SQL_PYTHON_ECOMMERCE_PROJECT

In [5]:
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import mysql.connector
import numpy as np


db = mysql.connector.connect(
    host='localhost',
    user='root',
    password='12345',
    database='ecommerce'
)
cursor = db.cursor()
```

BASIC

List all unique cities where customers are located.

In [6]:
```
query = '''select distinct customer_city from customers'''
cursor.execute(query)
c = cursor.fetchall()
c
```

Out[6]:
```
[('franca',),
 ('sao bernardo do campo',),
 ('sao paulo',),
 ('mogi das cruzes',),
 ('campinas',),
 ('jaragua do sul',),
 ('timoteo',),
 ('curitiba',),
 ('belo horizonte',),
 ('montes claros',),
 ('rio de janeiro',),
 ('lencois paulista',),
 ('caxias do sul',),
 ('piracicaba',),
 ('guarulhos',),
 ('pacaja',),
 ('florianopolis',),
 ('aparecida de goiania',),
 ('santo andre',),
```

Count the number of orders placed in 2017.

In [7]:
```
 query = '''select count(order_id) from orders where year(order_purchase_timestamp) = 2017'''
cursor.execute(query)
c = cursor.fetchall()
"total is ",c
```

Out[7]:
```
('total is ', [(225505,)])
```

Find the total sales per category.

In [8]:
```
 query = '''select upper(products.product_category )as pcat,
 round(sum(payments.payment_value),1) as pval
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 pcat '''

cursor.execute(query)

c = cursor.fetchall()
c

df=pd.DataFrame(c)
df.head(11)
```

Out[8]:

	0	1
0	ART	309929.3
1	COOL STUFF	7796980.0
2	GAMES CONSOLES	1954803.8
3	TELEPHONY	4868820.5
4	SPORT LEISURE	13921275.6
5	PERFUMERY	5067386.6
6	TOYS	6190376.9
7	COMPUTER ACCESSORIES	15853304.5
8	BABIES	5398456.6
9	BED TABLE BATH	17125536.7
10	FURNITURE DECORATION	14301763.9

Calculate the percentage of orders that were paid in installments.

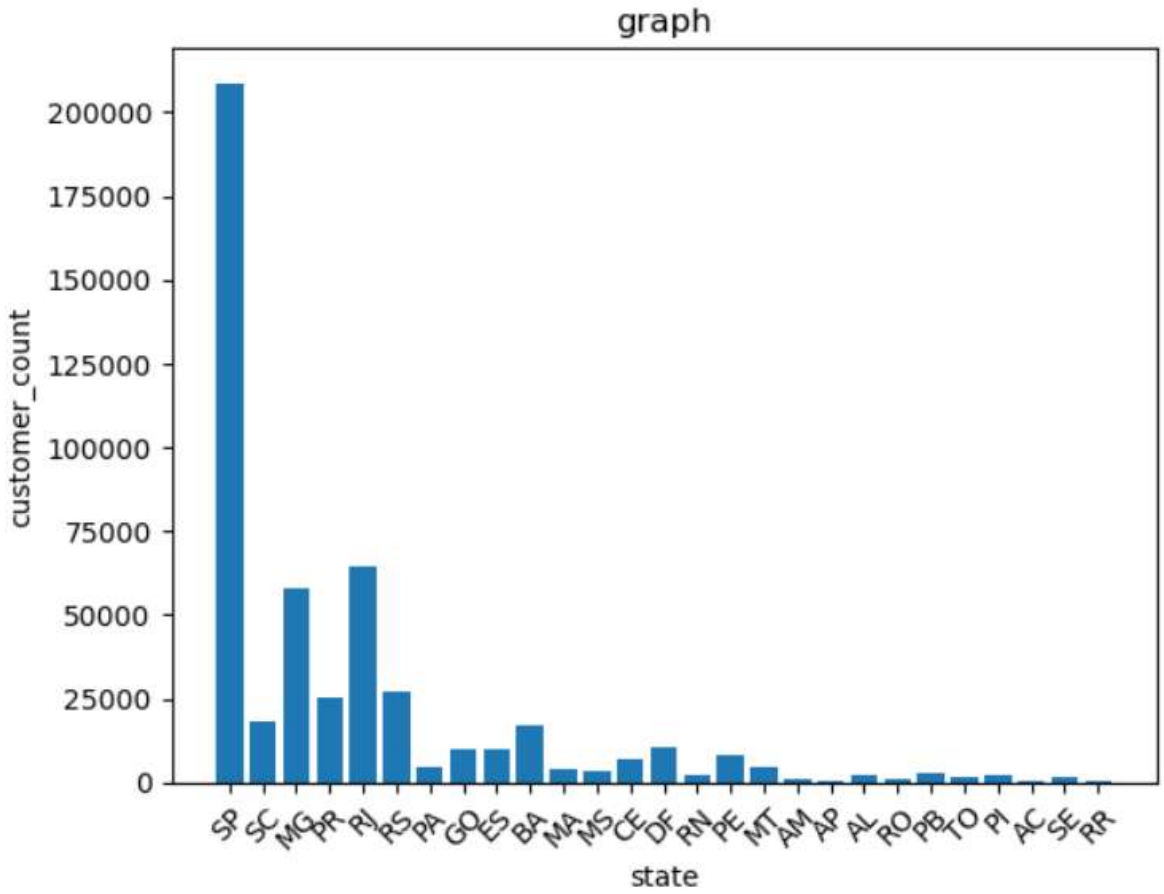
In [9]:
```
query = '''select (sum(case when payment_installments >=1 then 1
else 0 end))/count(*)*100 from payments '''

cursor.execute(query)

c = cursor.fetchall()
c
```

Out[9]:
```
[(Decimal('99.9981'),)]
```

Count the number of customers from each state.

In [10]:
```
query = '''select customer_state,count(customer_id) from
customers group by customer_state'''

cursor.execute(query)
c = cursor.fetchall()
c

df=pd.DataFrame(c,columns=['state','customer_count'])
plt.xlabel("state")
plt.ylabel("customer_count")
plt.title("graph")
plt.bar(df['state'],df['customer_count'])
plt.xticks(rotation=45)
df=df.sort_values(by='customer_count',ascending= False)
plt.figure(figsize = (9,7))


plt.show()
```

```
df=df.sort_values(by= 'customer_count' ,ascending= False)
plt.figure(figsize = (9,7))


plt.show()
```

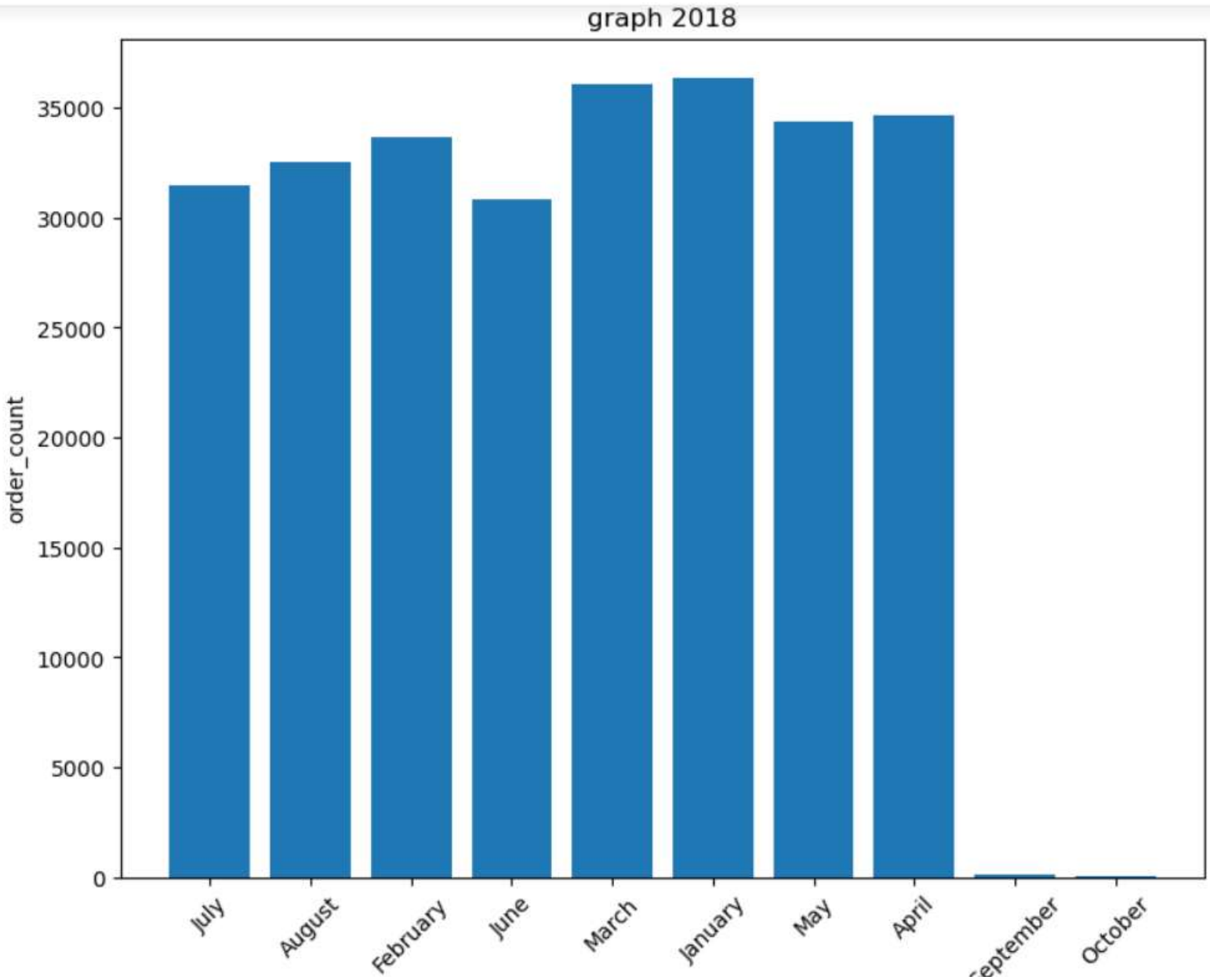



```
<Figure size 900x700 with 0 Axes>
```

INTERMEDIATE

Calculate the number of orders per month in 2018.

In [11]:
```
query = '''select monthname(order_purchase_timestamp) as month, count(order_id) as number_of_orders
from orders
where year(order_purchase_timestamp)= 2018 group by month '''

cursor.execute(query)
c = cursor.fetchall()
df=pd.DataFrame(c,columns=['month','order_count'])

plt.figure(figsize = (9,7))
plt.xticks(rotation=45)
plt.xlabel("month")
plt.ylabel("order_count")
plt.title("graph 2018")
plt.bar(df['month'],df['order_count'])
plt.show()
```


graph 2018
order_count
35000
30000
25000
20000
15000
10000
5000
0
July
August
February
June
March
January
May
April
September
October

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

In [12]:
```
query = '''with cpo as (SELECT orders.order_id,orders.customer_id ,
count(order_items.order_id) as cof 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(cpo.cof),2)
 from customers
 join cpo on customers.customer_id = cpo.customer_id
 group by customers.customer_city '''

cursor.execute(query)
c = cursor.fetchall()
df=pd.DataFrame(c,columns=["customer_city","average_orders"])
df
```

Out[12]:

	customer_city	average_orders
0	treze tilias	6.36
1	indaial	5.58
2	sao jose dos campos	5.69
3	sao paulo	5.78
4	porto alegre	5.87
...	...	...
4105	tibau do sul	5.00
4106	sao mamede	5.00
4107	guairaca	10.00
4108	sambaiba	5.00
4109	japaratuba	5.00

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

In [13]:
```
query = '''select upper(products.product_category )as pcat,
 round((sum(payments.payment_value)/(select sum(payment_value) from payments))*100,2)  as pval
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 pcat
order by pval desc'''

cursor.execute(query)

c = cursor.fetchall()
c
df=pd.DataFrame(c,columns = ["category", " sales %"])
df.head(11)
```

Out[13]:

Out[13]:

	category	sales %
0	BED TABLE BATH	53.49
1	HEALTH BEAUTY	51.76
2	COMPUTER ACCESSORIES	49.51
3	FURNITURE DECORATION	44.67
4	WATCHES PRESENT	44.64
5	SPORT LEISURE	43.48
6	HOUSEWARES	34.19
7	AUTOMOTIVE	26.62
8	GARDEN TOOLS	26.18
9	COOL STUFF	24.35
10	FURNITURE OFFICE	20.20

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

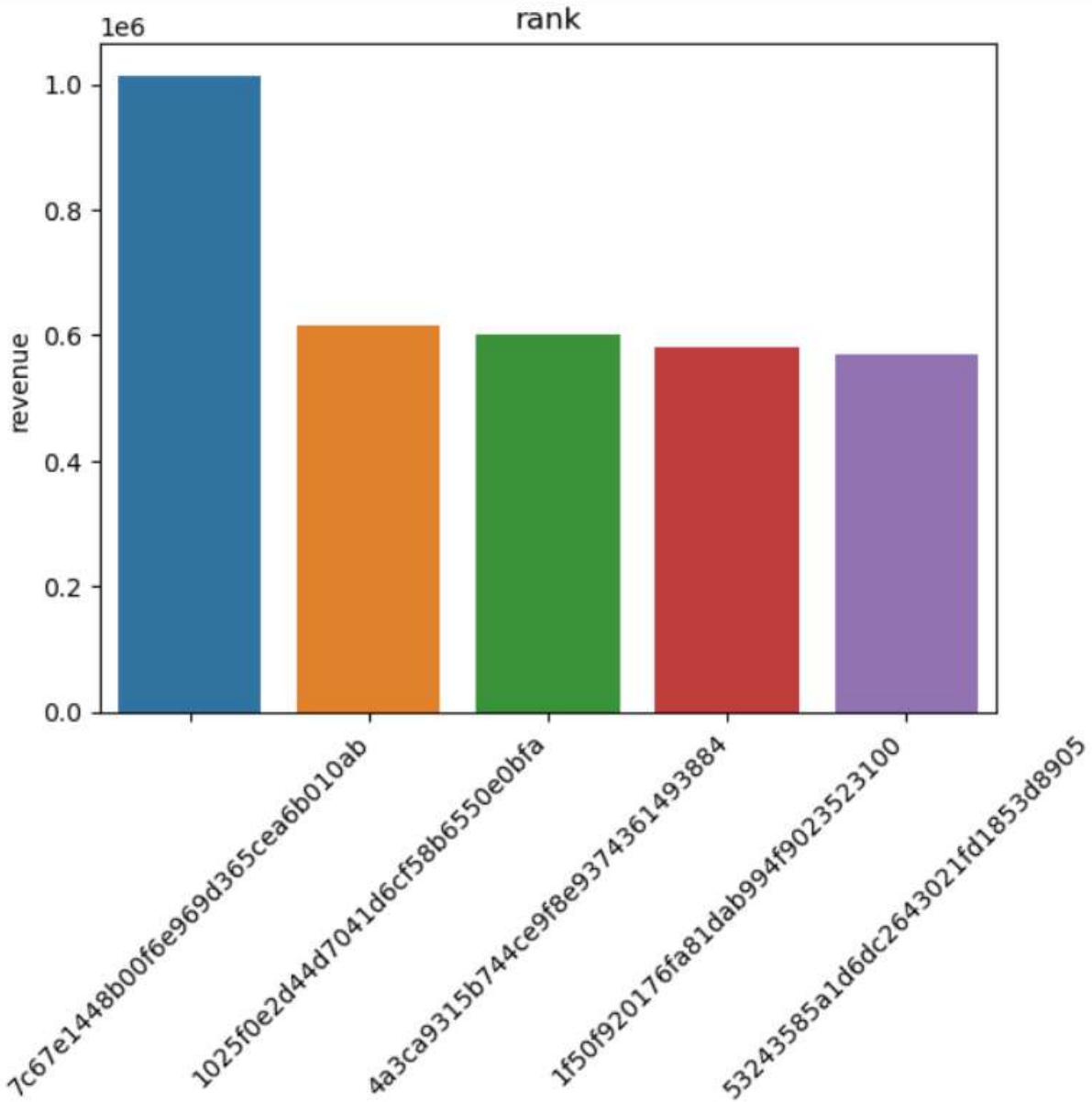
Identify the correlation between product price and the number of times a product has been purchased.

In [14]:
```
query = '''select products.product_category,count(order_items.product_id),round(sum(order_items.price),0)
from products join order_items on products.product_id = order_items.product_id
group by products.product_category '''

cursor.execute(query)

c = cursor.fetchall()
c
df=pd.DataFrame(c,columns = ["catg", " order_count ","price"])
df
a = df[" order_count "]
b = df["price"]
np.corrcoef([a,b])
```

Out[14]:
```
array([[1.        , 0.95033015],
       [0.95033015, 1.        ]])
```

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

In [15]:
```
query = '''select * , dense_rank() over(order by revenue desc) as t 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 f;'''

cursor.execute(query)

c = cursor.fetchall()

df=pd.DataFrame(c,columns = ["seller_id", " revenue ","rank"])
df=df.head(5)
sns.barplot(x = "seller_id", y = " revenue ", data = df)
plt.xticks(rotation= 45)
plt.xlabel("seller_id")
plt.ylabel("revenue")
plt.title("rank")
plt.show()
```


rank
1e6
1.0
0.8
0.6
0.4
0.2
0.0
revenue
7c67e1448b00f6e969d365cea6b010ab
1025f0e2d44d7041d6cf58b6550e0bfa
4a3ca9315b744ce9f8e9374361493884
1f50f920176fa81dab994f9023523100
53243585a1d6dc2643021fd1853d8905

ADVANCE

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

In [16]:
```
query = '''select customer_id, order_purchase_timestamp,
avg(payments) over(partition by customer_id order by order_purchase_timestamp
rows between 2 preceding and current row)
as moving_avg from

(select orders.customer_id,orders.order_purchase_timestamp,
payments.payment_value as payments
from payments join orders on payments.order_id = orders.order_id) as a '''

cursor.execute(query)

c = cursor.fetchall()
df=pd.DataFrame(c,columns = ["customer_id", "order_purchase_timestamp","moving_avg"])
df
```

Out[16]:

	customer_id	order_purchase_timestamp	moving_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

Out[16]:

	customer_id	order_purchase_timestamp	moving_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
...	...	...	...
1038855	ffffe8b65bbe3087b653a978c870db99	2017-09-29 14:07:03	18.370001
1038856	ffffe8b65bbe3087b653a978c870db99	2017-09-29 14:07:03	18.370001
1038857	ffffe8b65bbe3087b653a978c870db99	2017-09-29 14:07:03	18.370001
1038858	ffffe8b65bbe3087b653a978c870db99	2017-09-29 14:07:03	18.370001
1038859	ffffe8b65bbe3087b653a978c870db99	2017-09-29 14:07:03	18.370001

1038860 rows × 3 columns

Calculate the cumulative sales per month for each year.

In [17]:
```
query = '''select years,months,payment,sum(payment)    over (order by years,months)   as cumulative_sale 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 months,years
order by years,months) as a'''

cursor.execute(query)
c = cursor.fetchall()
df=pd.DataFrame(c,columns = ["years", "months","payments","cumulative_sales"])
df
```

Out[17]:

	years	months	payments	cumulative_sales
0	2016	9	2522.40	2.522400e+03
1	2016	10	590904.80	5.934272e+05
2	2016	12	196.20	5.936234e+05
3	2017	1	1384880.40	1.978504e+06
4	2017	2	2919080.10	4.897584e+06
5	2017	3	4498636.00	9.396220e+06
6	2017	4	4177880.29	1.357410e+07
7	2017	5	5929188.20	1.950329e+07
8	2017	6	5112763.80	2.461605e+07
9	2017	7	5923829.19	3.053988e+07
10	2017	8	6743963.20	3.728384e+07

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

In [18]:

```
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


'''


cursor.execute(query)
c = cursor.fetchall()
df=pd.DataFrame(c,columns = ["years","year by growth"])
df
```

Out[18]:

	years	year by growth
0	2016	NaN
1	2017	12112.703758
2	2018	20.000924

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

In [19]:
```
query = ''' select years,customer_id,payment,ranks from
(select year(orders.order_purchase_timestamp) as years, orders.customer_id,
sum(payments.payment_value) as payment,
dense_rank() over(partition by year (orders.order_purchase_timestamp) order by sum(payments.payment_value) desc) as ranks
from orders
join payments on payments.order_id = orders.order_id
group by year(orders.order_purchase_timestamp),orders.customer_id ) as a where ranks<=3

'''


cursor.execute(query)
c = cursor.fetchall()
df=pd.DataFrame(c,columns = ["years","customer_id","payment","ranks"])
df
```

Out[19]:

	years	customer_id	payment	ranks
0	2016	a9dc96b027d1252bbac0a9b72d837fc6	14235.500488	1
1	2016	1d34ed25963d5aae4cf3d7f3a4cda173	14007.399902	2
2	2016	4a06381959b6670756de02e07b83815f	12277.800293	3
3	2017	1617b1357756262bfa56ab541c47bc16	136640.800781	1
4	2017	c6e2731c5b391845f6800c97401a43a9	69293.100586	2
5	2017	3fd6777bbce08a352fddd04e4a7cc8f6	67266.601562	3
6	2018	ec5b2ba62e574342386871631fafd3fc	72748.798828	1
7	2018	f48d464a0baaea338cb25f816991ab1f	69222.099609	2
8	2018	e0a2412720e9ea4f26c1ac985f6a7358	48094.399414	3