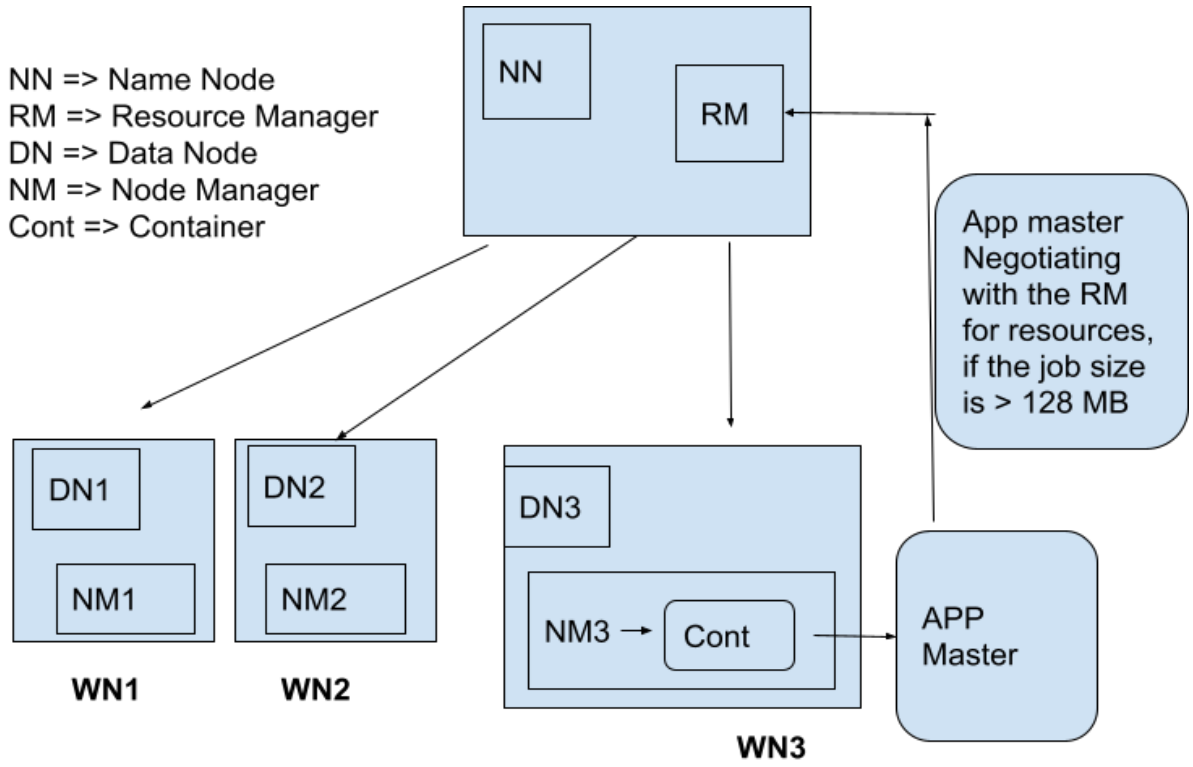
Uber Mode:

This mode will execute any application within the container of the Node Manager, if the spark job is under the default block size. So the Node manager doesn't request any additional resources(Containers) from the Resource manager.

Here the size of the job is determined as the size of the Block and also a job with less than 10 mappers and 1 Reducer.

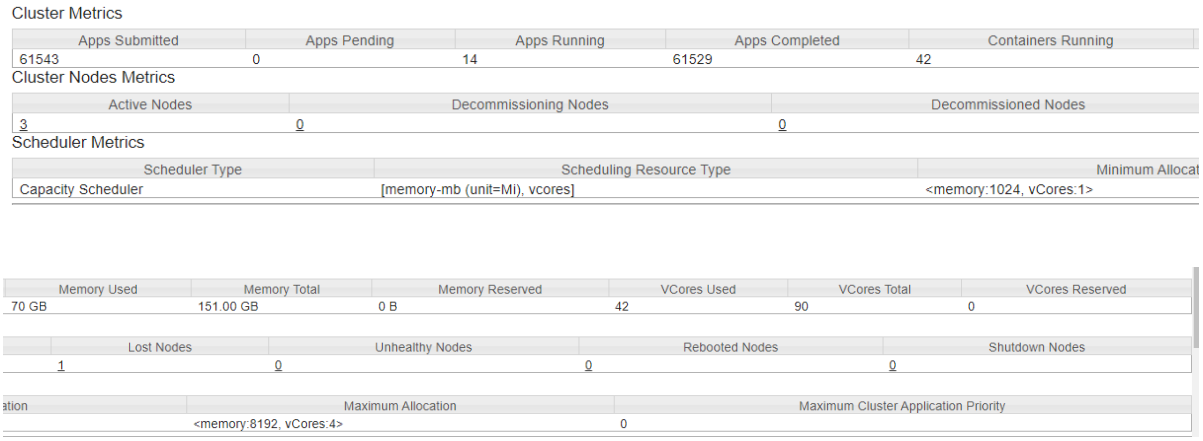
Also this helps the application master to optimise the job instead of allocating and running the tasks in a new container.



The above diagram shows the workflow of a specific job (size <128 MB) has a separate Application Manager which is created by the Node manager in the Worker Node 3. Here the Application master will negotiate if the job size is more than 128 MB or else it will execute the job in the same container and hence it does not require any additional resource. This process is called Uber Mode.

Multi Node Cluster Exploration:

Properties of the Resource manager:



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running
61543	0	14	61529	42

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes
3	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocat
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>

Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
70 GB	151.00 GB	0 B	42	90	0

	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
	1	0	0	0

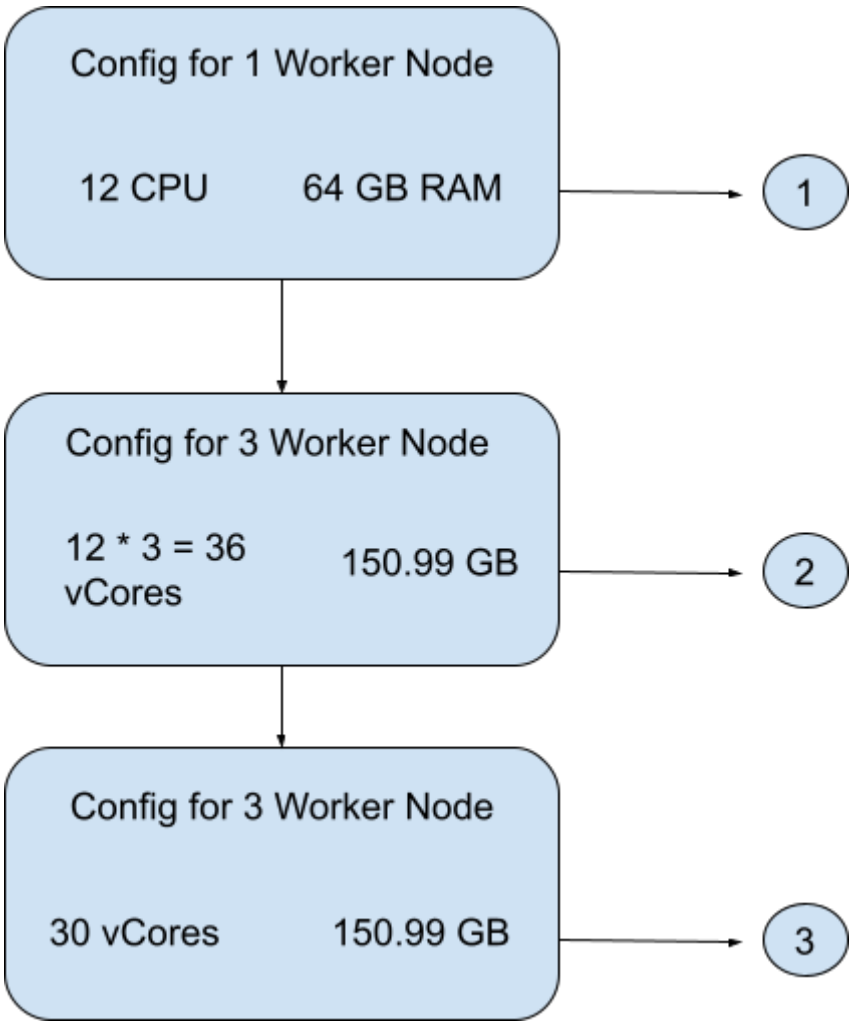
ation	Maximum Allocation	Maximum Cluster Application Priority
	<memory:8192, vCores:4>	0

Below are the properties of the Multi node clusters,

Total memory => 151GB
Total vCores => 90 (CPU)

This cluster is set up to have 1 Master node and 3 worker nodes. The properties are defined below.

Worker Node config:



Step 1:
Here the memory size is set to be 64 GB and the CPU core is 12 for each worker node before any operation begins.

Step 2:
In this step we are multiplying each cores * 3 (No.of worker nodes) to get the total vCores for our job. Also the RAM is further optimised to have 50.33 GB specially used for executor memory and remaining will be not the part of the container and that will be kept for garbage management. So in total we will have 36 vCores & 150.33 GB for our 3 worker Nodes.

Step 3:
Again here out of the 36 vCores only 30 Vores will be allocated to the executor to run the jobs. This is because out of 36 vCores , 6 vCores will be kept as a part of other administrative works which will involve non-Jobs related works.

Show 20 entries Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Allocation Tags	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
	/default-rack	RUNNING	w01.itversity.com:35127	w01.itversity.com:8042	Fri Jun 16 14:15:41 -0400 2023		5		10 GB	40.33 GB	5	25	3.3.0
	/default-rack	RUNNING	w03.itversity.com:41791	w03.itversity.com:8042	Fri Jun 16 14:14:55 -0400 2023		5		10 GB	40.33 GB	5	25	3.3.0
	/default-rack	RUNNING	w02.itversity.com:46669	w02.itversity.com:8042	Fri Jun 16 14:16:29 -0400 2023		17		25 GB	25.33 GB	17	13	3.3.0

Executors:

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	g02.itversity.com:37063	Active	0	145 KiB / 397.5 MiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	w02.itversity.com:37041	Active	0	145 KiB / 366.3 MiB	0.0 B	1	0	4	207	211	6 s (0.1 s)	64.1 MiB	740.9 KiB	740.9 KiB	stdout stderr	Thread Dump
2	w03.itversity.com:38721	Active	0	538.2 KiB / 366.3 MiB	0.0 B	1	0	1	6	7	4 s (0.1 s)	72.1 MiB	3.5 KiB	836.8 KiB	stdout stderr	Thread Dump
3	w02.itversity.com:43845	Active	0	0.0 B / 366.3 MiB	0.0 B	1	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

- If we look into any jobs which are currently running on the Spark Application master the driver will let us know where it is currently running (Client or Cluster Mode).
- Based on the above screenshot the Job is running on a Client mode, it is because the Spark application is accessed through jupyter notebook. Also it gives the address of the gateway node on which the notebook is being accessed.
- If we also note that there would be no cores allocated to the driver, it's because the driver is running in a client mode not the cluster node.
- Now if we look at the executor, we can see that there are 3 executors allocated for the current job and each of them has 1 core. By default the cluster is allocated to have minimum 2 & maximum 10 executors for function. We can also define the allocation based on our requirements.

spark.dynamicAllocation.enabled	true
spark.dynamicAllocation.maxExecutors	10
spark.dynamicAllocation.minExecutors	2

- Storage memory is set to have a minimum & maximum for each container.

Minimum Allocation	Maximum Allocation
<memory:1024, vCores:1>	<memory:8192, vCores:4>

- So for each container the minimum vCores is 1 and the maximum will be 4. Likewise the minimum storage allocation is 1024 (ie, 1GB) and the maximum will be 8192 (ie, 8GB). A note to remember that not all of the memory will be used by the executor only a portion will be used for the executor memory and the rest will be kept for other admin related activities.

Grouping & Various Aggregate Functions:

We can categorise the aggregate functions into 3 types as mentioned below, lets see an example for one by one each.

- Simple Function
- Grouping Function
- Windows function

Also there are three types to write the functions in our Apache Spark.

- Programmatic Style
- Column Expression Style
- Spark Sql Style

Simple Function & Grouping Function:

This function is used to get a single output out of the entire dataframe. Mostly the output will be the final metrics and no further transformation is required.

```
# Aggregate and grouping function on sales data

gro_df = spark.read.format("csv").option("header","True").option("inferSchema","True") \
.load("/public/trendytech/groceries.csv")

gro_df.show(5)
```

```
+--------+--------+--------+----------+--------+
|order_id|location|    item|order_date|quantity|
+--------+--------+--------+----------+--------+
|      o1| Seattle| Bananas|01/01/2017|       7|
|      o2|    Kent|  Apples|02/01/2017|      20|
|      o3|Bellevue| Flowers|02/01/2017|      10|
|      o4| Redmond|    Meat|03/01/2017|      40|
|      o5| Seattle|Potatoes|04/01/2017|       9|
+--------+--------+--------+----------+--------+
only showing top 5 rows
```

Let's look at the groceries data and use all of the three types of the Simple function such as Count, Count Distinct, Sum, Average and Grouping Aggregation.

To use these functions we need to import them using the below code.

```
from pyspark.sql.functions import * #import columns, expr functions to call columns
from pyspark.sql import *
```

Programmatic Style:

```
# simple aggregation Programatic style:
sap_df = gro_df.select(count("*").alias("row_count"),countDistinct("item").alias("Unique Items"),\
                       sum("quantity").alias("Total Quantities"), avg("quantity").alias("Avg quantity"))

sap_df.show()
```

```
+---------+------------+----------------+------------+
|row_count|Unique Items|Total Quantities|Avg quantity|
+---------+------------+----------------+------------+
|       21|           9|             273|        13.0|
+---------+------------+----------------+------------+
```

```
#grouping function Programatic style:
sum_df = gro_df.groupBy("location","item").agg(sum("quantity").alias("Total Quantity"))
sum_df.show()
```

```
+---------+--------+--------------+
| location|    item|Total Quantity|
+---------+--------+--------------+
|     Kent|  Apples|            20|
|Sammamish|   Bread|             5|
| Issaquah|    Meat|            14|
|   Renton|   Bread|             5|
|  Seattle| Bananas|             7|
|  Seattle|Potatoes|             9|
|  Redmond|  Cheese|            15|
| Issaquah|   Onion|            12|
|  Redmond|   Bread|             5|
|  Redmond|    Meat|            40|
| Bellevue| Flowers|            10|
| Issaquah|  Tomato|             6|
| Bellevue|   Bread|           125|
+---------+--------+--------------+
```

By using this approach we can call the built in functions like Count, Count Distinct, Sum, Average and Agg and write the code in a programmatic way.

Column Expression Style:

```
#column Expression style
sac_df = gro_df.selectExpr("count(*) as row_num", "count(Distinct(item)) as Unique_Items",\
                           "sum(quantity) as total_quantiny", "avg(quantity) as avg_quantity")

sac_df.show()
```

```
+-------+------------+--------------+------------+
|row_num|Unique_Items|total_quantiny|avg_quantity|
+-------+------------+--------------+------------+
|     21|           9|           273|        13.0|
+-------+------------+--------------+------------+
```

```
#grouping function column Expression style:
sum_df1 = gro_df.groupBy("location","item").agg(expr("sum(quantity)  as Total_Quantity"))
sum_df1.show()
```

```
+---------+--------+--------------+
| location|    item|Total_Quantity|
+---------+--------+--------------+
|     Kent|  Apples|            20|
|Sammamish|   Bread|             5|
| Issaquah|    Meat|            14|
|   Renton|   Bread|             5|
|  Seattle| Bananas|             7|
|  Seattle|Potatoes|             9|
|  Redmond|  Cheese|            15|
| Issaquah|   Onion|            12|
|  Redmond|   Bread|             5|
|  Redmond|    Meat|            40|
| Bellevue| Flowers|            10|
| Issaquah|  Tomato|             6|
| Bellevue|   Bread|           125|
+---------+--------+--------------+
```

This type is most similar to the SQL way of calling the aggregate functions but using it in the Data frames.

Spark Sql Style:

```
#spark sql style
gro_df.createOrReplaceGlobalTempView("itv005986_grocery_table")

spark.sql("""select count(*) as row_count, count(distinct(item)) as Unique_Items,
sum(quantity) as total_quantiny, avg(quantity) as avg_quantity
from global_temp.itv005986_grocery_table""").show()
```

```
+---------+------------+--------------+------------+
|row_count|Unique_Items|total_quantiny|avg_quantity|
+---------+------------+--------------+------------+
|       21|           9|           273|        13.0|
+---------+------------+--------------+------------+
```

```
##grouping function spark sql style:

spark.sql("""select location, item, sum(quantity) as total_quantity
from global_temp.itv005986_grocery_table group by location, item""").show()
```

```
+---------+--------+--------------+
| location|    item|total_quantity|
+---------+--------+--------------+
|     Kent|  Apples|            20|
|Sammamish|   Bread|             5|
| Issaquah|    Meat|            14|
|   Renton|   Bread|             5|
|  Seattle| Bananas|             7|
|  Seattle|Potatoes|             9|
|  Redmond|  Cheese|            15|
| Issaquah|   Onion|            12|
|  Redmond|   Bread|             5|
|  Redmond|    Meat|            40|
| Bellevue| Flowers|            10|
| Issaquah|  Tomato|             6|
| Bellevue|   Bread|           125|
+---------+--------+--------------+
```

For using the Spark SQL style, we need to convert the Data frame to a temp table and then write a Spark SQL for our functions to work.

Windows Function:

This type of function is little advanced from the above function. Running Total, Rank, Dense Rank, Lead & Lag are some examples of the windows function.
To use the window function we need to first set the window in which the aggregation should work.

Window properties :

1. Partition by =>This will help to partition the data by a specified columns
2. Order By => We can use any Measure to order the partitioned column either Ascending or Descending.
3. Window Size => Specifying the size of the window is very important here, as this will play a crucial role to get out function to work as per the requirement.
    - Current Row => This property will call the measure from the current row
    - Unbounded Preceding => This property will call the measure from the first row
    - Unbounded Following => This property will call the measure from the last row
    - Likewise we can use -1 or 1 to change the direction of the row in which the function needs to select the row values

Once we set the Window, then we can create a new column and use any aggregate function and call the window.

Please refer to the below snap shots of the usage of various Windows function.

Running Total:

```
#windows function
#running total to see the quantity of the items per locaction & items orders
gro_window = Window.partitionBy("location").orderBy("item") \
.rowsBetween(Window.unboundedPreceding,Window.currentRow)

sum_df2 = gro_df.withColumn("running_total",sum("quantity").over(gro_window))
sum_df2.show()
```

```
+--------+---------+--------+----------+--------+-------------+
|order_id| location|    item|order_date|quantity|running_total|
+--------+---------+--------+----------+--------+-------------+
|     o15| Issaquah|    Meat|08/01/2017|       3|            3|
|     o16| Issaquah|    Meat|09/01/2017|       5|            8|
|     o17| Issaquah|    Meat|10/01/2017|       6|           14|
|      o8| Issaquah|   Onion|05/01/2017|       4|           18|
|     o10| Issaquah|   Onion|06/01/2017|       4|           22|
|     o12| Issaquah|   Onion|07/01/2017|       4|           26|
|     o14| Issaquah|  Tomato|07/01/2017|       6|           32|
|     o13|Sammamish|   Bread|07/01/2017|       5|            5|
|      o7|  Redmond|   Bread|05/01/2017|       5|            5|
|      o9|  Redmond|  Cheese|05/01/2017|      15|           20|
|      o4|  Redmond|    Meat|03/01/2017|      40|           60|
|      o1|  Seattle| Bananas|01/01/2017|       7|            7|
|      o5|  Seattle|Potatoes|04/01/2017|       9|           16|
|      o2|     Kent|  Apples|02/01/2017|      20|           20|
|      o6| Bellevue|   Bread|04/01/2017|       5|            5|
|     o18| Bellevue|   Bread|11/01/2017|       7|           12|
|     o19| Bellevue|   Bread|12/01/2017|      54|           66|
|     o20| Bellevue|   Bread|13/01/2017|      34|          100|
|     o21| Bellevue|   Bread|14/01/2017|      25|          125|
|      o3| Bellevue| Flowers|02/01/2017|      10|          135|
+--------+---------+--------+----------+--------+-------------+
```

We are using the same grocery data to see the running total of the quantity partitioned by location, and ordered by Item. Window size is called from the current row to the previous row.

We can also use the same Running total in Spark SQL style.

```
#running total on spark sql
spark.sql("""select location, item, order_date,quantity,
sum(quantity) over(partition by location order by item,order_date) as running_total
from global_temp.itv005986_grocery_table group by location, item,order_date,quantity""").show()
```

```
+---------+--------+----------+--------+-------------+
| location|    item|order_date|quantity|running_total|
+---------+--------+----------+--------+-------------+
| Issaquah|    Meat|08/01/2017|       3|            3|
| Issaquah|    Meat|09/01/2017|       5|            8|
| Issaquah|    Meat|10/01/2017|       6|           14|
| Issaquah|   Onion|05/01/2017|       4|           18|
| Issaquah|   Onion|06/01/2017|       4|           22|
| Issaquah|   Onion|07/01/2017|       4|           26|
| Issaquah|  Tomato|07/01/2017|       6|           32|
|Sammamish|   Bread|07/01/2017|       5|            5|
|  Redmond|   Bread|05/01/2017|       5|            5|
|  Redmond|  Cheese|05/01/2017|      15|           20|
|  Redmond|    Meat|03/01/2017|      40|           60|
|  Seattle| Bananas|01/01/2017|       7|            7|
|  Seattle|Potatoes|04/01/2017|       9|           16|
|     Kent|  Apples|02/01/2017|      20|           20|
| Bellevue|   Bread|04/01/2017|       5|            5|
| Bellevue|   Bread|11/01/2017|       7|           12|
| Bellevue|   Bread|12/01/2017|      54|           66|
| Bellevue|   Bread|13/01/2017|      34|          100|
| Bellevue|   Bread|14/01/2017|      25|          125|
| Bellevue| Flowers|02/01/2017|      10|          135|
+---------+--------+----------+--------+-------------+
only showing top 20 rows
```

Rank, Dense Rank & Row Number:

We can use the same window properties to work the Rank, Dense Rank & Row number. But we need to call the specific function when creating the columns and use them accordingly.

Here we are using the SuperStore data set to explore the Advanced windows function

```
#data to explore advance windows function
ss_df = spark.read.format("csv").option("header","true").option("inferSchema","true") \
.load("/user/itv005986/super_store/super_store_data1.csv")

ss_df.show(5)
```

```
+--------------+-------------+--------+-------------+---------------+------+--------+--------+-------+
|      Order_ID|Customer_Name| Segment|      Country|       Category| Sales|Quantity|Discount| Profit|
+--------------+-------------+--------+-------------+---------------+------+--------+--------+-------+
|CA-2014-156587|Aaron Bergman|Consumer|United States|      Furniture|48.712|       1|     0.2| 5.4801|
|CA-2014-156587|Aaron Bergman|Consumer|United States|Office Supplies| 17.94|       3|     0.0| 4.6644|
|CA-2014-156587|Aaron Bergman|Consumer|United States|Office Supplies| 17.94|       3|     0.0| 4.6644|
|CA-2014-156587|Aaron Bergman|Consumer|United States|Office Supplies|242.94|       3|     0.0| 4.8588|
|CA-2014-152905|Aaron Bergman|Consumer|United States|Office Supplies|12.624|       2|     0.2|-2.5248|
+--------------+-------------+--------+-------------+---------------+------+--------+--------+-------+
only showing top 5 rows
```

Dataframe:

```
#Rank, dense rank, rownum functions
adv_window = Window.partitionBy("Customer_Name").orderBy(desc("Sales"))

rnk_df1 = ss_df.select("Order_ID","Customer_Name","Segment","Category","Sales") \
.withColumn("rank",rank().over(adv_window)).withColumn("Dense_Rank",dense_rank().over(adv_window)) \
.withColumn("Row_Number",row_number().over(adv_window))
rnk_df1.show()
```

Order_ID	Customer_Name	Segment	Category	Sales	rank	Dense_Rank	Row_Number
CA-2015-130113	Aaron Hawkins	Corporate	Technology	668.16	1	1	1
CA-2015-130113	Aaron Hawkins	Corporate	Office Supplies	323.1	2	2	2
CA-2015-130113	Aaron Hawkins	Corporate	Office Supplies	323.1	2	2	3
CA-2014-113768	Aaron Hawkins	Corporate	Furniture	279.456	4	3	4
CA-2014-122070	Aaron Hawkins	Corporate	Office Supplies	247.84	5	4	5
CA-2016-162747	Aaron Hawkins	Corporate	Furniture	86.45	6	5	6
US-2014-158400	Aaron Hawkins	Corporate	Office Supplies	49.408	7	6	7
CA-2014-157644	Aaron Hawkins	Corporate	Technology	34.77	8	7	8
CA-2014-157644	Aaron Hawkins	Corporate	Technology	34.77	8	7	9
CA-2014-157644	Aaron Hawkins	Corporate	Office Supplies	18.9	10	8	10
CA-2017-164000	Aaron Hawkins	Corporate	Office Supplies	18.704	11	9	11
CA-2014-122070	Aaron Hawkins	Corporate	Office Supplies	9.912	12	10	12
CA-2014-113768	Aaron Hawkins	Corporate	Office Supplies	8.0	13	11	13
CA-2017-162691	Aaron Smayling	Corporate	Technology	1439.982	1	1	1
CA-2017-113481	Aaron Smayling	Corporate	Technology	695.7	2	2	2
CA-2016-148747	Aaron Smayling	Corporate	Furniture	477.666	3	3	3
CA-2017-101749	Aaron Smayling	Corporate	Furniture	171.288	4	4	4
US-2017-147655	Aaron Smayling	Corporate	Office Supplies	88.074	5	5	5
US-2017-147655	Aaron Smayling	Corporate	Office Supplies	88.074	5	5	6
US-2014-150126	Aaron Smayling	Corporate	Office Supplies	65.78	7	6	7

Spark SQL:

```
# rank, dense rank, rownum using spark sql
ss_df.createOrReplaceGlobalTempView("itv005986_super_store_data")

spark.sql(f"""select Order_ID,Customer_Name,Segment,Category,Sales,
rank() over(partition by Customer_Name order by Sales desc) as rank,
dense_rank() over(partition by Customer_Name order by Sales desc) as dense_rank,
row_number() over(partition by Customer_Name order by Sales desc) as row_num
from global_temp.itv005986_super_store_data""").show()
```

```
+--------------+--------------+---------+---------------+--------+----+----------+-------+
|      Order_ID| Customer_Name|  Segment|       Category|   Sales|rank|dense_rank|row_num|
+--------------+--------------+---------+---------------+--------+----+----------+-------+
|CA-2015-130113| Aaron Hawkins|Corporate|     Technology|  668.16|   1|         1|      1|
|CA-2015-130113| Aaron Hawkins|Corporate|Office Supplies|   323.1|   2|         2|      2|
|CA-2015-130113| Aaron Hawkins|Corporate|Office Supplies|   323.1|   2|         2|      3|
|CA-2014-113768| Aaron Hawkins|Corporate|      Furniture| 279.456|   4|         3|      4|
|CA-2014-122070| Aaron Hawkins|Corporate|Office Supplies|  247.84|   5|         4|      5|
|CA-2016-162747| Aaron Hawkins|Corporate|      Furniture|   86.45|   6|         5|      6|
|US-2014-158400| Aaron Hawkins|Corporate|Office Supplies|  49.408|   7|         6|      7|
|CA-2014-157644| Aaron Hawkins|Corporate|     Technology|   34.77|   8|         7|      8|
|CA-2014-157644| Aaron Hawkins|Corporate|     Technology|   34.77|   8|         7|      9|
|CA-2014-157644| Aaron Hawkins|Corporate|Office Supplies|    18.9|  10|         8|     10|
|CA-2017-164000| Aaron Hawkins|Corporate|Office Supplies|  18.704|  11|         9|     11|
|CA-2014-122070| Aaron Hawkins|Corporate|Office Supplies|   9.912|  12|        10|     12|
|CA-2014-113768| Aaron Hawkins|Corporate|Office Supplies|     8.0|  13|        11|     13|
|CA-2017-162691|Aaron Smayling|Corporate|     Technology|1439.982|   1|         1|      1|
|CA-2017-113481|Aaron Smayling|Corporate|     Technology|   695.7|   2|         2|      2|
|CA-2016-148747|Aaron Smayling|Corporate|      Furniture| 477.666|   3|         3|      3|
|CA-2017-101749|Aaron Smayling|Corporate|      Furniture| 171.288|   4|         4|      4|
|US-2017-147655|Aaron Smayling|Corporate|Office Supplies|  88.074|   5|         5|      5|
|US-2017-147655|Aaron Smayling|Corporate|Office Supplies|  88.074|   5|         5|      6|
|US-2014-150126|Aaron Smayling|Corporate|Office Supplies|   65.78|   7|         6|      7|
+--------------+--------------+---------+---------------+--------+----+----------+-------+
only showing top 20 rows
```

Rank:

Rank function is used when there is same values for two categories and we want to rank them with the same number, but skip the following Rank. This kind of method is used in Races.

Dense Rank:

Dense Rank also works similar to the Rank function, but this function will not skip the next rank and assign the same rank to repeating values.

Row Number:

Row Number is widely used inorder to find the Top N values of a specific partitioned value. This will however give unique value to each row.

Lead & Lag Functions:

These two functions are used to play with measures and find some insights.
Lead => Lead helps to compare and get the next value in a new column of a partitioned data.

Lag => likewise lead, lag also helps us to get the previous value of the partitioned data.
We also need to use the Window properties with partition and order by in order to work with Lead & Lag

```
#Lead & Lag functions
ss_sch = 'Order_ID String, Order_Date Date, Customer_Name String, Segment String, Category String, Sales double'
ss_df1 = spark.read.format("csv").option("header","true").option("dateFormat","dd-MM-yyyy").schema(ss_sch) \
.load("/user/itv005986/super_store/super_store_data2.csv")

ss_df1.show()
```

```
+--------------+----------+--------------+---------+---------------+-------+
|      Order_ID|Order_Date| Customer_Name|  Segment|       Category|  Sales|
+--------------+----------+--------------+---------+---------------+-------+
|CA-2014-156587|2014-03-07| Aaron Bergman| Consumer|      Furniture| 48.712|
|CA-2014-156587|2014-03-07| Aaron Bergman| Consumer|Office Supplies|  17.94|
|CA-2014-156587|2014-03-07| Aaron Bergman| Consumer|Office Supplies| 242.94|
|CA-2014-152905|2014-02-18| Aaron Bergman| Consumer|Office Supplies| 12.624|
|CA-2016-140935|2016-11-10| Aaron Bergman| Consumer|     Technology| 221.98|
|CA-2016-140935|2016-11-10| Aaron Bergman| Consumer|      Furniture| 341.96|
|CA-2017-164000|2017-12-18| Aaron Hawkins|Corporate|Office Supplies| 18.704|
|CA-2016-162747|2016-03-20| Aaron Hawkins|Corporate|      Furniture|  86.45|
|CA-2014-113768|2014-05-13| Aaron Hawkins|Corporate|      Furniture|279.456|
|CA-2014-113768|2014-05-13| Aaron Hawkins|Corporate|Office Supplies|    8.0|
|US-2014-158400|2014-10-25| Aaron Hawkins|Corporate|Office Supplies| 49.408|
|CA-2014-122070|2014-04-22| Aaron Hawkins|Corporate|Office Supplies| 247.84|
|CA-2014-122070|2014-04-22| Aaron Hawkins|Corporate|Office Supplies|  9.912|
|CA-2014-157644|2014-12-31| Aaron Hawkins|Corporate|     Technology|  34.77|
|CA-2014-157644|2014-12-31| Aaron Hawkins|Corporate|Office Supplies|   18.9|
|CA-2015-130113|2015-12-27| Aaron Hawkins|Corporate|Office Supplies|  323.1|
|CA-2015-130113|2015-12-27| Aaron Hawkins|Corporate|     Technology| 668.16|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|     Technology|  695.7|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|Office Supplies|  15.66|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|Office Supplies| 28.854|
+--------------+----------+--------------+---------+---------------+-------+
```

Lead:

```
adv_window = Window.partitionBy("Customer_Name").orderBy("Order_Date")

ll_df = ss_df1.withColumn("next_category_Sales",lead("Sales").over(adv_window))

ll_df.show()
```

```
+--------------+----------+--------------+---------+---------------+--------+-------------------+
|      Order_ID|Order_Date| Customer_Name|  Segment|       Category|   Sales|next_category_Sales|
+--------------+----------+--------------+---------+---------------+--------+-------------------+
|CA-2014-122070|2014-04-22| Aaron Hawkins|Corporate|Office Supplies|  247.84|              9.912|
|CA-2014-122070|2014-04-22| Aaron Hawkins|Corporate|Office Supplies|   9.912|            279.456|
|CA-2014-113768|2014-05-13| Aaron Hawkins|Corporate|      Furniture| 279.456|                8.0|
|CA-2014-113768|2014-05-13| Aaron Hawkins|Corporate|Office Supplies|     8.0|             49.408|
|US-2014-158400|2014-10-25| Aaron Hawkins|Corporate|Office Supplies|  49.408|              34.77|
|CA-2014-157644|2014-12-31| Aaron Hawkins|Corporate|     Technology|   34.77|               18.9|
|CA-2014-157644|2014-12-31| Aaron Hawkins|Corporate|Office Supplies|    18.9|              323.1|
|CA-2015-130113|2015-12-27| Aaron Hawkins|Corporate|Office Supplies|   323.1|             668.16|
|CA-2015-130113|2015-12-27| Aaron Hawkins|Corporate|     Technology|  668.16|              86.45|
|CA-2016-162747|2016-03-20| Aaron Hawkins|Corporate|      Furniture|   86.45|             18.704|
|CA-2017-164000|2017-12-18| Aaron Hawkins|Corporate|Office Supplies|  18.704|               null|
|US-2014-150126|2014-07-27|Aaron Smayling|Corporate|Office Supplies|   65.78|               31.4|
|CA-2016-162901|2016-03-28|Aaron Smayling|Corporate|Office Supplies|    31.4|            477.666|
|CA-2016-148747|2016-09-25|Aaron Smayling|Corporate|      Furniture| 477.666|              695.7|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|     Technology|   695.7|              15.66|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|Office Supplies|   15.66|             28.854|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|Office Supplies|  28.854|            1439.982|
|CA-2017-162691|2017-08-01|Aaron Smayling|Corporate|     Technology|1439.982|             36.288|
|CA-2017-162691|2017-08-01|Aaron Smayling|Corporate|Office Supplies|  36.288|             88.074|
|US-2017-147655|2017-09-04|Aaron Smayling|Corporate|Office Supplies|  88.074|            171.288|
+--------------+----------+--------------+---------+---------------+--------+-------------------+
```

Lag:

```
adv_window = Window.partitionBy("Customer_Name").orderBy("Order_Date")

ll_df1 = ss_df1.withColumn("prev_category_Sales",lag("Sales").over(adv_window))

ll_df1.show()
```

```
+--------------+----------+--------------+---------+---------------+--------+-------------------+
|      Order_ID|Order_Date| Customer_Name|  Segment|       Category|   Sales|prev_category_Sales|
+--------------+----------+--------------+---------+---------------+--------+-------------------+
|CA-2014-122070|2014-04-22| Aaron Hawkins|Corporate|Office Supplies|  247.84|               null|
|CA-2014-122070|2014-04-22| Aaron Hawkins|Corporate|Office Supplies|   9.912|             247.84|
|CA-2014-113768|2014-05-13| Aaron Hawkins|Corporate|      Furniture| 279.456|              9.912|
|CA-2014-113768|2014-05-13| Aaron Hawkins|Corporate|Office Supplies|     8.0|            279.456|
|US-2014-158400|2014-10-25| Aaron Hawkins|Corporate|Office Supplies|  49.408|                8.0|
|CA-2014-157644|2014-12-31| Aaron Hawkins|Corporate|     Technology|   34.77|             49.408|
|CA-2014-157644|2014-12-31| Aaron Hawkins|Corporate|Office Supplies|    18.9|              34.77|
|CA-2015-130113|2015-12-27| Aaron Hawkins|Corporate|Office Supplies|   323.1|               18.9|
|CA-2015-130113|2015-12-27| Aaron Hawkins|Corporate|     Technology|  668.16|              323.1|
|CA-2016-162747|2016-03-20| Aaron Hawkins|Corporate|      Furniture|   86.45|             668.16|
|CA-2017-164000|2017-12-18| Aaron Hawkins|Corporate|Office Supplies|  18.704|              86.45|
|US-2014-150126|2014-07-27|Aaron Smayling|Corporate|Office Supplies|   65.78|               null|
|CA-2016-162901|2016-03-28|Aaron Smayling|Corporate|Office Supplies|    31.4|              65.78|
|CA-2016-148747|2016-09-25|Aaron Smayling|Corporate|      Furniture| 477.666|               31.4|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|     Technology|   695.7|            477.666|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|Office Supplies|   15.66|              695.7|
|CA-2017-113481|2017-01-02|Aaron Smayling|Corporate|Office Supplies|  28.854|              15.66|
|CA-2017-162691|2017-08-01|Aaron Smayling|Corporate|     Technology|1439.982|             28.854|
|CA-2017-162691|2017-08-01|Aaron Smayling|Corporate|Office Supplies|  36.288|            1439.982|
|US-2017-147655|2017-09-04|Aaron Smayling|Corporate|Office Supplies|  88.074|             36.288|
+--------------+----------+--------------+---------+---------------+--------+-------------------+
only showing top 20 rows
```

Dealing With Null values in Apache Spark:

LikeWise in any form of data we have the same problem of NULL and nan values in Apache spark Data frames. Identifying and dealing with them wisely would help us to maintain the data.

Let's look at the sales data below,

```
# dealing with null values on sales data
sales_sch = 'store_id long, product String, quantity int, revenue double'
sales_df = spark.read.format("json").schema(sales_sch).option("header","True") \
.load("/public/trendytech/datasets/sales_data.json")

sales_df.show(22)
```

```
+--------+----------+--------+-------+
|store_id|   product|quantity|revenue|
+--------+----------+--------+-------+
|       1|     Apple|      10|  100.0|
|       2|    Banana|      15|   75.0|
|       3|    Orange|      12|   90.0|
|       4|     Mango|       8|  120.0|
|       5|     Grape|      20|  150.0|
|       6|Watermelon|       5|   50.0|
|       7|Strawberry|      18|  108.0|
|       8| Pineapple|      14|  140.0|
|       9|    Cherry|       7|  105.0|
|      10|      Pear|       9|   81.0|
|      11| Blueberry|      11|   88.0|
|      12|      Kiwi|      16|  128.0|
|      13|     Peach|      13|   91.0|
|      14|      Plum|       6|   54.0|
|      15|     Lemon|      10|   70.0|
|      16| Raspberry|      17|  136.0|
|      17|   Coconut|       4|   80.0|
|      18|   Avocado|      11|   99.0|
|      19|Blackberry|       8|   64.0|
|      20|         G|    null|    NaN|
|    null|      null|    null|   null|
|      22|Watermelon|       5|   null|
+--------+----------+--------+-------+
```

The sales data has Null values in all of the columns. And it is present in both String and Integer data types.

Type 1:

We can change the null values in the integer data type by replacing it with any value. Here the replacement value majorly would be 0, and in some scenarios it can be a mean of a particular category. This will be dependent on the requirements.

```
null_df = sales_df.fillna(0)
null_df.show(22)
```

```
+--------+----------+--------+-------+
|store_id|   product|quantity|revenue|
+--------+----------+--------+-------+
|       1|     Apple|      10|  100.0|
|       2|    Banana|      15|   75.0|
|       3|    Orange|      12|   90.0|
|       4|     Mango|       8|  120.0|
|       5|     Grape|      20|  150.0|
|       6|Watermelon|       5|   50.0|
|       7|Strawberry|      18|  108.0|
|       8| Pineapple|      14|  140.0|
|       9|    Cherry|       7|  105.0|
|      10|      Pear|       9|   81.0|
|      11| Blueberry|      11|   88.0|
|      12|      Kiwi|      16|  128.0|
|      13|     Peach|      13|   91.0|
|      14|      Plum|       6|   54.0|
|      15|     Lemon|      10|   70.0|
|      16| Raspberry|      17|  136.0|
|      17|   Coconut|       4|   80.0|
|      18|   Avocado|      11|   99.0|
|      19|Blackberry|       8|   64.0|
|      20|         G|       0|    0.0|
|       0|      null|       0|    0.0|
|      22|Watermelon|       5|    0.0|
+--------+----------+--------+-------+
```

Here we have filled the null values with 0.

Type 2:

If we find any null value is not helping or providing any insights to us, then we can simply delete them by dropping it.

```
sales_df.filter(isnull("quantity")).show()
```

```
+--------+-------+--------+-------+
|store_id|product|quantity|revenue|
+--------+-------+--------+-------+
|      20|      G|    null|    NaN|
|    null|   null|    null|   null|
+--------+-------+--------+-------+
```

```
null_df1 = null_df.dropna()
null_df1.show()
```

```
+--------+----------+--------+-------+
|store_id|   product|quantity|revenue|
+--------+----------+--------+-------+
|       1|     Apple|      10|  100.0|
|       2|    Banana|      15|   75.0|
|       3|    Orange|      12|   90.0|
|       4|     Mango|       8|  120.0|
|       5|     Grape|      20|  150.0|
|       6|Watermelon|       5|   50.0|
|       7|Strawberry|      18|  108.0|
|       8| Pineapple|      14|  140.0|
|       9|    Cherry|       7|  105.0|
|      10|      Pear|       9|   81.0|
|      11| Blueberry|      11|   88.0|
|      12|      Kiwi|      16|  128.0|
|      13|     Peach|      13|   91.0|
|      14|      Plum|       6|   54.0|
|      15|     Lemon|      10|   70.0|
|      16| Raspberry|      17|  136.0|
|      17|   Coconut|       4|   80.0|
|      18|   Avocado|      11|   99.0|
|      19|Blackberry|       8|   64.0|
|      20|         G|       0|    0.0|
+--------+----------+--------+-------+
only showing top 20 rows
```