Step1: Importing libraries

```
import matplotlib.pyplot as plt

from pyspark.sql.functions import log, ntile, col, when

from pyspark.sql import functions as F

from pyspark.sql.window import Window

from mpl_toolkits.mplot3d import Axes3D

import pandas as pd

import numpy as np
```

Step2: Importing file

```
df = spark.read.csv("s3://humber-lfb-databricks-class-files/sales_06_FY2020_21.csv", header=True, inferSchema=True)

display(df)
```

Table

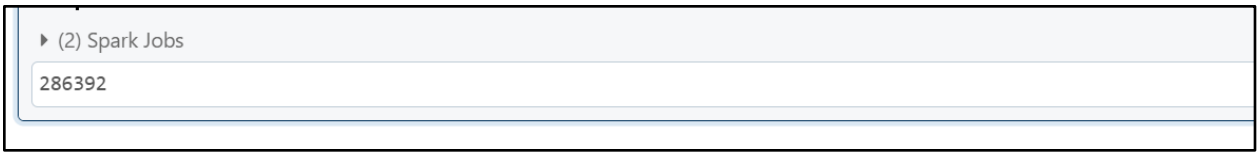
	order_id	order_date	status	item_id	sku	qty_ordered
1	100354678	2020-10-01	received	574772	oasis_Oasis-064-36	2
2	100354678	2020-10-01	received	574774	Fantastic_FT-48	1
3	100354680	2020-10-01	complete	574777	mdeal_DMC-610-8	
4	100354680	2020-10-01	complete	574779	oasis_Oasis-061-36	
5	100367357	2020-11-13	received	595185	MEFNAR59C38B6CA08CD	
6	100367357	2020-11-13	received	595186	MEFBUY59B7C3DDC2CA3-42	
7	100367360	2020-11-13	order_refunded	595192	MATDAN59C3C845B38F0	
8	100354677	2020-10-01	canceled	574769	GFE_19_USBLEDLight	
9	100354677	2020-10-01	canceled	574770	oasis_Kingston-32GB-DTIG4	
10	100354677	2020-10-01	canceled	574771	Geepas_GSB5420	
11	100356116	2020-10-08	order_refunded	577467	Geepas_GSB5420	
12	100358724	2020-10-21	order_refunded	581862	APPGEE59ADBEEE2EB9D	
13	100403034	2020-12-24	canceled	656937	APPWES59CA4D59CB41A	
14	100403034	2020-12-24	canceled	656938	APPPHI5A09939191938	

5,752+ rows | Truncated data due to byte limit | 18.75 seconds runtime

Refreshed 22 hours ago

Step3: Display row counts

```
display(df.count())
```

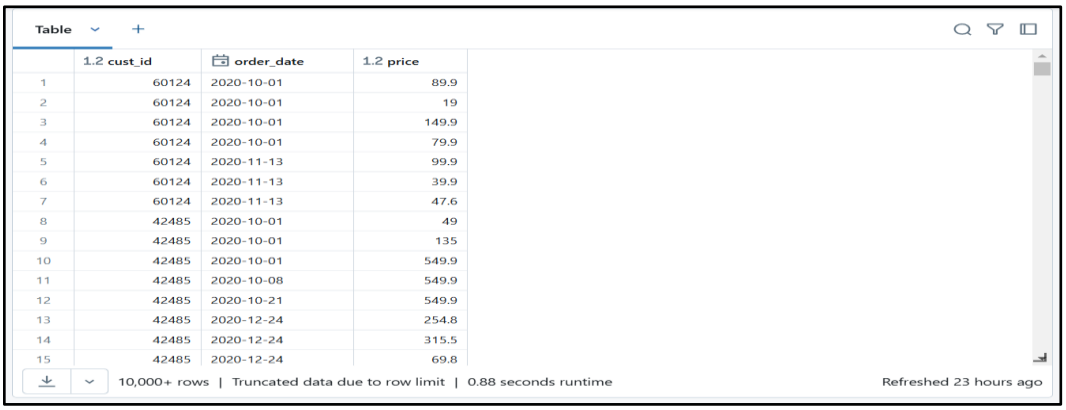


Step4: Display unique rows

```
%python
distinct_count = df.distinct().count()
display(distinct_count)
```

Step5: Create a new table "cus_info" with columns "cust_id", "order_date","price"

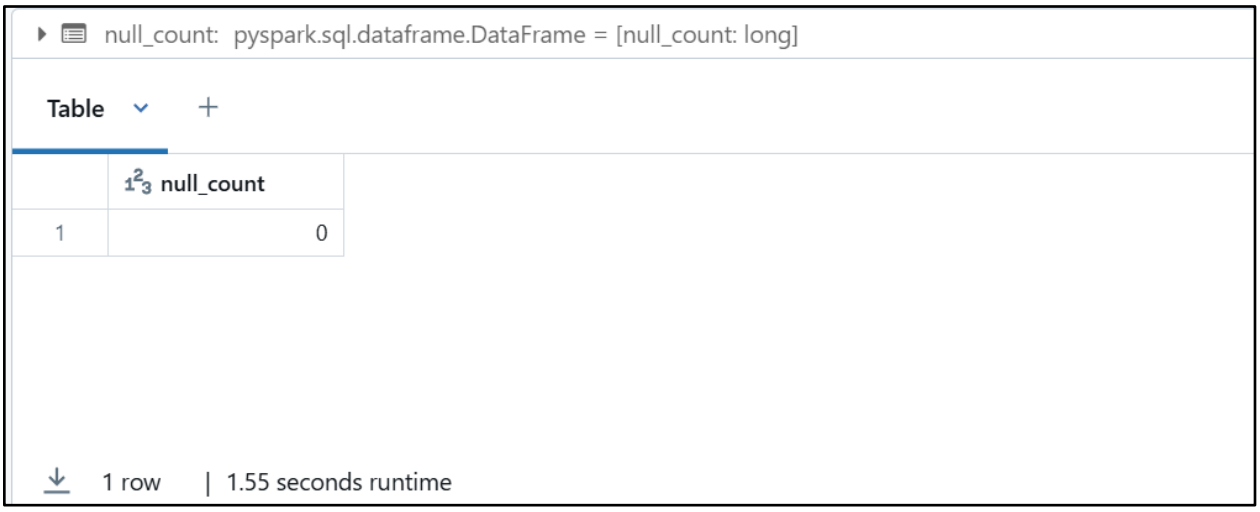
```
df.select("cust_id", "order_date", "price").createOrReplaceTempView("cus_info")
display(spark.sql("SELECT * FROM cus_info"))
```



Step6: Checking for missing data

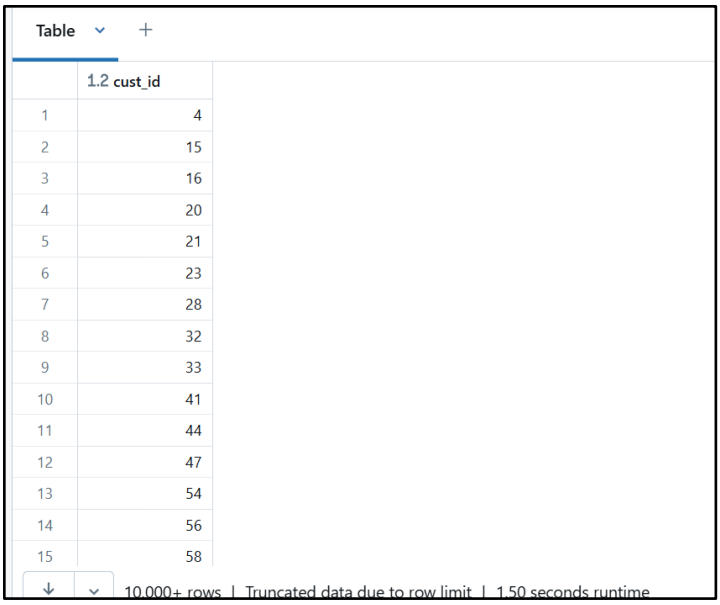
```
null_count = spark.sql("""
    SELECT
        COUNT(*) AS null_count
```

```
    FROM
        cus_info
    WHERE
        cust_id IS NULL OR
        order_date IS NULL OR
        price IS NULL
""")
display(null_count)
```



Step7: Creating new table "rfm_info" with unique cust_id from "cus_info"

```
rfm_info = spark.sql("SELECT DISTINCT cust_id FROM cus_info ORDER BY cust_id ASC")
rfm_info.createOrReplaceTempView("rfm_info")
display(spark.sql("SELECT * FROM rfm_info"))
```



Step8: Creating a new column recency in the table "rfm_info"

```
rfm_info = spark.sql("""
    SELECT
        r.cust_id,
        MAX(c.order_date) AS Recency
    FROM
        rfm_info r
    JOIN
        cus_info c
    ON
        r.cust_id = c.cust_id
    GROUP BY
        r.cust_id
    ORDER BY
        r.cust_id ASC
""")
```

```
rfm_info.createOrReplaceTempView("rfm_info")

display(spark.sql("SELECT * FROM rfm_info"))
```

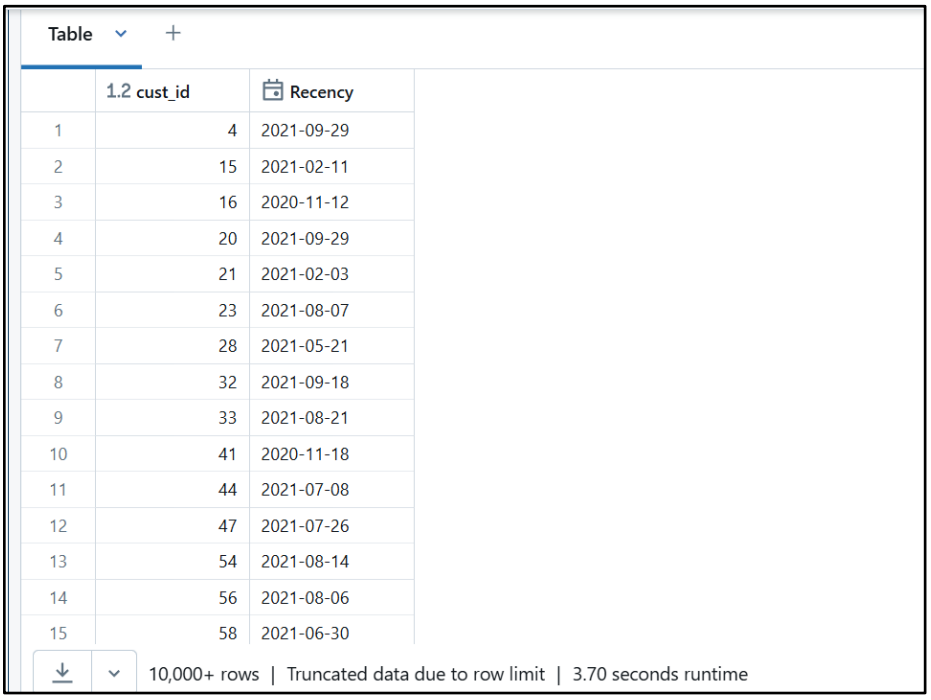


Table +

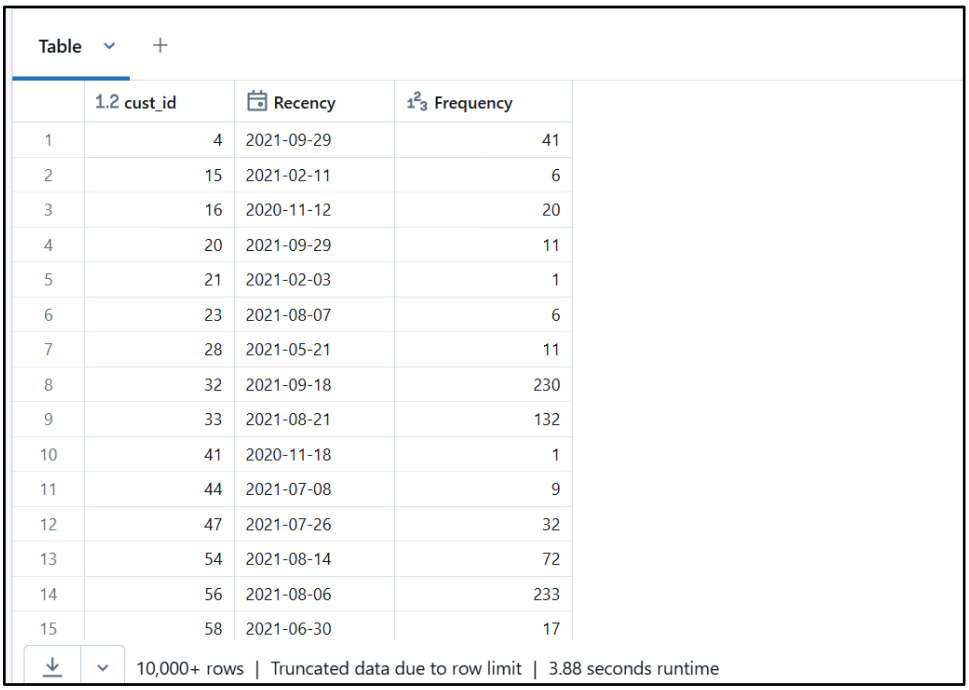
	1.2 cust_id	Recency
1	4	2021-09-29
2	15	2021-02-11
3	16	2020-11-12
4	20	2021-09-29
5	21	2021-02-03
6	23	2021-08-07
7	28	2021-05-21
8	32	2021-09-18
9	33	2021-08-21
10	41	2020-11-18
11	44	2021-07-08
12	47	2021-07-26
13	54	2021-08-14
14	56	2021-08-06
15	58	2021-06-30

10,000+ rows | Truncated data due to row limit | 3.70 seconds runtime

Step9: Creating a new column for Frequency in the table "rfm_info"

```
rfm_info = spark.sql("""
    SELECT
        r.cust_id,
        MAX(c.order_date) AS Recency,
        COUNT(c.cust_id) AS Frequency
    FROM
        rfm_info r
    JOIN
        cus_info c
    ON
```

```
    r.cust_id = c.cust_id
GROUP BY
    r.cust_id
ORDER BY
    r.cust_id ASC
""")
rfm_info.createOrReplaceTempView("rfm_info")
display(spark.sql("SELECT * FROM rfm_info"))
```



Table

	1.2 cust_id	Recency	1²3 Frequency
1	4	2021-09-29	41
2	15	2021-02-11	6
3	16	2020-11-12	20
4	20	2021-09-29	11
5	21	2021-02-03	1
6	23	2021-08-07	6
7	28	2021-05-21	11
8	32	2021-09-18	230
9	33	2021-08-21	132
10	41	2020-11-18	1
11	44	2021-07-08	9
12	47	2021-07-26	32
13	54	2021-08-14	72
14	56	2021-08-06	233
15	58	2021-06-30	17

10,000+ rows | Truncated data due to row limit | 3.88 seconds runtime

Step10: Creating a new column Monetary in the table "rfm_info"

```
rfm_info = spark.sql("""
    SELECT
        r.cust_id,
        MAX(c.order_date) AS Recency,
```

```
    COUNT(c.cust_id) AS Frequency,
    SUM(c.price) AS Monetary
  FROM
    rfm_info r
  JOIN
    cus_info c
  ON
    r.cust_id = c.cust_id
  GROUP BY
    r.cust_id
  ORDER BY
    r.cust_id ASC
""")
rfm_info.createOrReplaceTempView("rfm_info")
display(spark.sql("SELECT * FROM rfm_info"))
```

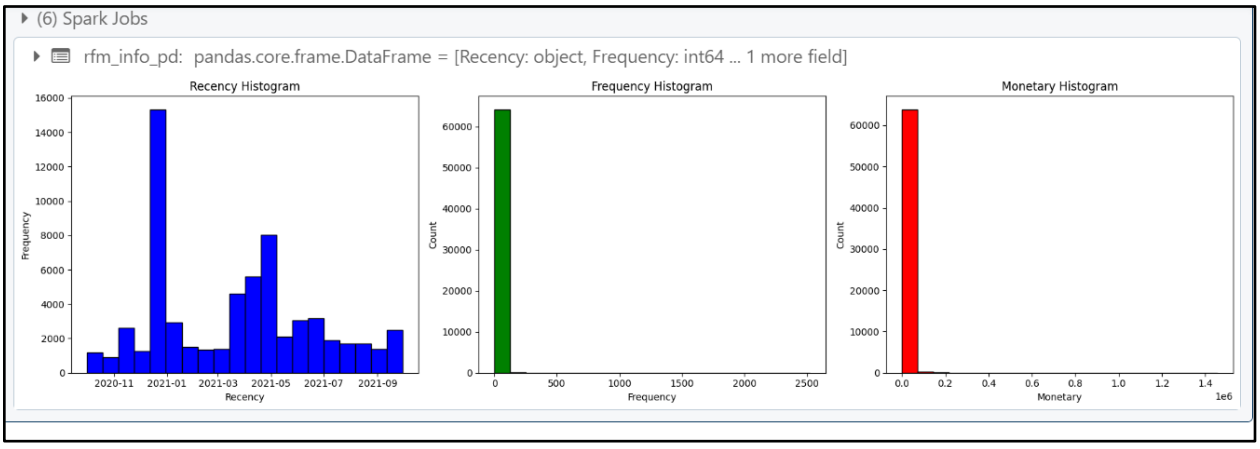
Table +

	1.2 cust_id	Recency	Frequency	1.2 Monetary
1	4	2021-09-29	41	47400.2999999999...
2	15	2021-02-11	6	198.3
3	16	2020-11-12	20	16106.9
4	20	2021-09-29	11	31594.7
5	21	2021-02-03	1	21
6	23	2021-08-07	6	677.14
7	28	2021-05-21	11	4335.3
8	32	2021-09-18	230	171271.899999999...
9	33	2021-08-21	132	82769.20000000007
10	41	2020-11-18	1	219.9
11	44	2021-07-08	9	13813.4999999999...
12	47	2021-07-26	32	25298.40000000001
13	54	2021-08-14	72	29055.9999999999...
14	56	2021-08-06	233	162885.7999999996
15	58	2021-06-30	17	16536.7399999999...

Step11: Generate histograms to visualize the distributions of "Recency," "Frequency," and "Monetary" metrics

```
# Convert Spark DataFrame to Pandas DataFrame for plotting

rfm_info_pd = rfm_info.select("Recency", "Frequency", "Monetary").toPandas()

# Plot histograms

fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Recency histogram

axes[0].hist(rfm_info_pd['Recency'], bins=20, color='blue', edgecolor='black')

axes[0].set_title('Recency Histogram')

axes[0].set_xlabel('Recency')

axes[0].set_ylabel('Frequency')

# Frequency histogram

axes[1].hist(rfm_info_pd['Frequency'], bins=20, color='green', edgecolor='black')

axes[1].set_title('Frequency Histogram')
```

```
axes[1].set_xlabel('Frequency')

axes[1].set_ylabel('Count')

# Monetary histogram

axes[2].hist(rfm_info_pd['Monetary'], bins=20, color='red', edgecolor='black')

axes[2].set_title('Monetary Histogram')

axes[2].set_xlabel('Monetary')

axes[2].set_ylabel('Count')

plt.tight_layout()

plt.show()
```



Recency Histogram:

- Most values are concentrated around specific dates, with a significant spike around early 2021. This suggests that many customers made recent purchases during that timeframe.
- Recency values are distributed unevenly, indicating varying activity levels among customers

Frequency Histogram:

- Many customers have very low purchase frequencies, indicating that most have made purchases only a few times.
- The distribution is highly skewed, with very few customers having high frequencies.

Monetary Histogram:

- Most monetary values are very small, indicating that the majority of customers spend relatively low.

- The graph shows an extreme right skew, with a few outliers contributing significantly higher amounts.

Step12 : Apply log transformation to normalize data in "rfm_info" table

```
rfm_info = rfm_info.withColumn("Log_Frequency", log("Frequency"))

rfm_info = rfm_info.withColumn("Log_Monetary", log("Monetary"))

rfm_info.createOrReplaceTempView("rfm_info")

display(spark.sql("SELECT * FROM rfm_info"))
```

Table +

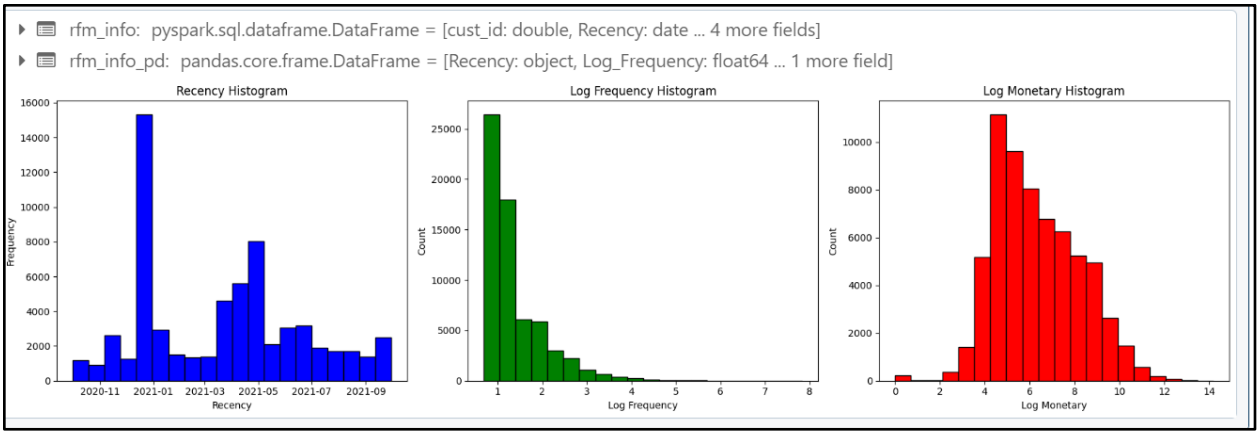
	1.2 cust_id	Recency	Frequency	1.2 Monetary	1.2 Log_Frequency	1.2 Log_Monetary
1	4	2021-09-29	41	47400.30000000002	3.713572066704308	10.766383836777063
2	15	2021-02-11	6	198.3	1.791759469228055	5.28978103552575
3	16	2020-11-12	20	16106.9000000000...	2.995732273553991	9.687003030590622
4	20	2021-09-29	11	31594.7000000000...	2.3978952727983707	10.36074466398919
5	21	2021-02-03	1	21	0	3.044522437723423
6	23	2021-08-07	6	677.14	1.791759469228055	6.517878046215625
7	28	2021-05-21	11	4335.299999999999	2.3978952727983707	8.374546090972395
8	32	2021-09-18	230	171271.899999999...	5.438079308923196	12.051007631159788
9	33	2021-08-21	132	82769.20000000001	4.882801922586371	11.32381129049505
10	41	2020-11-18	1	219.9	0	5.393172897560715
11	44	2021-07-08	9	13813.5	2.1972245773362196	9.533401653829532
12	47	2021-07-26	32	25298.4	3.4657359027997265	10.13849643160909
13	54	2021-08-14	72	29056	4.276666119016055	10.27698028140102
14	56	2021-08-06	233	162885.7999999999	5.4510384535657	12.000804620747635
15	58	2021-06-30	17	16536.7399999999...	2.833213344056216	9.713339851206953

10,000+ rows | Truncated data due to row limit | 3.65 seconds runtime Refreshed 23 hours a

Step 13: Visualize normalized data

```
# Add Log_Frequency and Log_Monetary columns

rfm_info = rfm_info.withColumn("Log_Frequency", F.log(F.col("Frequency") + 1))

rfm_info = rfm_info.withColumn("Log_Monetary", F.log(F.col("Monetary") + 1))

# Convert Spark DataFrame to Pandas DataFrame for plotting

rfm_info_pd = rfm_info.select("Recency", "Log_Frequency", "Log_Monetary").toPandas()

# Plot histograms

fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Recency histogram

axes[0].hist(rfm_info_pd['Recency'], bins=20, color='blue', edgecolor='black')
```

```
axes[0].set_title('Recency Histogram')
axes[0].set_xlabel('Recency')
axes[0].set_ylabel('Frequency')
# Log_Frequency histogram
axes[1].hist(rfm_info_pd['Log_Frequency'], bins=20, color='green', edgecolor='black')
axes[1].set_title('Log Frequency Histogram')
axes[1].set_xlabel('Log Frequency')
axes[1].set_ylabel('Count')
# Log_Monetary histogram
axes[2].hist(rfm_info_pd['Log_Monetary'], bins=20, color='red', edgecolor='black')
axes[2].set_title('Log Monetary Histogram')
axes[2].set_xlabel('Log Monetary')
axes[2].set_ylabel('Count')
plt.tight_layout()
plt.show()
```



Step 14: Renaming the columns

```
rfm_info = rfm_info.withColumnRenamed("Frequency", "Actual Frequency") \
                   .withColumnRenamed("Monetary", "Actual Monetary") \
```

```
        .withColumnRenamed("Log_Frequency", "Frequency") \
        .withColumnRenamed("Log_Monetary", "Monetary")
rfm_info.createOrReplaceTempView("rfm_info")
```

Step 15: Reordering columns

```
rfm_info = rfm_info.select("cust_id", "Recency", "Frequency", "Monetary", "Actual Frequency", "Actual Monetary")
```

Step 16: Calculate percentile rank and rank customers by Recency and assign them into any one of the four tiers based on their percentile rank

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window
# Calculate the quartiles for Recency
window = Window.orderBy(F.col("Recency").desc())
rfm_info = rfm_info.withColumn("Recency_rank", F.percent_rank().over(window))
# Assign Recency_tier based on quartiles
rfm_info = rfm_info.withColumn(
    "Recency_tier",
    F.when(F.col("Recency_rank") <= 0.25, "R-Tier-1")
    .when((F.col("Recency_rank") > 0.25) & (F.col("Recency_rank") <= 0.50), "R-Tier-2")
    .when((F.col("Recency_rank") > 0.50) & (F.col("Recency_rank") <= 0.75), "R-Tier-3")
    .otherwise("R-Tier-4")
).drop("Recency_rank")
rfm_info.createOrReplaceTempView("rfm_info")
display(spark.sql("SELECT * FROM rfm_info"))
```

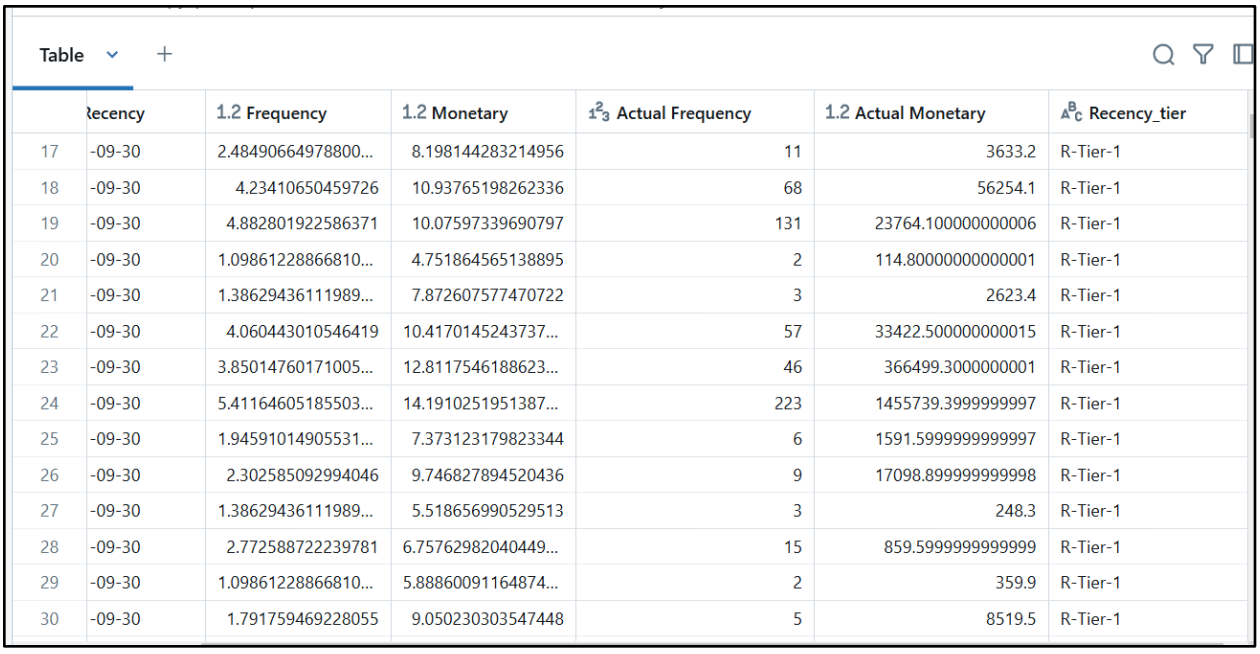
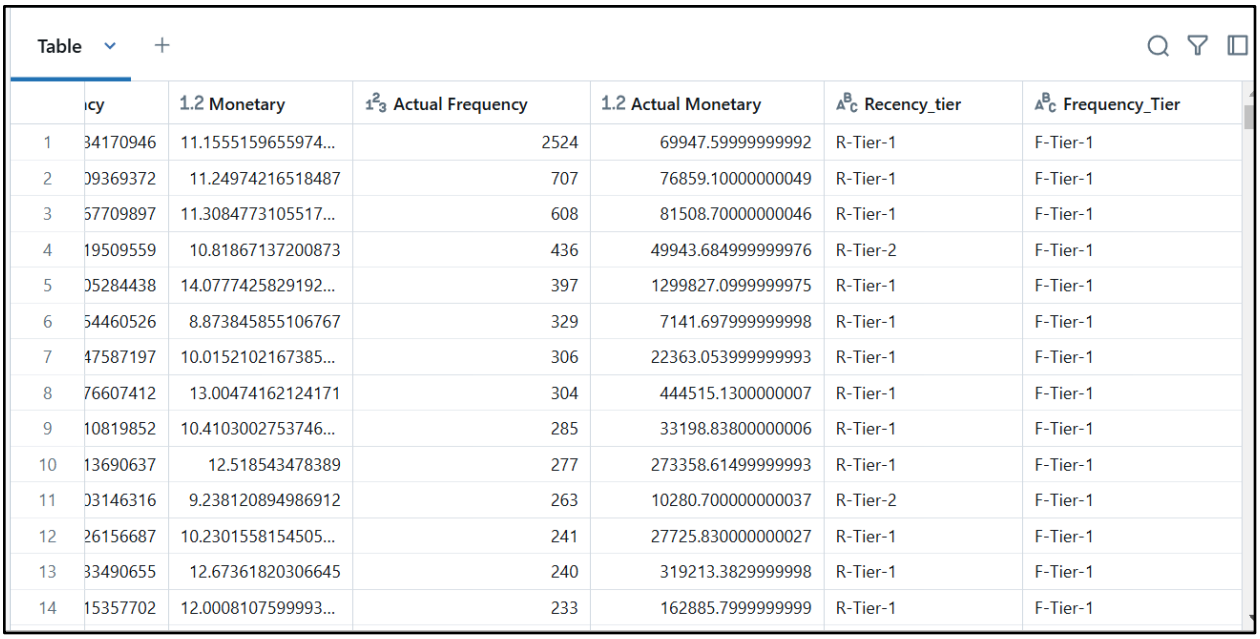


Table +

	ecency	1.2 Frequency	1.2 Monetary	1²3 Actual Frequency	1.2 Actual Monetary	ABC Recency_tier
17	-09-30	2.48490664978800...	8.198144283214956	11	3633.2	R-Tier-1
18	-09-30	4.23410650459726	10.93765198262336	68	56254.1	R-Tier-1
19	-09-30	4.882801922586371	10.07597339690797	131	23764.100000000006	R-Tier-1
20	-09-30	1.09861228866810...	4.751864565138895	2	114.80000000000001	R-Tier-1
21	-09-30	1.38629436111989...	7.872607577470722	3	2623.4	R-Tier-1
22	-09-30	4.060443010546419	10.4170145243737...	57	33422.500000000015	R-Tier-1
23	-09-30	3.85014760171005...	12.8117546188623...	46	366499.3000000001	R-Tier-1
24	-09-30	5.41164605185503...	14.1910251951387...	223	1455739.3999999997	R-Tier-1
25	-09-30	1.94591014905531...	7.373123179823344	6	1591.5999999999997	R-Tier-1
26	-09-30	2.302585092994046	9.746827894520436	9	17098.899999999998	R-Tier-1
27	-09-30	1.38629436111989...	5.518656990529513	3	248.3	R-Tier-1
28	-09-30	2.772588722239781	6.75762982040449...	15	859.5999999999999	R-Tier-1
29	-09-30	1.09861228866810...	5.88860091164874...	2	359.9	R-Tier-1
30	-09-30	1.791759469228055	9.050230303547448	5	8519.5	R-Tier-1

Step 17 : Divide customers into quartiles based on their purchase Frequency and assign them to any one of the four tiers

```
%python
from pyspark.sql.functions import ntile, col, when
from pyspark.sql.window import Window
# Define a window specification
window_spec = Window.orderBy(col("Frequency").desc())
# Add a new column "Frequency_Tier" with ntile function to divide into 4 quartiles
rfm_info = rfm_info.withColumn("Frequency_Tier", ntile(4).over(window_spec))
# Map the quartile numbers to the desired tier labels
rfm_info = rfm_info.withColumn("Frequency_Tier",
                               when(col("Frequency_Tier") == 1, "F-Tier-1")
                               .when(col("Frequency_Tier") == 2, "F-Tier-2")
                               .when(col("Frequency_Tier") == 3, "F-Tier-3")
 .otherwise("F-Tier-4"))
display(rfm_info)
```



Table

	ıcy	1.2 Monetary	1²3 Actual Frequency	1.2 Actual Monetary	ABC Recency_tier	ABC Frequency_Tier
1	34170946	11.1555159655974...	2524	69947.59999999992	R-Tier-1	F-Tier-1
2	09369372	11.24974216518487	707	76859.10000000049	R-Tier-1	F-Tier-1
3	57709897	11.3084773105517...	608	81508.70000000046	R-Tier-1	F-Tier-1
4	19509559	10.81867137200873	436	49943.684999999976	R-Tier-2	F-Tier-1
5	05284438	14.0777425829192...	397	1299827.0999999975	R-Tier-1	F-Tier-1
6	54460526	8.873845855106767	329	7141.697999999998	R-Tier-1	F-Tier-1
7	47587197	10.0152102167385...	306	22363.053999999993	R-Tier-1	F-Tier-1
8	76607412	13.00474162124171	304	444515.1300000007	R-Tier-1	F-Tier-1
9	10819852	10.4103002753746...	285	33198.83800000006	R-Tier-1	F-Tier-1
10	13690637	12.518543478389	277	273358.61499999993	R-Tier-1	F-Tier-1
11	03146316	9.238120894986912	263	10280.700000000037	R-Tier-2	F-Tier-1
12	26156687	10.2301558154505...	241	27725.830000000027	R-Tier-1	F-Tier-1
13	33490655	12.67361820306645	240	319213.3829999998	R-Tier-1	F-Tier-1
14	15357702	12.0008107599993...	233	162885.7999999999	R-Tier-1	F-Tier-1

Step 18 : Segment customers into four Monetary based tiers based on their spending

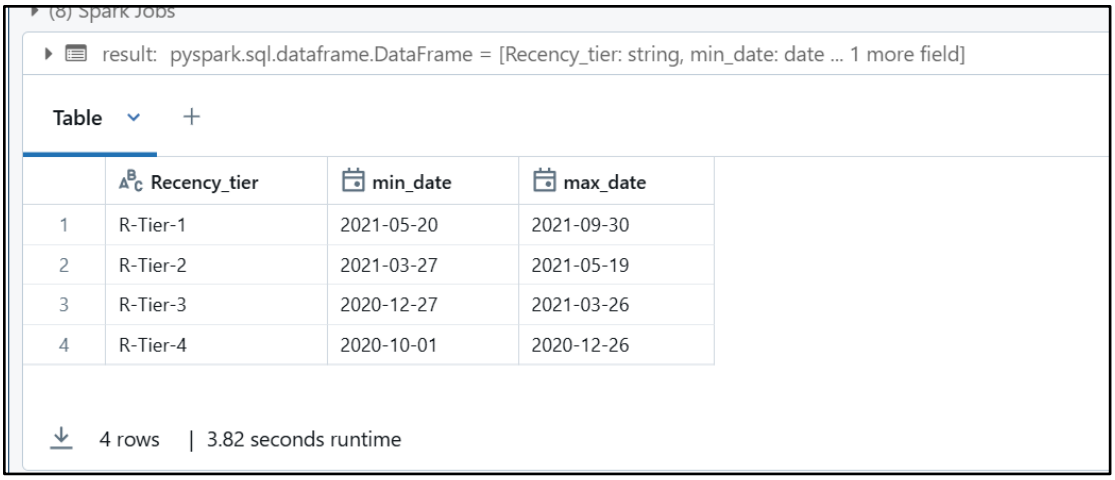
```
window = Window.orderBy(F.col("Monetary").desc())
rfm_info = rfm_info.withColumn("Monetary_rank", F.percent_rank().over(window))
# Assign Monetary_tier based on quartiles
rfm_info = rfm_info.withColumn(
    "Monetary_tier",
    F.when(F.col("Monetary_rank") <= 0.25, "M-Tier-1")
    .when((F.col("Monetary_rank") > 0.25) & (F.col("Monetary_rank") <= 0.50), "M-Tier-2")
    .when((F.col("Monetary_rank") > 0.50) & (F.col("Monetary_rank") <= 0.75), "M-Tier-3")
    .otherwise("M-Tier-4")
).drop("Monetary_rank")
rfm_info.createOrReplaceTempView("rfm_info")
display(spark.sql("SELECT * FROM rfm_info"))
```

Table +

		Actual Frequency	Actual Monetary	Recency_tier	Frequency_Tier	Monetary_tier
9	29207	116	593717.5000000008	R-Tier-1	F-Tier-1	M-Tier-1
10	788...	59	551750.6000000003	R-Tier-1	F-Tier-1	M-Tier-1
11	24171	304	444515.1300000007	R-Tier-1	F-Tier-1	M-Tier-1
12	30719	211	436156.2000000003	R-Tier-1	F-Tier-1	M-Tier-1
13	41009	47	432130.1000000003	R-Tier-1	F-Tier-1	M-Tier-1
14	255...	52	407645.2000000001	R-Tier-1	F-Tier-1	M-Tier-1
15	735...	36	401852.4	R-Tier-1	F-Tier-1	M-Tier-1
16	576...	47	398604.9000000001	R-Tier-1	F-Tier-1	M-Tier-1
17	354...	54	394199.3000000001	R-Tier-1	F-Tier-1	M-Tier-1
18	353...	123	391904.9000000002	R-Tier-1	F-Tier-1	M-Tier-1
19	285...	41	384152.9000000002	R-Tier-1	F-Tier-1	M-Tier-1
20	896...	136	372842.0000000003	R-Tier-1	F-Tier-1	M-Tier-1
21	819...	212	369709.199999999	R-Tier-1	F-Tier-1	M-Tier-1
22	623...	46	366499.3000000001	R-Tier-1	F-Tier-1	M-Tier-1

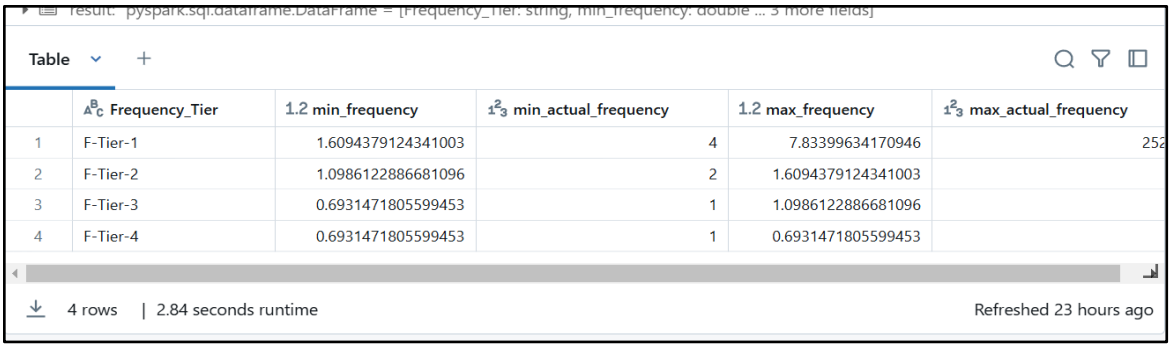
Step 19: Summarize the Recency distribution for each tier

```
query = """
SELECT
  Recency_tier,
  MIN(Recency) AS min_date,
  MAX(Recency) AS max_date
FROM rfm_info
GROUP BY Recency_tier
ORDER BY Recency_tier
"""
result = spark.sql(query)
display(result)
```



Step 20: Summarise the Frequency distribution for each tier

```
query = """
SELECT
  Frequency_Tier,
  MIN(Frequency) AS min_frequency,
  MIN(`Actual Frequency`) AS min_actual_frequency,
  MAX(Frequency) AS max_frequency,
  MAX(`Actual Frequency`) AS max_actual_frequency
FROM rfm_info
GROUP BY Frequency_Tier
ORDER BY Frequency_Tier
"""
result = spark.sql(query)
display(result)
```



result: pyspark.sql.dataframe.DataFrame = [Frequency_Tier: string, min_frequency: double ... 3 more fields]

Table

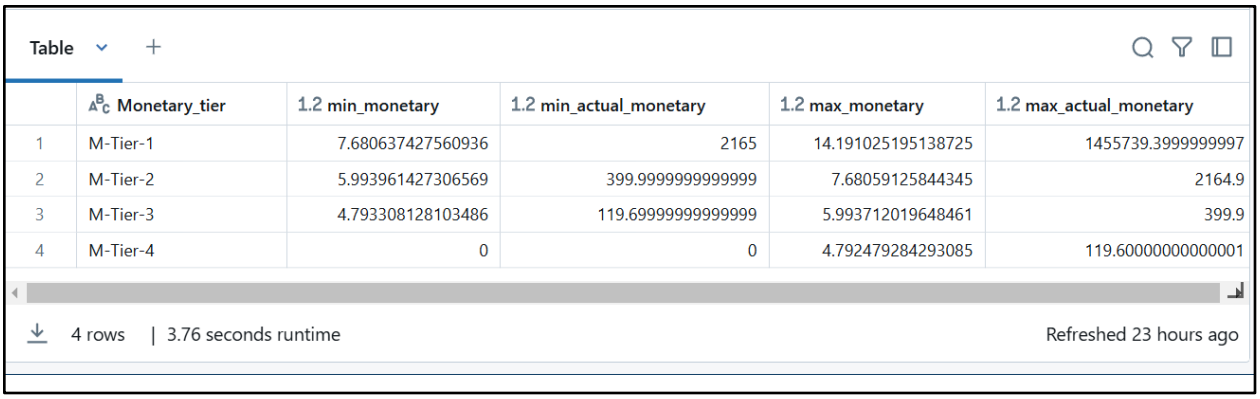
	Frequency_Tier	min_frequency	min_actual_frequency	max_frequency	max_actual_frequency
1	F-Tier-1	1.6094379124341003	4	7.83399634170946	252
2	F-Tier-2	1.0986122886681096	2	1.6094379124341003	
3	F-Tier-3	0.6931471805599453	1	1.0986122886681096	
4	F-Tier-4	0.6931471805599453	1	0.6931471805599453	

4 rows | 2.84 seconds runtime

Refreshed 23 hours ago

Step 21 : Summarize the Monetary distribution for each tier

```
%python

query = """

SELECT

  Monetary_tier,

  MIN(Monetary) AS min_monetary,

  MIN(`Actual Monetary`) AS min_actual_monetary,

  MAX(Monetary) AS max_monetary,

  MAX(`Actual Monetary`) AS max_actual_monetary

FROM rfm_info

GROUP BY Monetary_tier

ORDER BY Monetary_tier

"""

result = spark.sql(query)

display(result)
```



Table

	Monetary_tier	min_monetary	min_actual_monetary	max_monetary	max_actual_monetary
1	M-Tier-1	7.680637427560936	2165	14.191025195138725	1455739.3999999997
2	M-Tier-2	5.993961427306569	399.9999999999999	7.68059125844345	2164.9
3	M-Tier-3	4.793308128103486	119.69999999999999	5.993712019648461	399.9
4	M-Tier-4	0	0	4.792479284293085	119.60000000000001

4 rows | 3.76 seconds runtime

Refreshed 23 hours ago

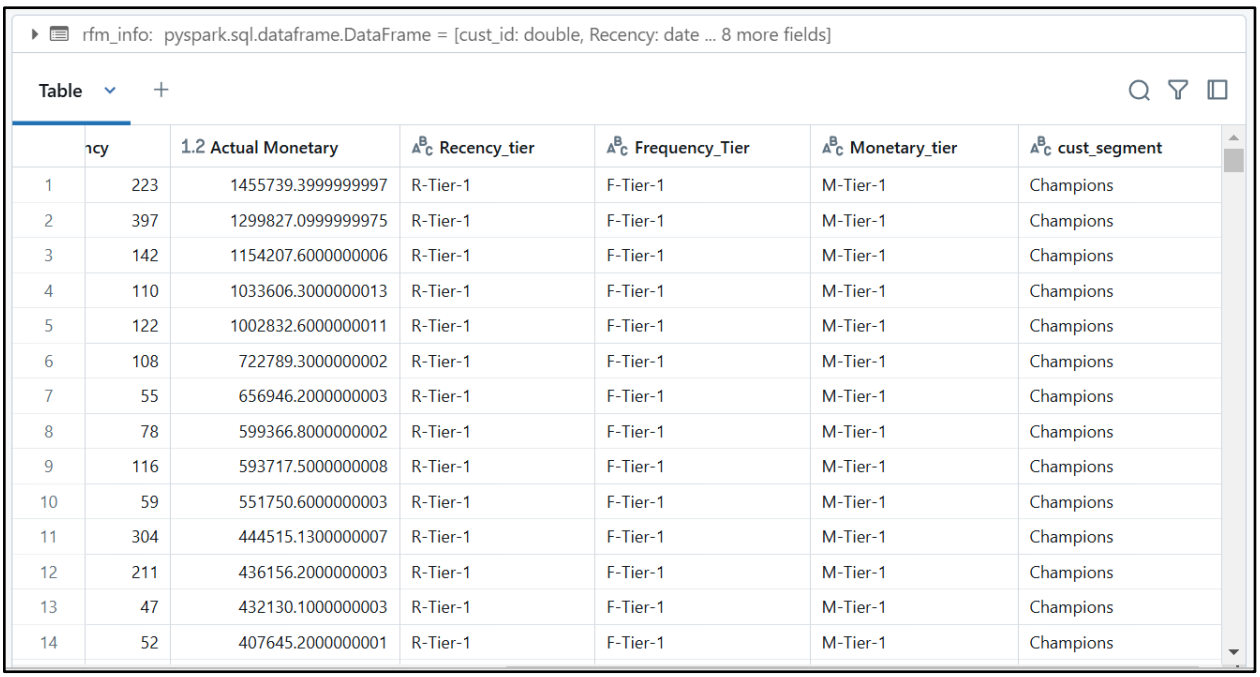
Step 22 : Divide customers into different segments based on RFM tiers

```
from pyspark.sql.functions import when

rfm_info = rfm_info.withColumn(

    "cust_segment",

    when(

        (rfm_info.Recency_tier == "R-Tier-1") &

        (rfm_info.Frequency_Tier == "F-Tier-1") &

        (rfm_info.Monetary_tier == "M-Tier-1"), "Champions"

    ).when(

        (rfm_info.Frequency_Tier == "F-Tier-1"), "Loyal Customers"

    ).when(

        (rfm_info.Recency_tier == "R-Tier-1") &

        ((rfm_info.Monetary_tier == "M-Tier-1") | (rfm_info.Monetary_tier == "M-Tier-2")),
"Potential loyalist"

    ).when(

        (rfm_info.Recency_tier == "R-Tier-1") &

        (rfm_info.Frequency_Tier == "F-Tier-3"), "New Customers"

    ).when(

        (rfm_info.Frequency_Tier == "F-Tier-1") &
```

```
        ((rfm_info.Monetary_tier == "M-Tier-2") | (rfm_info.Monetary_tier == "M-Tier-3")),
"Promising"

    ).when(

        (rfm_info.Recency_tier == "R-Tier-4") &

        (rfm_info.Frequency_Tier == "F-Tier-4") &

        (rfm_info.Monetary_tier == "M-Tier-4"), "Lost"

    ).when(

        (rfm_info.Recency_tier == "R-Tier-4") &

        ((rfm_info.Frequency_Tier == "F-Tier-3") | (rfm_info.Frequency_Tier == "F-Tier-4")) &

        ((rfm_info.Monetary_tier == "M-Tier-3") | (rfm_info.Monetary_tier == "M-Tier-4")),
"Hibernate"

    ).when(

        ((rfm_info.Recency_tier == "R-Tier-3") | (rfm_info.Recency_tier == "R-Tier-4")) &

        ((rfm_info.Frequency_Tier == "F-Tier-3") | (rfm_info.Frequency_Tier == "F-Tier-4")) &

        ((rfm_info.Monetary_tier == "M-Tier-1") | (rfm_info.Monetary_tier == "M-Tier-2")), "At
Risk"

    ).when(

        (rfm_info.Recency_tier == "R-Tier-4") &

        ((rfm_info.Frequency_Tier == "F-Tier-3") | (rfm_info.Frequency_Tier == "F-Tier-4")) &

        (rfm_info.Monetary_tier == "M-Tier-1"), "Can't loose them"

    ).when(

        ((rfm_info.Recency_tier == "R-Tier-3") | (rfm_info.Recency_tier == "R-Tier-4")) &

        ((rfm_info.Frequency_Tier == "F-Tier-2") | (rfm_info.Frequency_Tier == "F-Tier-3") |
(rfm_info.Frequency_Tier == "F-Tier-4")) &

        ((rfm_info.Monetary_tier == "M-Tier-2") | (rfm_info.Monetary_tier == "M-Tier-3") |
(rfm_info.Monetary_tier == "M-Tier-4")), "Needs attention"

    ).otherwise("Unknown")
```

)

```
rfm_info.createOrReplaceTempView("rfm_info")

display(spark.sql("SELECT * FROM rfm_info"))
```



rfm_info: pyspark.sql.dataframe.DataFrame = [cust_id: double, Recency: date ... 8 more fields]

Table

	ıcy	1.2 Actual Monetary	Recency_tier	Frequency_Tier	Monetary_tier	cust_segment
1	223	1455739.3999999997	R-Tier-1	F-Tier-1	M-Tier-1	Champions
2	397	1299827.0999999975	R-Tier-1	F-Tier-1	M-Tier-1	Champions
3	142	1154207.6000000006	R-Tier-1	F-Tier-1	M-Tier-1	Champions
4	110	1033606.3000000013	R-Tier-1	F-Tier-1	M-Tier-1	Champions
5	122	1002832.6000000011	R-Tier-1	F-Tier-1	M-Tier-1	Champions
6	108	722789.3000000002	R-Tier-1	F-Tier-1	M-Tier-1	Champions
7	55	656946.2000000003	R-Tier-1	F-Tier-1	M-Tier-1	Champions
8	78	599366.8000000002	R-Tier-1	F-Tier-1	M-Tier-1	Champions
9	116	593717.5000000008	R-Tier-1	F-Tier-1	M-Tier-1	Champions
10	59	551750.6000000003	R-Tier-1	F-Tier-1	M-Tier-1	Champions
11	304	444515.1300000007	R-Tier-1	F-Tier-1	M-Tier-1	Champions
12	211	436156.2000000003	R-Tier-1	F-Tier-1	M-Tier-1	Champions
13	47	432130.1000000003	R-Tier-1	F-Tier-1	M-Tier-1	Champions
14	52	407645.2000000001	R-Tier-1	F-Tier-1	M-Tier-1	Champions

Step 23: The RFM segmentation results for the customers with IDs: 60149, 2844, 60767 and 39707 (answer to the requirement)

```
query = """

SELECT *

FROM rfm_info

WHERE cust_id IN (60149, 2844, 60767, 39707)

"""

result = spark.sql(query)

display(result)
```

Table

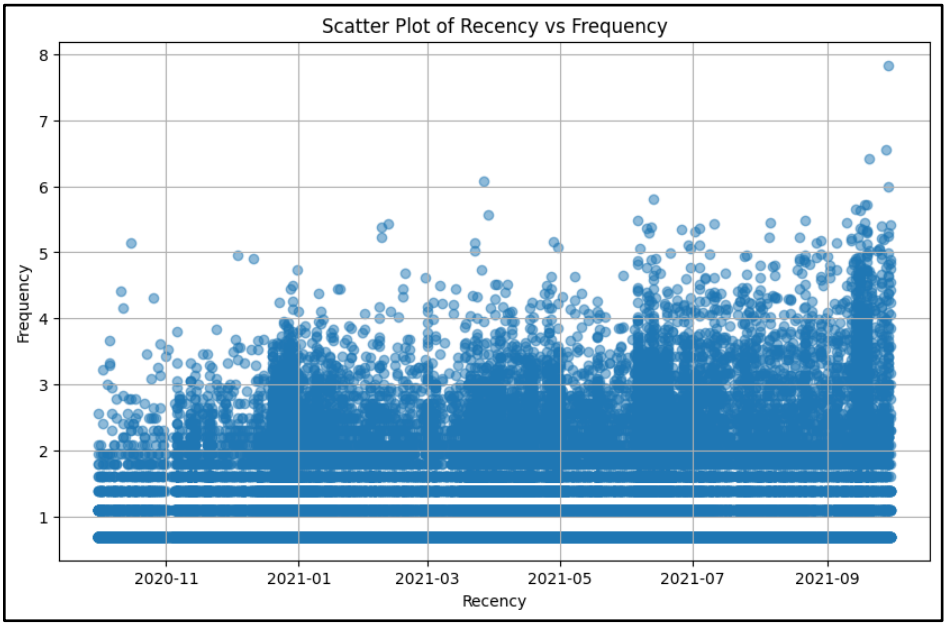
	1.2 cust_id	Recency	1.2 Frequency	1.2 Monetary
1	39707	2021-09-29	5.986452005284438	14.0777425829192...
2	2844	2021-09-29	4.532599493153256	12.2508648284123...
3	60767	2020-10-12	2.39789527279837...	11.1302145064806...
4	60149	2020-10-01	1.09861228866810...	6.187442430349423

Recency_tier	Frequency_Tier	Monetary_tier	cust_segment
R-Tier-1	F-Tier-1	M-Tier-1	Champions
R-Tier-1	F-Tier-1	M-Tier-1	Champions
R-Tier-4	F-Tier-1	M-Tier-1	Loyal Customers
R-Tier-4	F-Tier-3	M-Tier-2	At Risk

Step 24: Generate a scatter plot to visualize the relationship between the "Recency" and "Frequency"

```
# Select relevant columns

recency_frequency_df = rfm_info.select("Recency", "Frequency")

# Convert to Pandas DataFrame for plotting

recency_frequency_pd = recency_frequency_df.toPandas()

# Create scatter plot

plt.figure(figsize=(10, 6))

plt.scatter(recency_frequency_pd["Recency"], recency_frequency_pd["Frequency"], alpha=0.5)

plt.title("Scatter Plot of Recency vs Frequency")

plt.xlabel("Recency")

plt.ylabel("Frequency")

plt.grid(True)
```

plt.show()



Step 25 : Generate a scatter plot to visualize the relationship between Frequency and Monetary values

Select relevant columns

frequency_monetary_df = rfm_info.select("Frequency", "Monetary")

Convert to Pandas DataFrame for plotting

frequency_monetary_pd = frequency_monetary_df.toPandas()

Create scatter plot

plt.figure(figsize=(10, 6))

plt.scatter(frequency_monetary_pd["Frequency"], frequency_monetary_pd["Monetary"], alpha=0.5)

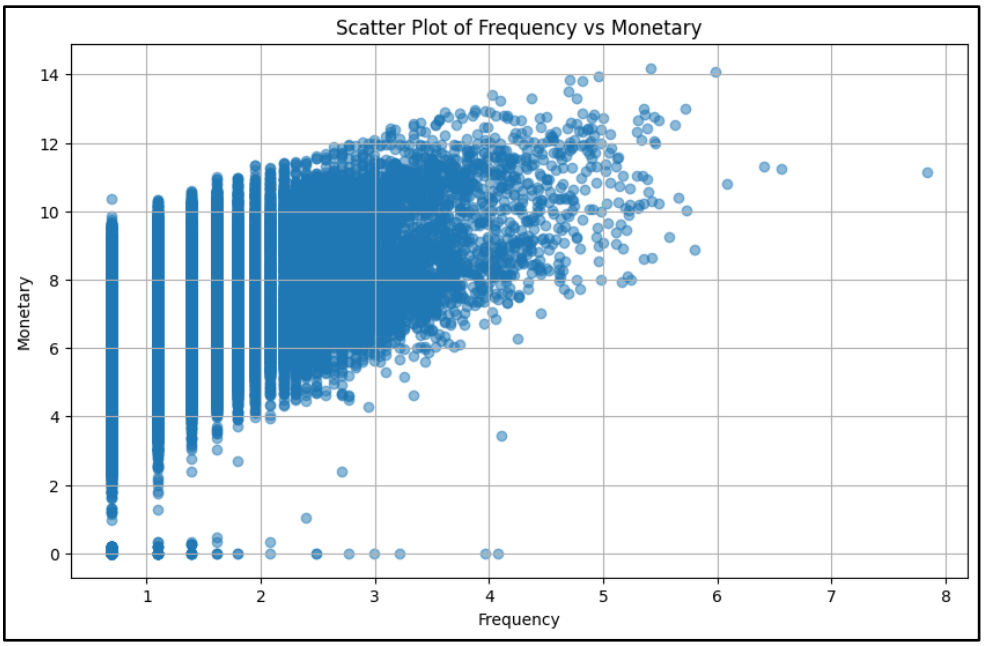
plt.title("Scatter Plot of Frequency vs Monetary")

plt.xlabel("Frequency")

plt.ylabel("Monetary")

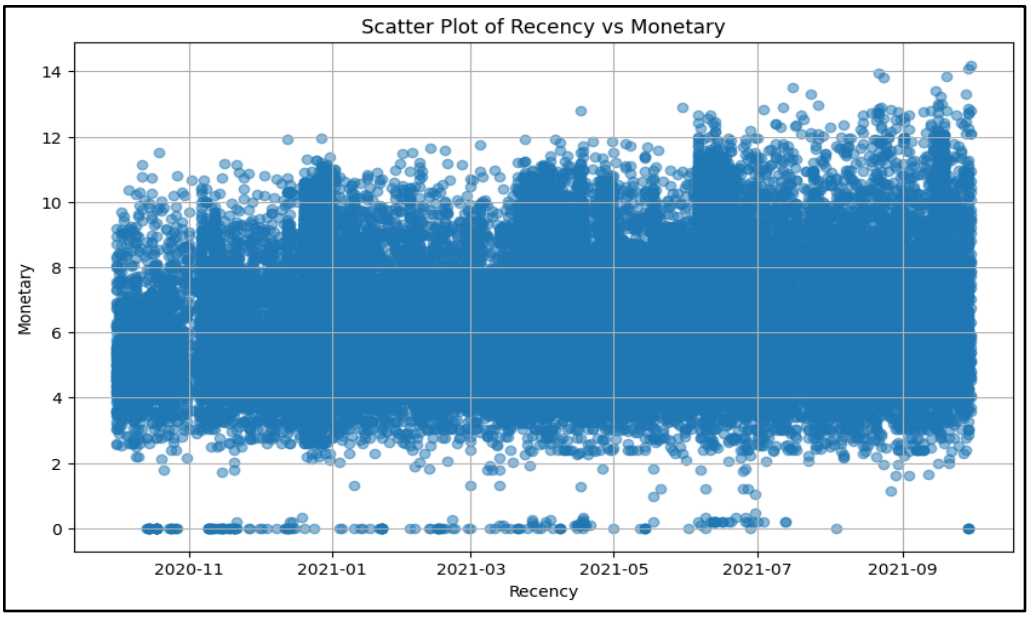
plt.grid(True)

plt.show()



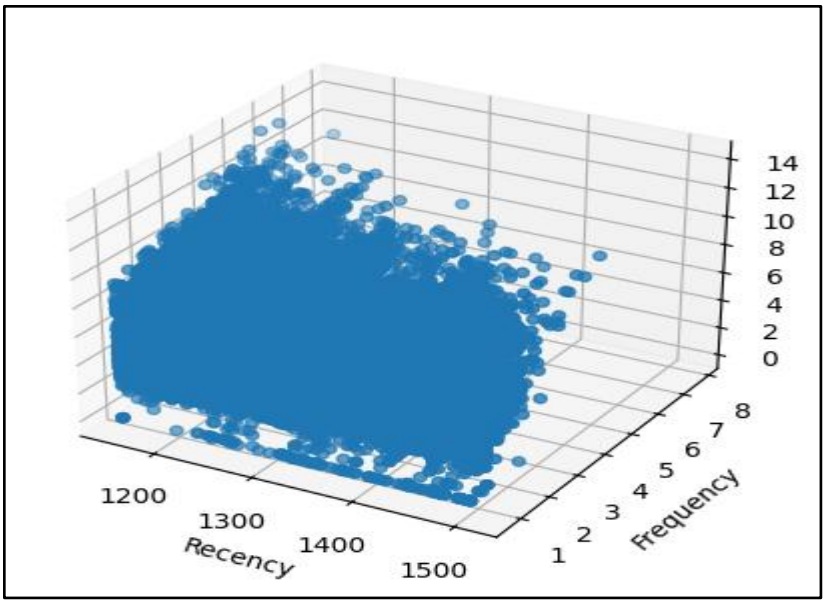
Step 26: Generate a scatter plot to visualize the relationship between Recency and Monetary values

```
# Select relevant columns

recency_monetary_df = rfm_info.select("Recency", "Monetary")

# Convert to Pandas DataFrame for plotting

recency_monetary_pd = recency_monetary_df.toPandas()

# Create scatter plot

plt.figure(figsize=(10, 6))

plt.scatter(recency_monetary_pd["Recency"], recency_monetary_pd["Monetary"], alpha=0.5)

plt.title("Scatter Plot of Recency vs Monetary")

plt.xlabel("Recency")

plt.ylabel("Monetary")

plt.grid(True)

plt.show()
```



Step 27 :Create a 3D scatter plot to visualize the relationship between Recency, Frequency and Monetary values

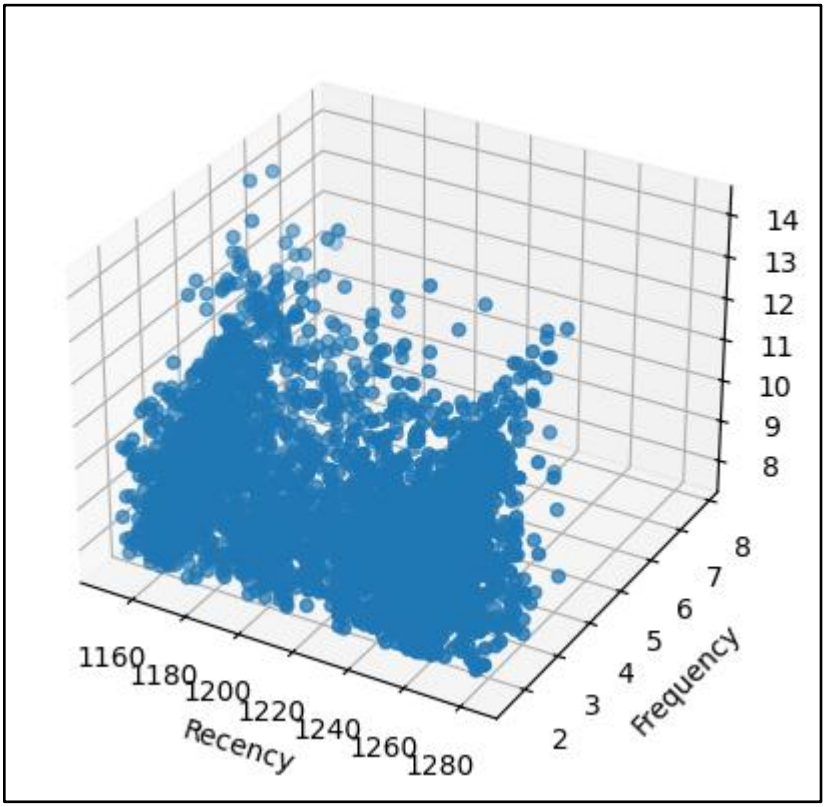
```
# Convert Spark DataFrame to Pandas DataFrame

rfm_info_pd = rfm_info.toPandas()

# Convert 'Recency' column to numeric values (days since a reference date)

reference_date = pd.to_datetime('2024-11-23')

rfm_info_pd['Recency'] = (reference_date - pd.to_datetime(rfm_info_pd['Recency'])).dt.days

# Create 3D scatter plot

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.scatter(rfm_info_pd['Recency'], rfm_info_pd['Frequency'], rfm_info_pd['Monetary'])

ax.set_xlabel('Recency')

ax.set_ylabel('Frequency')

ax.set_zlabel('Monetary')

plt.show()
```



Step 28 : Create a 3D scatter plot with 'Recency', 'Frequency', and 'Monetary' as the axes to visualize the data for the 'Champions' segment.

```
# Filter data for "Champions" segment

champions_df = rfm_info.filter(rfm_info.cust_segment == "Champions")

# Convert Spark DataFrame to Pandas DataFrame

champions_pd = champions_df.toPandas()

# Convert 'Recency' column to numeric values (days since a reference date)

reference_date = pd.to_datetime('2024-11-23')

champions_pd['Recency'] = (reference_date - pd.to_datetime(champions_pd['Recency'])).dt.days

# Create 3D scatter plot

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.scatter(champions_pd['Recency'], champions_pd['Frequency'], champions_pd['Monetary'])

ax.set_xlabel('Recency')

ax.set_ylabel('Frequency')

ax.set_zlabel('Monetary')
```

plt.show()



Step 29 : Create a 3D scatter plot with 'Recency', 'Frequency', and 'Monetary' as the axes to visualize the data for the 'Lost' segment.

```
# Filter the DataFrame for 'Lost' customer segment

rfm_lost_pd = rfm_info.filter(rfm_info['cust_segment'] == 'Lost').toPandas()

# Convert 'Recency' column to numeric values (days since a reference date)

rfm_lost_pd['Recency'] = (reference_date - pd.to_datetime(rfm_lost_pd['Recency'])).dt.days

# Create 3D scatter plot

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.scatter(rfm_lost_pd['Recency'], rfm_lost_pd['Frequency'], rfm_lost_pd['Monetary'])

ax.set_xlabel('Recency')

ax.set_ylabel('Frequency')
```

```
ax.set_zlabel('Monetary')

plt.show()
```