MySQL Install:

```
pip install mysql-connector-python
```

packages:

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

Connect to MYSQL Data Source and import MYSQL:

```
# 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 comma between these
entries
]

# Connect to the MySQL database
conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='12345@',
    database='ecommerce'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'C:/Users/lenovo/Desktop/E-Commerce project
sep,2024/archive'

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
```

Connect Database.

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

cursor = db.cursor()
```

1. List all unique cities where customers are located.

```
query = """SELECT DISTINCT customer_city FROM customers"""

cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns =['City'])
df
```

```
                        City
0                     franca
1     sao bernardo do campo
2                  sao paulo
3            mogi das cruzes
4                   campinas
...                       ...
4114                  siriji
4115    natividade da serra
4116            monte bonito
4117              sao rafael
4118      eugenio de castro

[4119 rows x 1 columns]
```

2. Count the number of orders placed in 2017.

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

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
'Total order purchesd in 2017 ->', data[0][0]
```

```
('Total order purchesd in 2017 ->', 90202)
```

3. Find the total sales per category.

```
query = """ select upper(products.product_category) categorey ,
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 categorey
"""

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

df = pd.DataFrame(data, columns = ['Categorey', 'Sales'])
df
```

	Categorey	Sales
0	PERFUMERY	4053909.28
1	FURNITURE DECORATION	11441411.13
2	TELEPHONY	3895056.41
3	BED TABLE BATH	13700429.37
4	AUTOMOTIVE	6818354.65
..	...	...
69	CDS MUSIC DVDS	9595.44
70	LA CUISINE	23308.24
71	FASHION CHILDREN'S CLOTHING	6285.36
72	PC GAMER	17395.44
73	INSURANCE AND SERVICES	2596.08

```
[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 """

cursor.fetchall()
```

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

```
[(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 """

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns = ['State', 'Cust_Count'])
df
```

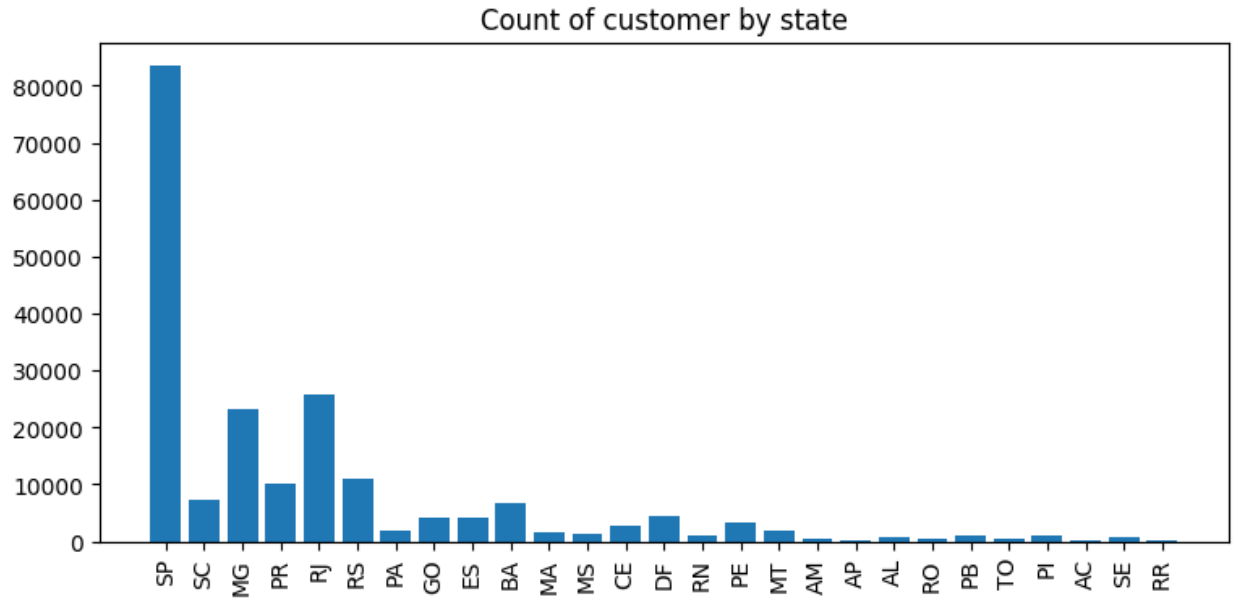
	State	Cust_Count
0	SP	83492
1	SC	7274
2	MG	23270
3	PR	10090
4	RJ	25704
5	RS	10932
6	PA	1950
7	GO	4040
8	ES	4066
9	BA	6760
10	MA	1494
11	MS	1430
12	CE	2672
13	DF	4280
14	RN	970
15	PE	3304
16	MT	1814
17	AM	296
18	AP	136
19	AL	826
20	RO	506
21	PB	1072
22	TO	560
23	PI	990
24	AC	162
25	SE	700
26	RR	92

Visualization Comperision;

```
plt.figure(figsize=(9,4))
plt.bar(df['State'], df['Cust_Count'])
```

```
plt.xticks(rotation = 90)
plt.title('Count of customer by state')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



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"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns = ['Month', 'Order_counts'])
o = ["January",
"February","March","April","May","June","July","August","September","O
ctober"]
df
```

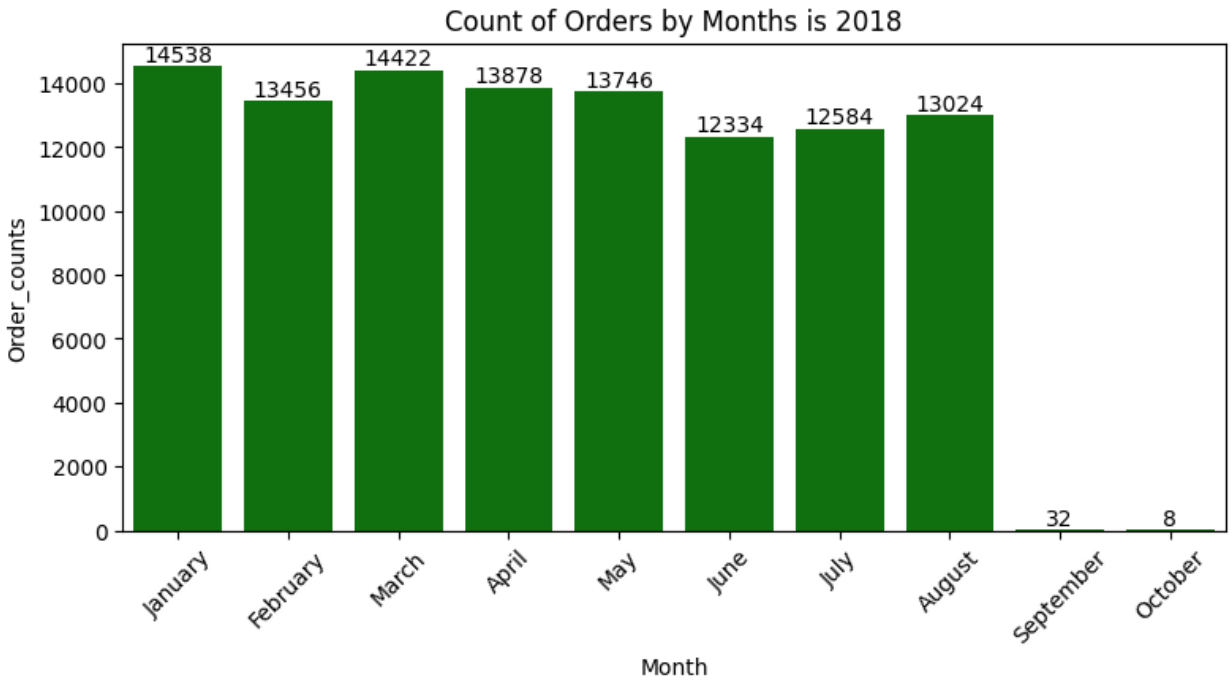
	Month	Order_counts
0	July	12584
1	August	13024
2	February	13456
3	June	12334
4	March	14422
5	January	14538
6	May	13746
7	April	13878

```
8  September            32
9    October             8
```

Visualization Comperision;

```
plt.figure(figsize=(9,4))
ax = sns.barplot(x = df["Month"],y =  df["Order_counts"], data = df,
order = o, color = "green")
plt.xticks(rotation = 45)
ax.bar_label(ax.containers[0])
plt.title("Count of Orders by Months is 2018")

plt.show()
```



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

```
query = """ with order_count 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(order_count.oc),2)
avg_order_count
from customers join order_count
on customers.customer_id = order_count.customer_id
```

```
group by customers.customer_city"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns = ['Products', 'AVG_count'])
df.head(10)
```

	Products	AVG_count
0	sao paulo	4.62
1	sao jose dos campos	4.55
2	porto alegre	4.70
3	indaial	4.46
4	treze tilias	5.09
5	rio de janeiro	4.59
6	mario campos	5.33
7	guariba	4.00
8	cuiaba	4.79
9	franca	5.01

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

```
query = """ select upper(products.product_category) categorey ,
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 categorey"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
data
df = pd.DataFrame(data, columns = ['categorey',
'Percentage_Distributaion'])
df.head(10)
```

	categorey	Percentage_Distributaion
0	PERFUMERY	12.66
1	FURNITURE DECORATION	35.73
2	TELEPHONY	12.17
3	BED TABLE BATH	42.79
4	AUTOMOTIVE	21.30
5	COMPUTER ACCESSORIES	39.61
6	HOUSEWARES	27.35
7	BABIES	13.49

```
8                   TOYS                        15.47
9      FURNITURE OFFICE                         16.16
```

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 """

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns = ['categorey',
'Count_orders_item','Averege_order_items'])
df.head(10)
```

	categorey	Count_orders_item	Averege_order_items
0	HEALTH BEAUTY	38680	130.16
1	sport leisure	34564	114.34
2	Cool Stuff	15184	167.36
3	computer accessories	31308	116.51
4	Watches present	23964	201.14
5	housewares	27856	90.79
6	electronics	11068	57.91
7	None	6412	112.00
8	toys	16468	117.55
9	bed table bath	44460	93.30

The correlation is.

```
arr1 = df["Count_orders_item"]
arr2 = df["Averege_order_items"]

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

```
-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 c
from
(select order_items.seller_id , sum(payments.payment_value) Revenue
from order_items join payments
```

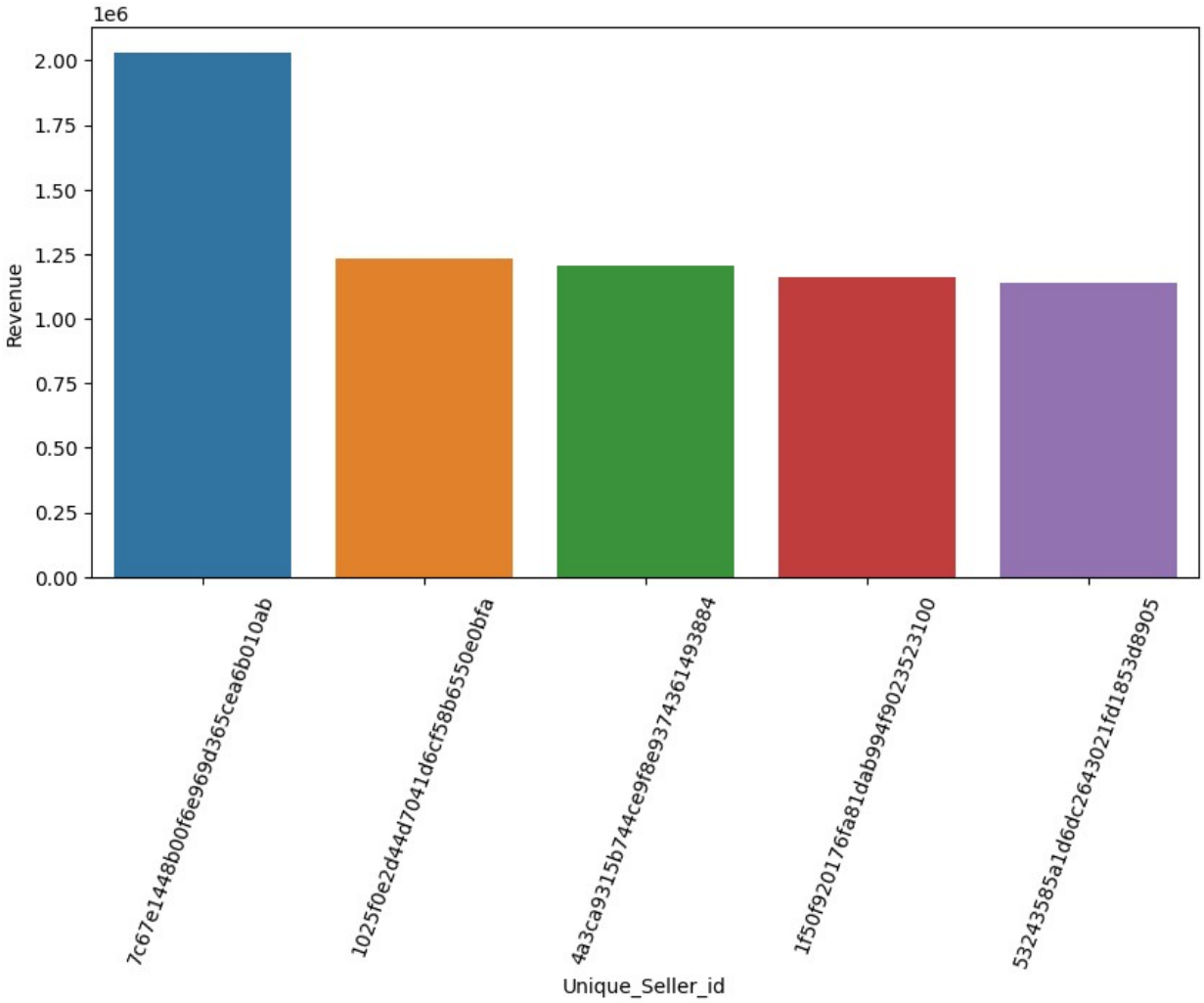
```
on order_items.order_id = payments.order_id
group by order_items.seller_id ) as x"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns = ['Unique_Seller_id',
'Revenue','Ranks'])
x = df.head()
x
```

	Unique_Seller_id	Revenue	Ranks
0	7c67e1448b00f6e969d365cea6b010ab	2.028668e+06	1
1	1025f0e2d44d7041d6cf58b6550e0bfa	1.232888e+06	2
2	4a3ca9315b744ce9f8e9374361493884	1.204981e+06	3
3	1f50f920176fa81dab994f9023523100	1.161014e+06	4
4	53243585a1d6dc2643021fd1853d8905	1.139612e+06	5

Analysis of Visuals

```
plt.figure(figsize = (10,5))
sns.barplot(x = "Unique_Seller_id", y = "Revenue", data = x)
plt.xticks(rotation = 70)
plt.show()
```



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

```
query = """ select customer_id, order_purchase_timestamp, Amount,
avg(Amount) 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 Amount
from payments join orders
on payments.order_id = orders.order_id) as a"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns =
```

```
['Customer_id','Order_Purchase_Time','Price','Moving_Avg'])
df.head()
```

```
                         Customer_id  Order_Purchase_Time   Price
Moving_Avg
0  00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74
114.739998
1  00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74
114.739998
2  00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74
114.739998
3  00012a2ce6f8dcda20d059ce98491703  2017-11-14 16:08:26  114.74
114.739998
4  000161a058600d5901f007fab4c27140  2017-07-16 09:40:32   67.41
67.410004
```

12. Calculate the cumulative sales per month for each year.

```
query = """ select years , months, price, sum(price)
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 price from orders join
payments
on orders.order_id = payments.order_id
group by years, months ) as x"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns =
['Years','Months','Price','Cumulative_sales'])
df.head()
```

```
   Years  Months       Price  Cumulative_sales
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
```

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

```
query = """ with x as
(select year(orders.order_purchase_timestamp) as years,
round(sum(payments.payment_value),2) as price from orders join
payments
on orders.order_id = payments.order_id
group by years)
```

```
select years, ((price - lag(price,1) over(order by years))/
lag(price,1) over(order by years))*100 from x"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns = ['Years','Percentage_of_growth'])
df
```

	Years	Percentage_of_growth
0	2016	NaN
1	2017	12112.703757
2	2018	20.000924

14. 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)
AS first_order
    FROM customers
    JOIN orders ON customers.customer_id = orders.customer_id
    GROUP BY customers.customer_id
),
b AS (
    SELECT orders.customer_id, COUNT(DISTINCT
orders.order_purchase_timestamp) AS next_order
    FROM orders
    JOIN a 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 orders.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;"""

cursor.fetchall()
cursor.execute(query)
data = cursor.fetchall()
print('None of the customer repited',data)
```

```
None of the customer repited [(None,)]
```

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

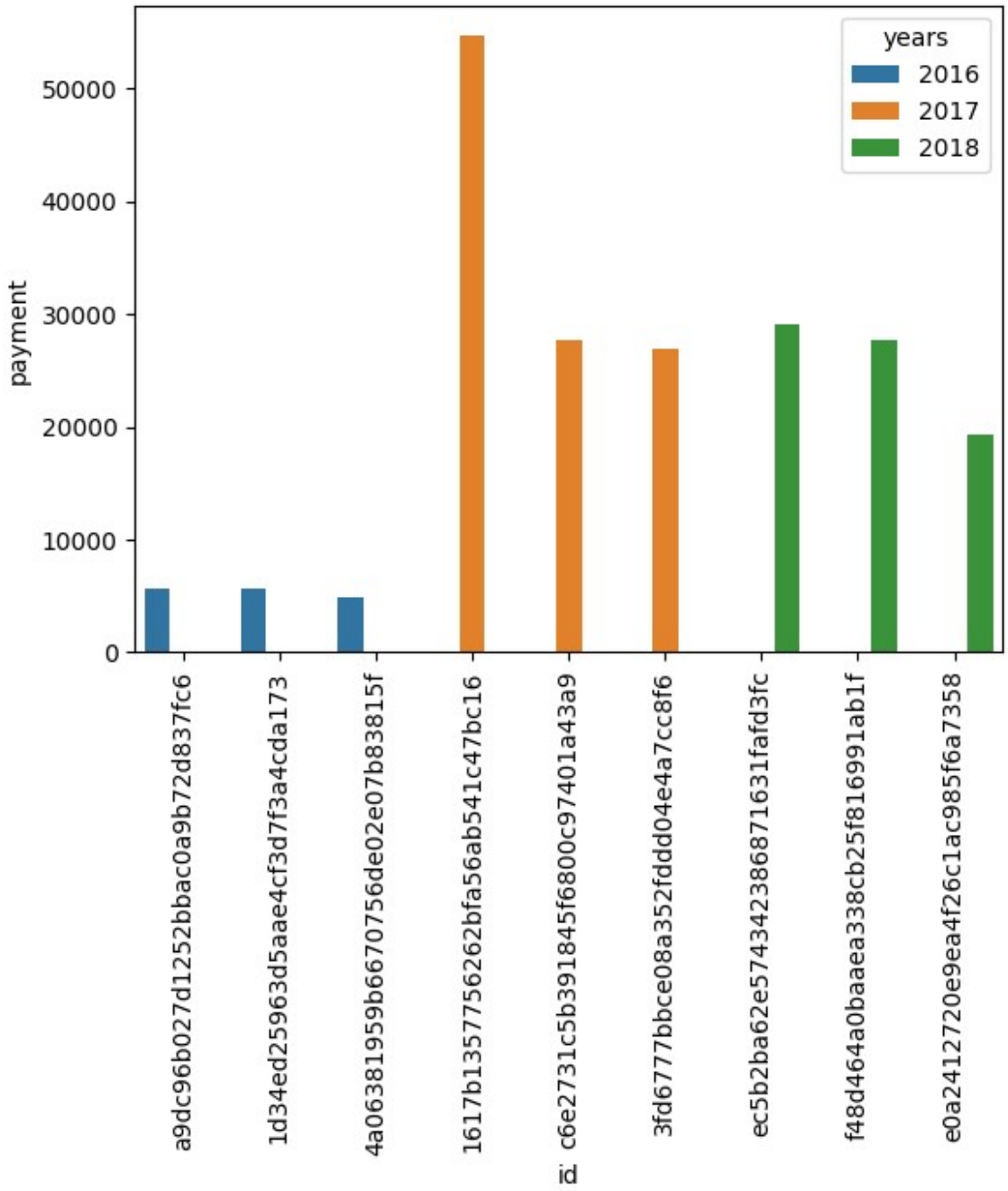
```
query = """select years, customer_id, payment, d_rank
from
(select year(orders.order_purchase_timestamp) years,
orders.customer_id, sum(payments.payment_value) payment,
dense_rank() over(partition by year(orders.order_purchase_timestamp)
order by sum(payments.payment_value) desc) d_rank
from orders 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 ;"""

cursor.execute(query)
data = cursor.fetchall()
df = pd.DataFrame(data, columns = ["years","id","payment","rank"])
df
```

	years	id	payment	rank
0	2016	a9dc96b027d1252bbac0a9b72d837fc6	5694.200195	1
1	2016	1d34ed25963d5aae4cf3d7f3a4cda173	5602.959961	2
2	2016	4a06381959b6670756de02e07b83815f	4911.120117	3
3	2017	1617b1357756262bfa56ab541c47bc16	54656.320312	1
4	2017	c6e2731c5b391845f6800c97401a43a9	27717.240234	2
5	2017	3fd6777bbce08a352fddd04e4a7cc8f6	26906.640625	3
6	2018	ec5b2ba62e574342386871631fafd3fc	29099.519531	1
7	2018	f48d464a0baaea338cb25f816991ab1f	27688.839844	2
8	2018	e0a2412720e9ea4f26c1ac985f6a7358	19237.759766	3

Analysis of Visuals.

```
sns.barplot(x = "id", y = "payment", data = df, hue = "years")
plt.xticks(rotation = 90)
plt.show()
```


years
2016
2017
2018
payment
50000
40000
30000
20000
10000
0
a9dc96b027d1252bbac0a9b72d837fc6
1d34ed25963d5aae4cf3d7f3a4cda173
4a063819596670756de02e07b83815f
1617b1357756262bfa56ab541c47bc16
c6e2731c5b391845f6800c97401a43a9
3fd6777bbce08a352fddd04e4a7cc8f6
ec5b2ba62e574342386871631fafd3fc
f48d464a0baaea338cb25f816991ab1f
e0a2412720e9ea4f26c1ac985f6a7358
id