Sale data analysis using sql and python

About dataset

Data set: https://www.kaggle.com/datasets/devarajv88/target-dataset?select=products.csv

Target is a globally recognized brand and a leading retailer in the United States, known for offering exceptional value, inspiration, innovation, and a unique shopping experience.

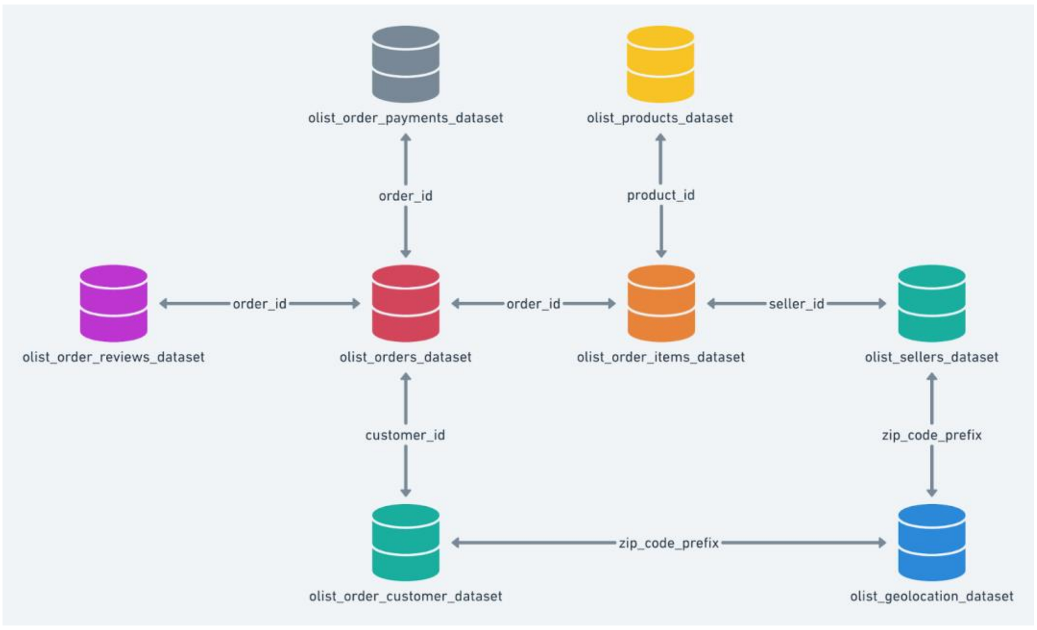
This dataset focuses on Target's operations in Brazil, covering 100,000 orders placed between 2016 and 2018. It includes detailed information on order status, pricing, payment and shipping performance, customer locations, product attributes, and customer reviews.

Features

The data is available in 8 csv files:

- customers.csv
- sellers.csv
- order_items.csv
- geolocation.csv
- payments.csv
- orders.csv
- products.csv
- Potential Use Cases

Data schema



Question to be answered

1. List all unique cities where customers are located.
2. Count the number of orders placed in 2017.
3. Find the total sales per category.
4. Calculate the percentage of orders that were paid in instalments.
5. Count the number of customers from each state.
6. Calculate the number of orders per month in 2018.
7. Find the average number of products per order, grouped by customer city.
8. Calculate the percentage of total revenue contributed by each product category.
9. Identify the correlation between product price and the number of times a product has been purchased.
10. Calculate the total revenue generated by each seller and rank them by revenue.
11. Calculate the moving average of order values for each customer over their order history.
12. Calculate the cumulative sales per month for each year.
13. Calculate the year-over-year growth rate of total sales.
14. Calculate the retention rate of customers, defined as the percentage of customers who make another purchase within 6 months of their first purchase.
15. Identify the top 3 customers who spent the most money in each year.

Setup

Before running the python create the database

Dumping the csv files directly in SQL database can take time as well as the efficiency of the dumping process might be low along with issues like

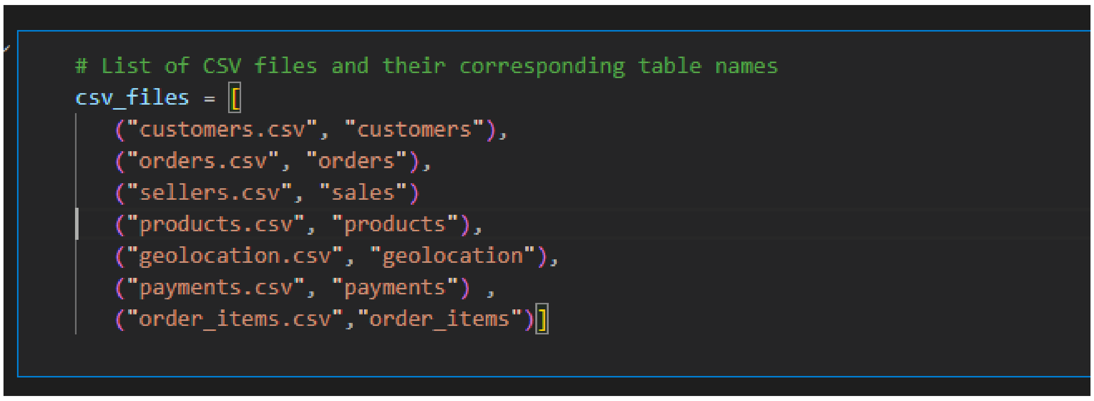
- **Data Quality**: CSV files may contain errors or inconsistencies.
- **Transformation Needs**: Data might require cleaning or conversion.
- **Performance Issues**: Large files can be slow to import directly.
- **Complex Structures**: CSV files are flat; databases often require structured data.
- **Error Handling**: Direct imports might not handle errors effectively.
- **Data Security**: Direct imports can expose sensitive data if not managed properly.
- **Data Mapping**: Fields in the CSV might need to be mapped to the database schema.

```
#!pip install mysql-connector-python
import pandas as pd
import mysql.connector
import os
```

Brief overview of the Lib used

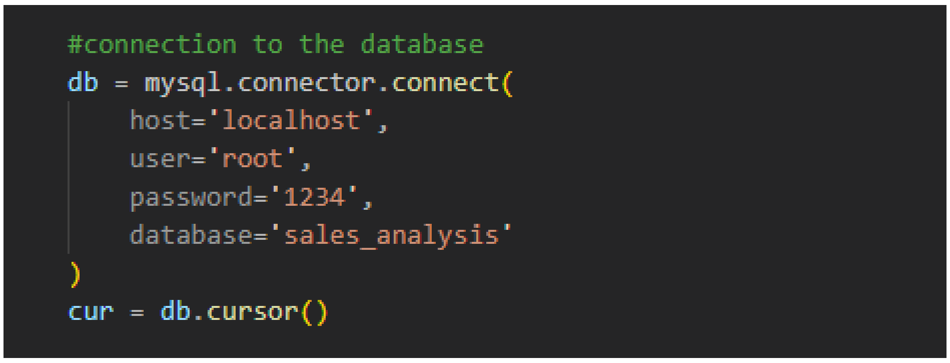
- **Pandas:** for data manipulation and analysis.
- **mysql.connector :** connect to MySQL databases from Python .

Add the respective csv and the table name in which you want to dump the csv files



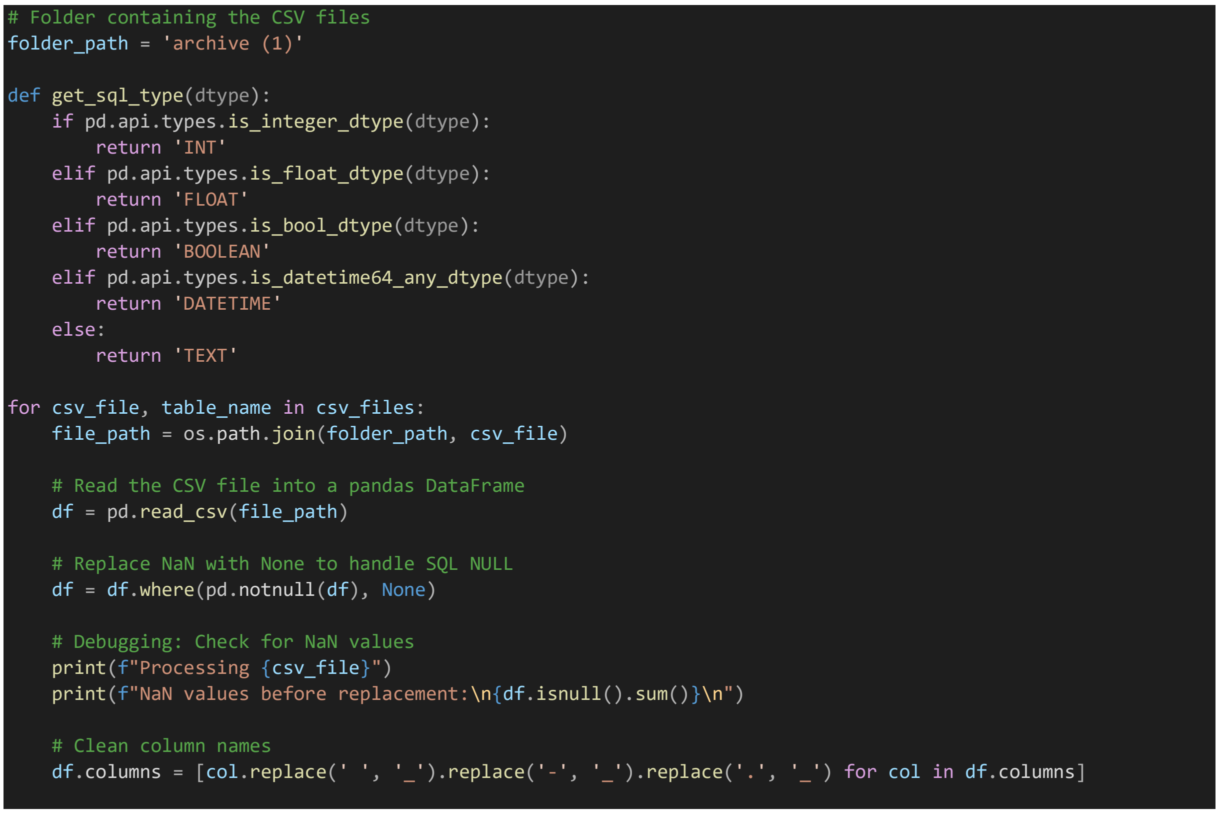
```python
# List of CSV files and their corresponding table names
csv_files = [
    ("customers.csv", "customers"),
    ("orders.csv", "orders"),
    ("sellers.csv", "sales")
    ("products.csv", "products"),
    ("geolocation.csv", "geolocation"),
    ("payments.csv", "payments") ,
    ("order_items.csv","order_items")]
```

Add the connection details to your sql workbench



```python
#connection to the database
db = mysql.connector.connect(
    host='localhost',
    user='root',
    password='1234',
    database='sales_analysis'
)
cur = db.cursor()
```

Process CSV files and insert their data into a MySQL database.



```python
# Folder containing the CSV files
folder_path = 'archive (1)'

def get_sql_type(dtype):
    if pd.api.types.is_integer_dtype(dtype):
        return 'INT'
    elif pd.api.types.is_float_dtype(dtype):
        return 'FLOAT'
    elif pd.api.types.is_bool_dtype(dtype):
        return 'BOOLEAN'
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return 'DATETIME'
    else:
        return 'TEXT'

for csv_file, table_name in csv_files:
    file_path = os.path.join(folder_path, csv_file)

    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(file_path)

    # Replace NaN with None to handle SQL NULL
    df = df.where(pd.notnull(df), None)

    # Debugging: Check for NaN values
    print(f"Processing {csv_file}")
    print(f"NaN values before replacement:\n{df.isnull().sum()}\n")

    # Clean column names
    df.columns = [col.replace(' ', '_').replace('-', '_').replace('.', '_') for col in df.columns]
    # Generate the CREATE TABLE statement with appropriate data types
    columns = ', '.join([f'`{col}` {get_sql_type(df[col].dtype)}' for col in df.columns])
    create_table_query = f'CREATE TABLE IF NOT EXISTS `{table_name}` ({columns})'
    cursor.execute(create_table_query)

    # Insert DataFrame data into the MySQL table
for _, row in df.iterrows():
        # Convert row to tuple and handle NaN/None explicitly
        values = tuple(None if pd.isna(x) else x for x in row)
        sql = f"INSERT INTO `{table_name}` ({', '.join(['`' + col + '`' for col in df.columns])}) VALUES ({', 
'.join(['%s'] * len(row))})"
        cursor.execute(sql, values)

    # Commit the transaction for the current CSV file
conn.commit()

# Close the connection
conn.close()
```

Define Folder Path:

- folder_path specifies the directory containing the CSV files.

get_sql_type Function:

- This function maps Pandas data types to SQL data types for creating tables in MySQL:
    - Integer types are mapped to INT.
    - Float types are mapped to FLOAT.
    - Boolean types are mapped to BOOLEAN.
    - DateTime types are mapped to DATETIME.
    - Other types are mapped to TEXT.

Processing Each CSV File:

- The code iterates over csv_files, which is assumed to be a list of tuples where each tuple contains the name of a CSV file and the corresponding table name in the database.
- For each CSV file:
    - The file is read into a Pandas DataFrame.
    - NaN values in the DataFrame are replaced with None to be handled as SQL NULL values.
    - Column names are cleaned by replacing spaces, hyphens, and periods with underscores.

Create SQL Table:

- The CREATE TABLE SQL query is dynamically generated based on the DataFrame's columns and their inferred SQL data types.
- The table is created in MySQL if it does not already exist.

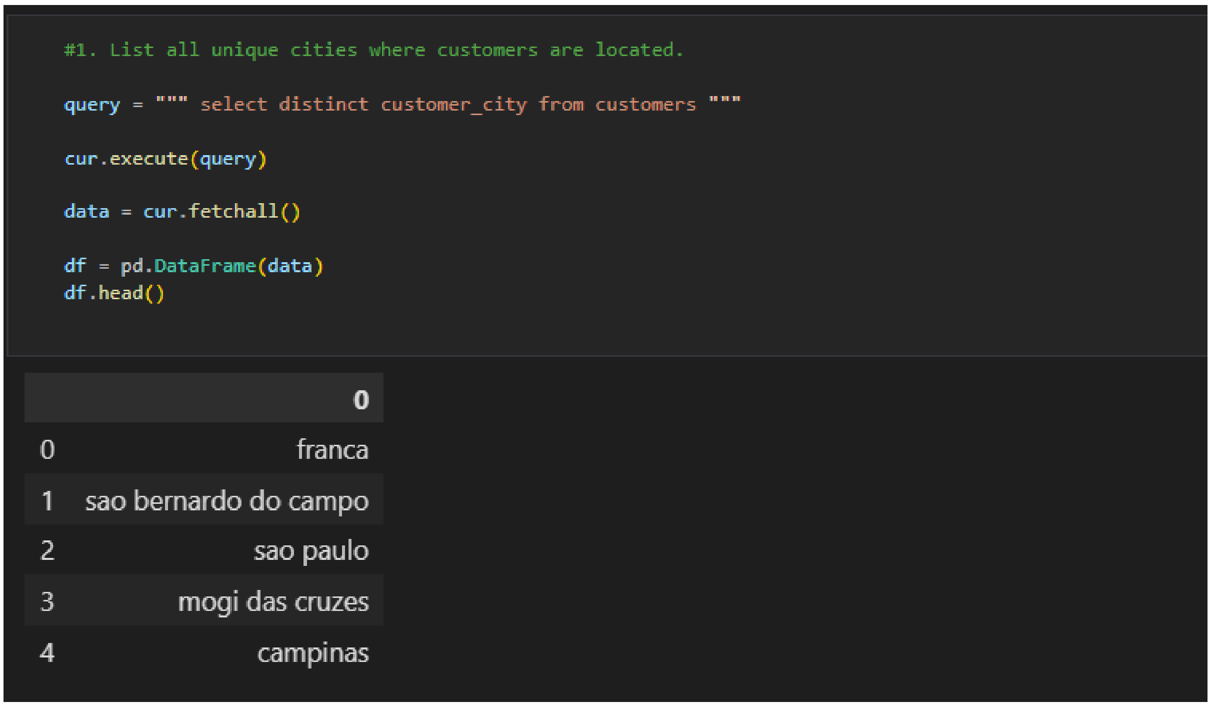
Insert Data into MySQL Table:

- For each row in the DataFrame:

    - The row is converted to a tuple, with NaN values explicitly set to None.
    - An INSERT INTO SQL query is prepared and executed to insert the row data into the MySQL table.

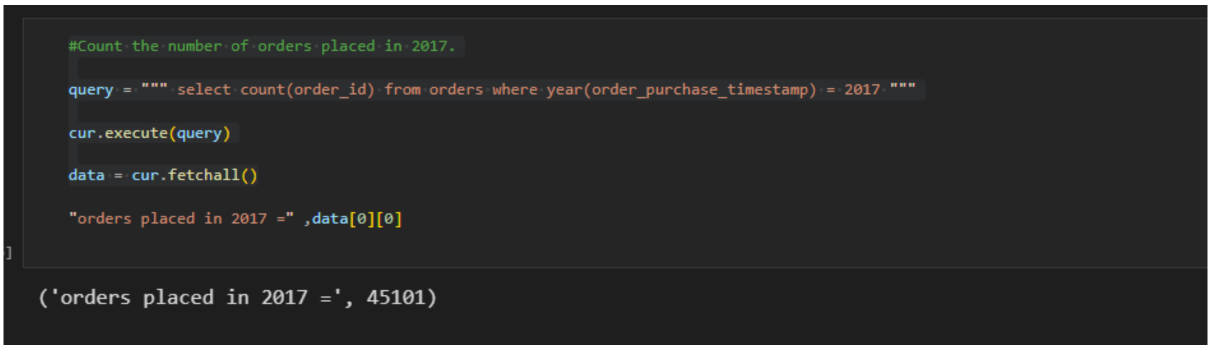
Commit and Close:

- After processing all rows for all CSV files, the transaction is committed to the database.
- The connection to the MySQL database is closed.



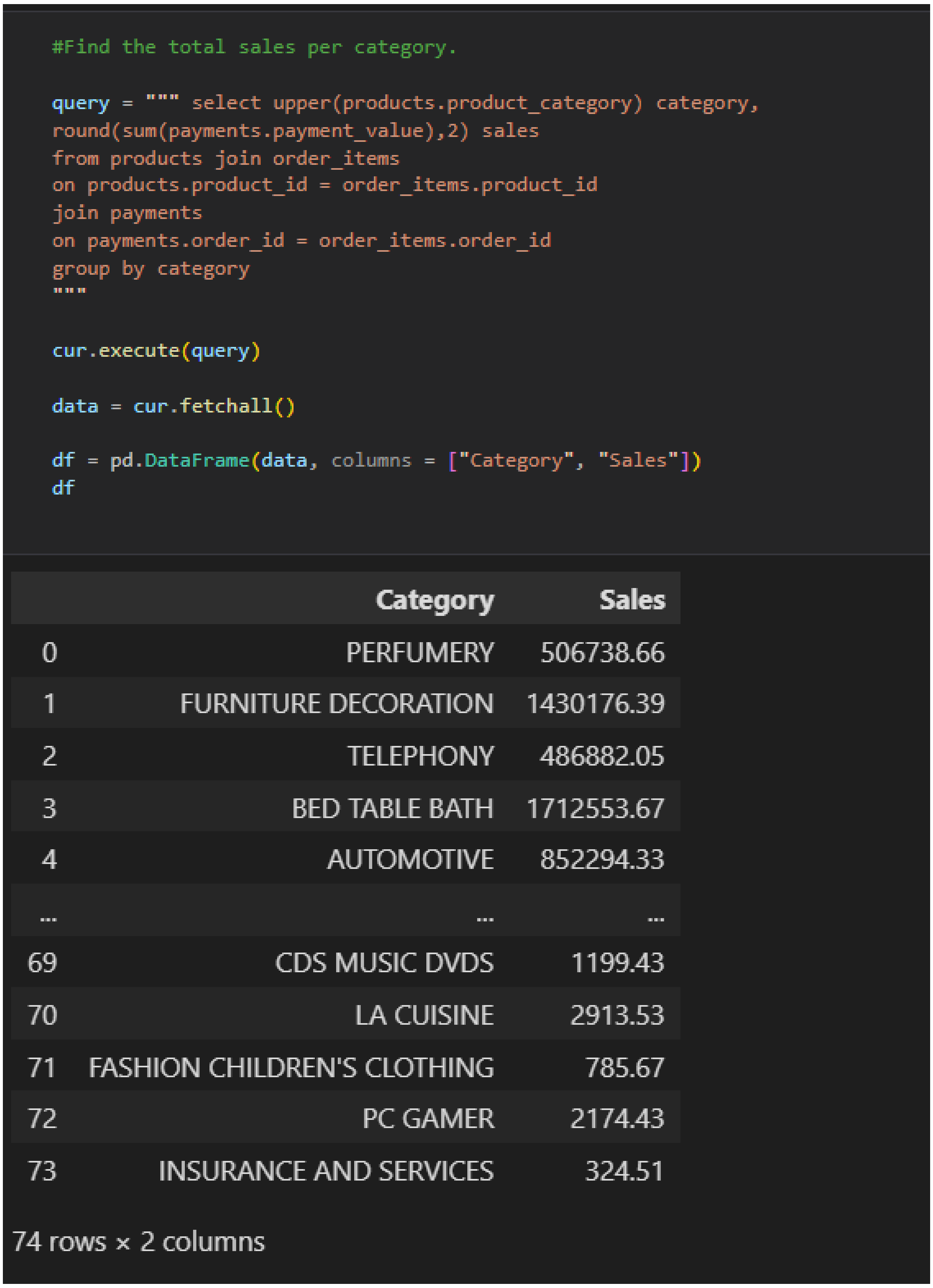
```
#1. List all unique cities where customers are located.

query = """ select distinct customer_city from customers """

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data)
df.head()
```

	0
0	franca
1	sao bernardo do campo
2	sao paulo
3	mogi das cruzes
4	campinas



```
#Count the number of orders placed in 2017.

query = """ select count(order_id) from orders where year(order_purchase_timestamp) = 2017 """

cur.execute(query)

data = cur.fetchall()

"orders placed in 2017 =" ,data[0][0]
```

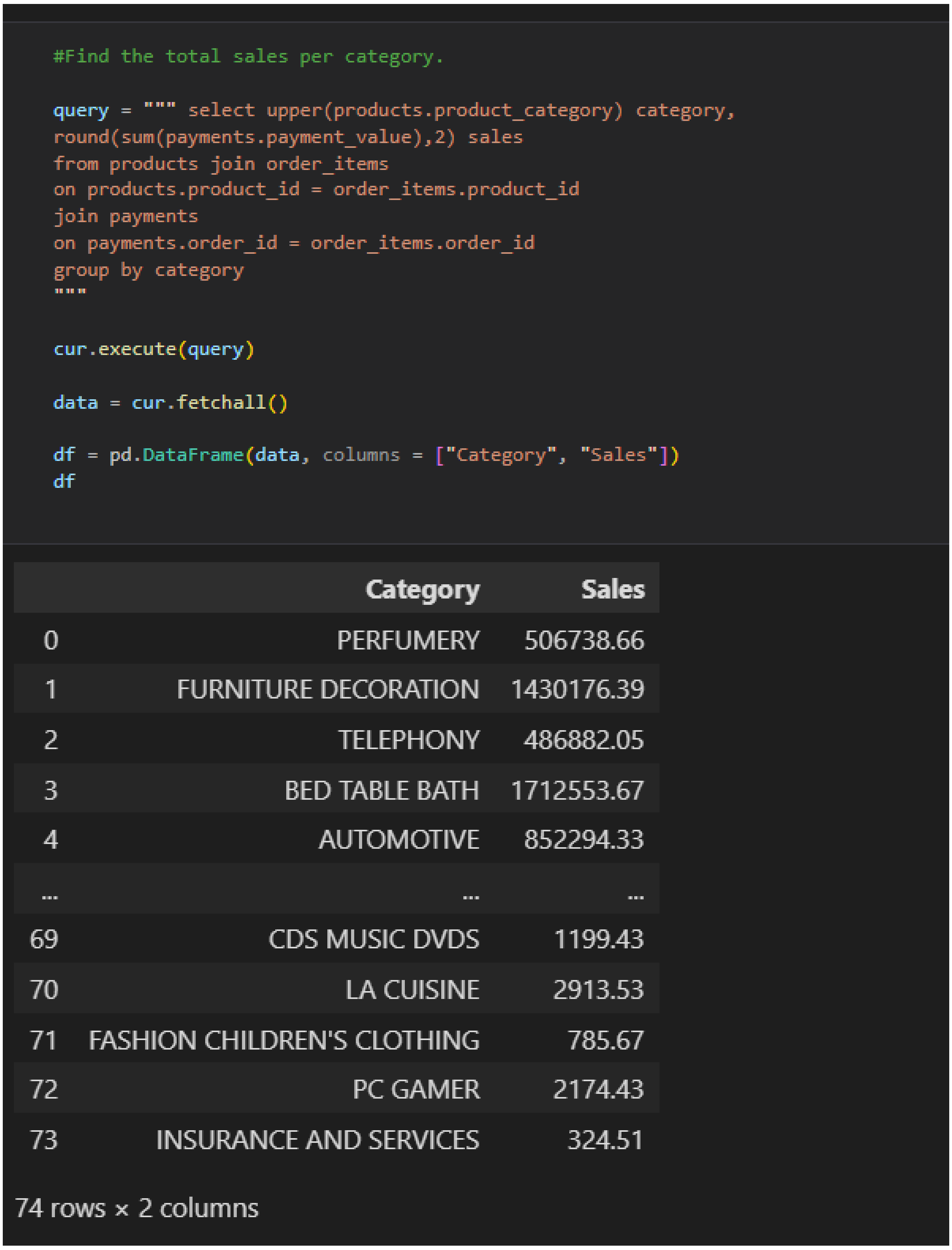
```
('orders placed in 2017 =', 45101)
```



```
#Find the total sales per category.

query = """ select upper(products.product_category) category,
round(sum(payments.payment_value),2) sales
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category
"""

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data, columns = ["Category", "Sales"])
df
```

	Category	Sales
0	PERFUMERY	506738.66
1	FURNITURE DECORATION	1430176.39
2	TELEPHONY	486882.05
3	BED TABLE BATH	1712553.67
4	AUTOMOTIVE	852294.33
...	...	...
69	CDS MUSIC DVDS	1199.43
70	LA CUISINE	2913.53
71	FASHION CHILDREN'S CLOTHING	785.67
72	PC GAMER	2174.43
73	INSURANCE AND SERVICES	324.51

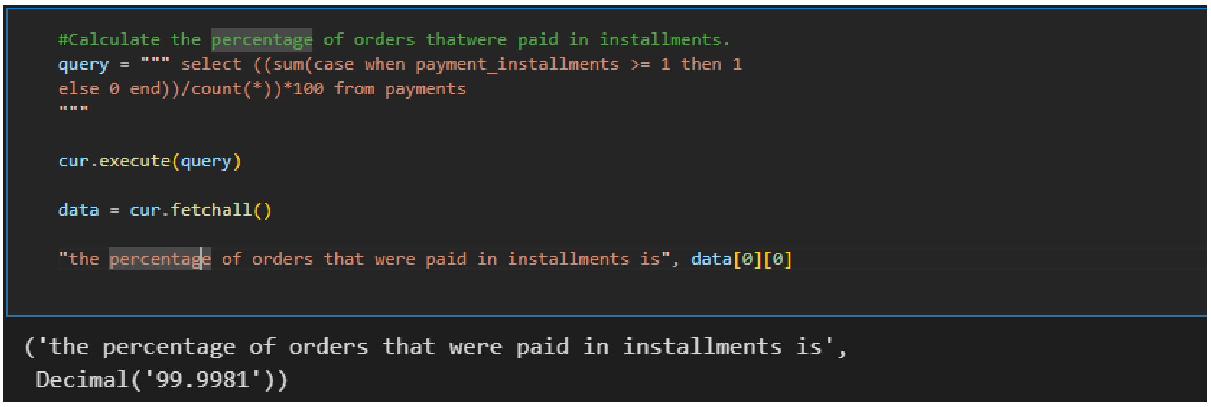
74 rows × 2 columns



```
#Find the total sales per category.

query = """ select upper(products.product_category) category,
round(sum(payments.payment_value),2) sales
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category
"""

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data, columns = ["Category", "Sales"])
df
```

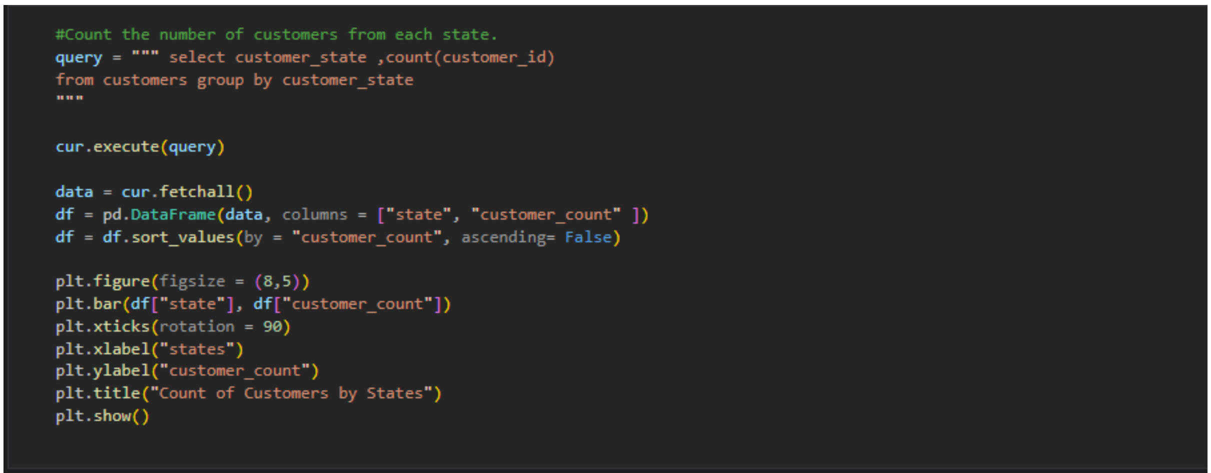
	Category	Sales
0	PERFUMERY	506738.66
1	FURNITURE DECORATION	1430176.39
2	TELEPHONY	486882.05
3	BED TABLE BATH	1712553.67
4	AUTOMOTIVE	852294.33
...	...	...
69	CDS MUSIC DVDS	1199.43
70	LA CUISINE	2913.53
71	FASHION CHILDREN'S CLOTHING	785.67
72	PC GAMER	2174.43
73	INSURANCE AND SERVICES	324.51

74 rows × 2 columns

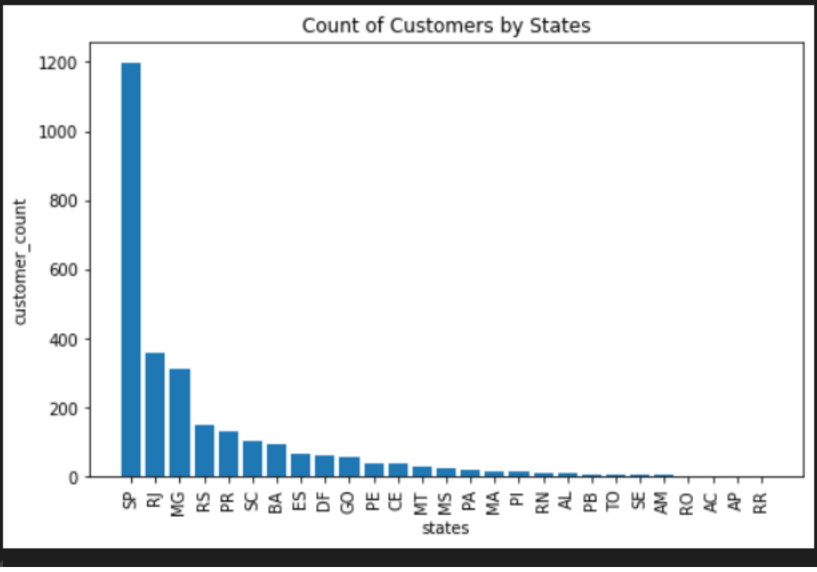


```
#Calculate the percentage of orders thatwere paid in installments.
query = """ select ((sum(case when payment_installments >= 1 then 1
else 0 end))/count(*))*100 from payments
"""

cur.execute(query)

data = cur.fetchall()

"the percentage of orders that were paid in installments is", data[0][0]
```

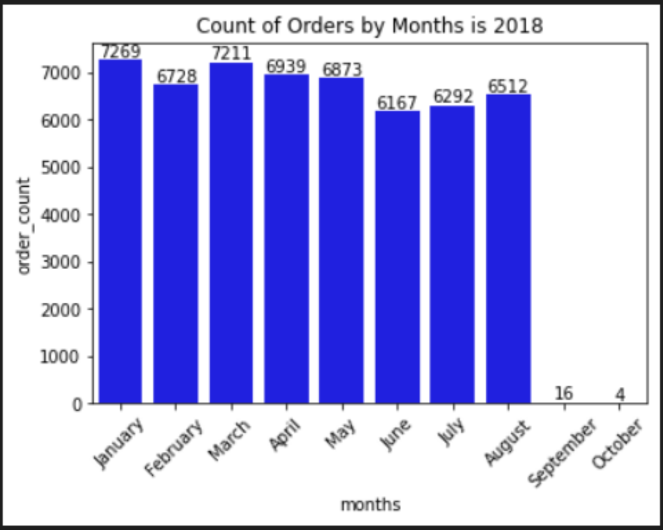
```
('the percentage of orders that were paid in installments is',
 Decimal('99.9981'))
```

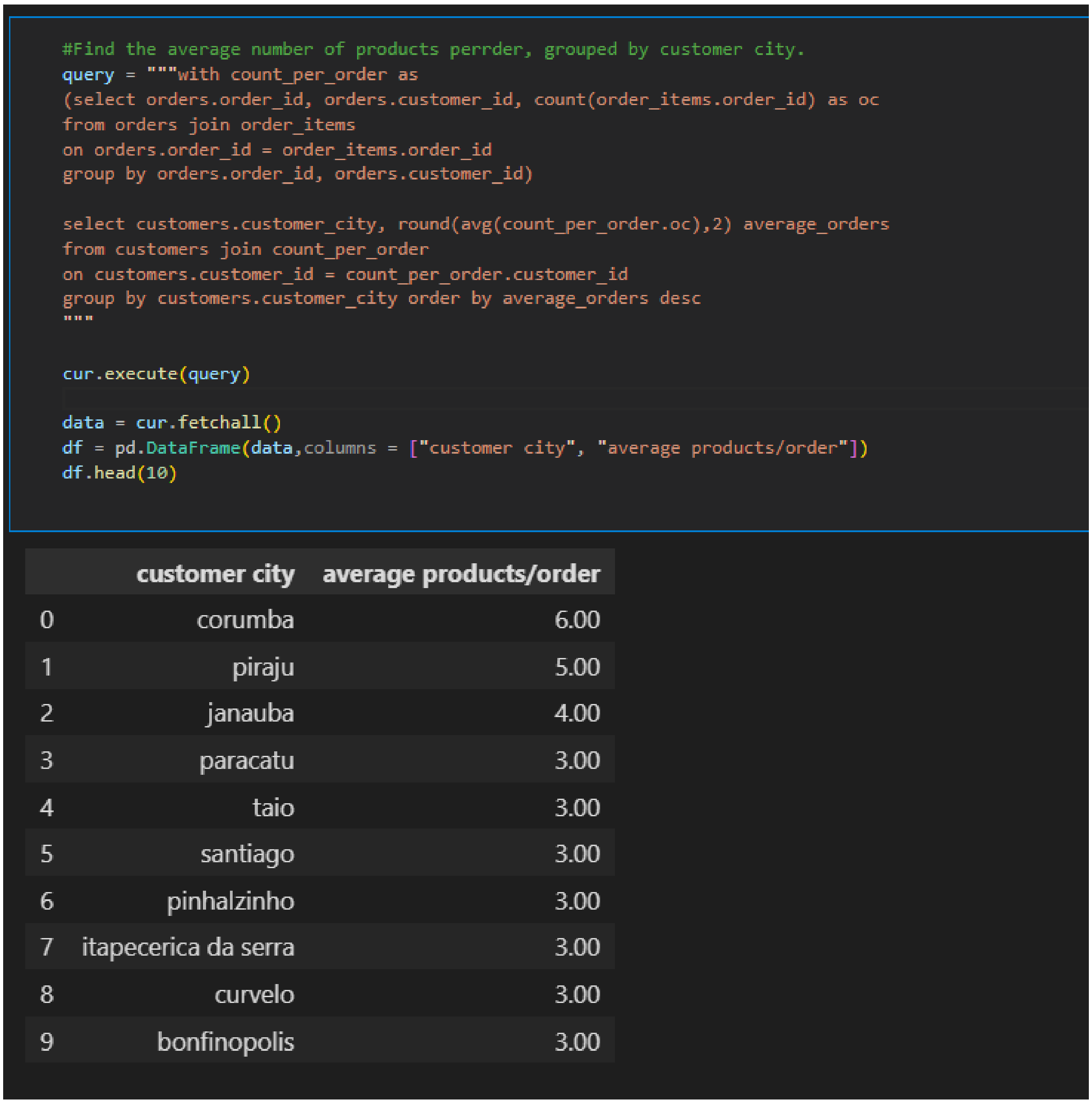


```
#Count the number of customers from each state.
query = """ select customer_state ,count(customer_id)
from customers group by customer_state
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["state", "customer_count" ])
df = df.sort_values(by = "customer_count", ascending= False)

plt.figure(figsize = (8,5))
plt.bar(df["state"], df["customer_count"])
plt.xticks(rotation = 90)
plt.xlabel("states")
plt.ylabel("customer_count")
plt.title("Count of Customers by States")
plt.show()
```



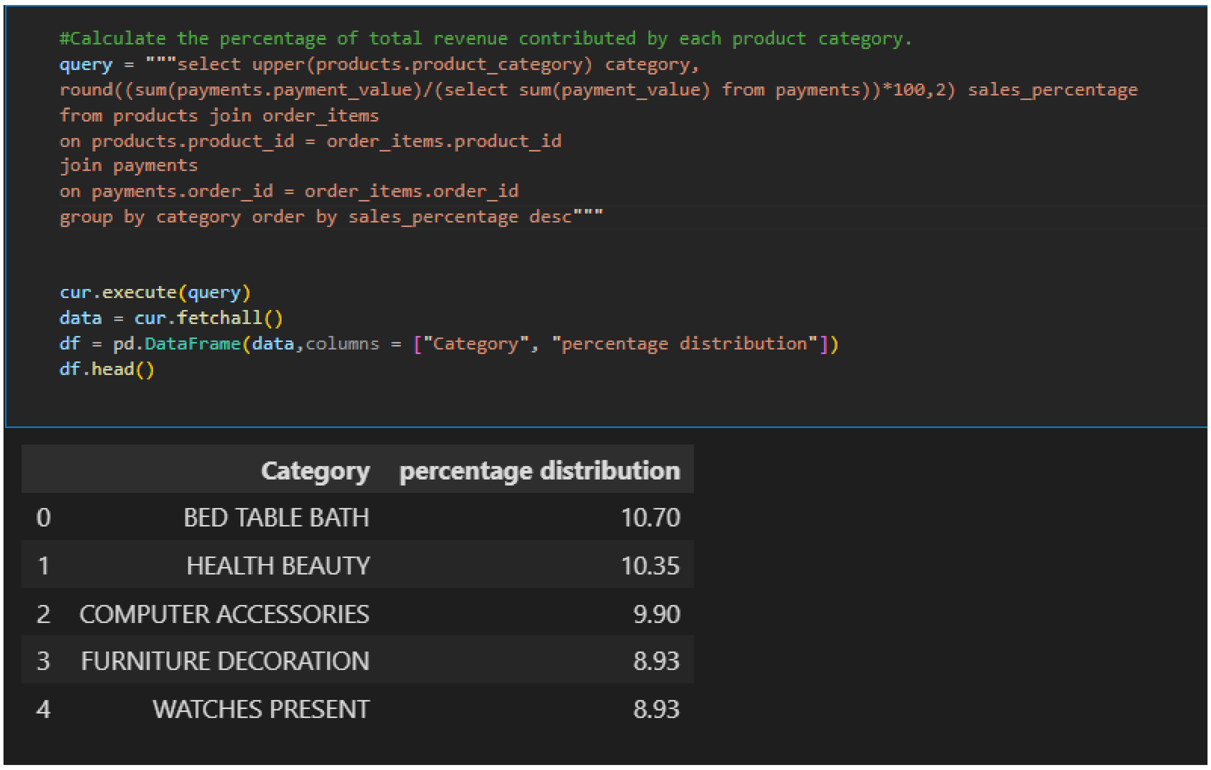
```
#Calculate the number of orders per month in 2018.
query = """ select monthname(order_purchase_timestamp) months, count(order_id) order_count
from orders where year(order_purchase_timestamp) = 2018
group by months
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["months", "order_count"])
o = ["January", "February","March","April","May","June","July","August","September","October"]

ax = sns.barplot(x = df["months"],y =  df["order_count"], data = df, order = o, color = "blue")
plt.xticks(rotation = 45)
ax.bar_label(ax.containers[0])
plt.title("Count of Orders by Months is 2018")

plt.show()
```





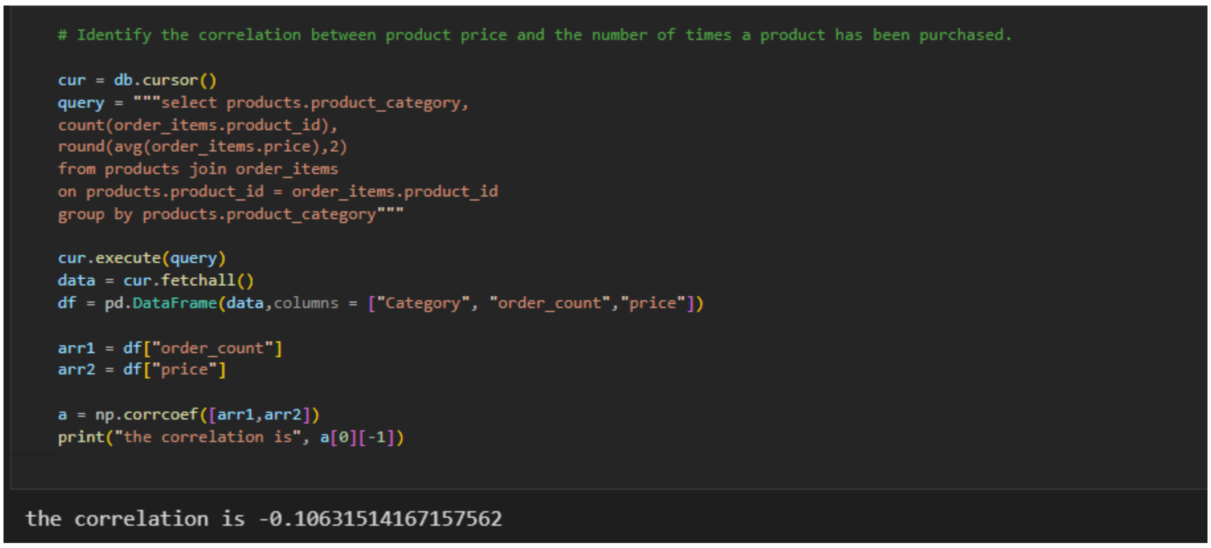
```
#Find the average number of products perrder, grouped by customer city.
query = """with count_per_order as
(select orders.order_id, orders.customer_id, count(order_items.order_id) as oc
from orders join order_items
on orders.order_id = order_items.order_id
group by orders.order_id, orders.customer_id)

select customers.customer_city, round(avg(count_per_order.oc),2) average_orders
from customers join count_per_order
on customers.customer_id = count_per_order.customer_id
group by customers.customer_city order by average_orders desc
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data,columns = ["customer city", "average products/order"])
df.head(10)
```

	customer city	average products/order
0	corumba	6.00
1	piraju	5.00
2	janauba	4.00
3	paracatu	3.00
4	taio	3.00
5	santiago	3.00
6	pinhalzinho	3.00
7	itapecerica da serra	3.00
8	curvelo	3.00
9	bonfinopolis	3.00



```python
#Calculate the percentage of total revenue contributed by each product category.
query = """select upper(products.product_category) category,
round((sum(payments.payment_value)/(select sum(payment_value) from payments))*100,2) sales_percentage
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category order by sales_percentage desc"""


cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns = ["Category", "percentage distribution"])
df.head()
```

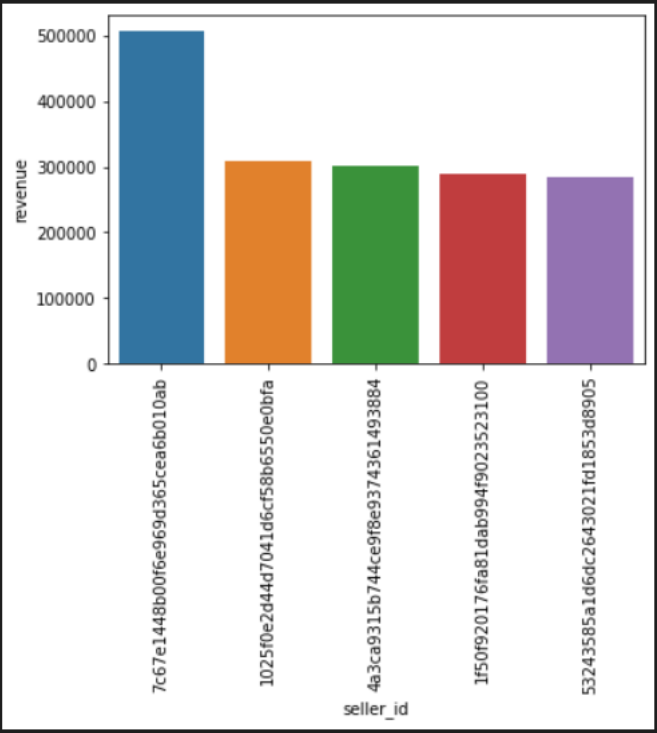
	Category	percentage distribution
0	BED TABLE BATH	10.70
1	HEALTH BEAUTY	10.35
2	COMPUTER ACCESSORIES	9.90
3	FURNITURE DECORATION	8.93
4	WATCHES PRESENT	8.93

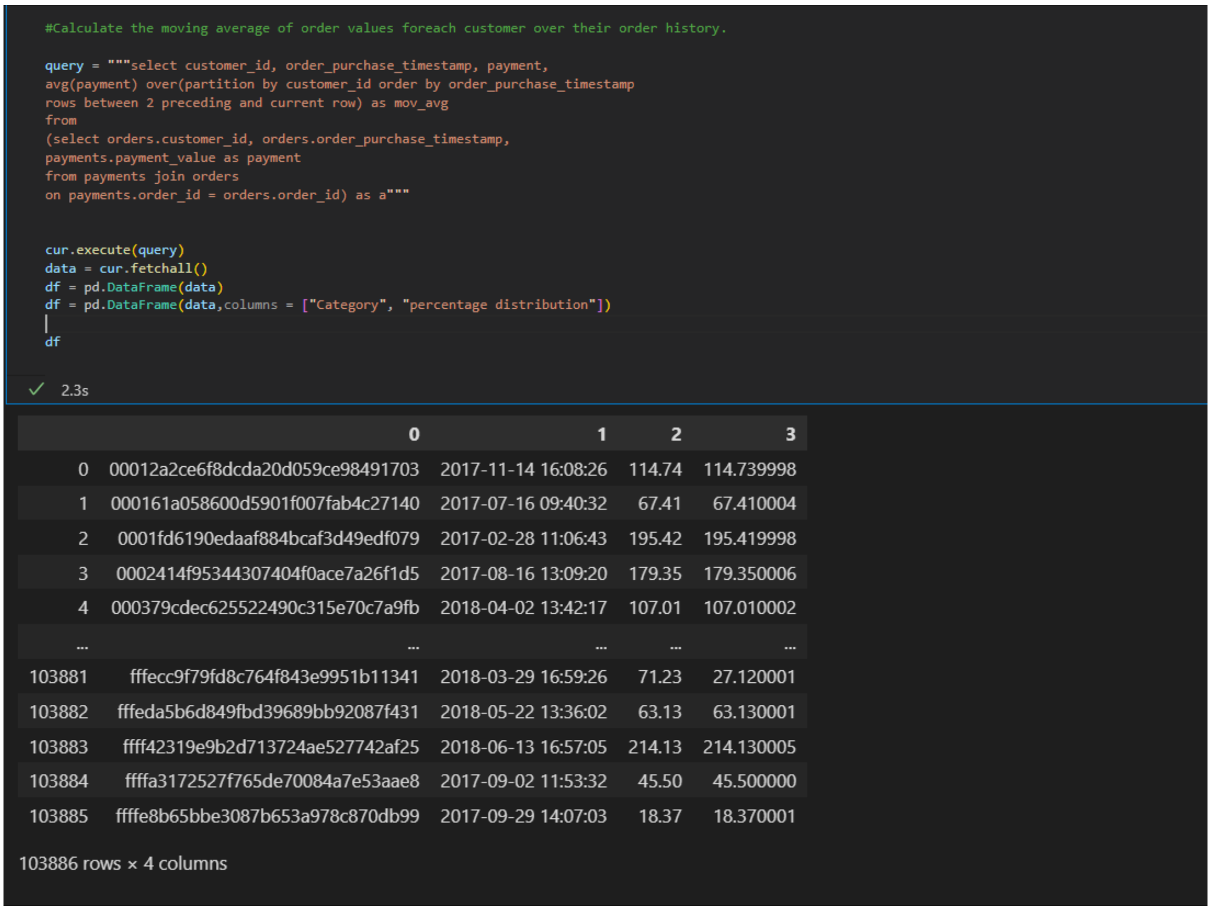


```python
# Identify the correlation between product price and the number of times a product has been purchased.

cur = db.cursor()
query = """select products.product_category,
count(order_items.product_id),
round(avg(order_items.price),2)
from products join order_items
on products.product_id = order_items.product_id
group by products.product_category"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns = ["Category", "order_count","price"])

arr1 = df["order_count"]
arr2 = df["price"]

a = np.corrcoef([arr1,arr2])
print("the correlation is", a[0][-1])
```

```
the correlation is -0.10631514167157562
```

```python
# Calculate the total revenue generated by each seller, and rank them by revenue
query = """ select *, dense_rank() over(order by revenue desc) as rn from
(select order_items.seller_id, sum(payments.payment_value)
revenue from order_items join payments
on order_items.order_id = payments.order_id
group by order_items.seller_id) as a """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["seller_id", "revenue", "rank"])
df = df.head()
sns.barplot(x = "seller_id", y = "revenue", data = df)
plt.xticks(rotation = 90)
plt.show()
```

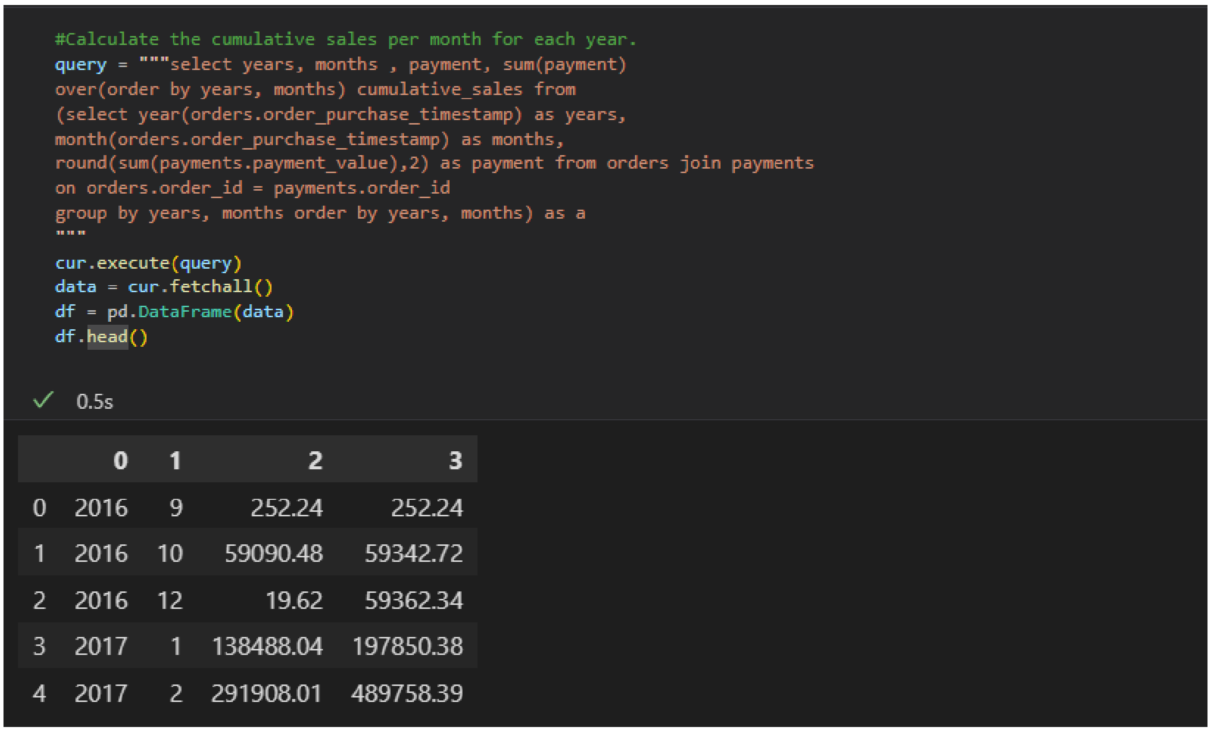




```
#Calculate the moving average of order values foreach customer over their order history.

query = """select customer_id, order_purchase_timestamp, payment,
avg(payment) over(partition by customer_id order by order_purchase_timestamp
rows between 2 preceding and current row) as mov_avg
from
(select orders.customer_id, orders.order_purchase_timestamp,
payments.payment_value as payment
from payments join orders
on payments.order_id = orders.order_id) as a"""


cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data)
df = pd.DataFrame(data,columns = ["Category", "percentage distribution"])

df
```

```
✓ 2.3s
```

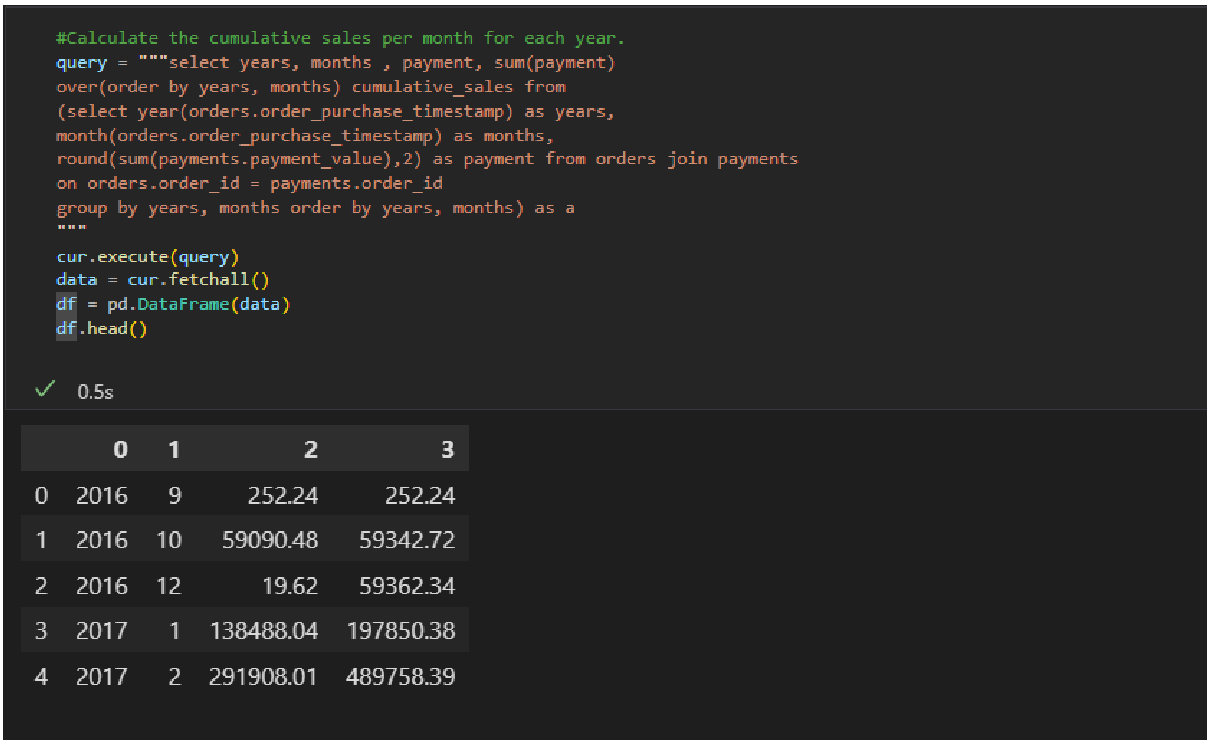
```
                                       0                    1       2           3
     0  00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74  114.739998
     1  000161a058600d5901f007fab4c27140  2017-07-16 09:40:32   67.41   67.410004
     2  0001fd6190edaaf884bcaf3d49edf079  2017-02-28 11:06:43  195.42  195.419998
     3  0002414f95344307404f0ace7a26f1d5  2017-08-16 13:09:20  179.35  179.350006
     4  000379cdec625522490c315e70c7a9fb  2018-04-02 13:42:17  107.01  107.010002
   ...                               ...                  ...     ...         ...
103881  fffecc9f79fd8c764f843e9951b11341  2018-03-29 16:59:26   71.23   27.120001
103882  fffeda5b6d849fbd39689bb92087f431  2018-05-22 13:36:02   63.13   63.130001
103883  ffff42319e9b2d713724ae527742af25  2018-06-13 16:57:05  214.13  214.130005
103884  ffffa3172527f765de70084a7e53aae8  2017-09-02 11:53:32   45.50   45.500000
103885  ffffe8b65bbe3087b653a978c870db99  2017-09-29 14:07:03   18.37   18.370001

103886 rows × 4 columns
```



```
#Calculate the cumulative sales per month for each year.
query = """select years, months , payment, sum(payment)
over(order by years, months) cumulative_sales from
(select year(orders.order_purchase_timestamp) as years,
month(orders.order_purchase_timestamp) as months,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by years, months order by years, months) as a
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data)
df.head()
```

```
✓ 0.5s
```

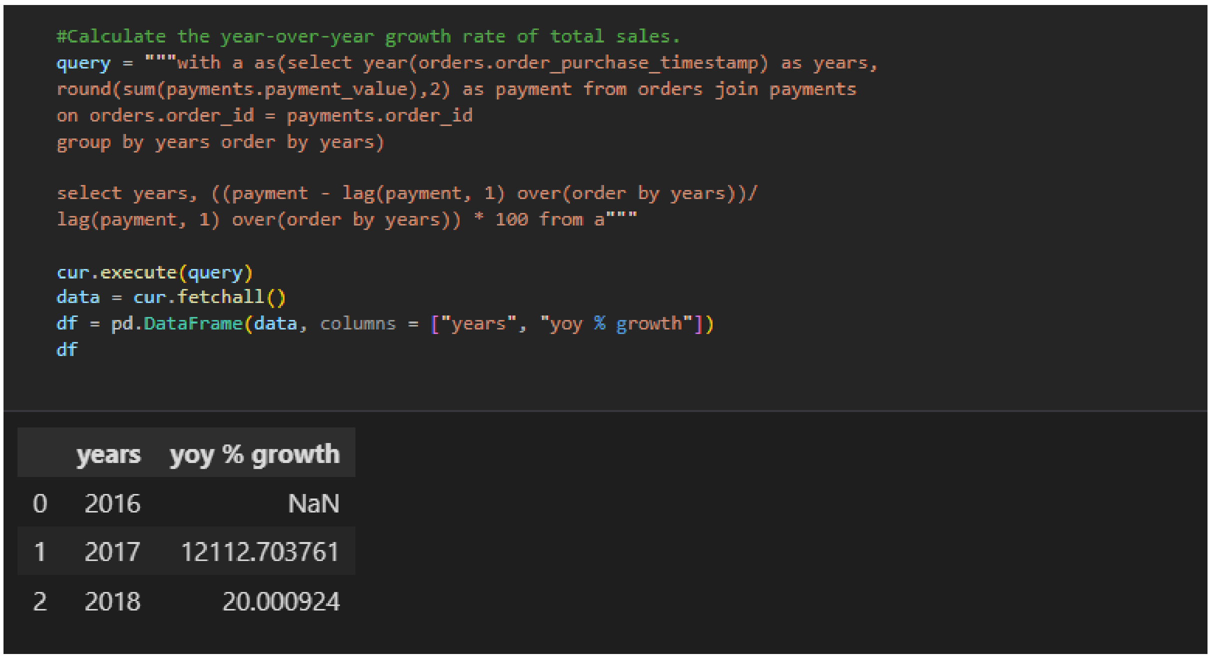
```
      0   1          2          3
0  2016   9     252.24     252.24
1  2016  10   59090.48   59342.72
2  2016  12      19.62   59362.34
3  2017   1  138488.04  197850.38
4  2017   2  291908.01  489758.39
```



```
#Calculate the cumulative sales per month for each year.
query = """select years, months , payment, sum(payment)
over(order by years, months) cumulative_sales from
(select year(orders.order_purchase_timestamp) as years,
month(orders.order_purchase_timestamp) as months,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by years, months order by years, months) as a
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data)
df.head()
```

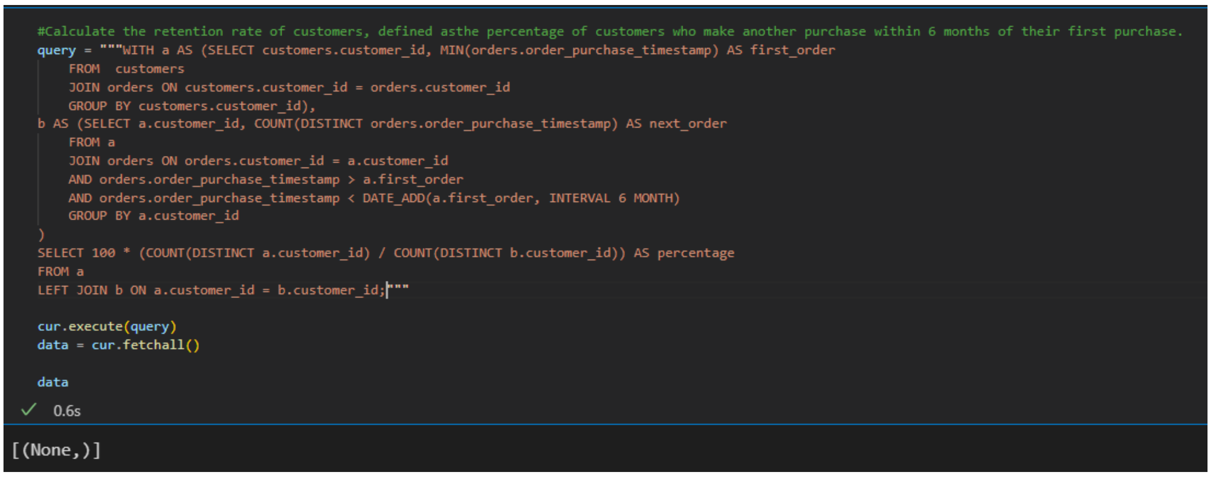
✓ 0.5s

```
      0   1          2          3
0  2016   9     252.24     252.24
1  2016  10   59090.48   59342.72
2  2016  12      19.62   59362.34
3  2017   1  138488.04  197850.38
4  2017   2  291908.01  489758.39
```



```
#Calculate the year-over-year growth rate of total sales.
query = """with a as(select year(orders.order_purchase_timestamp) as years,
round(sum(payments.payment_value),2) as payment from orders join payments
on orders.order_id = payments.order_id
group by years order by years)

select years, ((payment - lag(payment, 1) over(order by years))/
lag(payment, 1) over(order by years)) * 100 from a"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["years", "yoy % growth"])
df
```

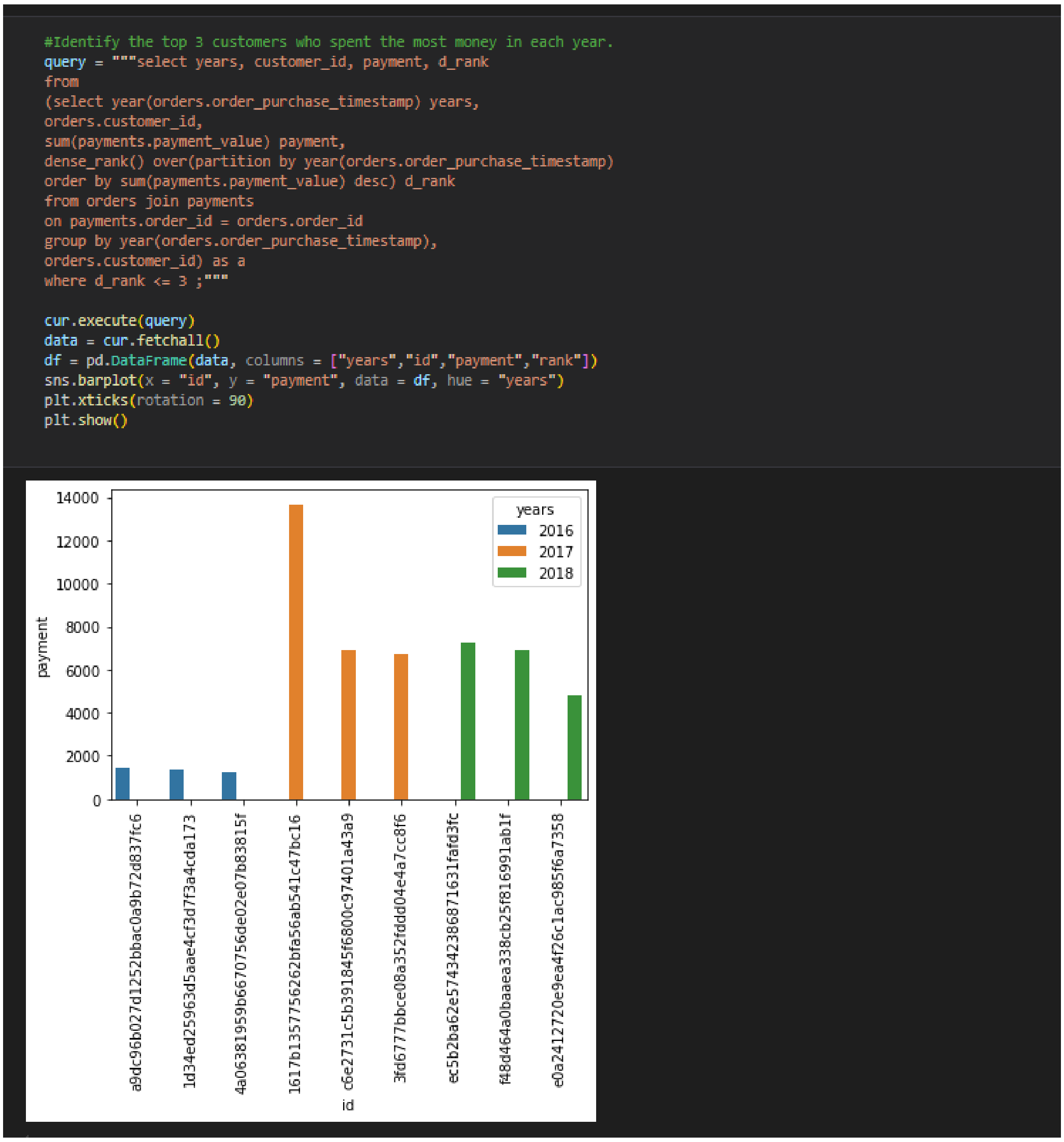
```
   years  yoy % growth
0   2016           NaN
1   2017  12112.703761
2   2018     20.000924
```



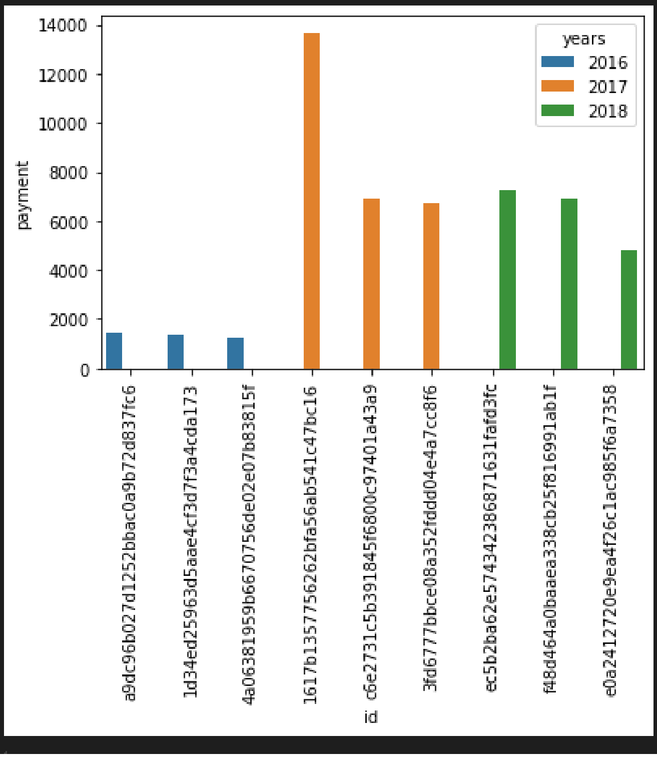
```
#Calculate the retention rate of customers, defined asthe percentage of customers who make another purchase within 6 months of their first purchase.
query = """WITH a AS (SELECT customers.customer_id, MIN(orders.order_purchase_timestamp) AS first_order
    FROM  customers
    JOIN orders ON customers.customer_id = orders.customer_id
    GROUP BY customers.customer_id),
b AS (SELECT a.customer_id, COUNT(DISTINCT orders.order_purchase_timestamp) AS next_order
    FROM a
    JOIN orders ON orders.customer_id = a.customer_id
    AND orders.order_purchase_timestamp > a.first_order
    AND orders.order_purchase_timestamp < DATE_ADD(a.first_order, INTERVAL 6 MONTH)
    GROUP BY a.customer_id
)
SELECT 100 * (COUNT(DISTINCT a.customer_id) / COUNT(DISTINCT b.customer_id)) AS percentage
FROM a
LEFT JOIN b ON a.customer_id = b.customer_id;"""

cur.execute(query)
data = cur.fetchall()

data
```

✓ 0.6s

```
[(None,)]
```



```
#Identify the top 3 customers who spent the most money in each year.
query = """select years, customer_id, payment, d_rank
from
(select year(orders.order_purchase_timestamp) years,
orders.customer_id,
sum(payments.payment_value) payment,
dense_rank() over(partition by year(orders.order_purchase_timestamp)
order by sum(payments.payment_value) desc) d_rank
from orders join payments
on payments.order_id = orders.order_id
group by year(orders.order_purchase_timestamp),
orders.customer_id) as a
where d_rank <= 3 ;"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["years","id","payment","rank"])
sns.barplot(x = "id", y = "payment", data = df, hue = "years")
plt.xticks(rotation = 90)
plt.show()
```


payment
years
a9dc96b027d1252bbac0a9b72d837fc6
1d34ed25963d5aae4cf3d7f3a4cda173
4a06381959b6670756de02e07b83815f
1617b1357756262bfa56ab541c47bc16
c6e2731c5b391845f6800c97401a43a9
3fd6777bbce08a352fddd04e4a7cc8f6
ec5b2ba62e574342386871631fafd3fc
f48d464a0baaea338cb25f816991ab1f
e0a2412720e9ea4f26c1ac985f6a7358
id