SQL PROJECT

Q1. List all unique cities where customer are located.

```
SELECT DISTINCT
  customer_city
FROM
  customers
ORDER BY customer_city;
```

Q2. Count number of order placed in 2017

```
SELECT
  COUNT(*) AS total_orders_2017
FROM
  orders
WHERE
  YEAR(order_purchase_timestamp) = 2017;
```

Q3. Find Total Sales Per Category

```
SELECT
  COUNT(*)
FROM
  order_items;
SELECT
  COUNT(*)
FROM
  products;

SELECT
  p.product_category, SUM(oi.price) AS total_sales
FROM
  order_items oi
    LEFT JOIN
```

```
    products p ON oi.product_id = p.product_id
GROUP BY p.product_category
ORDER BY total_sales DESC;
```

Q4. Calculate the percentage of orders that were paid in instalments

```
SELECT
  (COUNT(CASE
    WHEN
      payment_type = 'credit_card'
        AND payment_installments > 1
    THEN
      order_id
  END) * 100.0 / COUNT(order_id)) AS installment_percentage
FROM
  payments;
```

Q5. Count the number of customer from each state

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

Q6. Calculate the number of orders per month in 2018

```
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;
```

Q7. Find the average number of products per order grouped by customer city

```
SELECT
  c.customer_city,
  AVG(order_item_count) AS avg_products_per_order
FROM
  (SELECT
    o.order_id,
      o.customer_id,
      COUNT(oi.product_id) AS order_item_count
  FROM
    orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY o.order_id , o.customer_id) AS order_counts
    JOIN
  customers c ON c.customer_id = order_counts.customer_id
GROUP BY c.customer_city
ORDER BY avg_products_per_order DESC;
```

Q8. Calculate the Percentage of total revenue contributed by each product category

```
SELECT
  p.product_category AS category,
  SUM(oi.price) AS total_revenue,
  (SUM(oi.price) * 100.0 / (SELECT
      SUM(price)
    FROM
      order_items)) AS revenue_percentage
FROM
  order_items oi
    JOIN
  products p ON oi.product_id = p.product_id
GROUP BY p.product_category
ORDER BY revenue_percentage DESC;
```

Q9. Identify the correlation between product price and the number of times product has been purchased?

```
SELECT
  p.product_id,
  p.product_category,
  AVG(oi.price) AS avg_price,
  COUNT(oi.order_id) AS purchase_count
FROM
  order_items oi
    JOIN
  products p ON oi.product_id = p.product_id
GROUP BY p.product_id , p.product_category
ORDER BY avg_price DESC;
```

Q10. Calculate the total revenue generated by each seller, and rank them by revenue

```
SELECT
  s.seller_id,
  s.seller_city,
  SUM(oi.price) AS total_revenue,
  RANK() OVER (ORDER BY SUM(oi.price) DESC) AS revenue_rank
FROM order_items oi
JOIN sellers s ON oi.seller_id = s.seller_id
GROUP BY s.seller_id, s.seller_city
ORDER BY total_revenue DESC;
```

Q11. Calculate the moving average of order value for each customer over their order history

```
WITH OrderValues AS (
  SELECT
    o.customer_id,
    o.order_id,
    SUM(oi.price) AS order_value,
    o.order_purchase_timestamp
  FROM orders o
```

```
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY o.customer_id, o.order_id, o.order_purchase_timestamp
),
MovingAvg AS (
  SELECT
    customer_id,
    order_id,
    order_value,
    order_purchase_timestamp,
    AVG(order_value) OVER (
      PARTITION BY customer_id
      ORDER BY order_purchase_timestamp
      ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS moving_avg_3_orders
  FROM OrderValues
)
SELECT * FROM MovingAvg;
```

Q12. Calculate the cumulative sales per month of each year

```
SELECT
  YEAR(o.order_purchase_timestamp) AS order_year,
  MONTH(o.order_purchase_timestamp) AS order_month,
  SUM(oi.price) AS monthly_sales
FROM
  orders o
    JOIN
  order_items oi ON o.order_id = oi.order_id
GROUP BY order_year , order_month;

WITH MonthlySales AS (
  SELECT
    YEAR(o.order_purchase_timestamp) AS order_year,
```

```
    MONTH(o.order_purchase_timestamp) AS order_month,
    SUM(oi.price) AS monthly_sales
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY order_year, order_month
)
SELECT
  order_year,
  order_month,
  monthly_sales,
  SUM(monthly_sales) OVER (
    PARTITION BY order_year
    ORDER BY order_month
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ) AS cumulative_sales
FROM MonthlySales
ORDER BY order_year, order_month;
```

Q13. Calculate the year-over-year growth rate of total sales

```
WITH YearlySales AS (
  SELECT
    YEAR(o.order_purchase_timestamp) AS order_year,
    SUM(oi.price) AS total_sales
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY order_year
),
SalesGrowth AS (
  SELECT
    order_year,
    total_sales,
```

```
    LAG(total_sales) OVER (ORDER BY order_year) AS prev_year_sales,
    ((total_sales - LAG(total_sales) OVER (ORDER BY order_year)) /
      LAG(total_sales) OVER (ORDER BY order_year)) * 100 AS yoy_growth
  FROM YearlySales
)
SELECT * FROM SalesGrowth
ORDER BY order_year;
```

Q14. Calculate the retention rate of customer, defined as the percentage of customer who make another purchase within 6 months of their first purchase?

```
SELECT customer_id, COUNT(order_id) AS order_count
FROM orders
GROUP BY customer_id
HAVING order_count > 1
ORDER BY order_count DESC;

WITH FirstPurchase AS (
  SELECT
    customer_id,
    MIN(order_purchase_timestamp) AS first_purchase_date
  FROM orders
  GROUP BY customer_id
),
RepeatCustomers AS (
  SELECT DISTINCT o.customer_id
  FROM orders o
  JOIN FirstPurchase fp ON o.customer_id = fp.customer_id
  WHERE o.order_purchase_timestamp > fp.first_purchase_date
    AND o.order_purchase_timestamp <= DATE_ADD(fp.first_purchase_date, INTERVAL
6 MONTH)
)
SELECT
```

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

Q15. Identify the top 3 customer who spent the most money in each year

```
WITH CustomerYearlySpending AS (
  SELECT
    YEAR(o.order_purchase_timestamp) AS order_year,
    o.customer_id,
    SUM(oi.price) AS total_spent
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY order_year, o.customer_id
),
RankedCustomers AS (
  SELECT
    order_year,
    customer_id,
    total_spent,
    RANK() OVER (PARTITION BY order_year ORDER BY total_spent DESC) AS
customer_rank
  FROM CustomerYearlySpending
)
SELECT order_year, customer_id, total_spent
FROM RankedCustomers
WHERE customer_rank <= 3
ORDER BY order_year, customer_rank;
```

PYTHON PROJECT

Q1. List all unique cities where customer are located.

```
unique_cities = customers["customer_city"].unique()

print(unique_cities)
```

Q2. Count number of order placed in 2017

```
orders["order_purchase_timestamp"] =
pd.to_datetime(orders["order_purchase_timestamp"])

orders_2017 = orders[orders["order_purchase_timestamp"].dt.year == 2017]

order_count_2017 = len(orders_2017)

print(f"Number of orders placed in 2017: {order_count_2017}")
```

Q3. Find Total Sales Per Category

```
merged_df = order_items.merge(products, on="product_id")

sales_per_category =
merged_df.groupby("product_category")["price"].sum().reset_index()

sales_per_category.columns = ["product_category", "total_sales"]

print(sales_per_category)
```

Q4. Calculate the percentage of orders that were paid in instalments

```
total_orders = payments["order_id"].nunique()

installment_orders = payments[payments["payment_installments"] >
1]["order_id"].nunique()

installment_percentage = (installment_orders / total_orders) * 100

print(f"Percentage of orders paid in installments: {installment_percentage:.2f}%")
```

Q5. Count the number of customers from each state

```
customers_per_state =
customers.groupby("customer_state")["customer_id"].nunique().reset_index()

customers_per_state.columns = ["state", "customer_count"]

print(customers_per_state)
```

Q6. Calculate the number of orders per month in 2018

```
orders["order_purchase_timestamp"] =
pd.to_datetime(orders["order_purchase_timestamp"])

orders_2018 = orders[orders["order_purchase_timestamp"].dt.year == 2018]

orders_per_month =
orders_2018.groupby(orders_2018["order_purchase_timestamp"].dt.strftime("%Y-
%m"))["order_id"].count().reset_index()

orders_per_month.columns = ["month", "order_count"]

print(orders_per_month)
```

Q7. Find the average number of products per order grouped by customer city

```
customers_orders = customers.merge(orders, on="customer_id")

orders_products = customers_orders.merge(order_items, on="order_id")

products_per_order = orders_products.groupby(["customer_city",
"order_id"])["order_id"].count().reset_index(name="product_count")

avg_products_per_city =
products_per_order.groupby("customer_city")["product_count"].mean().reset_index()

avg_products_per_city.columns = ["customer_city", "avg_products_per_order"]

print(avg_products_per_city)
```

Q8. Calculate the Percentage of total revenue contributed by each product category

```
merged_df = order_items.merge(products, on="product_id")

sales_per_category =
merged_df.groupby("product_category")["price"].sum().reset_index()

total_revenue = sales_per_category["price"].sum()

sales_per_category["percentage_revenue"] = (sales_per_category["price"] /
total_revenue) * 100

sales_per_category.columns = ["product_category", "total_sales",
"percentage_revenue"]
```

```
print(sales_per_category)
```

Q9. Identify the correlation between product price and the number of times product has been purchased?

```
product_purchase_count =
order_items.groupby("product_id")["order_id"].count().reset_index(name="purchase_
count")

merged_df = product_purchase_count.merge(order_items[["product_id",
"price"]].drop_duplicates(), on="product_id")

correlation = merged_df["price"].corr(merged_df["purchase_count"])

print(f"Correlation between product price and purchase count: {correlation:.2f}")
```

Q10. Calculate the total revenue generated by each seller, and rank them by revenue

```
seller_revenue = order_items.groupby("seller_id")["price"].sum().reset_index()

seller_revenue = seller_revenue.sort_values(by="price", ascending=False)

seller_revenue["Rank"] = seller_revenue["price"].rank(method="dense",
ascending=False).astype(int)

seller_revenue.columns = ["Seller ID", "Total Revenue", "Rank"]


print(seller_revenue)
```

Q11. Calculate the moving average of order value for each customer over their order history

```
order_values = order_items.groupby("order_id")["price"].sum().reset_index()

order_values = order_values.rename(columns={"price": "order_value"})

merged_df = orders[["order_id", "customer_id",
"order_purchase_timestamp"]].merge(

   order_values, on="order_id"

)

merged_df["order_purchase_timestamp"] =
pd.to_datetime(merged_df["order_purchase_timestamp"])

merged_df = merged_df.sort_values(by=["customer_id",
"order_purchase_timestamp"])
```

```
merged_df["moving_avg_3_orders"] =
merged_df.groupby("customer_id")["order_value"].transform(
    lambda x: x.rolling(window=3, min_periods=1).mean()
)
print(merged_df[["customer_id", "order_id", "order_purchase_timestamp",
"order_value", "moving_avg_3_orders"]])
```

Q12. Calculate the cumulative sales per month of each year

```
orders["order_purchase_timestamp"] =
pd.to_datetime(orders["order_purchase_timestamp"])
orders["year"] = orders["order_purchase_timestamp"].dt.year
orders["month"] = orders["order_purchase_timestamp"].dt.month
order_sales = order_items.groupby("order_id")["price"].sum().reset_index()
order_sales = order_sales.rename(columns={"price": "order_value"})

merged_df = orders[["order_id", "year", "month"]].merge(order_sales, on="order_id")
monthly_sales = merged_df.groupby(["year",
"month"])["order_value"].sum().reset_index()
monthly_sales["cumulative_sales"] =
monthly_sales.groupby("year")["order_value"].cumsum()
print(monthly_sales)
```

Q13. Calculate the year-over-year growth rate of total sales

```
orders["order_purchase_timestamp"] =
pd.to_datetime(orders["order_purchase_timestamp"])
orders["year"] = orders["order_purchase_timestamp"].dt.year
yearly_sales = order_items.merge(orders[["order_id", "year"]], on="order_id")
yearly_sales = yearly_sales.groupby("year")["price"].sum().reset_index()
yearly_sales = yearly_sales.rename(columns={"price": "total_sales"})
yearly_sales["yoy_growth"] = yearly_sales["total_sales"].pct_change() * 100
print(yearly_sales)
```

Q14. Calculate the retention rate of customer, defined as the percentage of customer who make another purchase within 6 months of their first purchase?

```
orders["order_purchase_timestamp"] =
pd.to_datetime(orders["order_purchase_timestamp"])


first_purchase =
orders.groupby("customer_id")["order_purchase_timestamp"].min().reset_index()
first_purchase = first_purchase.rename(columns={"order_purchase_timestamp":
"first_purchase_date"})


orders = orders.merge(first_purchase, on="customer_id")


orders["within_6_months"] = (orders["order_purchase_timestamp"] >
orders["first_purchase_date"]) & \
                        (orders["order_purchase_timestamp"] <=
orders["first_purchase_date"] + pd.DateOffset(months=6))


repeat_customers =
orders.groupby("customer_id")["within_6_months"].any().reset_index()


total_customers = repeat_customers["customer_id"].nunique()
retained_customers = repeat_customers["within_6_months"].sum()
retention_rate = (retained_customers / total_customers) * 100


print(f"Total Customers: {total_customers}")
print(f"Retained Customers: {retained_customers}")
print(f"Customer Retention Rate: {retention_rate:.2f}%")
```

Q15. Identify the top 3 customer who spent the most money in each year

```
orders["order_purchase_timestamp"] =
pd.to_datetime(orders["order_purchase_timestamp"])
orders["year"] = orders["order_purchase_timestamp"].dt.year
order_sales = order_items.groupby("order_id")["price"].sum().reset_index()
order_sales = order_sales.rename(columns={"price": "total_spent"})
```

```
merged_df = orders[["order_id", "customer_id", "year"]].merge(order_sales, on="order_id")

customer_spending = merged_df.groupby(["year", "customer_id"])["total_spent"].sum().reset_index()

customer_spending["rank"] = customer_spending.groupby("year")["total_spent"].rank(method="dense", ascending=False)

top_customers = customer_spending[customer_spending["rank"] <= 3].sort_values(["year", "rank"])

print(top_customers)
```