PYTHON + SQL END TO END PROJECT

[basic to Advance solved queries]

TARGET

List all Unique Cities where Customers are Located

```
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 = [redacted],
                             database = "Ecommerce")
cur = db.cursor()
```

```
query = """ select distinct customer_city from customers  """

cur.execute(query)

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

	customer_states
0	franca
1	sao bernardo do campo
2	sao paulo
3	mogi das cruzes
4	campinas

```
#1. List of all Unique Cities where customers are located
SELECT DISTINCT customer_city FROM customers;
```

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 ', 90202)
```

```sql
#2. Count the number of Orders placed in 2017
SELECT COUNT(order_id)
FROM orders
WHERE year(order_purchase_timestamp) = 2017;
```

Find the total sales per category.

[4]:
```
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
```

[4]:

	Category	Sales
0	PERFUMERY	4053909.28
1	FURNITURE DECORATION	11441411.13
2	TELEPHONY	3895056.41
3	FASHION BAGS AND ACCESSORIES	1745266.24
4	BED TABLE BATH	13700429.37

```sql
#3. Find the total sales per Category
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;
```

Calculate the percentage of orders that were paid in installments.

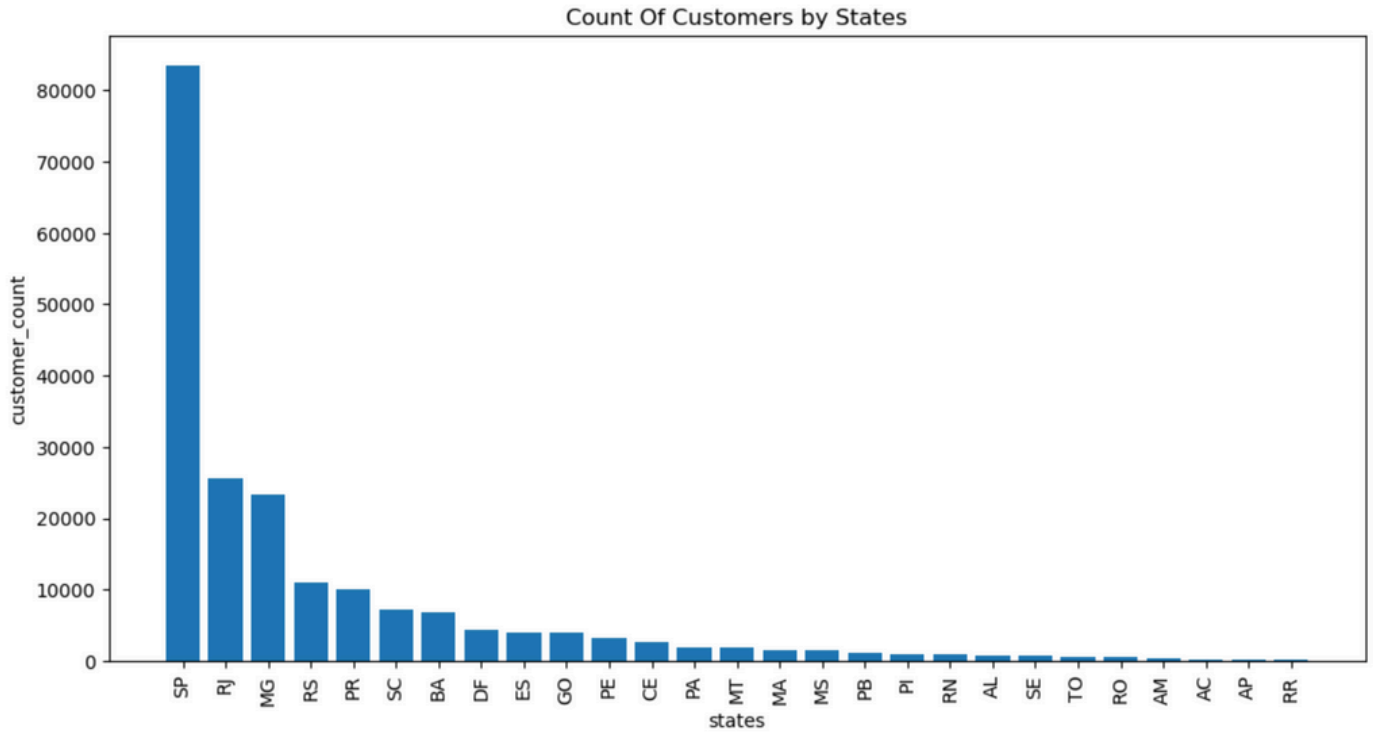
```
query = """ select (sum(case when payment_installments >=1 then 1
else 0 end))/count(*)*100 from payments """

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'))
```

```
#4. Calculate the percentage of orders that were paid in installments.
SELECT SUM(CASE WHEN payment_installments >= 1 THEN 1
ELSE 0 END) / COUNT(*)*100
FROM payments;
```

Count the number of customers from each state.

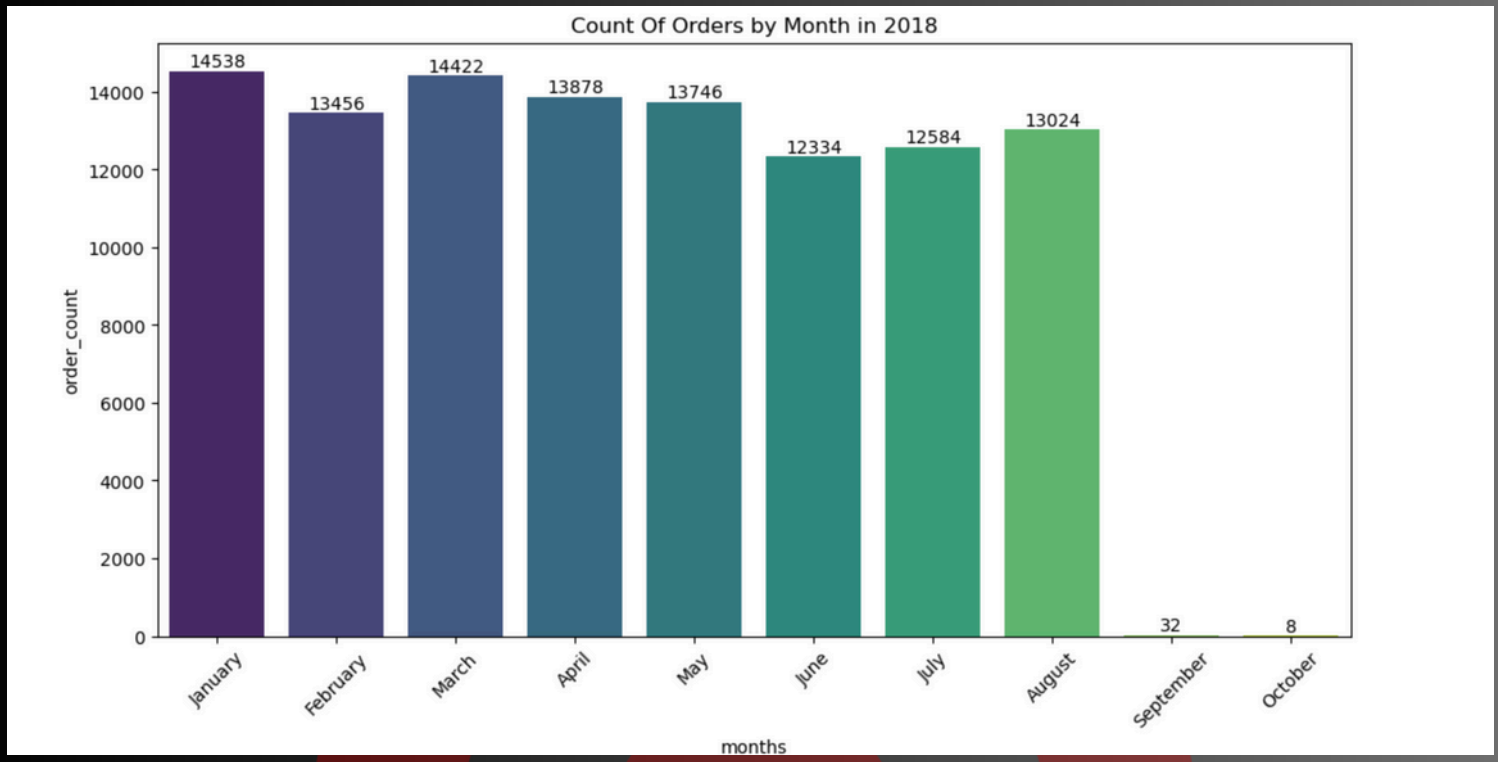
```
query = """ select customer_state, count(customer_id)
from customers
group by customer_state """

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 = (12,6))
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()
```




```sql
#5. Count the number of customers from each state
SELECT customer_state, COUNT(customer_id)
FROM customers
GROUP BY customer_state;
```

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()
plt.figure(figsize = (12,6))

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, palette = 'viridis')
plt.xticks(rotation = 45)
ax.bar_label(ax.containers[0])
plt.title('Count Of Orders by Month in 2018')

plt.show()
```




```sql
#6. Calculate the number of Orders per month in 2018.
SELECT monthname(order_purchase_timestamp) months, count(order_id) order_count
FROM orders
WHERE year(order_purchase_timestamp) = 2018
GROUP BY months;
```

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

[8]:
```
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 per order"])
df.head(10)
```

[8]:

	customer_city	average products per order
0	padre carvalho	28.00
1	celso ramos	26.00
2	datas	24.00
3	candido godoi	24.00
4	matias olimpio	20.00

```sql
#8. Calculate the percentage of total revenue contributed by each product category.
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;
```

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

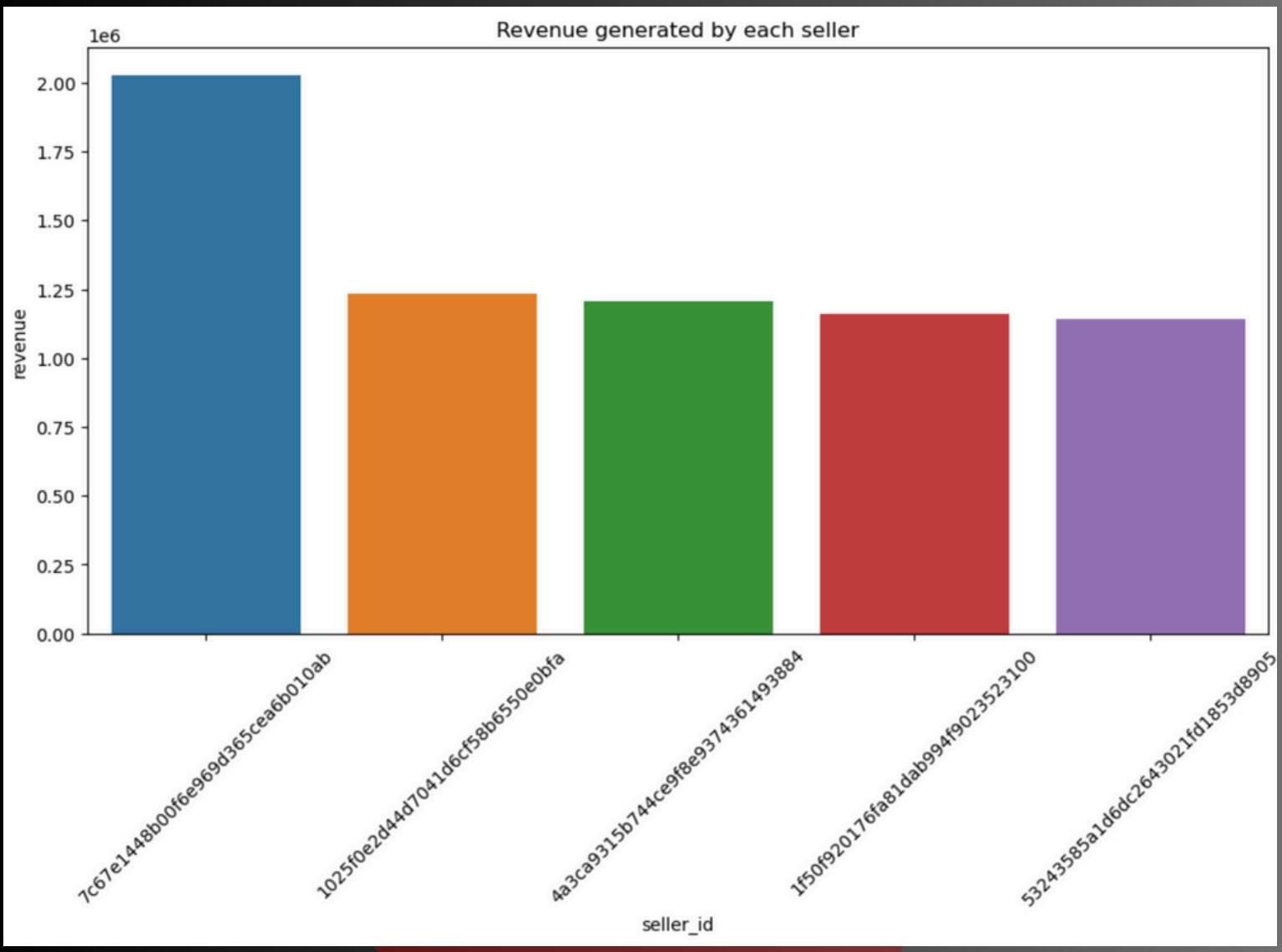
```python
import numpy as np
query = """ select products.product_category, count(order_items.product_id),
round(avg(order_items.price),2)
from products
join order_items
on products.product_id = order_items.product_id
group by products.product_category  """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["product_category", "order_count", "price"])

#corelation
arr1 = df["order_count"]
arr2 = df["price"]
a = np.corrcoef([arr1,arr2])
print("the correlation bw price and number of times a product has been purchased is", a)
```

```
the correlation bw price and number of times a product has been purchased is [[ 1.         -0.10631514]
 [-0.10631514  1.        ]]
```

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

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

```python
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()
plt.figure(figsize = (12,6))
sns.barplot(x = 'seller_id', y = "revenue", data = df)

plt.xticks(rotation = 45)
plt.show()
```




```sql
#10 Calculate the total revenue generated by each seller, and rank them by revenue.
SELECT *, DENSE_RANK() OVER(ORDER BY revenue DESC) AS rank_no
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;
```

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 = ["customer_id", "order_purchase_timestamp","payment","mov_avg"])

df.head(10)
```

	customer_id	order_purchase_timestamp	payment	mov_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
5	000161a058600d5901f007fab4c27140	2017-07-16 09:40:32	67.41	67.410004

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

Calculate the cumulative sales per month for each year.

```
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, columns = ["Year", "Months","Payment","Cumulative_Sales"])
df.head()
```

	Year	Months	Payment	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

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

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","year_over_year%"])
df
```

	years	year_over_year%
0	2016	NaN
1	2017	12112.703757
2	2018	20.000924

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

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)
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


# Note:- In This ques. Since None of our customer is repeated thats why our output is null.
```

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

```
#14. Calculate the retention rate of customers, defined as the percentage of customers who make another purchase within 6 months
#of their first purchase.
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)
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;
```

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

```
import numpy as np
query = """SELECT years, customer_id, payment, c_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)c_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 c_rank <= 3; """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns =["years","customer_id","payments","rank"])
plt.figure(figsize = (12,6))
plt.xticks(rotation = 45)
plt.title("Top 3 customer who spent most")
sns.barplot(x = "customer_id" , y= "payments", hue= "years", data = df)
plt.xlabel("customer_id")
plt.ylabel("payments")
plt.show();
```