In [ ]:
```python
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='pooja@12345',
    database='ecommerce'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'C:\\Users\\ABC\\Desktop\\python plus sql'

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 ({', '
        cursor.execute(sql, values)

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

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

In [ ]:
```
! pip install pandas
```

In [ ]:
```
!pip install mysql-connector-python
```

In [ ]:
```
!pip install matplotlib
!pip install seaborn
```

In [2]:
```python
import pandas as pd
```

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


db = mysql.connector.connect(host = "localhost",
                             username = "root",
                             password = "pooja@12345",
                             database= "ecommerce")

cur =db.cursor()
```

List all unique cities where customers are located.

In [3]:
```
query = """ select distinct customer_city from  customers"""

cur.execute(query)

data = cur.fetchall()


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

Out[3]:

	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.

In [5]:
```
query = """select count(order_id) from orders where year(order_purchase_timestamp)= 2017"""

cur.execute(query)

data = cur.fetchall()


"total orders plased in 2017 are" ,data[0][0]
```

Out[5]:
```
('total orders plased in 2017 are', 135303)
```

Find the total sales per category.

In [ ]:
```
query = """select upper(products.product_category) catagory,
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 catagory
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data,columns= ["catagory","sales"])
df
```

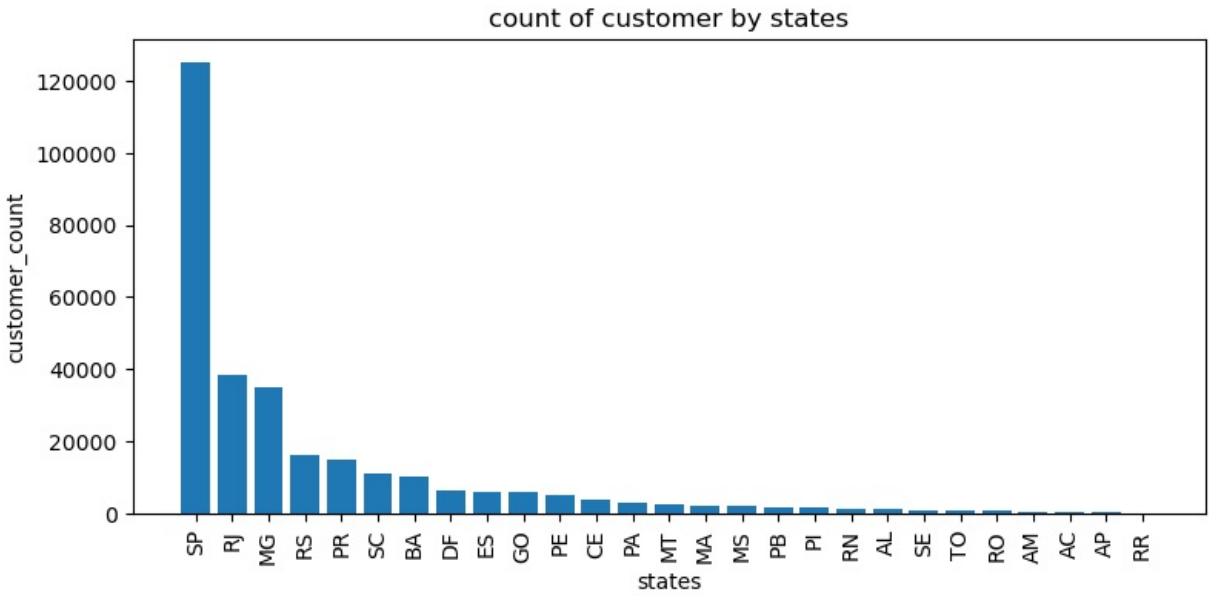
Calculate the percentage of orders that were paid in installments.

In [15]:
```
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]
```

Out[15]:
```
('the percentage of orders that were paid in installments is',
 Decimal('99.9981'))
```

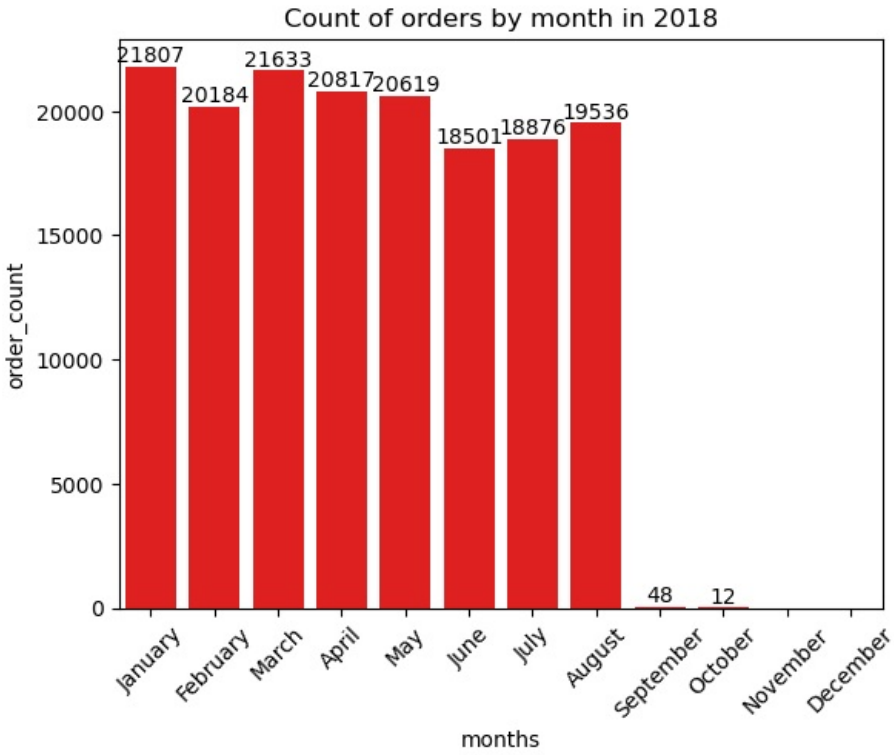
Count the number of customers from each state.

In [16]:
```
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.xticks(rotation = 90)
plt.xlabel("states")
plt.ylabel("customer_count")
plt.title("count of customer by states")
plt.show()
```



Calculate the number of orders per month in 2018.

In [17]:
```
query = """
select monthname(order_purchase_timestamp) as months,
       count(order_id) as 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","November","December"]

ax = sns.barplot(x="months", y="order_count", data=df, order=o,color = "r")
plt.xticks(rotation=45)
ax.bar_label(ax.containers[0])
plt.title("Count of orders by month in 2018")
plt.show()
```



Find the average number of products per order, grouped by customer city.

In [18]:
```
import pandas as pd

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)
```

Out[18]:

	customer city	average products/order
0	padre carvalho	63.00
1	celso ramos	58.50
2	datas	54.00
3	candido godoi	54.00
4	matias olimpio	45.00
5	cidelandia	36.00
6	curralinho	36.00
7	picarra	36.00
8	morro de sao paulo	36.00
9	teixeira soares	36.00

Calculate the percentage of total revenue contributed by each product category.

In [19]:
```
query= """select upper(products.product_category) catagory,
round((sum(payments.payment_value) /(select sum(payment_value) from payments))*100,2)
 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 catagory  order  by sales desc
"""

cur.execute(query)

df = pd.DataFrame(data,columns = ["category","percentage distribution"])
df.head(5)
```

Out[19]:

	category	percentage distribution
0	padre carvalho	63.00
1	celso ramos	58.50
2	datas	54.00
3	candido godoi	54.00
4	matias olimpio	45.00

In [ ]:

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

In [21]:

```
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 = "pooja@12345",
                             database= "ecommerce")

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 between price andnumber of times has been purchased is",a[0][-1])
```

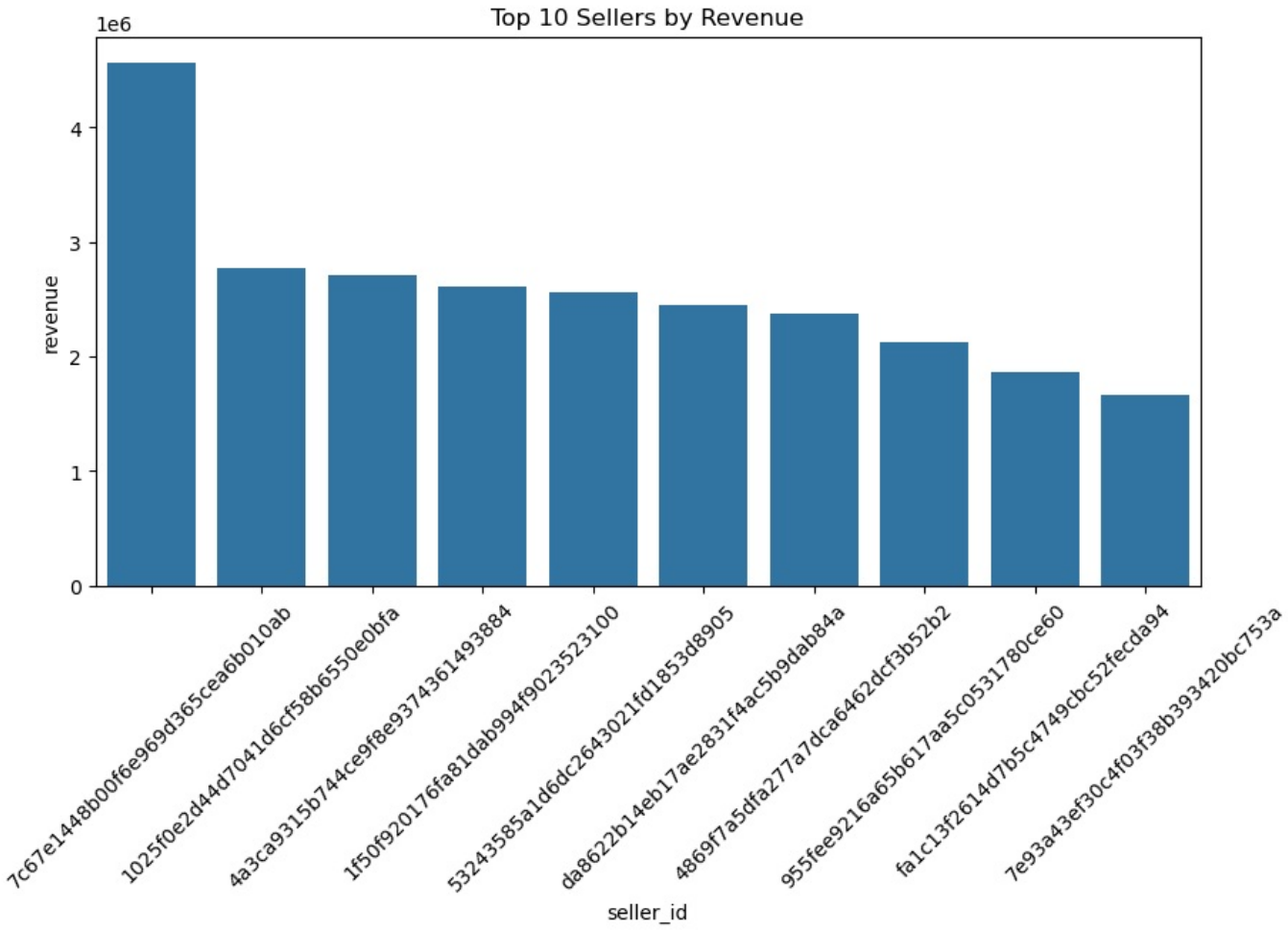
```
the correlation between price andnumber of times has been purchased is -0.10631514167157556
```

Calculate the total revenue generated by each seller, and rank them by revenue.

In [22]:

```
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"])

# Sort by revenue and take top 10 sellers
top_10 = df.sort_values("revenue", ascending=False).head(10)

plt.figure(figsize=(10,5))
```

```
sns.barplot(x="seller_id", y="revenue", data=top_10)

plt.title("Top 10 Sellers by Revenue")
plt.xticks(rotation=45)
plt.show()
```



Calculate the moving average of order values for each customer over their order history.

In [24]:
```
query = """
SELECT
    customer_id,
    order_purchase_timestamp,
    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
ORDER BY customer_id, order_purchase_timestamp;
"""

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns=["customer_id", "order_purchase_timestamp", "mov_avg"])
df.head()
```

Out[24]:

	customer_id	order_purchase_timestamp	mov_avg
0	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.739998
1	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.739998
2	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.739998
3	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.739998
4	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.739998

Calculate the cumulative sales per month for each year.

In [25]:

```
query = """ select years,months,payment,sum(payment)
over(order by years,months)  as 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
```

Out[25]:

	0	1	2	3
0	2016	9	2270.16	2.270160e+03
1	2016	10	531814.32	5.340845e+05
2	2016	12	176.58	5.342611e+05
3	2017	1	1246392.36	1.780653e+06
4	2017	2	2627172.09	4.407826e+06
5	2017	3	4048772.40	8.456598e+06
6	2017	4	3760092.27	1.221669e+07
7	2017	5	5336269.38	1.755296e+07
8	2017	6	4601487.42	2.215445e+07
9	2017	7	5331446.28	2.748589e+07
10	2017	8	6069566.88	3.355546e+07
11	2017	9	6549862.05	4.010532e+07
12	2017	10	7017100.92	4.712242e+07
13	2017	11	10753945.20	5.787637e+07
14	2017	12	7905613.32	6.578198e+07
15	2018	1	10035037.61	7.581702e+07
16	2018	2	8932170.06	8.474919e+07
17	2018	3	10436869.08	9.518606e+07
18	2018	4	10447069.33	1.056331e+08
19	2018	5	10385839.36	1.160190e+08
20	2018	6	9214924.51	1.252339e+08
21	2018	7	9598866.74	1.348328e+08
22	2018	8	9201827.89	1.440346e+08
23	2018	9	39955.86	1.440745e+08
24	2018	10	5307.03	1.440798e+08

Calculate the year-over-year growth rate of total sales.

In [26]:

```
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 = "pooja@12345",
                             database= "ecommerce")

cur =db.cursor()

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 = ["year","yoy % growth"])
df
```

Out[26]:

	year	yoy % growth
0	2016	NaN
1	2017	12112.703758
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.

In [27]:

```
 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
```

Out[27]:

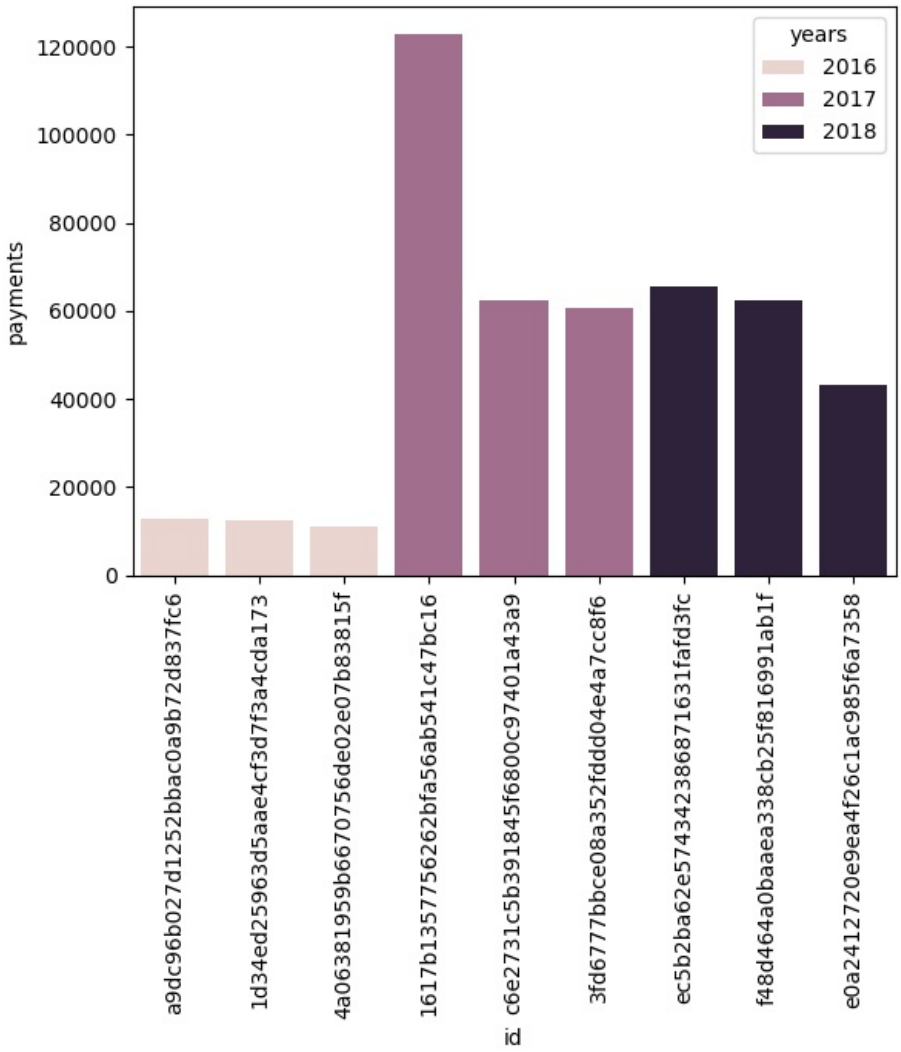
```
[(None,)]
```

Identify the top 3 customers who spent the most money in each year.

In [29]:

```
query = """
select  years,customer_id,payment,d_rank
from
(select
    year(o.order_purchase_timestamp) as years,
    o.customer_id,
    sum(p.payment_value) as payment,
    dense_rank() over (
        partition by year(o.order_purchase_timestamp)
        order by sum(p.payment_value) desc
    ) as d_rank
from orders o
join payments p
    on p.order_id = o.order_id
group by
    year(o.order_purchase_timestamp),
    o.customer_id) as a
    where d_rank<=3
    """
```

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



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: