Ecommerce Analysis

Objective

- *To compare the implementation of data analysis problems using **SQL and Python**, demonstrating the strengths, efficiency, and practical application of both technologies in handling real-world e-commerce datasets. The project aims to highlight differences in querying, data manipulation, aggregation, and advanced analytical techniques while showcasing proficiency in both programming environments.*

Problem Statement

- *Organizations often rely on multiple tools such as SQL databases and Python programming for data analysis. However, understanding when and how to use each tool effectively can be challenging. This project focuses on solving identical e-commerce data problems using both **SQL queries** and **Python (Pandas)** methods to evaluate their approaches, performance, and suitability for different analytical tasks such as sales calculation, customer behavior analysis, revenue trends, and advanced statistical operations.*

Basic Problems

Objective-Objective: Extract fundamental insights from the dataset

Methodology

- ***Dataset Understanding:*** *Studied the structure, fields, and relationships among key tables such as Customers, Orders, Order Items, Products, Sellers, and Payments.*
- ***Problem Definition:*** *Identified analytical questions covering basic metrics, intermediate aggregations, and advanced analytical scenarios.*
- ***Dual Implementation:*** *Solved each problem using both* **SQL queries** *and* **Python (Pandas)** *to enable direct comparison of approaches.*
- ***Data Preparation:*** *Performed data cleaning, datatype conversions, joins, filtering, grouping, and aggregations in both environments.*
- ***Comparative Analysis:*** *Evaluated differences in syntax, execution style, flexibility, and efficiency between SQL and Python solutions.*
- ***Result Validation:*** *Cross-checked outputs from both methods to ensure accuracy and consistency of results.*
- ***Insight Extraction:*** *Interpreted results to derive meaningful business insights related to sales trends, customer behavior, and revenue patterns.*

1. List of all unique cities where customers are located

CODE : Python

```
customers['customer_city'].unique()
```

```
array(['franca', 'sao bernardo do campo', 'sao paulo', ...,
       'monte bonito', 'sao rafael', 'eugenio de castro'], dtype=object)
```

CODE : SQL

```
SELECT DISTINCT
  customer_city
FROM
  customers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

customer_city
franca
sao bernardo do campo
sao paulo
mogi das cruzes
campinas
jaragua do sul
timoteo
curitiba
belo horizonte
montes claros

customers 2

2. Count number of orders placed in 2017

CODE : Python

```
orders_2017 = orders[
orders['order_purchase_timestamp'].astype(
str).str.startswith('2017')]len(orders_201
7)
```

45101

CODE : SQL

```
SELECT COUNT(*)FROM ordersWHERE
order_purchase_timestamp LIKE '2017%';
```

3.Total Sales Per Category

CODE : Python

```
merged = order_items.merge(products,
on='product_id')
sales_category = merged.groupby('product
category')['price'].sum().sort_values(ascending=Fa
lse)
```

```
product category
HEALTH BEAUTY                 1258681.34
Watches present               1205005.68
bed table bath                1036988.68
sport leisure                  988048.97
computer accessories           911954.32
                                 ...
flowers                          1110.04
House Comfort 2                   760.27
cds music dvds                    730.00
Fashion Children's Clothing       569.85
insurance and services            283.29
Name: price, Length: 73, dtype: float64
```

CODE : SQL

```
HOW COLUMNS FROM products;
SELECT p.`product category`, SUM(oi.price) AS
total_sales
FROM order_items oi JOIN products p ON
oi.product_id = p.product_id
GROUP BY p.`product category`
ORDER BY total_sales DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product category	total_sales
HEALTH BEAUTY	1258681.340969324
Watches present	1205005.6775493622
bed table bath	1036988.6800355911
sport leisure	988048.9688892365
computer accessories	911954.3174581528
Furniture Decoration	729762.4922962189
Cool Stuff	635290.8516969681
housewares	632248.6608705521
automotive	592720.1107084751
Garden tools	485256.4620256424

Result 4

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

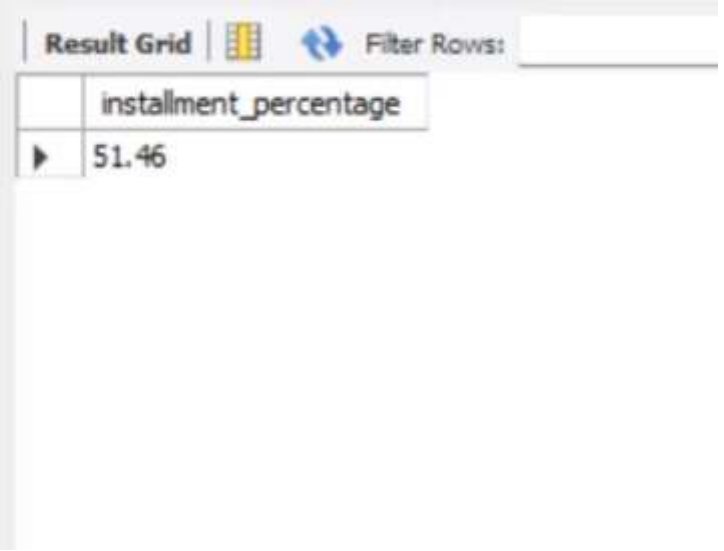
CODE : Python

```
total_orders = payments['order_id'].nunique()
installment_orders =
payments[payments['payment_installments'] >
1]['order_id'].nunique()
percentage = (installment_orders / total_orders) *
100
round(percentage, 2)
```

51.46

CODE : SQL

```
SELECT ROUND((COUNT(DISTINCT CASE
             WHEN payment_installments > 1 THEN
order_id
             END) * 100.0) / COUNT(DISTINCT
order_id),
             2) AS installment_percentage
FROM order_payments;
```



5. Count the number of customers from each state

CODE : Python

```
customers.groupby('customer_state')['customer_id'].count().sort_values(ascending=False)
```

```
customer_state
SP    41746
RJ    12852
MG    11635
RS     5466
PR     5045
SC     3637
BA     3380
DF     2140
ES     2033
GO     2020
PE     1652
CE     1336
PA      975
MT      907
MA      747
MS      715
PB      536
PI      495
RN      485
AL      413
SE      350
TO      280
RO      253
AM      148
AC       81
AP       68
RR       46
Name: customer_id, dtype: int64
```

CODE : SQL

```
SELECT customer_state, COUNT(customer_id) AS
total_customers
FROM customers
GROUP BY customer_state
ORDER BY total_customers DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_state	total_customers
SP	41746
RJ	12852
MG	11635
RS	5466
PR	5045
SC	3637
BA	3380
DF	2140
ES	2033
GO	2020

Result 6

Intermediate Problems

Objective: Dive deeper into sales and order trends.

1.Number of Orders Per Month in 2018

CODE : Python

```
orders['order_purchase_timestamp'] = pd.to_datetime(
    orders['order_purchase_timestamp'],
    format='%Y-%m-%d %H.%M.%S')
orders_2018 =
orders[orders['order_purchase_timestamp'].dt.year ==
2018].copy()
orders_2018.loc[:, 'month'] =
orders_2018['order_purchase_timestamp'].dt.to_period('M')
orders_2018.groupby('month')['order_id'].count()
```

```
[43]:  month
2018-01    7269
2018-02    6728
2018-03    7211
2018-04    6939
2018-05    6873
2018-06    6167
2018-07    6292
2018-08    6512
2018-09      16
2018-10       4
Freq: M, Name: order_id, dtype: int64
```

CODE : SQL

```
SELECT
    DATE_FORMAT(order_purchase_timestamp, '%Y-%m') AS
order_month,
    COUNT(order_id) AS total_orders
FROM
    orders
WHERE
    YEAR(order_purchase_timestamp) = 2018
GROUP BY
    order_month
ORDER BY
    order_month;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_month	total_orders
2018-01	7269
2018-02	6728
2018-03	7211
2018-04	6939
2018-05	6873
2018-06	6167
2018-07	6292
2018-08	6512
2018-09	16
2018-10	4

Result 7

2.Average Number of Products per Order by Customer City

CODE : Python

```python
# Count products per order
order_counts = (
    order_items
    .groupby('order_id')['product_id']
    .count()
    .reset_index(name='product_count')
)
# Merge with orders
merged = order_counts.merge(orders, on='order_id', how='left')
# Merge with customers
merged = merged.merge(customers, on='customer_id', how='left')
# Final result
avg_products_city = (
    merged
    .groupby('customer_city')['product_count']
    .mean()
    .sort_values(ascending=False)
)
avg_products_city
```

```
customer_city
padre carvalho    7.0
celso ramos       6.5
candido godoi     6.0
datas             6.0
matias olimpio    5.0
                 ...
indiana           1.0
indianopolis      1.0
indiapora         1.0
indiaroba         1.0
zortea            1.0
Name: product_count, Length: 4110, dtype: float64
```

CODE : SQL

```sql
SELECT c.customer_city,
       AVG(order_product_count) AS avg_products
FROM (
    SELECT order_id, COUNT(product_id) AS order_product_count
    FROM order_items
    GROUP BY order_id
) oi
JOIN orders o ON oi.order_id = o.order_id
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY c.customer_city;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

customer_city	avg_products
sao paulo	1.1562
bage	1.0476
macapa	1.1481
sao vendelino	1.0000
sao caetano do sul	1.1091
sao francisco do sul	1.2353
frederico westphalen	1.0714
coronel joao sa	1.0000
campo grande	1.1429
sao bernardo do campo	1.1422

3.Percentage of Total Revenue by Product Category

CODE : Python

```
merged = order_items.merge(products, on='product_id',
how='left')

category_revenue = ( merged.groupby('product
category')['price'] .sum())

percentage = (category_revenue /
category_revenue.sum()) * 100percentage =
percentage.sort_values(ascending=False)percentage
```

```
product category
HEALTH BEAUTY                9.384664
Watches present              8.984461
bed table bath               7.731735
sport leisure                7.366843
computer accessories         6.799485
                               ...
flowers                      0.008276
House Comfort 2              0.005669
cds music dvds               0.005443
Fashion Children's Clothing  0.004249
insurance and services       0.002112
Name: price, Length: 73, dtype: float64
```

CODE : SQL

```
SELECT p.`product category`, ROUND(SUM(oi.price) * 100
/ (SELECT SUM(price) FROM order_items), 2) AS
revenue_percentFROM order_items oi JOIN products p ON
oi.product_id = p.product_idGROUP BY p.`product
category`ORDER BY revenue_percent DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

product category	revenue_percent
HEALTH BEAUTY	9.26
Watches present	8.87
bed table bath	7.63
sport leisure	7.27
computer accessories	6.71
Furniture Decoration	5.37
Cool Stuff	4.67
housewares	4.65
automotive	4.36
Garden tools	3.57

Result 9 ×

4. Correlation Between Price and Purchase Count

CODE : Python

```python
product_stats =
order_items.groupby('product_id').agg({
   'price': 'mean',
   'order_id': 'count'})
product_stats.corr()
```

	price	order_id
price	1.00000	-0.03214
order_id	-0.03214	1.00000

CODE : SQL

```sql
SELECT    product_id,
   AVG(price) AS avg_price,
   COUNT(order_id) AS purchase_count
FROM
   order_items
GROUP BY product_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

product_id	avg_price	purchase_count
4244733e06e7ecb4970a6e2683c13e61	59.23333485921224	9
e5f2d52b802189ee658865ca93d83a8f	239.89999389648438	1
c777355d18b72b67abbeef9df44fd0fd	199	3
7634da152a4610f1595efa32f14722fc	12.989999771118164	2
ac6c3623068f30de03045865e4e10089	202.39999389648438	12
ef92defde845ab8450f9d70c526ef70f	21.899999618530273	5
8d4f2bb7e93e6710a28f34fa83ee7d28	18.56666628519694	3
557d850972a7d6f792fd18ae1400d9b6	810	1
310ae3c140ff94b03219ad0adc3c778f	145.9499969482422	2
4535b0e1091c278dfd193e5a1d63b39f	53.9900016784668	4

Result 10 ×

5.Total Revenue per Seller & Ranking

CODE : Python

```
seller_revenue =
order_items.groupby('seller_id')['price'].sum().so
rt_values(ascending=False)
seller_revenue.rank(ascending=False)
```

```
seller_id
4869f7a5dfa277a7dca6462dcf3b52b2       1.0
53243585a1d6dc2643021fd1853d8905       2.0
4a3ca9315b744ce9f8e9374361493884       3.0
fa1c13f2614d7b5c4749cbc52fecda94       4.0
7c67e1448b00f6e969d365cea6b010ab       5.0
                                       ...
34aefe746cd81b7f3b23253ea28bef39    3091.0
702835e4b785b67a084280efca355756    3092.0
1fa2d3def6adfa70e58c276bb64fe5bb    3093.0
77128dec4bec4878c37ab7d6169d6f26    3094.0
cf6f6bc4df3999b9c6440f124fb2f687    3095.0
Name: price, Length: 3095, dtype: float64
```

CODE : SQL

```
SELECT      seller_id,    SUM(price) AS
total_revenue,
    RANK() OVER (ORDER BY SUM(price) DESC) AS
revenue_rank
FROM order_items
GROUP BY seller_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

seller_id	total_revenue	revenue_rank
4869f7a5dfa277a7dca6462dcf3b52b2	229472.6283493042	1
53243585a1d6dc2643021fd1853d8905	222776.0495452881	2
4a3ca9315b744ce9f8e9374361493884	200472.921459198	3
fa1c13f2614d7b5c4749cbc52fecda94	194042.02939605713	4
7c67e1448b00f6e[illegible] fa1c13f2614d7b5c4749cbc52fecda94	[illegible]32	5
7e93a43ef30c4f03f38b393420bc753a	176431.86933135986	6
da8622b14eb17ae2831f4ac5b9dab84a	160236.5680885315	7
7a67c85e85bb2ce8582c35f2203ad736	141745.53166007996	8
1025f0e2d44d7041d6cf58b6550e0bfa	138968.55053710938	9
955fee9216a65b617aa5c0531780ce60	135171.7006969452	10

Result 11

Advance Problems

Objective: Generate strategic and customer-centric insights.

1.Moving Average of Order Value per Customer.

CODE : Python

```
df = orders.merge(order_payments, on='order_id',
how='left')df.head()

df = df.sort_values(['customer_id',
'order_purchase_timestamp'])

df['moving_avg'] = (
   df.groupby('customer_id')['payment_value']
     .rolling(3, min_periods=1)
     .mean()
     .reset_index(level=0, drop=True))

df[['customer_id','order_purchase_timestamp','paym
ent_value','moving_avg']].head(20)
```

CODE : SQL

```
SELECT   customer_id,   order_purchase_timestamp,
   payment_value,   AVG(payment_value) OVER
(
     PARTITION BY customer_id
     ORDER BY order_purchase_timestamp
     ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
) AS moving_avg_valueFROM orders oJOIN
order_payments pON o.order_id = p.order_id;
```

1.Moving Average of Order Value per Customer.

CODE : Python

[122]:

	customer_id	order_purchase_timestamp	payment_value	moving_avg
71588	00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.740000
10466	000161a058600d5901f007fab4c27140	2017-07-16 09:40:32	67.41	67.410000
68796	0001fd6190edaaf884bcaf3d49edf079	2017-02-28 11:06:43	195.42	195.420000
45160	0002414f95344307404f0ace7a26f1d5	2017-08-16 13:09:20	179.35	179.350000
6119	000379cdec625522490c315e70c7a9fb	2018-04-02 13:42:17	107.01	107.010000
76896	0004164d20a9e969af783496f3408652	2017-04-12 08:35:12	71.80	71.800000
48296	000419c5494106c306a97b5635748086	2018-03-02 17:47:40	49.40	49.400000
62658	00046a560d407e99b969756e0b10f282	2017-12-18 11:08:30	166.59	166.590000
82777	00050bf6e01e69d5c0fd612f1bcfb69c	2017-09-17 16:04:44	85.23	85.230000
83948	000598caf2ef4117407665ac33275130	2018-08-11 12:14:35	1255.71	1255.710000
869	0005aefbb696d34b3424dccd0a0e9fd0	2018-06-20 09:46:53	147.33	147.330000
92420	00062b33cb9f6fe976afdcff967ea74d	2017-03-15 23:44:09	58.95	58.950000
68729	00066ccbe787a588c52bd5ff404590e3	2018-02-06 16:10:09	270.00	270.000000
97466	00072d033fe2e59061ae5c3aff1a2be5	2017-09-01 09:24:39	106.97	106.970000
45768	0009a69b72033b2d0ec8c69fc70ef768	2017-04-28 13:36:30	173.60	173.600000
24135	000bf8121c3412d3057d32371c5d3395	2017-10-11 07:44:31	45.56	45.560000
20010	000e943451fc2788ca6ac98a682f2f49	2017-04-20 19:37:14	26.80	26.800000
20011	000e943451fc2788ca6ac98a682f2f49	2017-04-20 19:37:14	26.80	26.800000

CODE : SQL

customer_id	order_purchase_timestamp	payment_value	moving_avg_value
00012a2ce6f8dcda20d059ce98491703	2017-11-14 16:08:26	114.74	114.73999786376953
000161a058600d5901f007fab4c27140	2017-07-16 09:40:32	67.41	67.41000366210938
0001fd6190edaaf884bcaf3d49edf079	2017-02-28 11:06:43	195.42	195.4199981689453
0002414f95344307404f0ace7a26f1d5	2017-08-16 13:09:20	179.35	179.35000610351562
000379cdec625522490c315e70c7a9fb	2018-04-02 13:42:17	107.01	107.01000213623047
0004164d20a9e969af783496f3408652	2017-04-12 08:35:12	71.8	71.80000305175781
000419c5494106c306a97b5635748086	2018-03-02 17:47:40	49.4	49.40000152587906
00046a560d407e99b969756e0b10f282	2017-12-18 11:08:30	166.59	166.58999633789062
00050bf6e01e69d5c0fd612f1bcfb69c	2017-09-17 16:04:44	85.23	85.2300033569336
000598caf2ef4117407665ac33275130	2018-08-11 12:14:35	1255.71	1255.7099609375

Result 12 ×

2. Calculate the cumulative sales per month for each year

CODE : Python

```
df = orders.merge(order_payments, on='order_id', how='left')

df['year'] = df['order_purchase_timestamp'].dt.year
df['month'] = df['order_purchase_timestamp'].dt.month

df[['order_purchase_timestamp','year','month']].head()

monthly_sales = (
    df.groupby(['year','month'])['payment_value']
      .sum()
      .reset_index())

monthly_sales = monthly_sales.sort_values(['year','month'])

monthly_sales['cumulative_sales'] = (
    monthly_sales.groupby('year')['payment_value']
                 .cumsum())

monthly_sales.head()
```

	year	month	payment_value	cumulative_sales
0	2016	9	252.24	252.24
1	2016	10	59090.48	59342.72
2	2016	12	19.62	59362.34
3	2017	1	138488.04	138488.04
4	2017	2	291908.01	430396.05
5	2017	3	449863.60	880259.65
6	2017	4	417788.03	1298047.68
7	2017	5	592918.82	1890966.50
8	2017	6	511276.38	2402242.88
9	2017	7	592382.92	2994625.80
10	2017	8	674396.32	3669022.12
11	2017	9	727762.45	4396784.57
12	2017	10	779677.88	5176462.45
13	2017	11	1194882.80	6371345.25
14	2017	12	878401.48	7249746.73

CODE : SQL

```
FROM (
    SELECT
        YEAR(o.order_purchase_timestamp) AS year,
        MONTH(o.order_purchase_timestamp) AS month,
        SUM(p.payment_value) AS monthly_sales
    FROM orders o
    JOIN order_payments p
        ON o.order_id = p.order_id
    GROUP BY year, month) t
ORDER BY year, month;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year	month	monthly_sales	cumulative_sales
2016	9	252.23999404907227	252.23999404907227
2016	10	59090.47999930382	59342.71999335289
2016	12	19.6200008392334	59362.3399419212
2017	1	138488.04006415606	138488.04006415606
2017	2	291908.00950714946	430396.0495713055
2017	3	449863.5995282233	880259.6490995288
2017	4	417788.02949872613	1298047.678598255
2017	5	592918.8201363329	1890966.4987345878
2017	6	511276.38032871485	2402242.8790633027
2017	7	592382.9194870591	2994625.798550362

Result 14 ×

3.Year-over-Year Growth Rate

CODE : Python

```
yearly_sales = (
   df.groupby('year')['payment_value']
     .sum()
     .reset_index())

yearly_sales

yearly_sales = yearly_sales.sort_values('year')

yearly_sales['previous_year_sales'] =
yearly_sales['payment_value'].shift(1)

yearly_sales['yoy_growth_percent'] = (
   (yearly_sales['payment_value'] -
yearly_sales['previous_year_sales'])
   / yearly_sales['previous_year_sales']) * 100

yearly_sales
```

	year	payment_value	previous_year_sales	yoy_growth_percent
0	2016	59362.34	NaN	NaN
1	2017	7249746.73	59362.34	12112.703761
2	2018	8699763.05	7249746.73	20.000924

CODE : SQL

```
SELECT yr, total_sales, (total_sales - LAG(total_sales) OVER
(ORDER BY yr)) / LAG(total_sales) OVER (ORDER BY yr) * 100 AS
yoy_growthFROM yearly;
```

Result Grid | Filter Rows: | Export: | Wrap Cell C

yr	total_sales	yoy_growth
2016	59362.33999419212	NULL
2017	7249746.72820987	12112.70375952021
2018	8699763.051850779	20.00092386674247

4.Customer Retention Rate (6 Months)

CODE : Python

```
first_purchase = (
   df.groupby('customer_id')['order_purchase_timestamp']
     .min()
     .reset_index())
first_purchase.columns = ['customer_id', 'first_purchase_date']

merged = df.merge(first_purchase, on='customer_id', how='left')

repeat_orders = merged[
   (merged['order_purchase_timestamp'] >
merged['first_purchase_date']) &
   (merged['order_purchase_timestamp'] <=
     merged['first_purchase_date'] + pd.DateOffset(months=6))]

total_customers = first_purchase['customer_id'].nunique()
repeat_customers = repeat_orders['customer_id'].nunique()

# Count customers
total_customers = first_purchase['customer_id'].nunique()
repeat_customers = repeat_orders['customer_id'].nunique()

# Retention %
retention_rate = (repeat_customers / total_customers) * 100
round(retention_rate, 2)
```

```
]: 0.0
```

CODE : SQL

```
SELECT COUNT(DISTINCT r.customer_id) * 100.0 /
  COUNT(DISTINCT f.customer_id) AS retention_rate
FROM first_purchase f
LEFT JOIN repeat_purchase r
ON f.customer_id = r.customer_id;
```

Result Grid | Filter Rows: | Export:

retention_rate
0.00000

5.Top 3 Customers per Year by Spend

CODE : Python

```python
orders['order_purchase_timestamp'] =
pd.to_datetime(orders['order_purchase_timestamp'])
orders['year'] = orders['order_purchase_timestamp'].dt.year
df = orders.merge(order_payments, on='order_id', how='left')
spend = (
    df.groupby(['year','customer_id'])['payment_value']
    .sum()
    .reset_index()
)
spend['rank'] = spend.groupby('year')['payment_value'] \
    .rank(method='dense', ascending=False)
top3_customers = spend[spend['rank'] <= 3] \
    .sort_values(['year','rank'])
```

	year	customer_id	payment_value	rank
223	2016	a9dc96b027d1252bbac0a9b72d837fc6	1423.55	1.0
38	2016	1d34ed25963d5aae4cf3d7f3a4cda173	1400.74	2.0
84	2016	4a06381959b6670756de02e07b83815f	1227.78	3.0
4218	2017	1617b1357756262bfa56ab541c47bc16	13664.08	1.0
35453	2017	c6e2731c5b391845f6800c97401a43a9	6929.31	2.0
11541	2017	3fd6777bbce08a352fddd04e4a7cc8f6	6726.66	3.0
95349	2018	ec5b2ba62e574342386871631fafd3fc	7274.88	1.0
97087	2018	f48d464a0baaea338cb25f816991ab1f	6922.21	2.0
92873	2018	e0a2412720e9ea4f26c1ac985f6a7358	4809.44	3.0

CODE : SQL

```sql
SELECT *
FROM (
    SELECT *,
           RANK() OVER (PARTITION BY yr ORDER BY spend DESC) AS rnk
    FROM yearly_spend
) AS ranked
WHERE rnk <= 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	yr	spend	rnk
a9dc96b027d1252bbac0a9b72d837fc6	2016	1423.550048828125	1
1d34ed25963d5aae4cf3d7f3a4cda173	2016	1400.739990234375	2
4a06381959b6670756de02e07b83815f	2016	1227.780029296875	3
1617b1357756262bfa56ab541c47bc16	2017	13664.080078125	1
c6e2731c5b391845f6800c97401a43a9	2017	6929.31005859375	2
3fd6777bbce08a352fddd04e4a7cc8f6	2017	6726.66015625	3
ec5b2ba62e574342386871631fafd3fc	2018	7274.8798828125	1
f48d464a0baaea338cb25f816991ab1f	2018	6922.2099609375	2
e0a2412720e9ea4f26c1ac985f6a7358	2018	4809.43994140625	3

Result 17

Conclusion

- *This project successfully demonstrated the comparison of solving identical e-commerce data analysis problems using **SQL** and **Python (Pandas)**. Both tools proved effective for data manipulation, aggregation, and advanced analytical tasks, while differing in syntax style, flexibility, and execution approach.*
- *SQL showed strong capability in structured querying, joins, and database-level operations, whereas Python provided greater flexibility for data transformation, statistical analysis, and complex logic handling.*
- *Through this comparative approach, the project strengthened practical understanding of when and how to use each technology efficiently. The analysis also highlighted key business insights related to sales trends, customer behavior, and revenue distribution, reinforcing the importance of selecting the right analytical tool based on problem requirements.*