

```

import pandas as pd
import mysql.connector
import os

# List of CSV files and their corresponding table names
csv_files = [
    ('customers.csv', 'customers'),
    ('orders.csv', 'orders'),
    ('sellers.csv', 'sellers'),
    ('products.csv', 'products'),
    ('geolocation.csv', 'geolocation'),
    ('payments.csv', 'payments'),
    ('order_items.csv', 'order_items')# Added payments.csv for
specific handling
]

# Connect to the MySQL database
conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='Piyush@28',
    database='ecommerce'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'F:/E-commerce'

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()

```

```

Processing customers.csv
NaN values before replacement:
customer_id          0
customer_unique_id   0
customer_zip_code_prefix  0
customer_city         0
customer_state        0
dtype: int64

```

```

Processing orders.csv
NaN values before replacement:
order_id              0
customer_id           0
order_status          0
order_purchase_timestamp  0
order_approved_at     160
order_delivered_carrier_date  1783
order_delivered_customer_date  2965
order_estimated_delivery_date  0
dtype: int64

```

```

Processing sellers.csv

```

NaN values before replacement:

seller_id	0
seller_zip_code_prefix	0
seller_city	0
seller_state	0

dtype: int64

Processing products.csv

NaN values before replacement:

product_id	0
product_category	610
product_name_length	610
product_description_length	610
product_photos_qty	610
product_weight_g	2
product_length_cm	2
product_height_cm	2
product_width_cm	2

dtype: int64

Processing geolocation.csv

NaN values before replacement:

geolocation_zip_code_prefix	0
geolocation_lat	0
geolocation_lng	0
geolocation_city	0
geolocation_state	0

dtype: int64

Processing payments.csv

NaN values before replacement:

order_id	0
payment_sequential	0
payment_type	0
payment_installments	0
payment_value	0

dtype: int64

Processing order_items.csv

NaN values before replacement:

order_id	0
order_item_id	0
product_id	0
seller_id	0
shipping_limit_date	0
price	0
freight_value	0

dtype: int64

```

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mysql.connector
import numpy as np

db = mysql.connector.connect(host= "localhost",
                             username = "root",
                             password = "Piyush@28",
                             database = "ecommerce")

cur = db.cursor()

```

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()

```

	customer_city
0	franca
1	sao bernardo do campo
2	sao paulo
3	mogi das cruces
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()

"Total orders placed in 2017 are", data[0][0]

('Total orders placed in 2017 are', 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 x 2 columns]
```

Calculate the percentage of orders that were paid in installments.

```
query = """ select ((sum(case when payment_installments >= 1 then 1
else 0 end))/count(*))*100 from payments
"""
```

```
cur.execute(query)
```

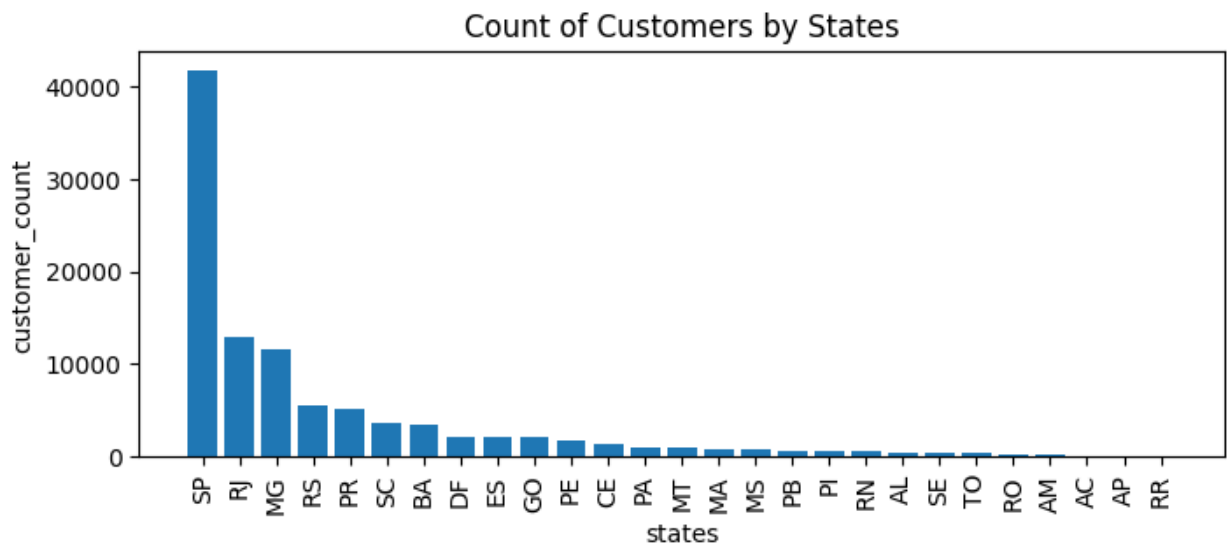
```
data = cur.fetchall()
```

```
"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 = (8,3))  
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.

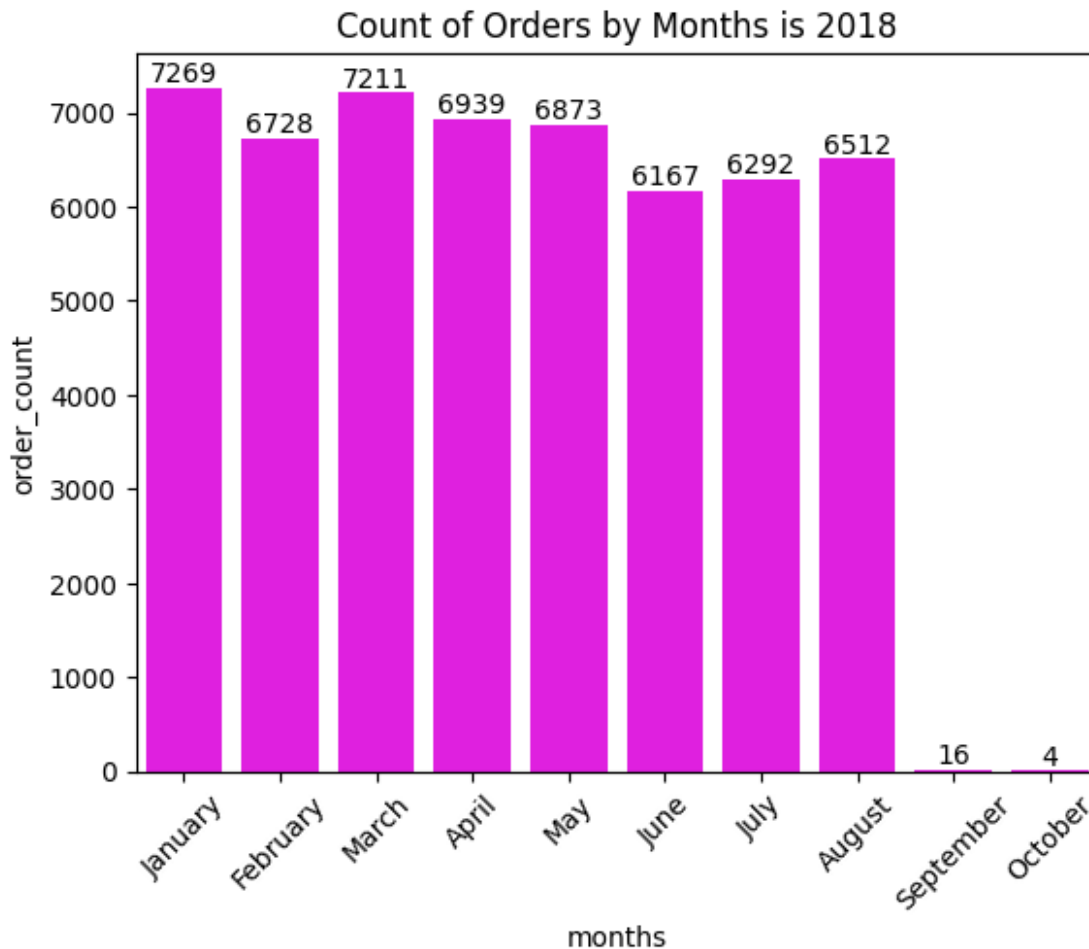
```
query = """ select monthname(order_purchase_timestamp) months,
count(order_id) order_count
from orders where year(order_purchase_timestamp) = 2018
group by months
"""

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 = "magenta")
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 per order, 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	padre carvalho	7.00
1	celso ramos	6.50
2	datas	6.00
3	candido godoi	6.00
4	matias olimpiao	5.00
5	cidelandia	4.00
6	picarra	4.00
7	morro de sao paulo	4.00
8	teixeira soares	4.00
9	curralinho	4.00

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

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"]

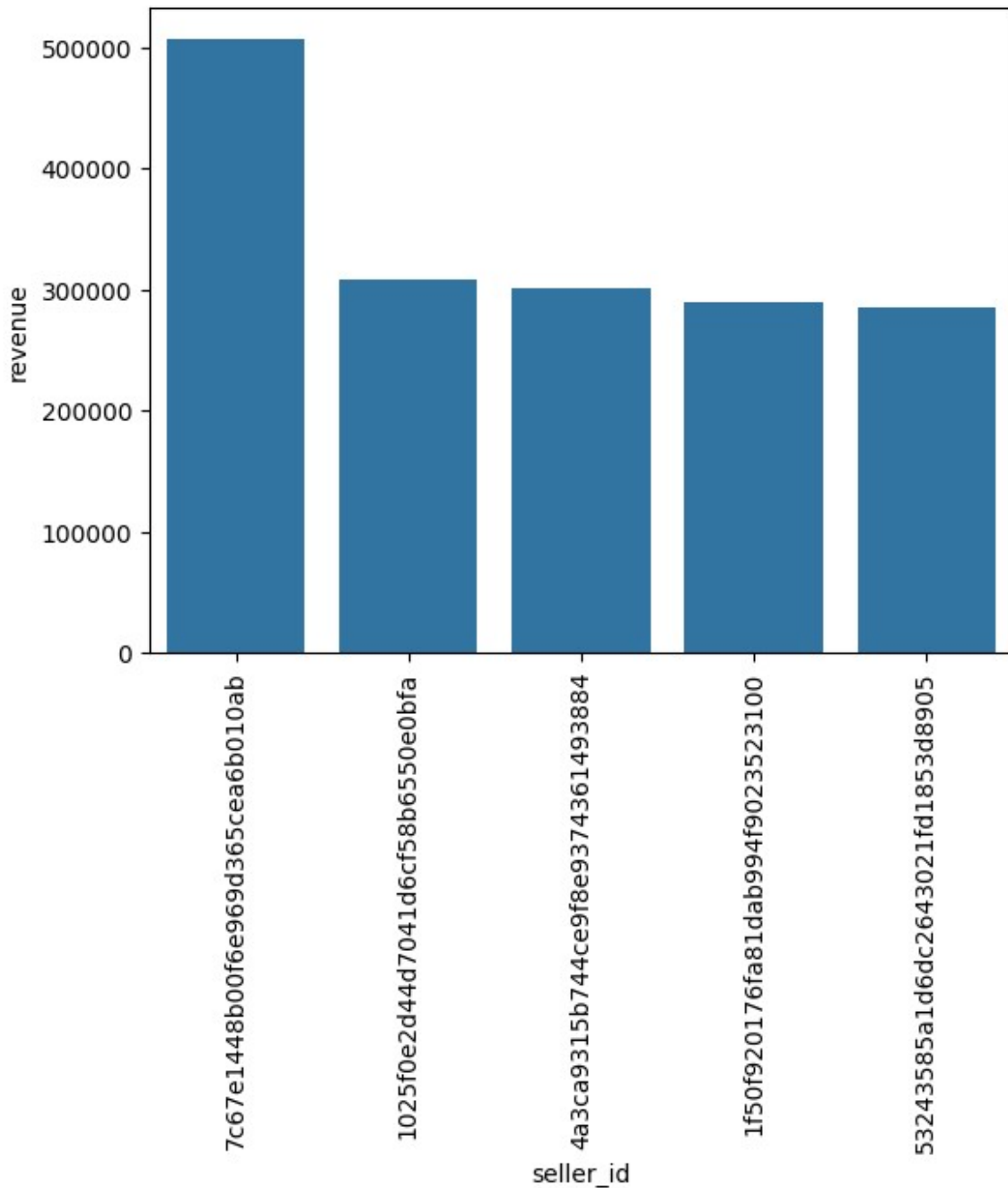
a = np.corrcoef([arr1, arr2])
print("the correlation is", a[0][-1])

the correlation is -0.10631514167157562
```

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 for each 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

```

	0	1	2
\			
0	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74
1	000161a058600d5901f007fab4c27140	2017-07-16 09:40:32	67.41
2	0001fd6190edaaf884bcaf3d49edf079	2017-02-28 11:06:43	195.42
3	0002414f95344307404f0ace7a26f1d5	2017-08-16 13:09:20	179.35
4	000379cdec625522490c315e70c7a9fb	2018-04-02 13:42:17	107.01
...
103881	fffec9f79fd8c764f843e9951b11341	2018-03-29 16:59:26	71.23
103882	fffeda5b6d849fbd39689bb92087f431	2018-05-22 13:36:02	63.13
103883	ffff42319e9b2d713724ae527742af25	2018-06-13 16:57:05	214.13
103884	ffffa3172527f765de70084a7e53aae8	2017-09-02 11:53:32	45.50
103885	ffffe8b65bbe3087b653a978c870db99	2017-09-29 14:07:03	18.37

	3
0	114.739998
1	67.410004
2	195.419998
3	179.350006
4	107.010002
...	...
103881	27.120001
103882	63.130001
103883	214.130005
103884	45.500000
103885	18.370001

[103886 rows x 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
```

	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
5	2017	3	449863.60	939621.99
6	2017	4	417788.03	1357410.02
7	2017	5	592918.82	1950328.84
8	2017	6	511276.38	2461605.22
9	2017	7	592382.92	3053988.14
10	2017	8	674396.32	3728384.46
11	2017	9	727762.45	4456146.91
12	2017	10	779677.88	5235824.79
13	2017	11	1194882.80	6430707.59
14	2017	12	878401.48	7309109.07
15	2018	1	1115004.18	8424113.25
16	2018	2	992463.34	9416576.59
17	2018	3	1159652.12	10576228.71
18	2018	4	1160785.48	11737014.19
19	2018	5	1153982.15	12890996.34
20	2018	6	1023880.50	13914876.84
21	2018	7	1066540.75	14981417.59
22	2018	8	1022425.32	16003842.91
23	2018	9	4439.54	16008282.45
24	2018	10	589.67	16008872.12

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 as the 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) 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) next_order
from a join orders
on orders.customer_id = a.customer_id
and orders.order_purchase_timestamp > first_order
and orders.order_purchase_timestamp <
date_add(first_order, interval 6 month)
group by a.customer_id)

select 100 * (count( distinct a.customer_id)/ count(distinct
```

```
b.customer_id))  
from a left join b  
on a.customer_id = b.customer_id ;"""
```

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

```
data
```

```
[(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()
```

