	3	5.9270270270270276	9
5	Brownwood	Food and beverages	6.4	7.799999999999999	9.2
6	Flower Mound	Health and beauty	6.4	7.949999999999999	9.5

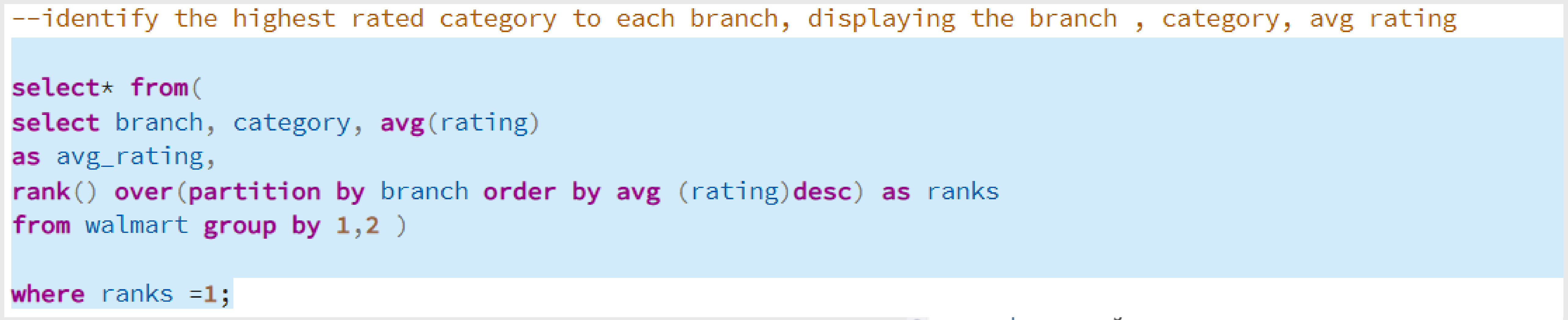
Total rows: 513 Query complete 00:00:00.267



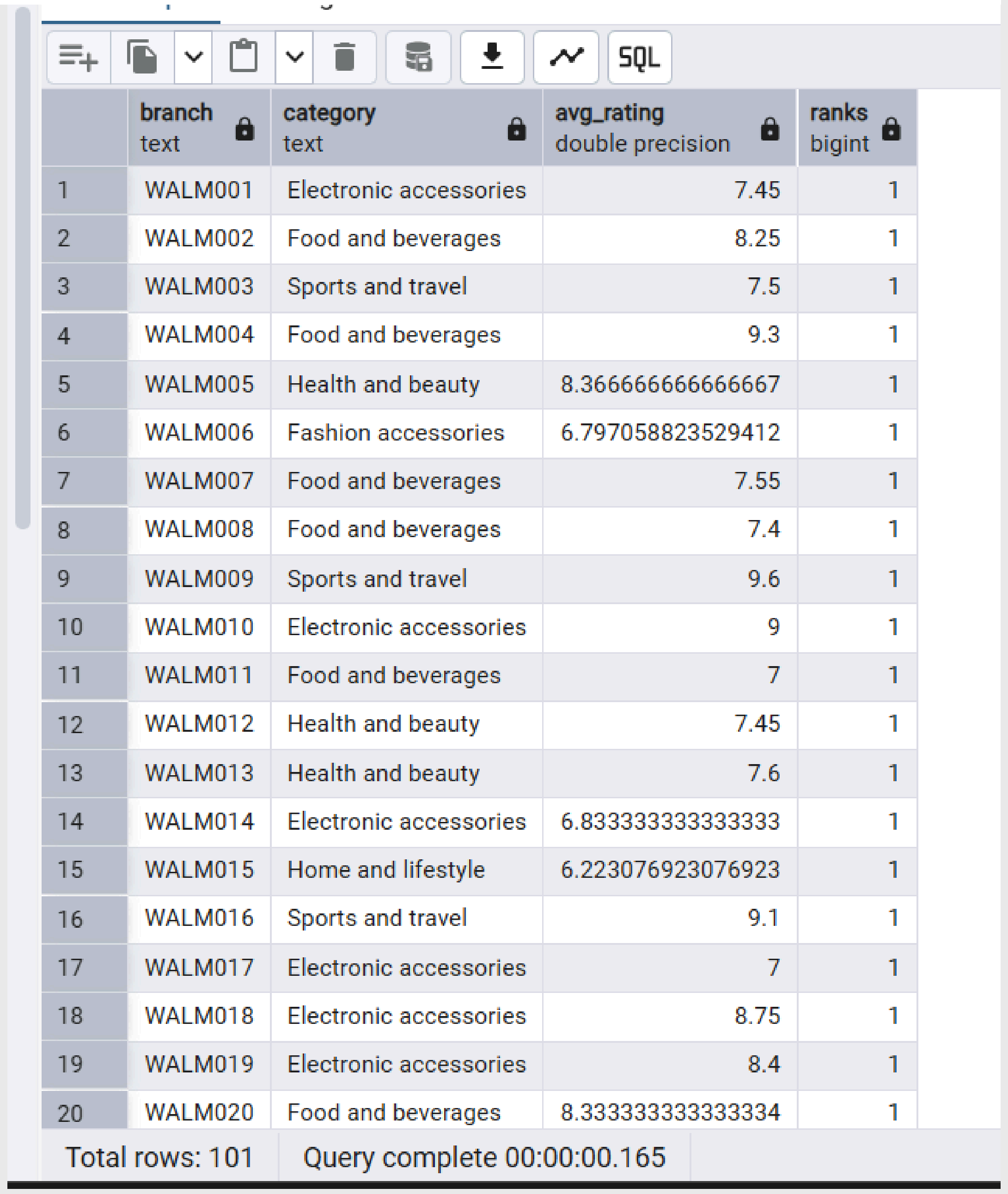
```sql
select payment_method,
--count(*) as payment_method,
sum(quantity)as total_qty
from walmart group by payment_method;
```

Data Output Messages Notifications

	payment_method text	total_qty double precision
1	Credit card	19134
2	Ewallet	17864
3	Cash	9968

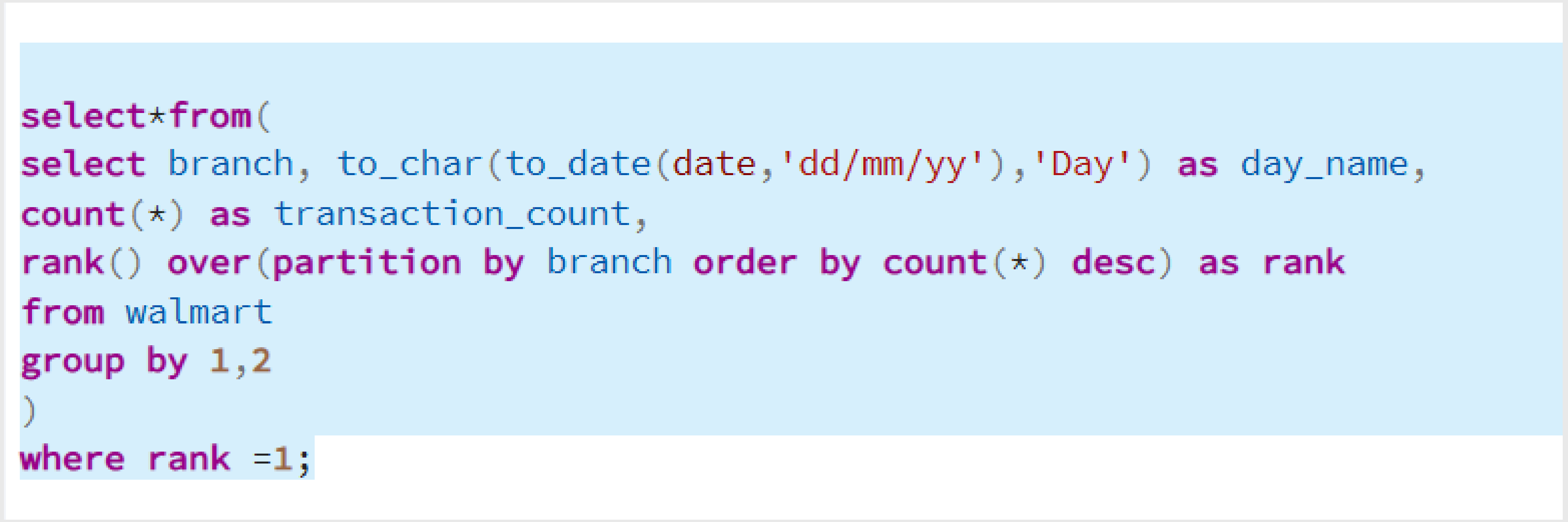


```sql
--identify the highest rated category to each branch, displaying the branch , category, avg rating

select* from(
select branch, category, avg(rating)
as avg_rating,
rank() over(partition by branch order by avg (rating)desc) as ranks
from walmart group by 1,2 )

where ranks =1;
```

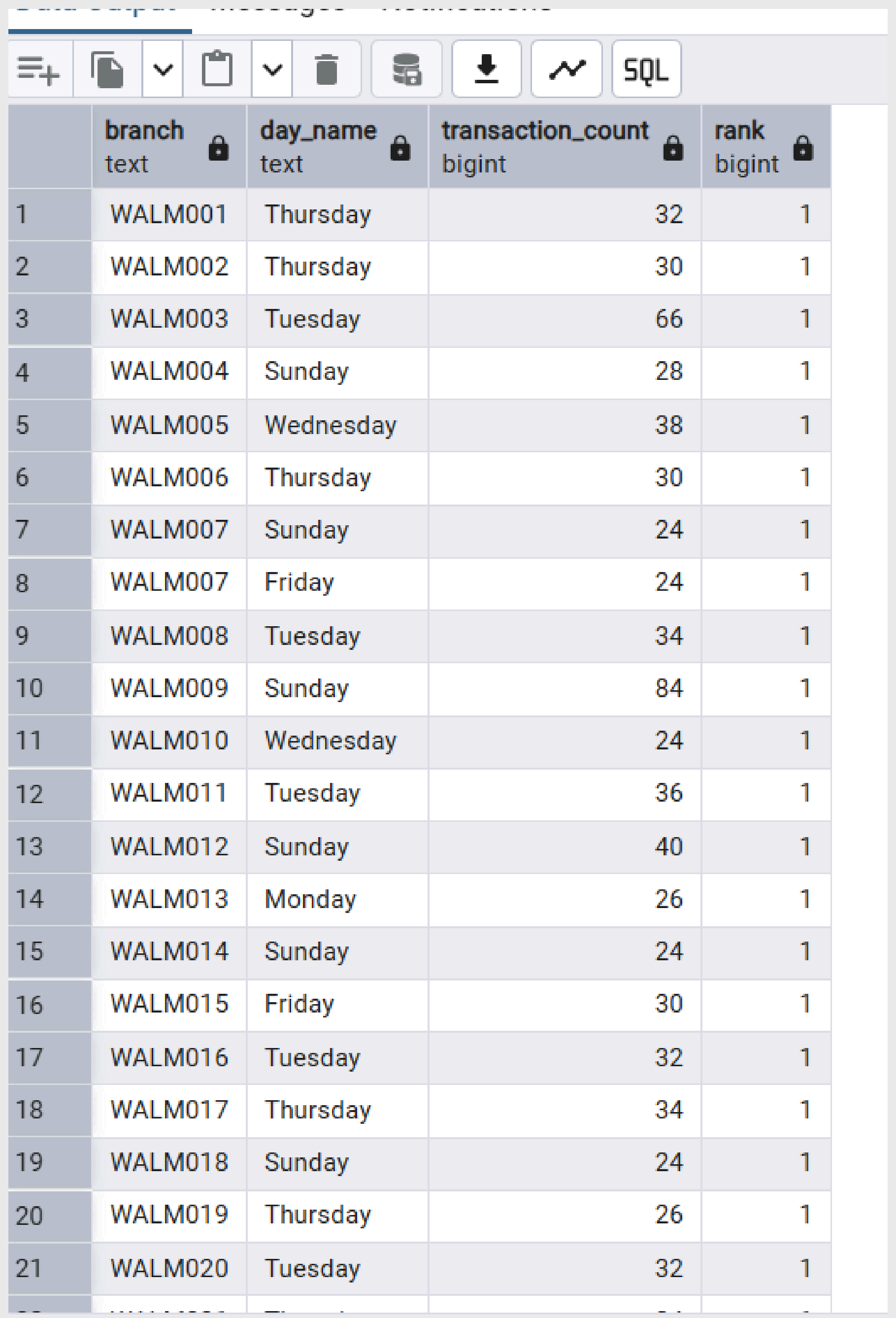


	branch text	category text	avg_rating double precision	ranks bigint
1	WALM001	Electronic accessories	7.45	1
2	WALM002	Food and beverages	8.25	1
3	WALM003	Sports and travel	7.5	1
4	WALM004	Food and beverages	9.3	1
5	WALM005	Health and beauty	8.366666666666667	1
6	WALM006	Fashion accessories	6.797058823529412	1
7	WALM007	Food and beverages	7.55	1
8	WALM008	Food and beverages	7.4	1
9	WALM009	Sports and travel	9.6	1
10	WALM010	Electronic accessories	9	1
11	WALM011	Food and beverages	7	1
12	WALM012	Health and beauty	7.45	1
13	WALM013	Health and beauty	7.6	1
14	WALM014	Electronic accessories	6.833333333333333	1
15	WALM015	Home and lifestyle	6.223076923076923	1
16	WALM016	Sports and travel	9.1	1
17	WALM017	Electronic accessories	7	1
18	WALM018	Electronic accessories	8.75	1
19	WALM019	Electronic accessories	8.4	1
20	WALM020	Food and beverages	8.333333333333334	1

Total rows: 101 Query complete 00:00:00.165



```sql
select*from(
select branch, to_char(to_date(date,'dd/mm/yy'),'Day') as day_name,
count(*) as transaction_count,
rank() over(partition by branch order by count(*) desc) as rank
from walmart
group by 1,2
)
where rank =1;
```



	branch text	day_name text	transaction_count bigint	rank bigint
1	WALM001	Thursday	32	1
2	WALM002	Thursday	30	1
3	WALM003	Tuesday	66	1
4	WALM004	Sunday	28	1
5	WALM005	Wednesday	38	1
6	WALM006	Thursday	30	1
7	WALM007	Sunday	24	1
8	WALM007	Friday	24	1
9	WALM008	Tuesday	34	1
10	WALM009	Sunday	84	1
11	WALM010	Wednesday	24	1
12	WALM011	Tuesday	36	1
13	WALM012	Sunday	40	1
14	WALM013	Monday	26	1
15	WALM014	Sunday	24	1
16	WALM015	Friday	30	1
17	WALM016	Tuesday	32	1
18	WALM017	Thursday	34	1
19	WALM018	Sunday	24	1
20	WALM019	Thursday	26	1
21	WALM020	Tuesday	32	1