EMR Cluster execution sample

- Upload this notebook to EMR Notebooks
- Set up Kernel as PySpark

In [1]:
```
## Info to run as Emr Serverless script export to python file and change EMR_SERVERLESS_EXECUTION to False
## all input data must be executed as parameters with EMR Serverless
import sys
from datetime import datetime


### EMR ServerLess Execution
EMR_SERVERLESS_EXECUTION = False ## True for emr_serverless or False for Emr Cluster - Jupyter Notebook

from pyspark.sql import SparkSession
```

```
VBox()
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1679779274242_0001	pyspark	idle	Link	Link	✓

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
SparkSession available as 'spark'.
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [2]:
```
spark.version
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
'3.3.1-amzn-0'
```

In [3]:
```
def fnc_validate_parameters(awsExec_v=EMR_SERVERLESS_EXECUTION):
    if (len(sys.argv) != 4) and awsExec_v:
        print("Usage: spark-etl ['input folder'] ['output folder'] ['rpt_folder']")
        sys.exit(-1)

    if not(awsExec_v):
        ## Emr Cluster execution
        input_args = ['python-script.py', '../s3_data/input/', '../s3_data/output/', '../s3_data/rpt/']
        input_location = input_args[1]
        output_location = input_args[2]
        rpt_location = input_args[3]
    else:
        ## Emr Serverless Execution
        input_location = sys.argv[1]
        output_location = sys.argv[2]
        rpt_location = sys.argv[3]

    return input_location, output_location, rpt_location
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [ ]:
```
(input_location, output_location, rpt_location) = fnc_validate_parameters()
```

In [5]:
```
def fshape(dataframe1):
    print('Shape : ', dataframe1.count(), len(dataframe1.columns))
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [6]:
```
dbname = 'DBM' ## database Marketing
tablename = 'TBP_CUSTOMER_CLV' ## Parquet table - Customer Lifetime Value
spark_ml_table = 'TB_ML_SPARK_SDF'
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [7]:
```
def fnc_show_db_tables():
    spark.sql( ' SHOW DATABASES ').show()
    spark.sql(' SHOW TABLES ').show()

def spark_sql_write_glue_database(db_name, table_name, parquet_output_location=output_location, temp_table=spark_ml_table):

    ## Create AWS GLUE table for Analytics - Ad hoc query for example using Athena SQL
    print(' database creation: ', db_name)
    spark.sql(f" CREATE database if not exists {db_name} ")

    print(' table name creation , ', table_name)
    spark.sql((
        f" CREATE TABLE IF NOT EXISTS {db_name}.{table_name} "
        f" USING PARQUET LOCATION '{parquet_output_location}' AS SELECT * FROM {temp_table}"
    ))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [8]:
```
# s3_bucket = 's3://S3_BUCKET_NAME/'
# parquet_file_name = 'S3_NAME_DIR/data_input/OnlineRetail__AWS.parquet'

# s3_filename = s3_bucket + parquet_file_name

s3_filename = input_location
s3_filename
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
's3://s3-bucket-ohio-v1/emr_serverless/s3_data/input/'
```

In [9]:
```
## read parquet filename
sdf = spark.read.parquet(s3_filename)
print(sdf.printSchema())
# fshape(sdf)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: timestamp (nullable = true)
 |-- UnitPrice: float (nullable = true)
 |-- CustomerID: double (nullable = true)
 |-- Country: string (nullable = true)

None
```

In [10]:
```
fshape(sdf)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
Shape :  541909 8
```

In [11]:
```
## ETL
sdf.createOrReplaceTempView('TB_SALES_SDF')
spark.sql('select max(TO_DATE(InvoiceDate)) as current_date_for_FRMV_CLV, current_date as not_today from TB_SALES_SDF').show()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
+-------------------------+----------+
|current_date_for_FRMV_CLV| not_today|
+-------------------------+----------+
|               2011-12-09|2023-03-25|
+-------------------------+----------+
```

In [12]:
```
## formula to calculate CLV
def fnc_customer_clv_udf(monetary_value_f, frequency_f, recency_f, discount_f=0.1):
    return round ( ( (monetary_value_f / frequency_f) * (1 - ((recency_f + 1) / 365)) / (1 + discount_f) ) , 2)

## Register the formula to be used by Spark-SQL
from pyspark.sql.types import FloatType

spark.udf.register('fnc_customer_clv_udf', fnc_customer_clv_udf, FloatType())

print("Catalog Entry:")
[print(r) for r in spark.catalog.listFunctions() if "fnc_customer_clv_udf" in r.name]
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
Catalog Entry:
Function(name='fnc_customer_clv_udf', description=None, className='org.apache.spark.sql.UDFRegistration$$Lambda$4837/1074466461', isTemporary=True)
[None]
```

In [13]:
```
## Apply some filters and create the main customer purchase history as an example
sql_query_clv = """
WITH TB_SALES_V AS
(
    SELECT CustomerID as customer_id
        , COUNT(DISTINCT (InvoiceDate))  as frequency
        , DATEDIFF( current_date , MAX (InvoiceDate) )  as recency_now
        , ROUND(SUM(Quantity * UnitPrice), 2) as monetary_value
        , ROUND(avg(Quantity * UnitPrice), 2) as avg_revenue
        , MIN(InvoiceDate) as dt_first_Invoice
        , MAX(InvoiceDate) as dt_last_Invoice
        -- , ROUND(AVG(Quantity), 2) as avg_items
        -- , ROUND(SUM(Quantity), 2) as total_items
    FROM TB_SALES_SDF
    WHERE 1 = 1
        AND InvoiceDate IS NOT NULL
        AND Quantity > 0
        AND UnitPrice > 0
    GROUP BY customer_id
)
SELECT tb3.*
  , ROUND ( ( (monetary_value / frequency) * (1 - ((recency_dt + 1) / 365)) / (1 + 0.1) ) , 2) AS CLV_SQL -- discount of 0.1
  , fnc_customer_clv_udf(monetary_value,frequency,recency_dt) AS CLV_UDF
FROM (
    SELECT tb1.*
        , CAST( DATEDIFF(tb2.dt_current_date , tb1.dt_last_Invoice ) as float) as recency_dt
    FROM TB_SALES_V as tb1
    CROSS JOIN (SELECT MAX(dt_last_Invoice) AS dt_current_date FROM TB_SALES_V) tb2
    ) tb3
WHERE 1 = 1
  AND monetary_value > 0
  AND frequency > 0
  AND customer_id IS NOT NULL
ORDER BY monetary_value DESC
```

```
"""

sdf_clv = spark.sql(sql_query_clv)
sdf_clv.printSchema()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
root
 |-- customer_id: double (nullable = true)
 |-- frequency: long (nullable = false)
 |-- recency_now: integer (nullable = true)
 |-- monetary_value: double (nullable = true)
 |-- avg_revenue: double (nullable = true)
 |-- dt_first_Invoice: timestamp (nullable = true)
 |-- dt_last_Invoice: timestamp (nullable = true)
 |-- recency_dt: float (nullable = true)
 |-- CLV_SQL: double (nullable = true)
 |-- CLV_UDF: float (nullable = true)
```

In [14]:
```
print('clv_SQL and clv_udf provide the same information - just show how to implement it using 2 solutions... SQL and UDF')
sdf_clv.show(3)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
clv_SQL and clv_udf provide the same information - just show how to implement it using 2 solutions... SQL and UDF
+-----------+---------+-----------+--------------+-----------+-------------------+-------------------+----------+-------+-------+
|customer_id|frequency|recency_now|monetary_value|avg_revenue|   dt_first_Invoice|    dt_last_Invoice|recency_dt|CLV_SQL|CLV_UDF|
+-----------+---------+-----------+--------------+-----------+-------------------+-------------------+----------+-------+-------+
|    14646.0|       51|       4125|      200541.0|     137.36|2010-12-20 10:09:00|2011-12-08 00:12:00|       1.0|3555.12|3555.12|
|    16446.0|        2|       4124|     168472.49|    56157.5|2011-05-18 09:52:00|2011-12-09 09:15:00|       0.0|76368.6|76368.6|
|    17450.0|       27|       4134|     121321.71|     588.94|2010-12-07 09:23:00|2011-11-29 09:56:00|      10.0| 3961.8| 3961.8|
+-----------+---------+-----------+--------------+-----------+-------------------+-------------------+----------+-------+-------+
only showing top 3 rows
```

In [15]:
```
sdf_clv.createOrReplaceTempView(spark_ml_table)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [16]:
```
def ml_sql_prediction():
    text_sql_ml2 = f"""
    SELECT
        {spark_ml_table}.*,
    ( CASE
    WHEN ( ( ( `frequency`  > 1.0e1 AND `frequency`  <= 1.14e2 ) ) ) THEN 9
    WHEN ( ((abs(year(`dt_first_Invoice`) - 2.01e3) <= 10e-9) OR ( (`dt_first_Invoice` IS NULL ) ) ) AND ((abs(`frequency` - 1.0e0) <= 10e-9) OR (abs(`frequency` - 2.0e0) <= 10e-9) OR (abs(`freq
    WHEN ( ((abs(`frequency` - 7.0e0) <= 10e-9) OR  ( `frequency`  >= 8.0e0 AND `frequency`  <= 1.3e1 ) ) ) THEN 3
    WHEN ( ( ( `recency_dt`  >= 0.0e0 AND `recency_dt`  <= 4.0e0 ) ) ) THEN 10
    WHEN ( ( ( `CLV_SQL`  > 9.0245000000000005e2 AND `CLV_SQL`  <= 7.49729e3 ) ) ) THEN 6
    WHEN ( ( ( `CLV_SQL`  > 3.8501999999999998e2 AND `CLV_SQL`  <= 9.0245000000000005e2 ) ) AND ((abs(`frequency` - 1.0e0) <= 10e-9) OR (abs(`frequency` - 2.0e0) <= 10e-9) OR (abs(`frequency` -
    WHEN ( ( ( (datediff(concat(year(`dt_first_Invoice`),'-',month(`dt_first_Invoice`),'-',day(`dt_first_Invoice`)),concat(year(`dt_first_Invoice`),'-01-01')) + 1)  > 1.3e1 AND (datediff(concat(
    WHEN ( ((abs(month(`dt_last_Invoice`) - 3.0e0) <= 10e-9) OR (abs(month(`dt_last_Invoice`) - 4.0e0) <= 10e-9) OR (abs(month(`dt_last_Invoice`) - 5.0e0) <= 10e-9) OR (abs(month(`dt_last_Invoic
    WHEN ( ( ( `recency_dt`  >= 3.0e0 AND `recency_dt`  <= 2.5e1 ) OR  ( `recency_dt`  > 3.1e1 AND `recency_dt`  <= 3.6e1 ) OR  ( `recency_dt`  > 3.25e2 AND `recency_dt`  <= 3.74e2 ) ) AND ((abs
    WHEN ( ( ( (datediff(concat(year(`dt_last_Invoice`),'-',month(`dt_last_Invoice`),'-',day(`dt_last_Invoice`)),concat(year(`dt_last_Invoice`),'-01-01')) + 1)  >= 4.0e0 AND (datediff(concat(yea
    ELSE 11
    END ) AS kc_monetary_value
    FROM {spark_ml_table}
"""
    return text_sql_ml2
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [17]:
```
sdf_ml = spark.sql(ml_sql_prediction())
```

```
sdf_ml.printSchema()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
root
 |-- customer_id: double (nullable = true)
 |-- frequency: long (nullable = false)
 |-- recency_now: integer (nullable = true)
 |-- monetary_value: double (nullable = true)
 |-- avg_revenue: double (nullable = true)
 |-- dt_first_Invoice: timestamp (nullable = true)
 |-- dt_last_Invoice: timestamp (nullable = true)
 |-- recency_dt: float (nullable = true)
 |-- CLV_SQL: double (nullable = true)
 |-- CLV_UDF: float (nullable = true)
 |-- kc_monetary_value: integer (nullable = false)
```

In [18]:
```
sdf_ml.show(3)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
+-----------+---------+-----------+-------------+-----------+-------------------+-------------------+----------+-------+-------+-----------------+
|customer_id|frequency|recency_now|monetary_value|avg_revenue|   dt_first_Invoice|    dt_last_Invoice|recency_dt|CLV_SQL|CLV_UDF|kc_monetary_value|
+-----------+---------+-----------+-------------+-----------+-------------------+-------------------+----------+-------+-------+-----------------+
|    14646.0|       51|       4125|      200541.0|     137.36|2010-12-20 10:09:00|2011-12-08 00:12:00|       1.0|3555.12|3555.12|                9|
|    16446.0|        2|       4124|     168472.49|    56157.5|2011-05-18 09:52:00|2011-12-09 09:15:00|       0.0|76368.6|76368.6|               10|
|    17450.0|       27|       4134|     121321.71|     588.94|2010-12-07 09:23:00|2011-11-29 09:56:00|      10.0| 3961.8| 3961.8|                9|
+-----------+---------+-----------+-------------+-----------+-------------------+-------------------+----------+-------+-------+-----------------+
only showing top 3 rows
```

In [19]:
```
s3_export_file = output_location
sdf_ml.write.mode('overwrite').parquet(s3_export_file)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [20]:
```
## Summary report
sdf_ml.createOrReplaceTempView('TB_CLV_SDF_ML')

ml_rpt_sql = """
WITH TB_CLUSTER AS
(
    select kc_monetary_value as cluster_number
    , count(distinct customer_id) as customer_count
    , avg(clv_sql) avg_clv
    , avg(monetary_value) avg_monetary_value
    -- , count(*) as qty_records
    FROM TB_CLV_SDF_ML
    group by kc_monetary_value
)
SELECT cluster_number
--    , customer_count
    , ROUND( customer_count / (select sum(customer_count) from TB_CLUSTER ) * 100, 2) as percent_of_customers
    , ROUND( avg_clv, 2) as avg_clv
    , ROUND( avg_monetary_value, 2) as avg_monetary_value
FROM TB_CLUSTER tb1
order by avg_clv desc
"""

sdf_ml_rpt = spark.sql(ml_rpt_sql)
sdf_ml_rpt.printSchema()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
root
 |-- cluster_number: integer (nullable = false)
 |-- percent_of_customers: double (nullable = true)
 |-- avg_clv: double (nullable = true)
 |-- avg_monetary_value: double (nullable = true)
```

In [22]:
```
sdf_ml_rpt.show()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
+--------------+--------------------+-------+------------------+
|cluster_number|percent_of_customers|avg_clv|avg_monetary_value|
+--------------+--------------------+-------+------------------+
|             6|                1.41|1627.96|           6390.09|
|            10|                4.78| 859.47|           2221.99|
|             9|                3.88| 607.03|          15079.84|
|             2|               10.07| 543.86|           1592.38|
|             3|                5.45| 359.51|           3559.05|
|             1|                8.47|  201.7|           1180.36|
|             5|               14.02| 181.48|             579.6|
|             4|               33.19| 156.01|            519.58|
|             7|               10.07| 145.49|            342.34|
|             8|                8.66|  126.8|            529.09|
+--------------+--------------------+-------+------------------+
```

AWS GLUE - Database and Table export

In [26]:
```
## EMR CLUSTER
### upload table to AWS GLUE - database
spark_sql_write_glue_database(db_name=dbname, table_name=tablename,
                              parquet_output_location=output_location)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
 database creation:  DBM
 table name creation ,  TBP_CUSTOMER_CLV
```

In [ ]:
```
!jupyter nbconvert --to html customer_value_emr_cluster_execution_sample.ipynb
```