```
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'),
    ('sallers.csv', 'sallers'),
    ('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='17032004',
    database='ecommerce'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'C:/Users/ganes/OneDrive/Desktop/SQL+pyhton Project'

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()



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 = "17032004",
                             database = "ecommerce")
cur = db.cursor()
```

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
```

2. 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()

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

```
('data orders placed in 2017 are', 45101)
```

3. 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  1013477.32
1           FURNITURE DECORATION  2860352.78
2                      TELEPHONY   973764.10
3                 BED TABLE BATH  3425107.34
4                     AUTOMOTIVE  1704588.66
..                           ...         ...
69                CDS MUSIC DVDS     2398.86
70                    LA CUISINE     5827.06
71  FASHION CHILDREN'S CLOTHING      1571.34
72                    PC GAMER     4348.86
73       INSURANCE AND SERVICES      649.02

[74 rows x 2 columns]
```

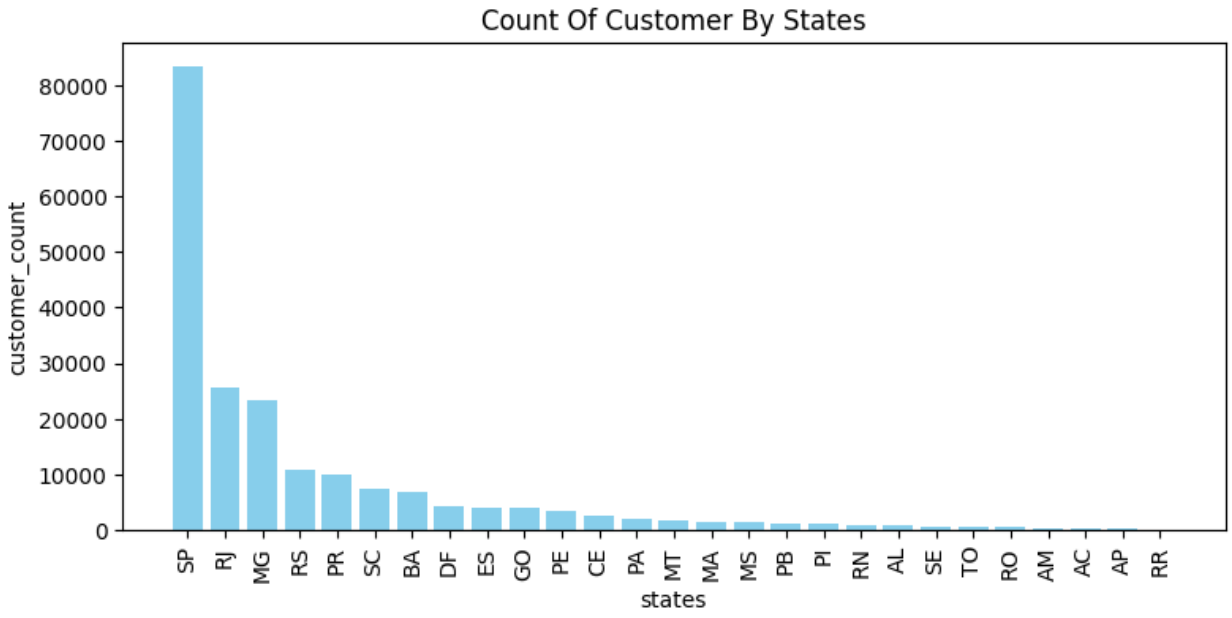
4. Calculate the percentage of orders that were paid in installments.

```
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'))
```

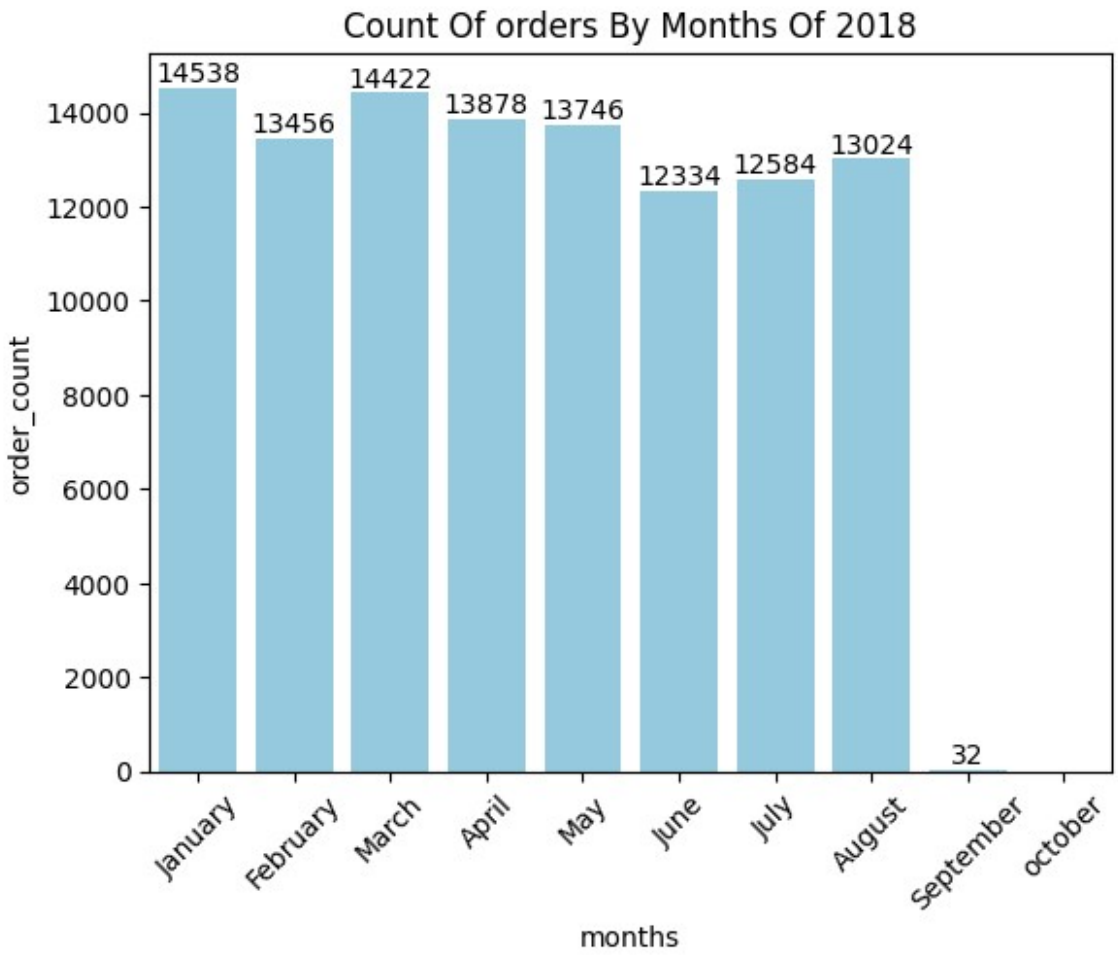
5. Count the number of customers from each state.

```
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 = (9,4))
plt.bar(df["state"], df["customer_count"])


plt.bar(df["state"], df["customer_count"], color='skyblue')


plt. xticks(rotation = 90)
plt.xlabel("states")
plt.ylabel("customer_count")
plt.title("Count Of Customer By States")
plt.show()
```



6. 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)
plt.xticks(rotation = 45)

ax.bar_label(ax.containers[0])
plt.title("Count Of orders By Months Of 2018")

ax = sns.barplot(x=df["months"], y=df["order_count"], data=df,
order=o, color='skyblue')




plt.show()
```



7. 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/Orders"])
df.head(10)
```

	Customer City	Average Products/Orders
0	padre carvalho	7.00
1	celso ramos	6.50
2	datas	6.00
3	candido godoi	6.00
4	matias olimpio	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

8. 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", "Sales Percentage"])
df
```

	Category	Sales Percentage
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
..	...	...

```
69               HOUSE COMFORT 2               0.01
70                CDS MUSIC DVDS               0.01
71                      PC GAMER               0.01
72  FASHION CHILDREN'S CLOTHING                  0.00
73       INSURANCE AND SERVICES                  0.00

[74 rows x 2 columns]
```

9. Identify the correlation between product price and the number of times a product has been purchased.

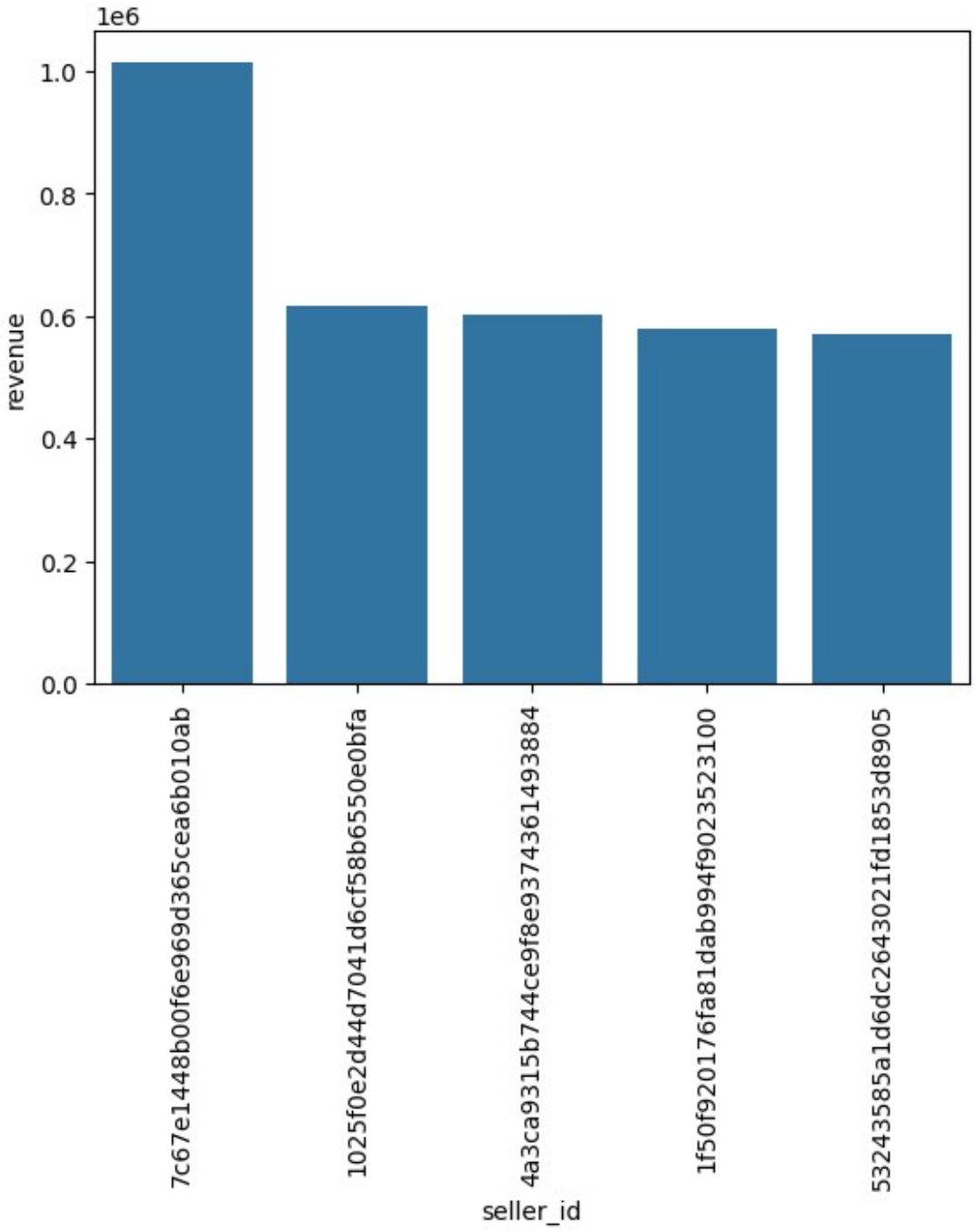
```
query = """select products.product_category,
count(order_items.product_id),
round(avg(order_items.price),2)
from products join order_items
on products.product_id = order_items.product_id
group by products.product_category;
"""


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 between price and number of times a product
has been puchased is" ,a[0][1])
```

```
The correlation between price and number of times a product has been
puchased is -0.10631514167157562
```

10. 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
```

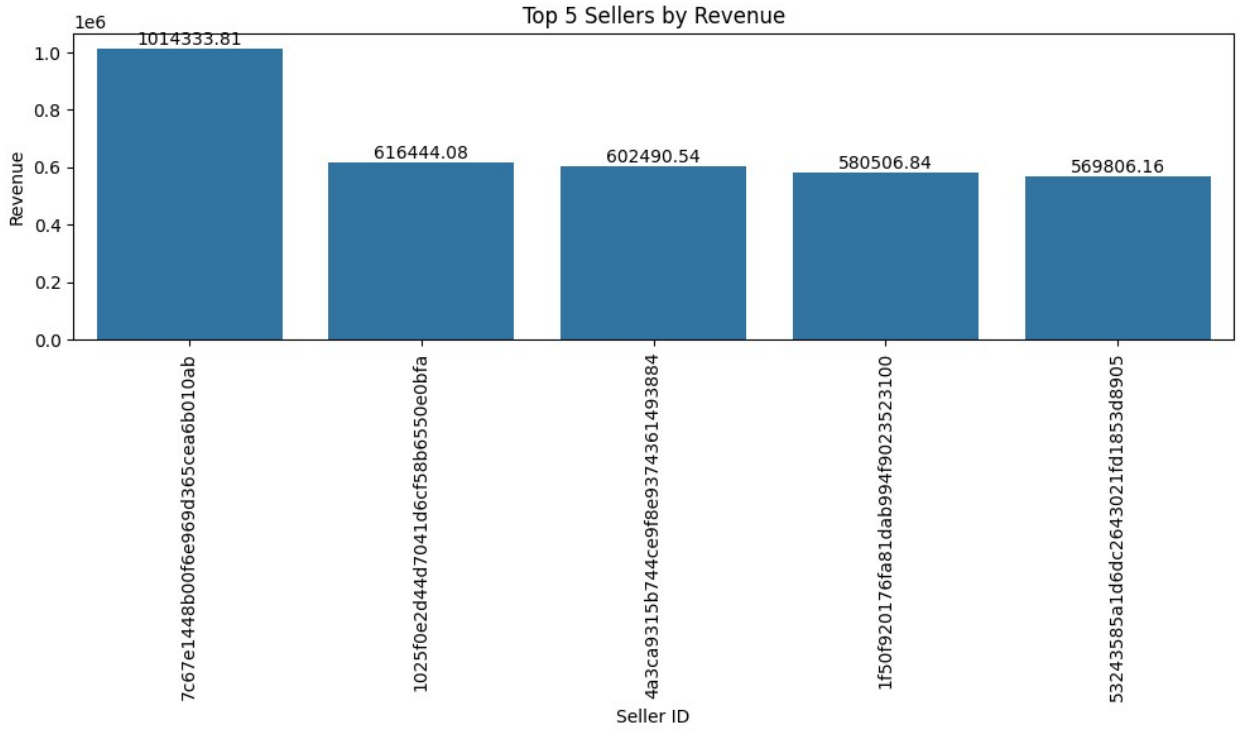
```
(select order_items.seller_id, sum(payments.payment_value)
revenue from order_items join payments
on order_items.order_id = payments.order_id
group by order_items.seller_id) as a
"""


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()
```



11. Calculate the total revenue generated by top f sellers and ranl them by revenue ¶

```
query = """
SELECT *, DENSE_RANK() OVER(ORDER BY revenue DESC) as rn
FROM (
```

```
    SELECT order_items.seller_id, SUM(payments.payment_value) AS
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()


plt.figure(figsize=(10, 6))
barplot = sns.barplot(x="seller_id", y="revenue", data=df)


for index, row in df.iterrows():
    barplot.text(index, row["revenue"], f'{row["revenue"]:.2f}',
color='black', ha="center", va="bottom")

plt.xticks(rotation=90)
plt.xlabel("Seller ID")
plt.ylabel("Revenue")
plt.title("Top 5 Sellers by Revenue")
plt.tight_layout()
plt.show()
```



12 1. 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, columns=["customerid", "timestamp", "price",
"movavg"])
df
```

```
                                customerid            timestamp   price
\
0       00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74

1       00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74

2       00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74

3       00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74

4       000161a058600d5901f007fab4c27140  2017-07-16 09:40:32   67.41

...                                  ...                  ...     ...

415539  ffffa3172527f765de70084a7e53aae8  2017-09-02 11:53:32   45.50

415540  ffffe8b65bbe3087b653a978c870db99  2017-09-29 14:07:03   18.37

415541  ffffe8b65bbe3087b653a978c870db99  2017-09-29 14:07:03   18.37

415542  ffffe8b65bbe3087b653a978c870db99  2017-09-29 14:07:03   18.37

415543  ffffe8b65bbe3087b653a978c870db99  2017-09-29 14:07:03   18.37


            movavg
0       114.739998
1       114.739998
2       114.739998
3       114.739998
4        67.410004
...            ...
415539   45.500000
415540   18.370001
415541   18.370001
415542   18.370001
415543   18.370001

[415544 rows x 4 columns]
```

13. 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	1008.96	1008.96
1	2016	10	236361.92	237370.88
2	2016	12	78.48	237449.36
3	2017	1	553952.16	791401.52
4	2017	2	1167632.04	1959033.56
5	2017	3	1799454.40	3758487.96
6	2017	4	1671152.12	5429640.08
7	2017	5	2371675.28	7801315.36
8	2017	6	2045105.52	9846420.88
9	2017	7	2369531.68	12215952.56
10	2017	8	2697585.28	14913537.84
11	2017	9	2911049.80	17824587.64
12	2017	10	3118711.52	20943299.16
13	2017	11	4779531.20	25722830.36
14	2017	12	3513605.92	29236436.28
15	2018	1	4460016.72	33696453.00
16	2018	2	3969853.36	37666306.36
17	2018	3	4638608.48	42304914.84
18	2018	4	4643141.92	46948056.76
19	2018	5	4615928.60	51563985.36
20	2018	6	4095522.00	55659507.36
21	2018	7	4266163.00	59925670.36
22	2018	8	4089701.29	64015371.65
23	2018	9	17758.16	64033129.81
24	2018	10	2358.68	64035488.49

14. 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	12112.703757
2	2018	20.000924

15. Calculate the retention rate of customers, defined as the percentage of customers who make another purchase within 6 months of their first purchase.

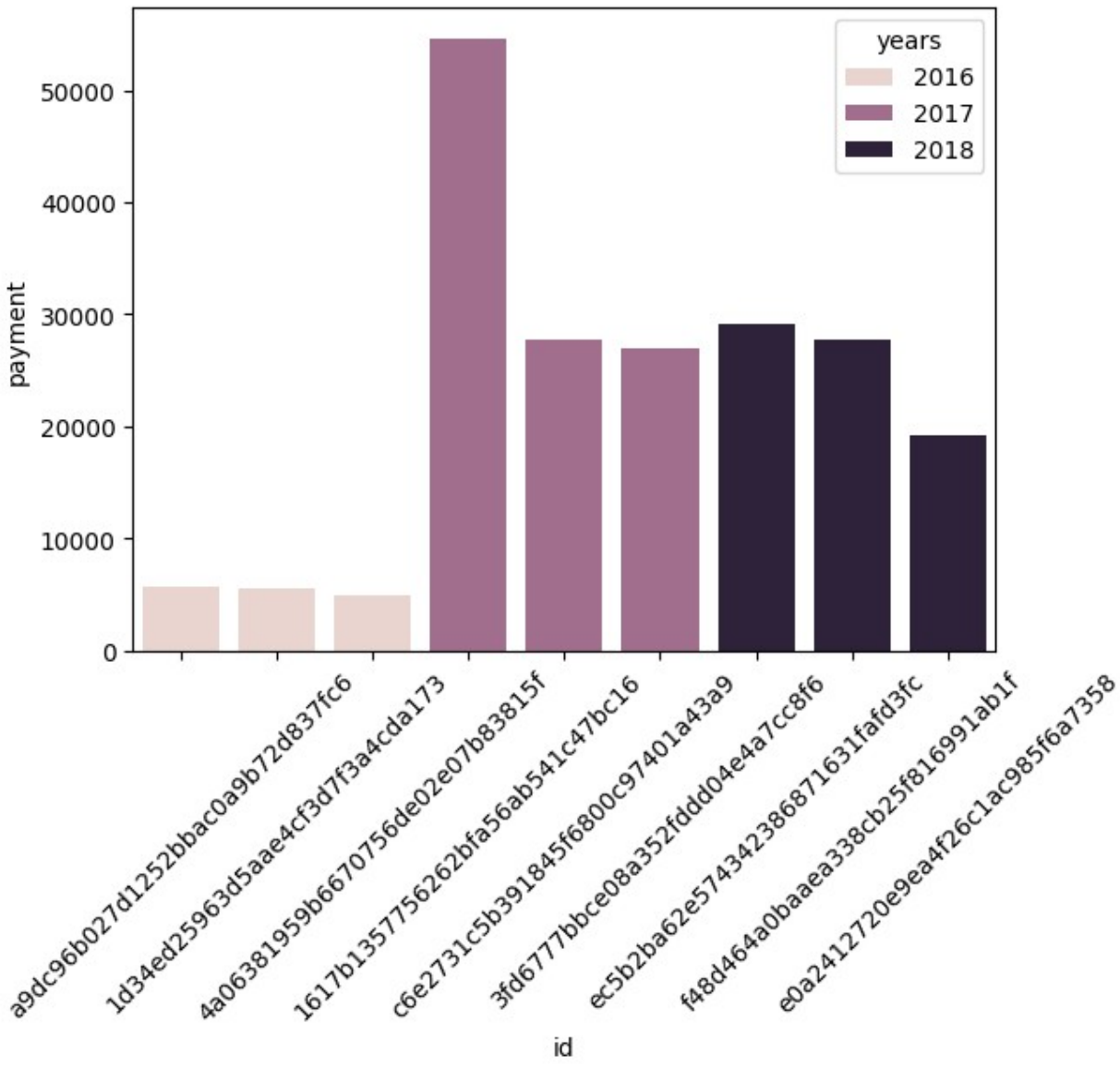
```
query = """ with a as (select customers.customer_id,
min(orders.order_purchase_timestamp) first_order
from customers join orders
on customers.customer_id = orders.customer_id
group by customers.customer_id),

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
# there is no customer who make another purchase within 6 months of
their first purchase.¶
```

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

16. 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 = 45)
plt.show()
```


years
2016
2017
2018
payment
50000
40000
30000
20000
10000
0
a9dc96b027d1252bbac0a9b72d837fc6
1d34ed25963d5aae4cf3d7f3a4cda173
4a06381959b6670756de02e07b83815f
1617b1357756262bfa56ab541c47bc16
c6e2731c5b391845f6800c97401a43a9
3fd6777bbce08a352fddd04e4a7cc8f6
ec5b2ba62e574342386871631fafd3fc
f48d464a0baaea338cb25f816991ab1f
e0a2412720e9ea4f26c1ac985f6a7358
id