Name : Nitish K
Jupyter notebook link:
https://colab.research.google.com/drive/1wNcbkOzXGKAjsColCLdoddqEv6wfkOB_

Initial Process

```
#importing the libraries
import matplotlib.pyplot as plt
import pandas as pd
import findspark
findspark.init()
findspark.find()
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *
import pyspark.pandas as ps


spark =(
        SparkSession
           .builder
           .appName("PracticeApp")
           .master("local[4]")
   .config("spark.dynamicAllocation.enabled","false")
   .config("spark.sql.adaptive.enabled","false")
           .getOrCreate())
sc= spark.sparkContext
spark

#load each of the dataset
customer = (spark.read.option("header","true").csv("datasets/Copy of customers.csv"))
geolocation = (spark.read.option("header","true").csv("datasets/Copy of geolocation.csv"))
order_items = (spark.read.option("header","true").csv("datasets/Copy of order_items.csv"))
order_reviews = (spark.read.option("header","true").csv("datasets/Copy of
order_reviews.csv"))
```

```
orders = (spark.read.option("header","true").csv("datasets/Copy of orders.csv"))
payments = (spark.read.option("header","true").csv("datasets/Copy of payments.csv"))
products = (spark.read.option("header","true").csv("datasets/Copy of products.csv"))
sellers = (spark.read.option("header","true").csv("datasets/Copy of sellers.csv"))

#printing the schema
print("Customers Schema")
print(customer.printSchema())
print("\n")
print("Geolocation Schema")
print(geolocation.printSchema())
print("\n")
print("Order_items Schema")
print(order_items.printSchema())
print("\n")
print("Order Reviews Schema")
print(order_reviews.printSchema())
print("\n")
print("orders schema")
print(orders.printSchema())
print("\n")
print("Payment schema")
print(payments.printSchema())
print("\n")
print("Products Schema")
print(products.printSchema())
print("\n")
print("Sellers Schema")
print(sellers.printSchema())
print("\n")

#Create a temp view for each of the dataset above

#Customer View
customer.createOrReplaceTempView("customerView")
```

```
#Geolocation View
geolocation.createOrReplaceTempView("geolocationView")

#Order Items View
order_items.createOrReplaceTempView("order_itemsView")

#Order Reviews View
order_reviews.createOrReplaceTempView("order_reviewsView")

#Orders View
orders.createOrReplaceTempView("ordersView")

#Payment View
payments.createOrReplaceTempView("paymentView")

#Products View
products.createOrReplaceTempView("productsView")

#Sellers View
sellers.createOrReplaceTempView("sellersView")
```

Question 1

#1) Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset:

#1.1) Data type of all columns in the "customers" table.

Soln)

```
print(customer.printSchema())
```

Output

```
print(customer.printSchema())

root
 |-- customer_id: string (nullable = true)
 |-- customer_unique_id: string (nullable = true)
 |-- customer_zip_code_prefix: string (nullable = true)
 |-- customer_city: string (nullable = true)
 |-- customer_state: string (nullable = true)

None
```

#1.2) Get the time range between which the orders were placed.
Soln)

```
timerange = spark.sql("""
        SELECT MIN(order_purchase_timestamp) as
Min_Time,MAX(order_purchase_timestamp) as Max_TIme
  FROM ordersView
  GROUP BY DATE(order_purchase_timestamp)
  ORDER BY DATE(order_purchase_timestamp)
""")
timerangedf = ps.DataFrame(timerange)
timerangedf
```

Output

Out[45]:

	Min_Time	Max_TIme
0	2016-09-04 21:15:19	2016-09-04 21:15:19
1	2016-09-05 00:15:34	2016-09-05 00:15:34
2	2016-09-13 15:24:19	2016-09-13 15:24:19
3	2016-09-15 12:16:38	2016-09-15 12:16:38
4	2016-10-02 22:07:52	2016-10-02 22:07:52
5	2016-10-03 09:44:50	2016-10-03 22:51:30
6	2016-10-04 09:06:10	2016-10-04 23:59:01
7	2016-10-05 00:32:31	2016-10-05 23:14:34
8	2016-10-06 00:06:17	2016-10-06 23:49:18
9	2016-10-07 00:54:40	2016-10-07 23:18:38
10	2016-10-08 01:28:14	2016-10-08 23:46:06

#1.3) Count the number of Cities and States in our dataset
Soln)

#For customer's dataset

```
count = spark.sql("""
        SELECT COUNT(DISTINCT customer_city) AS City_Count,COUNT(DISTINCT
customer_state) AS State_Count
   FROM customerView
""")
countDF = ps.DataFrame(count)
print("Count of city and states in customer dataset is")
countDF
```

Output:

```
Count of city and states in customer dataset is
Out[16]:
```

	City_Count	State_Count
0	4119	27

```
#For geolocation dataset
count2 = spark.sql("""
        SELECT COUNT(DISTINCT geolocation_city) AS City_Count,COUNT(DISTINCT
geolocation_state) AS State_Count
   FROM geolocationView
""")
countDF2 = ps.DataFrame(count2)
print("Count of city and states in Geolocation dataset is")
countDF2
```

Output:

```
Count of city and states in Geolocation dataset is
Out[17]:
```

	City_Count	State_Count
0	8011	27

```
#For Seller Dataset
count3 = spark.sql("""
        SELECT COUNT(DISTINCT seller_city) AS City_Count,COUNT(DISTINCT seller_state) AS
State_Count
   FROM sellersView
""")
countDF3 = ps.DataFrame(count3)
print("Count of city and states in Seller dataset is")
countDF3
```

Output:

```
Count of city and states in Seller dataset is

Out[18]:
```

	City_Count	State_Count
0	611	23

#2. In-depth Exploration:

#1. Is there a growing trend in the no. of orders placed over the past years?

SOln)

```
growingTrend = spark.sql(
  """
        SELECT YEAR(o.order_purchase_timestamp) as
year_placed,MONTH(o.order_purchase_timestamp) as
      month_placed,COUNT(o.order_id) as count_of_orders_placed
            FROM ordersView o
            INNER JOIN customerView c
            ON o.customer_id = c.customer_id
            GROUP BY year_placed,month_placed
            ORDER BY year_placed ASC
  """
)

growingTrendDF = ps.DataFrame(growingTrend)
growingTrendDF
```

#Solution :

#Based on the results obtained it can be seen the there is an increase in the number of orders placed from 4 to 324

 # and then dropped to a minimal of 1 in december

#FRom 2017 the number of orders shows a general increasing trend from January to November, with slight
#fluctuations in between.
#IN 2018 the number of orders starts high in January and remains relatively stable until May. After May,
#there is a decreasing trend in order volume.

Output:

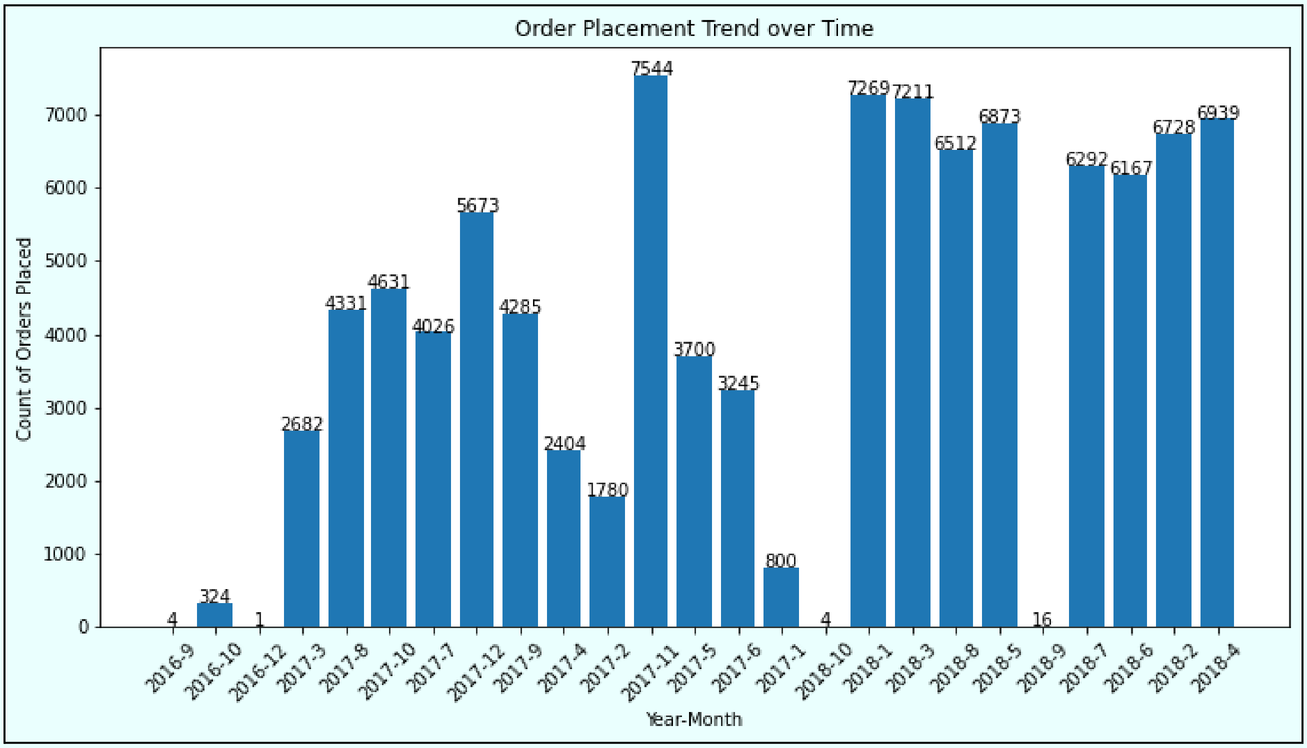
Out[38]:

	year_placed	month_placed	count_of_orders_placed
0	2016	9	4
1	2016	10	324
2	2016	12	1
3	2017	3	2682
4	2017	8	4331
5	2017	10	4631
6	2017	7	4026
7	2017	12	5673
8	2017	9	4285
9	2017	4	2404

Graphical Representation

```
plt.figure(figsize=(12, 6))
plt.bar(range(len(growingTrendDF)),
growingTrendDF['count_of_orders_placed'].to_numpy())

for i, count in enumerate(growingTrendDF['count_of_orders_placed'].to_numpy()):
    plt.text(i, count + 0.5, str(count), ha='center')

plt.xticks(range(len(growingTrendDF)),
growingTrendDF['year_placed'].astype(str).to_numpy() + '-' +
growingTrendDF['month_placed'].astype(str).to_numpy())
```

```
plt.xlabel('Year-Month')
plt.ylabel('Count of Orders Placed')
plt.title('Order Placement Trend over Time')

plt.xticks(rotation=45)

plt.show()
```



#2. Can we see some kind of monthly seasonality in terms of the no. of orders being placed?
Soln)
monthly seasonality refers to the regular and predictable fluctuations in certain variables or phenomena that can be observed
#from one month to another. From the above output
#In september the no of orders placed in september 2017 was more when compared to 2016 and 2018

#In october the no of orders placed in october 2017 was more when compared to 2016 and 2018
#In november it was only 2017
#In december 2017 had the highest no of orders placed

#3) During what time of the day, do the Brazilian customers mostly place their orders? (Dawn, Morning, Afternoon or Night)
#▪ 0-6 hrs : Dawn
#▪ 7-12 hrs : Mornings
#▪ 13-18 hrs : Afternoon
#▪ 19-23 hrs : Night

Soln)

```
brazil = spark.sql(
    """
    SELECT
        (CASE WHEN HOUR(o.order_purchase_timestamp) BETWEEN 0 AND 6 THEN 'Dawn'
              WHEN HOUR(o.order_purchase_timestamp) BETWEEN 7 AND 12 THEN 'Morning'
              WHEN HOUR(o.order_purchase_timestamp) BETWEEN 13 AND 18 THEN 'Afternoon'
              WHEN HOUR(o.order_purchase_timestamp) BETWEEN 19 AND 23 THEN 'Night'
        END) AS time_slot,
        COUNT(*) AS order_count
    FROM ordersView o
    INNER JOIN customerView c
    ON o.customer_id = c.customer_id
    GROUP BY time_slot
    ORDER BY order_count DESC
    """
)

brazilDF = ps.DataFrame(brazil)
brazilDF
```

#SOlution

#The highest number of orders placed by the brazilian customers is during Afternoon having order count of 38135

Output:

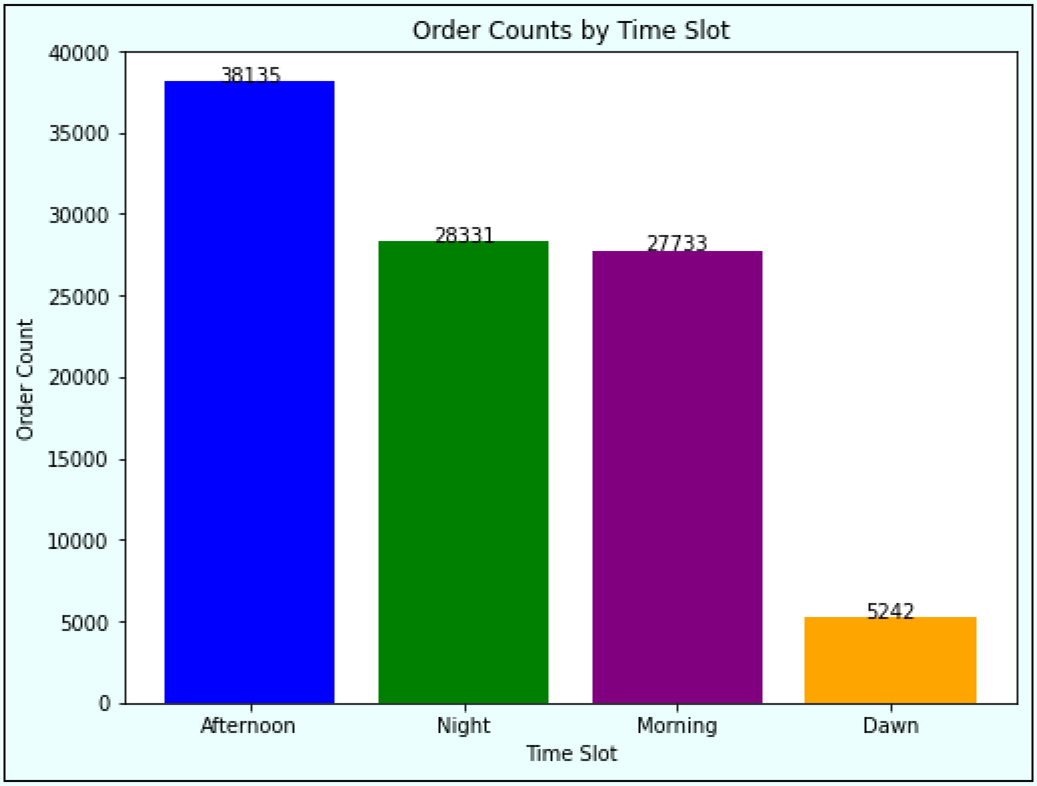
Out[21]:

	time_slot	order_count
0	Afternoon	38135
1	Night	28331
2	Morning	27733
3	Dawn	5242

Graphical representation

```
time_slots = brazilDF['time_slot'].to_numpy()
order_counts = brazilDF['order_count'].to_numpy()

colors = ['blue', 'green', 'purple', 'orange', 'purple']

plt.figure(figsize=(8, 6))
plt.bar(time_slots, order_counts,color = colors)
for i, count in enumerate(brazilDF['order_count'].to_numpy()):
    plt.text(i, count + 0.5, str(count), ha='center')
plt.xlabel('Time Slot')
plt.ylabel('Order Count')
plt.title('Order Counts by Time Slot')
plt.grid(False)

plt.show()
```

Output:



#3)Evolution of E-commerce orders in the Brazil region:

#1) Get the month on month no. of orders placed in each state.

Soln)

monthonmonth = spark.sql("""
  WITH cte AS (
    SELECT month(CAST(o.order_purchase_timestamp AS date)) AS purchase_month,
    c.customer_state,
    COUNT(o.order_id) AS no_of_orders,
    LAG(month(CAST(o.order_purchase_timestamp AS date)), 1) OVER(PARTITION BY
c.customer_state
    ORDER BY COUNT(o.order_id) DESC) AS lagged_month
    FROM ordersView o
    INNER JOIN customerView c ON o.customer_id = c.customer_id

```
      GROUP BY month(CAST(o.order_purchase_timestamp AS date)), c.customer_state
   )
   SELECT purchase_month,customer_state,no_of_orders FROM cte
   ORDER BY purchase_month ASC
""")


monthonmonthDF = ps.DataFrame(monthonmonth)
monthonmonthDF
```

Output:

Out[23]:

	purchase_month	customer_state	no_of_orders
0	1	SE	24
1	1	TO	19
2	1	SC	345
3	1	PI	55
4	1	RO	23
5	1	RN	51
6	1	AM	12
7	1	PE	113
8	1	PR	443
9	1	MT	96
10	1	MS	71

#2)How are the customers distributed across all the states?

Soln)

```
customercount = spark.sql(
   """
   SELECT customer_state,COUNT(customer_id) as count_of_customers
```

```
    FROM customerView
    GROUP BY customer_state
    Order BY count_of_customers ASC
    """
)
customercountDF = ps.DataFrame(customercount)
customercountDF.head(10)
```

Output:

Out[24]:

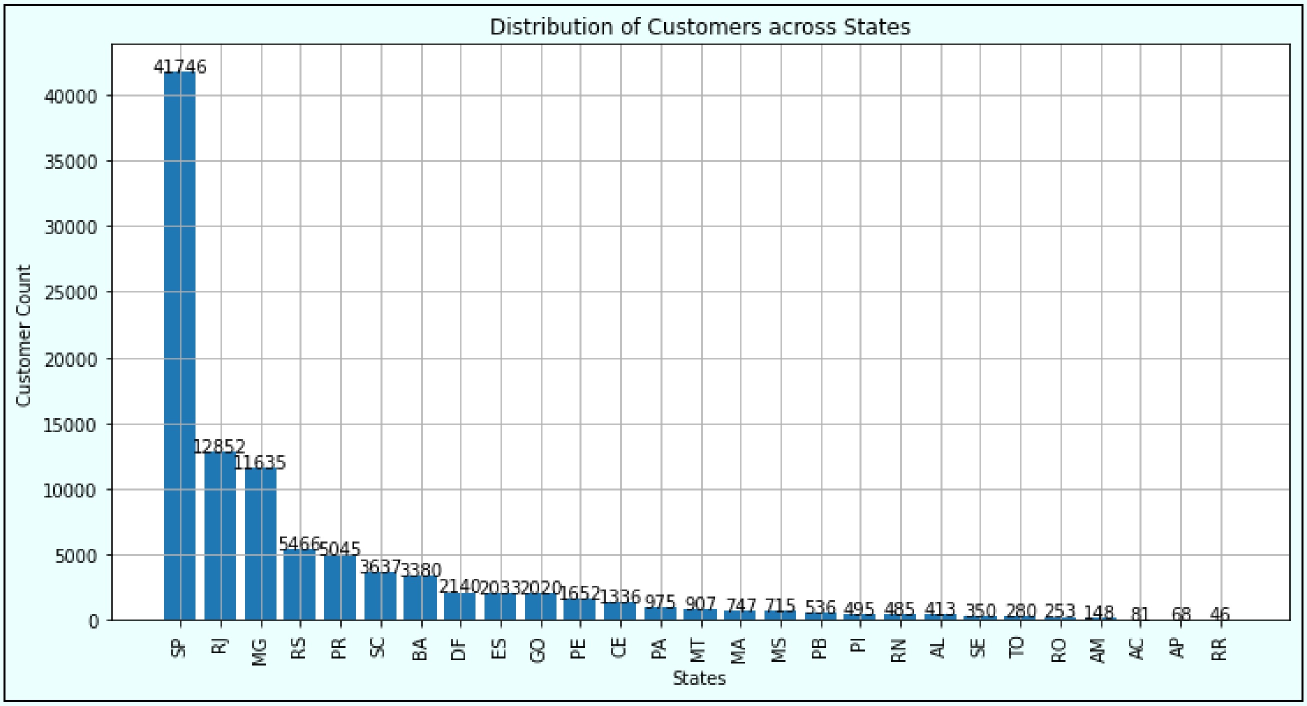
	customer_state	count_of_customers
0	RR	46
1	AP	68
2	AC	81
3	AM	148
4	RO	253
5	TO	280
6	SE	350
7	AL	413
8	RN	485
9	PI	495
10	PB	536

Graphical Representation

```
states = customercountDF['customer_state'].to_numpy()
customer_counts = customercountDF['count_of_customers'].to_numpy()

plt.figure(figsize=(12, 6))
plt.bar(states, customer_counts)
plt.xlabel('States')
plt.ylabel('Customer Count')
```

```
plt.title('Distribution of Customers across States')
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```

Output:



#4) Impact on Economy: Analyze the money movement by e-commerce by looking at order prices, freight and others.
#4.1) Get the % increase in the cost of orders from year 2017 to 2018 (include months between Jan to Aug only).
#You can use the "payment_value" column in the payments table to get the cost of orders

SOln)

```
percentIncrease = spark.sql("""
```

```
SELECT ((SUM(CASE WHEN YEAR(o.order_purchase_timestamp) = 2018 THEN
p.payment_value ELSE 0 END)
   - SUM(CASE WHEN YEAR(o.order_purchase_timestamp) = 2017 THEN p.payment_value
ELSE 0 END))
   / SUM(CASE WHEN YEAR(o.order_purchase_timestamp) = 2017 THEN p.payment_value
ELSE 0 END)) * 100
   AS cost_increase_percentage
FROM ordersView o
INNER JOIN paymentView p ON
o.order_id = p.order_id
WHERE YEAR(o.order_purchase_timestamp) IN (2017,2018)
AND MONTH(o.order_purchase_timestamp) BETWEEN 1 AND 8
""")

percentIncreaseDF = ps.DataFrame(percentIncrease)
percentIncreaseDF
```

Output)

Out[26]:

	cost_increase_percentage
0	136.976872

#4.2) Calculate the Total & Average value of order price for each state.
SOln)

```
data = spark.sql(
   """
         SELECT SUM(o.price) as Total_Price,AVG(o.price) as Average_Price
      FROM order_itemsView o
      INNER JOIN sellersView s
```

```
        ON o.seller_id = s.seller_id
        GROUP BY s.seller_state
    """
)

dataDF = ps.DataFrame(data)
dataDF.head(10)
```

Output:

Out[39]:

	Total_Price	Average_Price
0	6.324261e+05	155.196582
1	4.762200e+03	340.157143
2	2.522000e+03	210.166667
3	1.177000e+03	392.333333
4	6.639921e+04	127.690788
5	1.707072e+04	117.729103
6	8.753396e+06	108.951684
7	4.768961e+04	128.197876
8	1.709500e+04	449.868421
9	3.785595e+05	172.150769

#4.2) Calculate the Total & Average value of order freight for each state.
Soln)

```
data1 = spark.sql(
    """
        SELECT SUM(o.freight_value) as Total_Price,AVG(o.freight_value) as Average_Price
        FROM order_itemsView o
        INNER JOIN sellersView s
        ON o.seller_id = s.seller_id
```

```
    GROUP BY s.seller_state
  """
)

dataDF1 = ps.DataFrame(data1)
dataDF1
```

Output:

Out[40]:

	Total_Price	Average_Price
0	106547.06	26.146518
1	712.78	50.912857
2	443.32	36.943333
3	81.80	27.266667
4	12565.50	24.164423
5	4631.73	31.942966
6	1482487.67	18.452213
7	12171.13	32.718091
8	1489.15	39.188158
9	57243.09	26.031419

#5)Analysis based on sales, freight and delivery time.
#5.1)Find the no. of days taken to deliver each order from the order's purchase date as delivery time.
#Also, calculate the difference (in days) between the estimated & actual delivery date of an order.
#Do this in a single query.

Soln)

```
noofDays = spark.sql(
```

```
    """
    SELECT order_id,(DATE(order_delivered_customer_date) -
DATE(order_purchase_timestamp) ) as no_of_days,
    DATE(order_estimated_delivery_date) - DATE(order_delivered_customer_date) as
estimated_days
    FROM ordersView
    """
)

noofDaysDF = ps.DataFrame(noofDays)
noofDaysDF
```

Output

Out[29]:

	order_id	no_of_days	estimated_days
0	e481f51cbdc54678b7cc49136f2d6af7	8 days	8 days
1	53cdb2fc8bc7dce0b6741e2150273451	14 days	6 days
2	47770eb9100c2d0c44946d9cf07ec65d	9 days	18 days
3	949d5b44dbf5de918fe9c16f97b45f8a	14 days	13 days
4	ad21c59c0840e6cb83a9ceb5573f8159	3 days	10 days
5	a4591c265e18cb1dcee52889e2d8acc3	17 days	6 days
6	136cce7faa42fdb2cefd53fdc79a6098	NaT	NaT
7	6514b8ad8028c9f2cc2374ded245783f	10 days	12 days
8	76c6e866289321a7c93b82b54852dc33	10 days	32 days
9	e69bfb5eb88e0ed6a785585b27e16dbf	18 days	7 days
10	e6ce16cb79ec1d90b1da9085a6118aeb	13 days	9 days

#5.2)Find out the top 5 states with the highest & lowest average freight value.
SOln)

```
top5_states = spark.sql(
  """
  WITH state_avg_freight AS (
    SELECT c.customer_state, AVG(oi.freight_value) AS avg_freight
    FROM order_itemsView oi
    INNER JOIN ordersView o ON oi.order_id = o.order_id
    INNER JOIN customerView c ON c.customer_id = o.customer_id
    GROUP BY c.customer_state
  )
  SELECT customer_state, avg_freight
  FROM state_avg_freight
  ORDER BY avg_freight DESC
  LIMIT 5
  """
)
top5DF = ps.DataFrame(top5_states)
top5DF
```

Output:

Out[42]:

	customer_state	avg_freight
0	RR	42.984423
1	PB	42.723804
2	RO	41.069712
3	AC	40.073370
4	PI	39.147970

```
bottom5_states = spark.sql(
  """
  WITH state_avg_freight AS (
    SELECT c.customer_state, AVG(oi.freight_value) AS avg_freight
    FROM order_itemsView oi
    INNER JOIN ordersView o ON oi.order_id = o.order_id
    INNER JOIN customerView c ON c.customer_id = o.customer_id
```

```
        GROUP BY c.customer_state
    )
    SELECT customer_state, avg_freight
    FROM state_avg_freight
    ORDER BY avg_freight ASC
    LIMIT 5
    """
)
bottom5DF = ps.DataFrame(bottom5_states)
bottom5DF
```

Output:

Out[31]:

	customer_state	avg_freight
0	SP	15.147275
1	PR	20.531652
2	MG	20.630167
3	RJ	20.960924
4	DF	21.041355

#Find out the top 5 states with the highest & lowest average delivery time.
Soln)

```
top5avg = spark.sql(
    """
    WITH top5avg AS (
        SELECT c.customer_state, AVG(CAST(o.order_estimated_delivery_date AS timestamp))
AS avg_delivery_date
        FROM customerView c
        INNER JOIN ordersView o ON c.customer_id = o.customer_id
        GROUP BY c.customer_state
```

```
    )
    SELECT * FROM top5avg
    ORDER BY avg_delivery_date DESC
    LIMIT 5
    """
)
top5avg.show()

bottom5avg = spark.sql(
    """
    WITH top5avg AS (
        SELECT c.customer_state, AVG(CAST(o.order_estimated_delivery_date AS timestamp))
AS avg_delivery_date
        FROM customerView c
        INNER JOIN ordersView o ON c.customer_id = o.customer_id
        GROUP BY c.customer_state
    )
    SELECT * FROM top5avg
    ORDER BY avg_delivery_date ASC
    LIMIT 5
    """
)
bottom5avg.show()
```

Output

```
+--------------+-------------------+
|customer_state|  avg_delivery_date|
+--------------+-------------------+
|            AP|1.5179786470588236E9|
|            DF|1.5178605274766355E9|
|            MS|1.5172984523076923E9|
|            SP|1.5172669378910553E9|
|            PE|1.5172116886198547E9|
+--------------+-------------------+

+--------------+-------------------+
|customer_state|  avg_delivery_date|
+--------------+-------------------+
|            AC|1.5120527333333333E9|
|            RO| 1.514470090909091E9|
|            SE| 1.514615646857143E9|
|            AL|1.5151004324455206E9|
|            MA| 1.515196778313253E9|
+--------------+-------------------+
```

#6) Analysis based on the payments:
#6.1) Find the month on month no. of orders placed using different payment types.
SOln)

```
monthOnMonth = spark.sql(
  """

WITH cte AS (
    SELECT month(CAST(o.order_purchase_timestamp AS date)) AS purchase_month,
p.payment_type,
    COUNT(o.order_id) AS no_of_orders, LAG(month(CAST(o.order_purchase_timestamp
AS date)), 1) OVER(PARTITION
    BY p.payment_type ORDER BY COUNT(o.order_id) DESC) AS lagged_month
    FROM ordersView o
    INNER JOIN paymentView p
    ON o.order_id = p.order_id
    GROUP BY month(CAST(o.order_purchase_timestamp AS date)), p.payment_type
```

```
    )
    SELECT purchase_month,payment_type,no_of_orders FROM cte
    order by payment_type ASC,no_of_orders ASC
    """
)

monthOnMonthDF = ps.DataFrame(monthOnMonth)
monthOnMonthDF
```

Output:

Out[33]:

	purchase_month	payment_type	no_of_orders
0	9	UPI	903
1	10	UPI	1056
2	12	UPI	1160
3	11	UPI	1509
4	1	UPI	1715
5	2	UPI	1723
6	4	UPI	1783
7	6	UPI	1807
8	3	UPI	1942
9	5	UPI	2035
10	7	UPI	2074
11	8	UPI	2077
12	9	credit_card	3286
13	10	credit_card	3778
14	12	credit_card	4378
15	11	credit_card	5897
16	1	credit_card	6103
17	2	credit_card	6609
18	6	credit_card	7276
19	4	credit_card	7301
20	3	credit_card	7707

#6.2) Find the no. of orders placed on the basis of the payment installments that have been paid.
SOln)

```
paymentInstallments = spark.sql(
   """
         SELECT p.payment_installments,COUNT(o.order_id) as no_of_orders
                  FROM ordersView o
                  INNER JOIN paymentView p
                  ON o.order_id = p.order_id
      GROUP BY p.payment_installments
                  ORDER BY p.payment_installments
   """
)
paymentInstallmentsDF = ps.DataFrame(paymentInstallments)
paymentInstallmentsDF.head(30)
```

Output:

Out[34]:

	payment_installments	no_of_orders
0	0	2
1	1	52546
2	10	5328
3	11	23
4	12	133
5	13	16
6	14	15
7	15	74
8	16	5
9	17	8
10	18	27
11	2	12413

#Miscellaneous

#1)To find the count the number of orders which were paid using different methods

```
cards = spark.sql(
   """
         SELECT p.payment_type as payment_type,COUNT(p.order_id) as count
                  FROM paymentView p
                  INNER JOIN ordersView o
                  ON p.order_id = o.order_id
                  GROUP BY p.payment_type
                  ORDER BY count DESC
   """
)
cardsDF = ps.DataFrame(cards)
cardsDF
```

Output:

Out[43]:

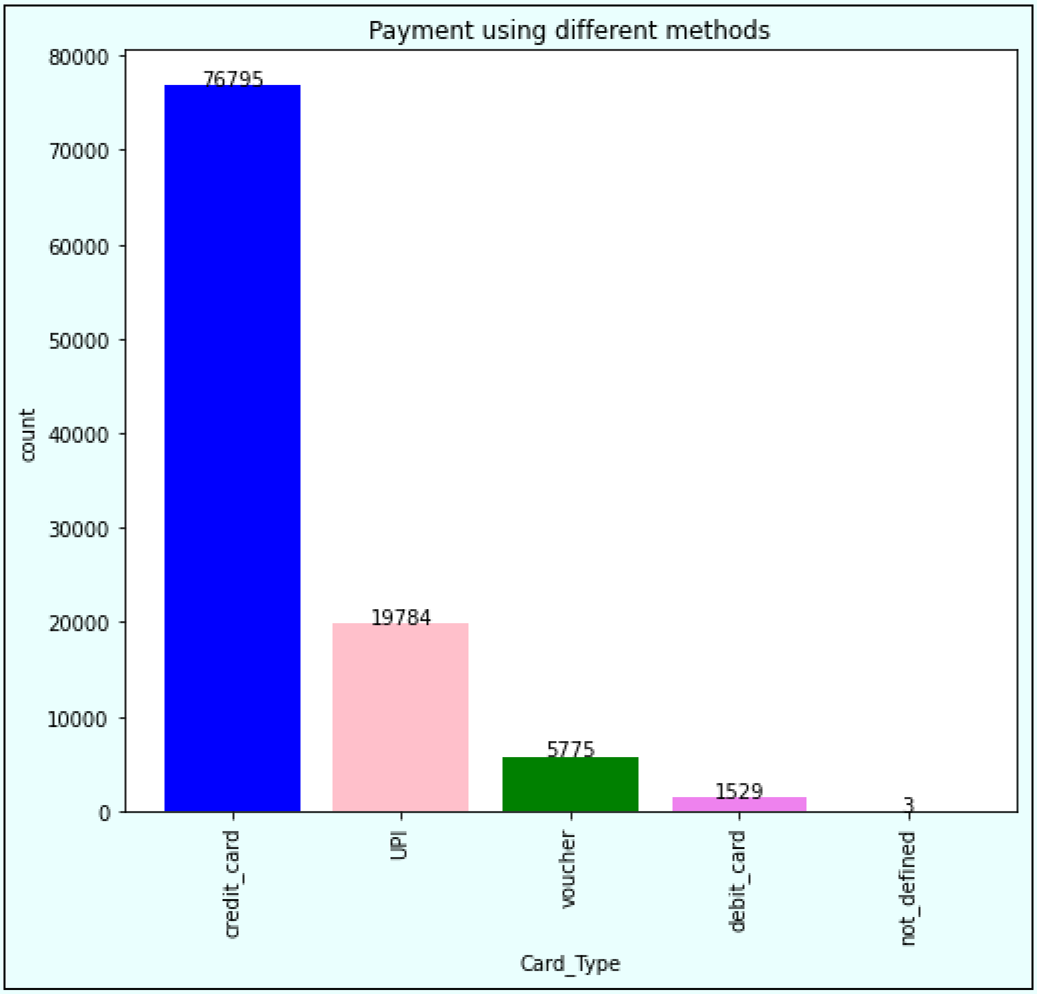
	payment_type	count
0	credit_card	76795
1	UPI	19784
2	voucher	5775
3	debit_card	1529
4	not_defined	3

Graphical Representation

```
color = ["blue","pink","green","violet","gray"]
Card_Type = cardsDF['payment_type'].to_numpy()
count = cardsDF['count'].to_numpy()
plt.figure(figsize=(8,7))
```

```
plt.bar(Card_Type, count,color = color)
for i, count in enumerate(cardsDF['count'].to_numpy()):
   plt.text(i, count + 0.5, str(count), ha='center')
plt.xlabel('Card_Type')
plt.ylabel('count')
plt.title('Payment using different methods')
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```

Output:



#2) Count of customers whose orders are delivered,shipped etc

SOln)

```
status = spark.sql(
    """
    SELECT o.order_status,COUNT(c.customer_id) as count_of_customers
            FROM ordersView o
            INNER JOIN customerView c
            ON o.customer_id = c.customer_id
            GROUP BY o.order_status
    ORDER BY count_of_customers DESC
    """
)
statusDF = ps.DataFrame(status)
statusDF
```

Output:

Out[109]:

	order_status	count_of_customers
0	delivered	96478
1	shipped	1107
2	canceled	625
3	unavailable	609
4	invoiced	314
5	processing	301
6	created	5
7	approved	2

#3) Count of products available in each category

SOln)

```
productCategory = spark.sql("""
SELECT product_category,COUNT(product_id) as count
FROM productsView
GROUP BY product_category
ORDER BY count DESC
""")
productCategoryDF = ps.DataFrame(productCategory)
productCategoryDF.head(75)
```

Output:

Out[104]:

	product_category	count
0	bed table bath	3029
1	sport leisure	2867
2	Furniture Decoration	2657
3	HEALTH BEAUTY	2444
4	housewares	2335
5	automotive	1900
6	computer accessories	1639
7	toys	1411
8	Watches present	1329
9	telephony	1134

#4) To find the count of seller cities in the seller states
Soln)

```
seller_city_count = spark.sql(
  """
  SELECT seller_state,COUNT(seller_city) as count_city
        FROM SellersView
        GROUP BY seller_state
```

```
        ORDER BY count_city DESC
  """
)
seller_city_countDF = ps.DataFrame(seller_city_count)
seller_city_countDF
```

Output:

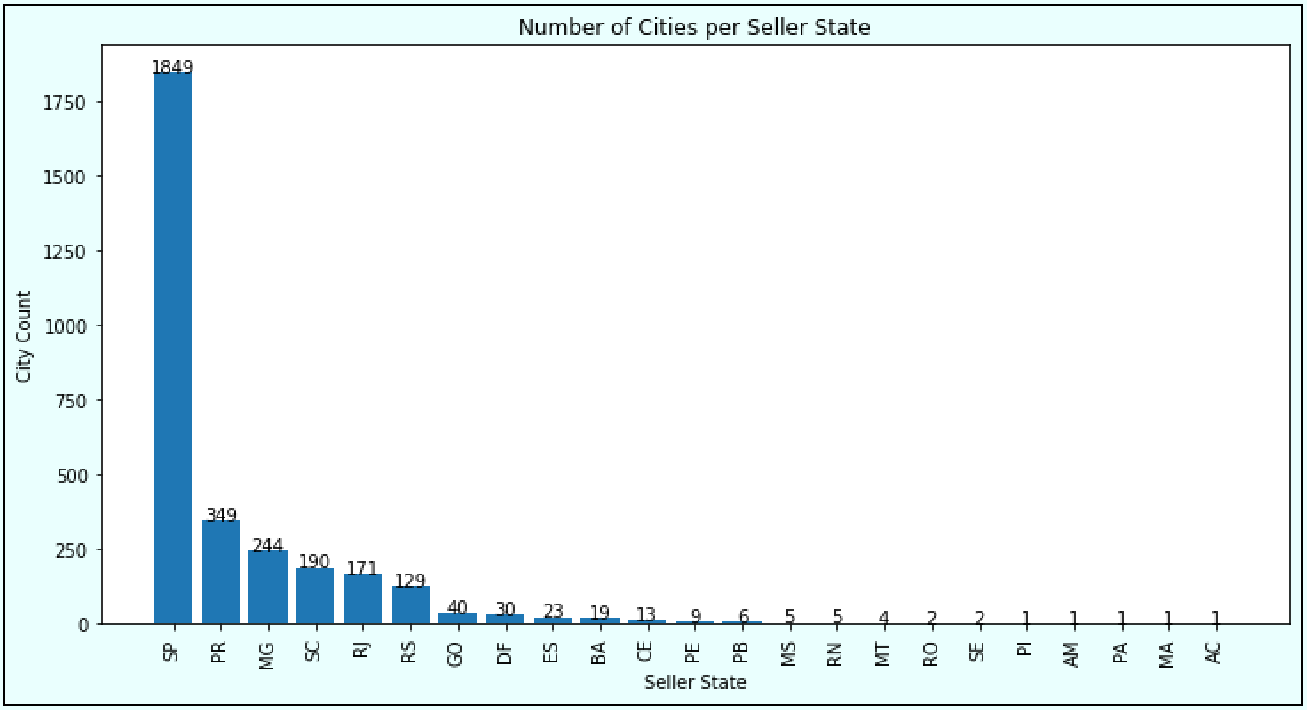
Out[50]:

	seller_state	count_city
0	SP	1849
1	PR	349
2	MG	244
3	SC	190
4	RJ	171
5	RS	129
6	GO	40
7	DF	30
8	ES	23
9	BA	19
10	CE	13
11	PE	9

Graphical Representation

```
plt.figure(figsize=(12, 6))
plt.bar(seller_city_countDF['seller_state'].to_numpy(),
seller_city_countDF['count_city'].to_numpy())

for i, count in enumerate(seller_city_countDF['count_city'].to_numpy()):
  plt.text(i, count + 0.5, str(count), ha='center')

plt.xlabel('Seller State')
```

```
plt.ylabel('City Count')
plt.title('Number of Cities per Seller State')

plt.xticks(rotation=90)

plt.show()
```



#5) THe count of order ids which got highest review score

SOln)

```
score = spark.sql(
   """
   SELECT review_score,COUNT(o.order_id) as count_of_orders
         FROM order_reviewsView r
         INNER JOIN ordersView o
         ON r.order_id = o.order_id
   GROUP BY review_score
   ORDER BY r.review_score ASC
   """
```

```
)
scoreDF = ps.DataFrame(score)
scoreDF
```

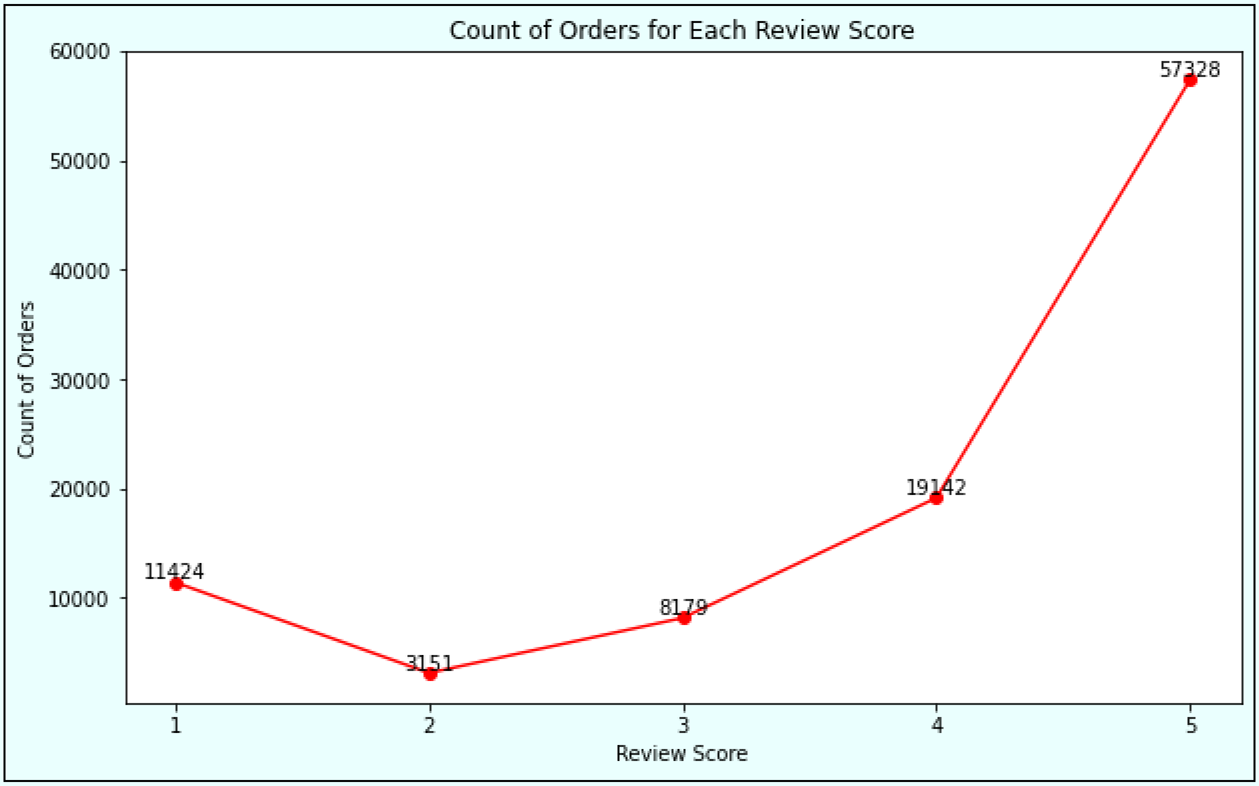
Output

Out[76]:

	review_score	count_of_orders
0	1	11424
1	2	3151
2	3	8179
3	4	19142
4	5	57328

Graphical Representation

```
plt.figure(figsize=(10, 6))
line_color = "red"
plt.plot(scoreDF['review_score'].to_numpy(), scoreDF['count_of_orders'].to_numpy(),
marker='o', linestyle='-',color = line_color)

for x, y in zip(scoreDF['review_score'].to_numpy(), scoreDF['count_of_orders'].to_numpy()):
    plt.text(x, y, str(y), ha='center', va='bottom')

plt.xlabel('Review Score')
plt.ylabel('Count of Orders')
plt.title('Count of Orders for Each Review Score')

# Display the plot
plt.show()
```



#6) Products which have the highest volume

```
volume = spark.sql("""
        WITH CTE AS
        (
                SELECT
product_id,product_length_cm,product_width_cm,product_height_cm,
                product_length_cm*product_width_cm*product_height_cm  as volume
     FROM productsView
        )

        SELECT product_id,volume
        FROM CTE
  ORDER BY volume DESC
""")
volumeDF = ps.DataFrame(volume)
```

volumeDF

Output

Out[99]:

	product_id	volume
0	256a9c364b75753b97bee410c9491ad8	296208.0
1	0b48eade13cfad433122f23739a66898	294000.0
2	f227e2d44f10f7dad30fb4dfa839e7a2	294000.0
3	3eb14e65e4208c6d94b7a32e41add538	294000.0
4	c1e0531cb1864fd3a0cae57dca55ca80	294000.0
5	90c1b4e040d1d1c45897ec2dad4a809d	293706.0
6	c6fdec160d0f8f488d9041316c85051d	288000.0
7	8d6f2c3454002d3f5aa7479a7fad7794	288000.0
8	99ff40856c47a638df807c0a144470cc	288000.0
9	0e9dfb804bafa3d68ef3ee7a621abfb2	287980.0
10	ab495f166205a883ffe5ab0b5b55f867	285138.0

#PRinting the maximum and minimum value of volume

```
max_volume_product = volumeDF.loc[volumeDF['volume'].idxmax()]
min_volume_product = volumeDF.loc[volumeDF['volume'].idxmin()]

print("The product with id", max_volume_product['product_id'], "has the maximum volume of", max_volume_product['volume'])
print("The product with id", min_volume_product['product_id'], "has the minimum volume of", min_volume_product['volume'])
```

Output:

The product with id 256a9c364b75753b97bee410c9491ad8 has the maximum volume of 296208.0

The product with id 106392145fca363410d287a815be6de4 has the minimum volume of 168.0

#8)To find which payment type had the highest installment

SOln)

```
installment = spark.sql("""
        SELECT payment_type,MAX(payment_installments) as count
        FROM paymentView
        GROUP BY payment_type
  ORDER BY count DESC
""")
installmentDF = ps.DataFrame(installment)
installmentDF
```

Output

Out[105]:

	payment_type	count
0	credit_card	9
1	not_defined	1
2	voucher	1
3	debit_card	1
4	UPI	1

#9)Count of people who gave reviews

SOln)

```
review = spark.sql("""SELECT * FROM order_reviewsView""")

review_countDF = review.toPandas()

extractDF = review_countDF["review_comment_title"].value_counts()
solnDF = pd.DataFrame(extractDF)
```

solnDF.head(40)

Output:

Out[138]:

review_comment_title	count
I recommend	1063
Good	609
Very good	479
Great	467
�In	457
Super recommend	358
super recommend	282
Excellent	200
10	162
excellent	120
Perfect	102
� on	97
very good	87
�In product	79
great	76
Good product	72
Satisfied	70
OK	69
Delivery R� � Pida	69
I RECOMMEND	65

#10) To print all schemas in one go

Soln)

```
dataFrames = {
        'customer':customer,
        'geolocation':geolocation,
        'order_items':order_items,
        'order_reviews':order_reviews,
        'orders':orders,
        'payments':payments,
        'products':products,
        'sellers':sellers
}
for table,dataframe in dataFrames.items():
   print(f"Schema for {table}:")
   dataframe.printSchema()
   print()
```

Output

```
Schema for customer:
root
 |-- customer_id: string (nullable = true)
 |-- customer_unique_id: string (nullable = true)
 |-- customer_zip_code_prefix: string (nullable = true)
 |-- customer_city: string (nullable = true)
 |-- customer_state: string (nullable = true)


Schema for geolocation:
root
 |-- geolocation_zip_code_prefix: string (nullable = true)
 |-- geolocation_lat: string (nullable = true)
 |-- geolocation_lng: string (nullable = true)
 |-- geolocation_city: string (nullable = true)
 |-- geolocation_state: string (nullable = true)


Schema for order_items:
root
 |-- order_id: string (nullable = true)
 |-- order_item_id: string (nullable = true)
 |-- product_id: string (nullable = true)
 |-- seller_id: string (nullable = true)
 |-- shipping_limit_date: string (nullable = true)
 |-- price: string (nullable = true)
 |-- freight_value: string (nullable = true)


Schema for order_reviews:
root
 |-- review_id: string (nullable = true)
 |-- order_id: string (nullable = true)
 |-- review_score: string (nullable = true)
 |-- review_comment_title: string (nullable = true)
 |-- review_creation_date: string (nullable = true)
 |-- review_answer_timestamp: string (nullable = true)


Schema for orders:
root
 |-- order_id: string (nullable = true)
 |-- customer_id: string (nullable = true)
 |-- order_status: string (nullable = true)
 |-- order_purchase_timestamp: string (nullable = true)
 |-- order_approved_at: string (nullable = true)
 |-- order_delivered_carrier_date: string (nullable = true)
```

Insights Generated

1) Based on the results obtained it can be seen the there is an increase in the number of orders placed from 4 to 324 and then dropped to a minimal of 1 in december
2) From 2017 the number of orders shows a general increasing trend from January to November, with slight fluctuations in between.
3) IN 2018 the number of orders starts high in January and remains relatively stable until May. After May,there is a decreasing trend in order volume.
4) The highest number of purchases are from people who are there in São Paulo (SP) of Brazil and the least are from Roraima (RR).
5) Most of the purchase are made in afternoon (38135) and the least no of purchases are made in the dawn (5242)
6) monthly seasonality refers to the regular and predictable fluctuations in certain variables or phenomena that can be observed from one month to another.
7) From the above output In september the no of orders placed in september 2017 was more when compared to 2016 and 2018 In october the no of orders placed in october 2017 was more when compared to 2016 and 2018 In november it was only 2017 In december 2017 had the highest no of orders placed
8) Also the highest count of products are for bed table bath and the least count of products are for cds music dvd
9) Most of the people make use of credit cards for their purchases and 3 of them haven’t defined their payment which can be seen through the bar graphs
10) The dataset sellers has the highest count of cities from Sao Paulo which means many people from Sao Paulo sell goods and the least count of sellers are from Piauí (PI), Manaus (AM), Acre (AC), Maranhão (MA)

11) Many order_ids have been given a review_score of 5 (57328) and very less order_ids have been given a review_score of 2 (3151)
12) The product with id 256a9c364b75753b97bee410c9491ad8 has the maximum volume of 296208.0
13) The product with id 106392145fca363410d287a815be6de4 has the minimum volume of 168.0
14) Credit card had the highest installment in it
15) The count of customers to whom the products are delivered has the highest count 9678.
16) The count of bed table bath products is the highest among all and it can be seen that more revenue is generated from it
17) San Paulo has the highest number of seller cities which means that most of the selling of products takes place in San paulo. The conditions are favorable for profitable trade of products in San Paulo
18) About 57328 orders have been given the highest rating of 5 which means the type of goods which are sent are of best value

Recommendations

1) Focus on boosting sales during the months of September, October, and December, as these months have shown higher order volumes compared to other years.
2) Attention has to be given to specific regions with lower order, such as Roraima (RR), and explore strategies to increase sales in those areas. This could involve tailoring marketing efforts, offering promotions, or partnering with local influencers or businesses to boost visibility such as advertisements.
3) Since most purchases are made in the afternoon, consider optimizing your customer service and support availability during these hours.

4) Analyze the product catalog to identify the top-selling products, such as bed table bath items, and allocate resources to further enhance their availability, marketing, and customer experience.
5) Develop targeted marketing strategies to promote the use of credit cards for purchases. Highlight the benefits of using credit cards, such as rewards programs, or additional security measures.
6) Focus on improving customer reviews and satisfaction. While a majority of orders have received a review score of 5, try to enhance the overall customer experience to maintain high levels of satisfaction. Monitor customer feedback and promptly address any issues or concerns to ensure that customers feel valued and have a positive impression on the brand.