Ecommerce Capstone

Analyzing E-Commerce dataset to draw insights about sales pattern across different geographic regions and locations. Expanding Customer reach by improving the sales of the business by identifying most valued regions and products.

Data Set - https://www.kaggle.com/olistbr/brazilian-ecommerce/home

Technology Stack – PySpark

IDE – PyCharm / Jupyter Lab

Spark Version – Spark 3.0

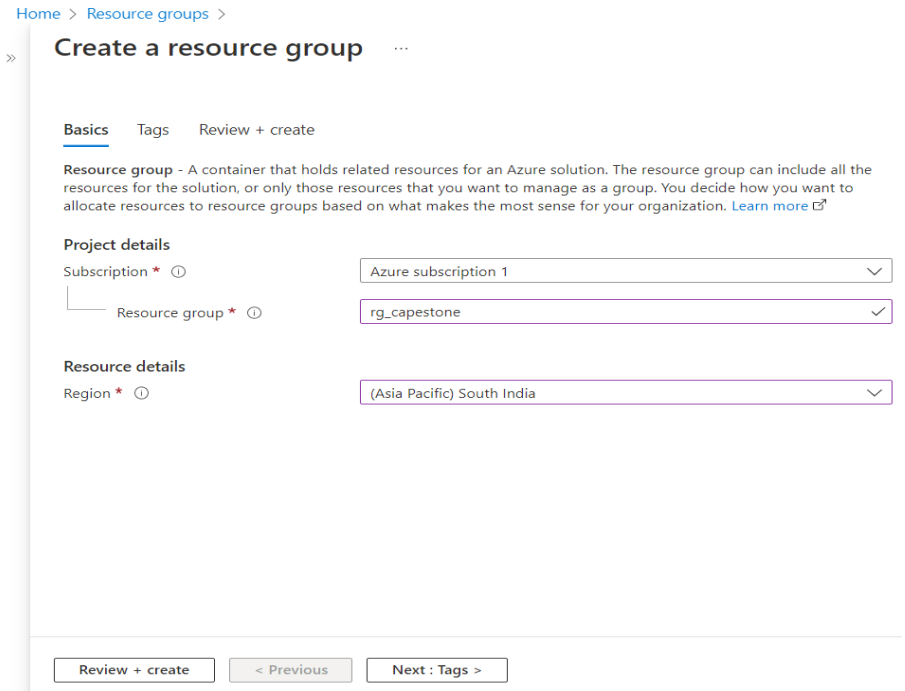
Capstone solution:

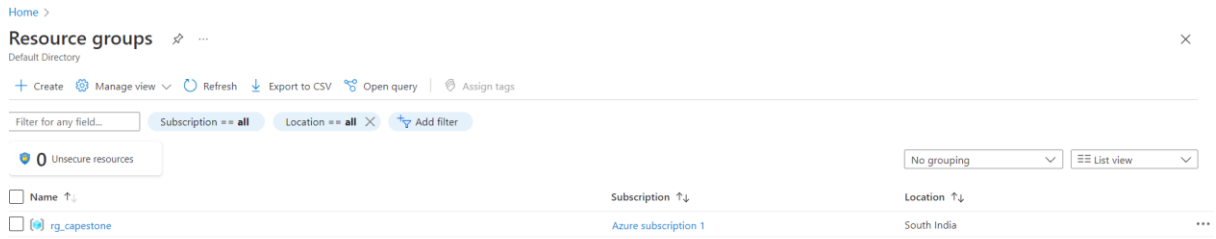
Requirements to setup the environment:

- Azure cloud:
    1. Azure Datalake
    2. Azure Databricks
    3. Key Vaults
    4. App Registration

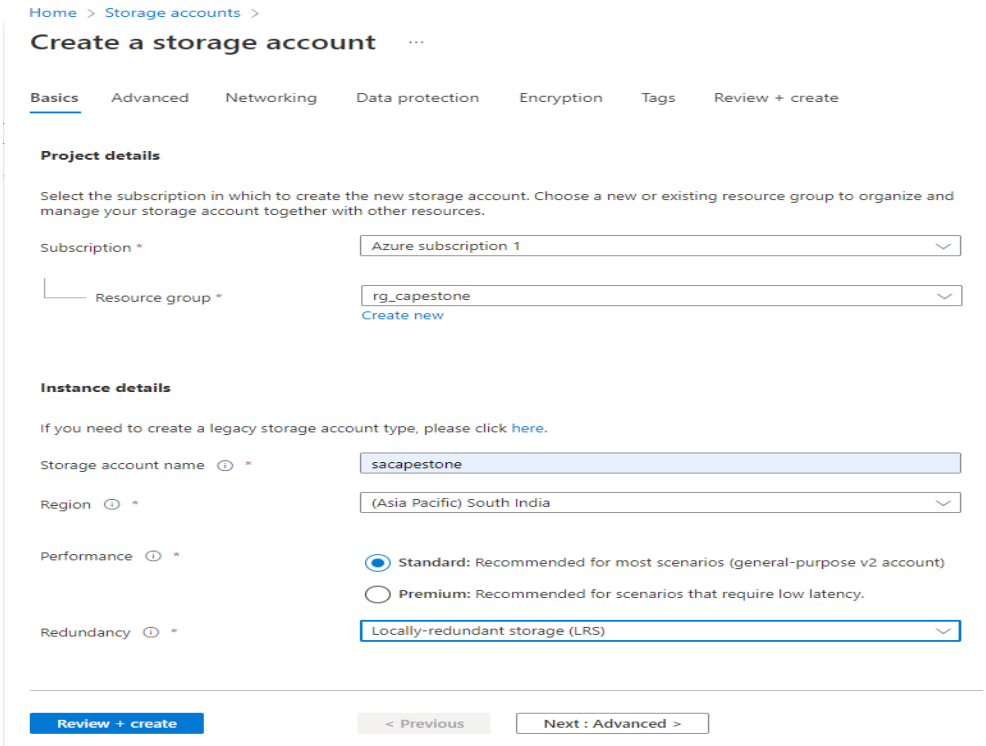
Setting up Azure Databricks and creating cluster:

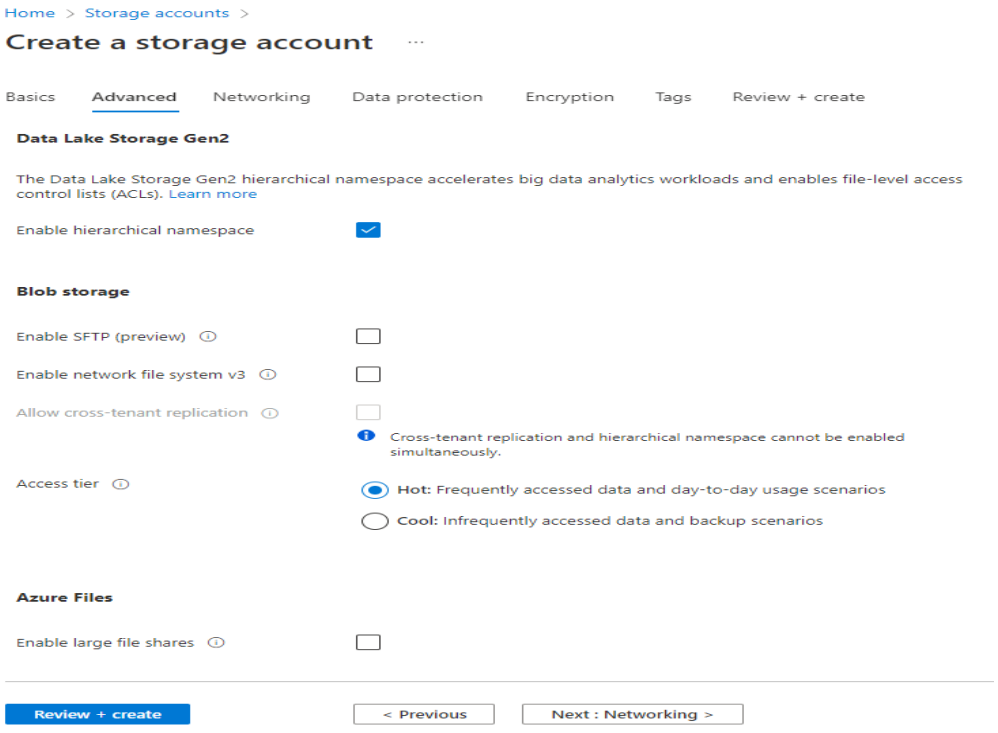
1. Create a resourse group as ‘rg_capestone’ in location South India:

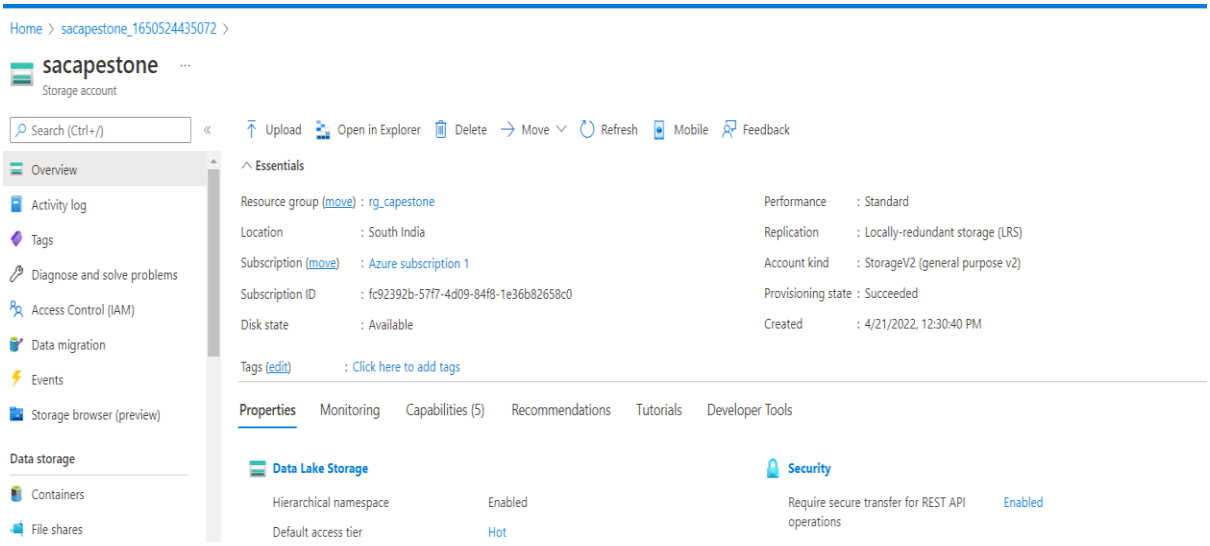




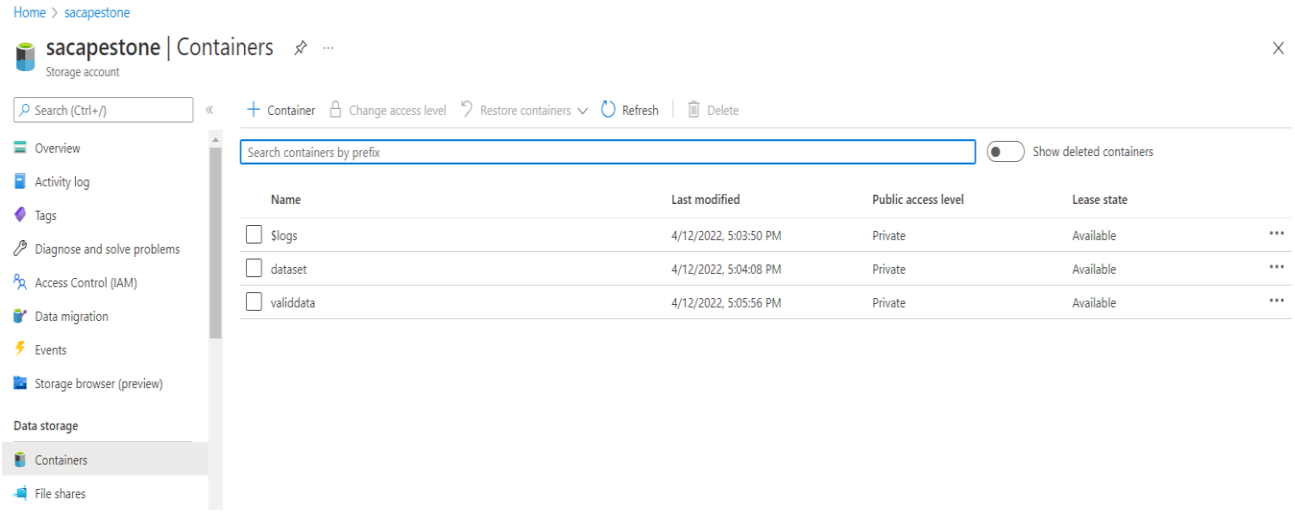
2. Create a datalake storage account with name “sacapestone”:



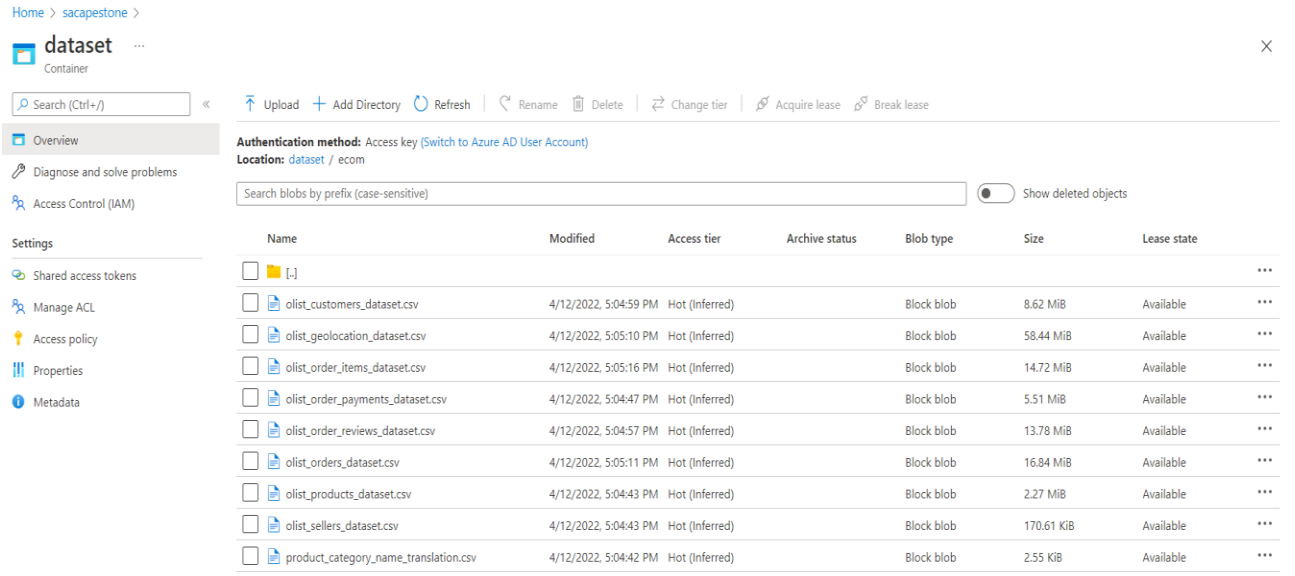




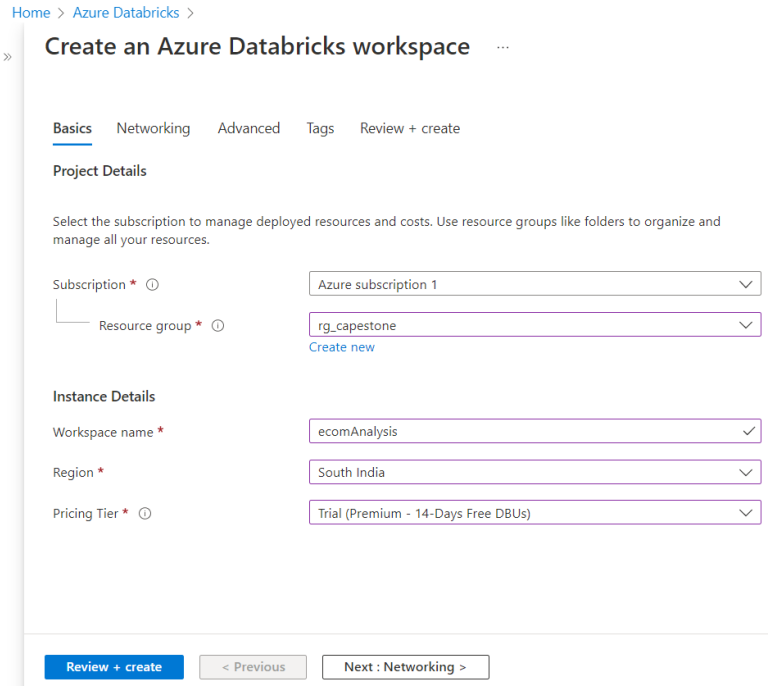
3. Create two containers, “dataset” for uploading input data and “validdata” for writing the output:

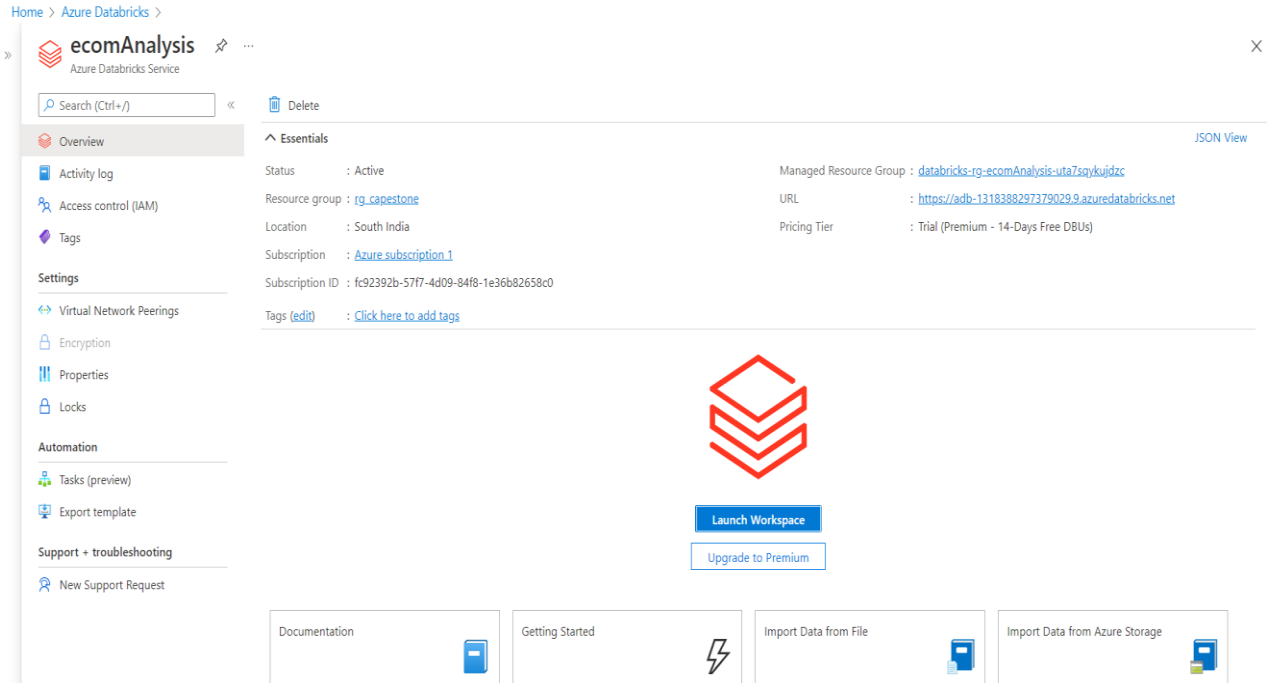


4. Upload the required input data into the “dataset” container inside “ecom” director

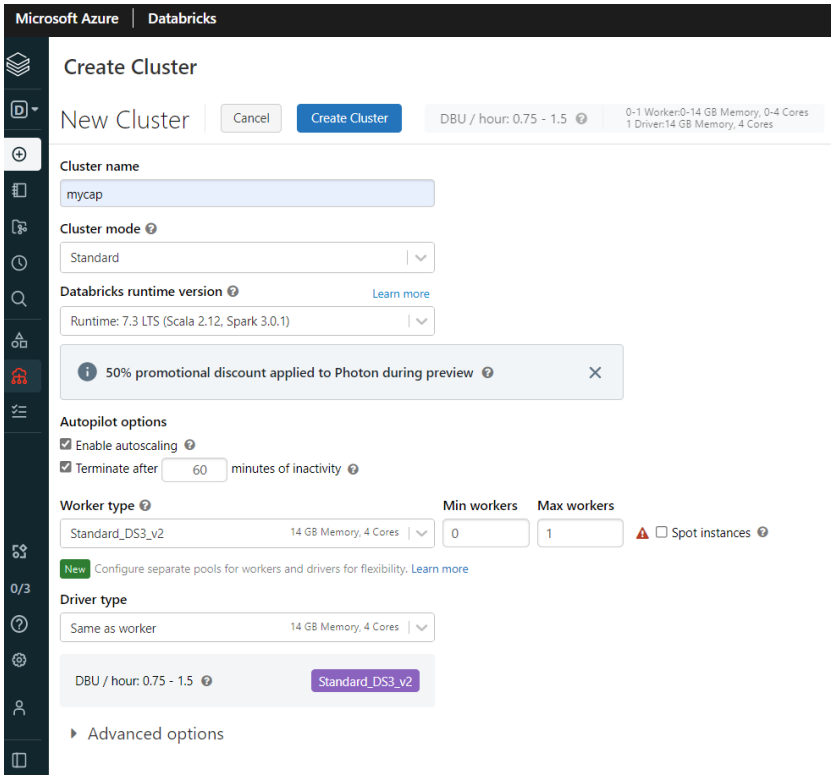


5. Create Dabricks workspace with name “ecomAnalysis” under trial pricing tier:

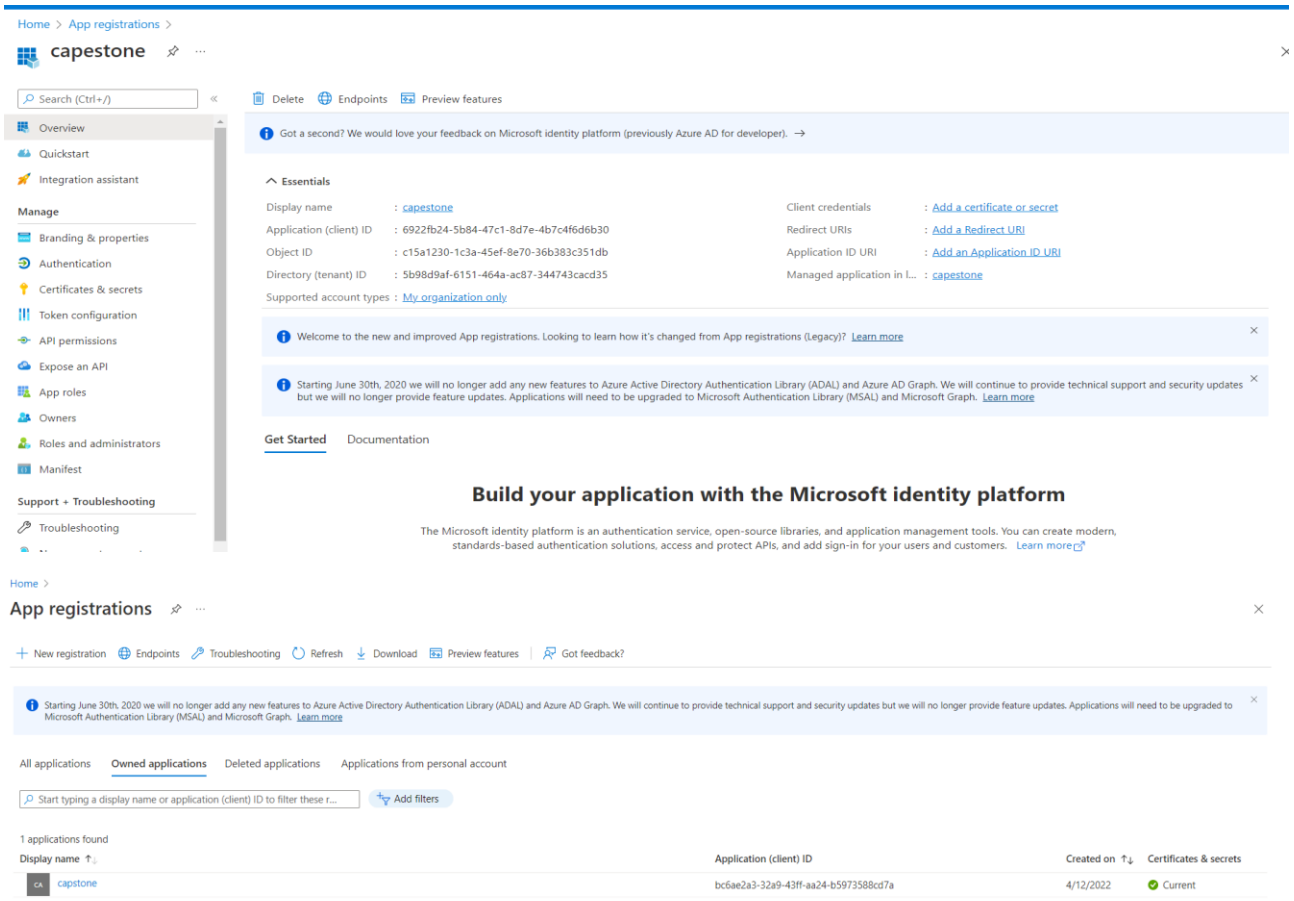




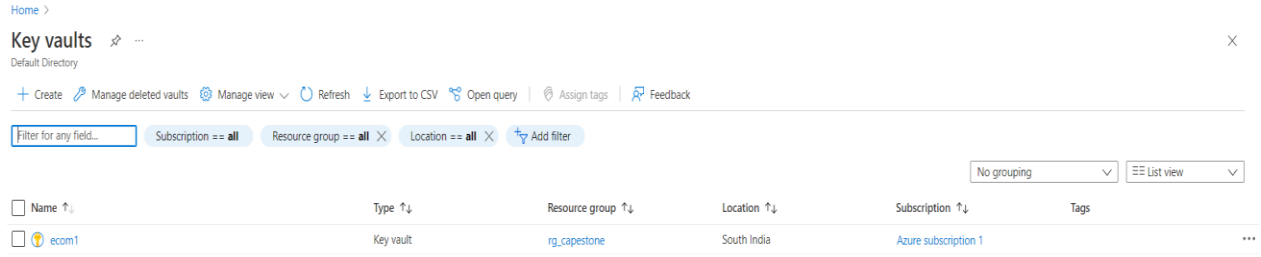
6. Launch the databricks workspace and create a cluster in standard mode with 1 driver and 1 worker nodes. Use runtime: 7.3 LTS (Scala 2.12, Spark 3.0.1):



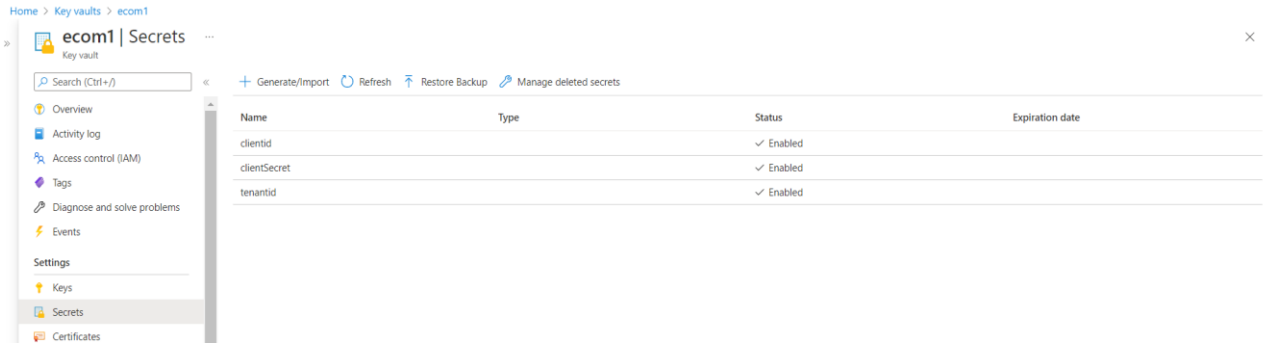
7. Create an App registration named “capestone”:



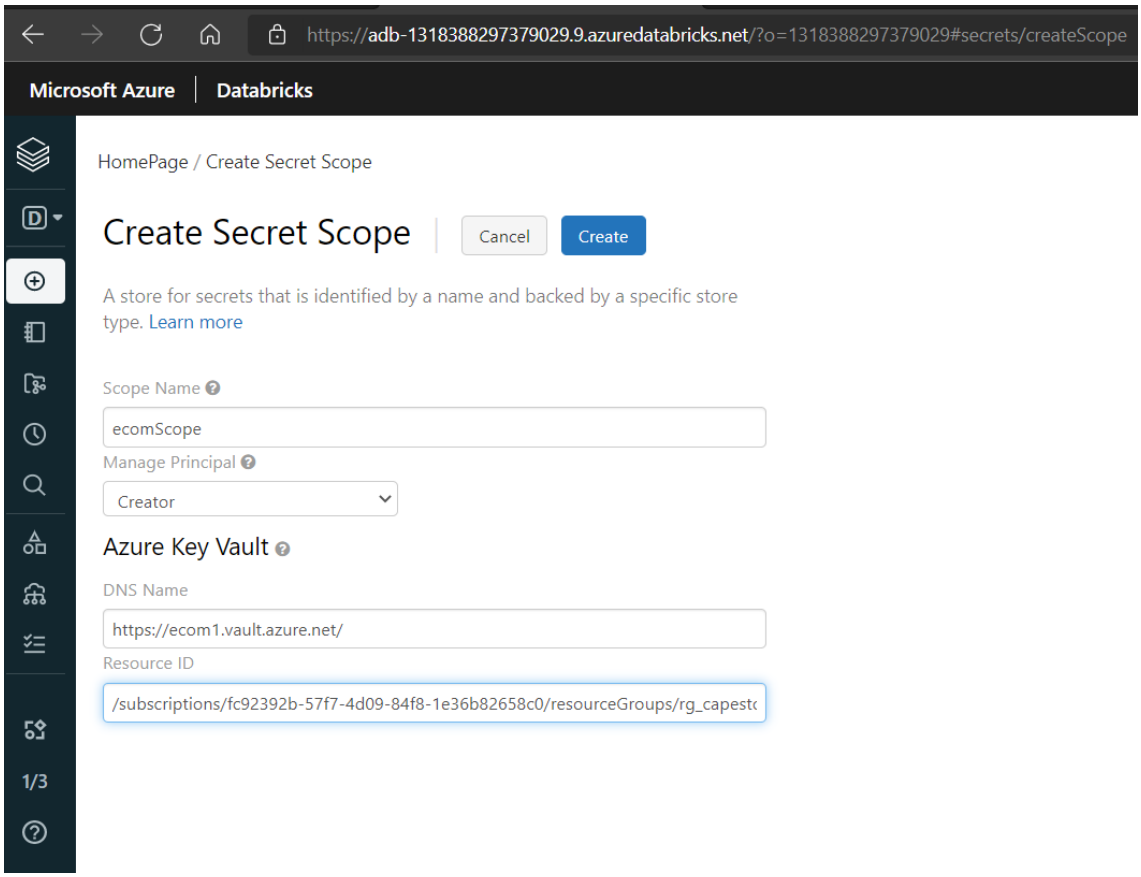
8. Create a key vault named “ecom1”:



9. Create secrets for clientid and tenantid from the registered app. Also create a clientSecret from certificates and secrets in the app “capestone”:

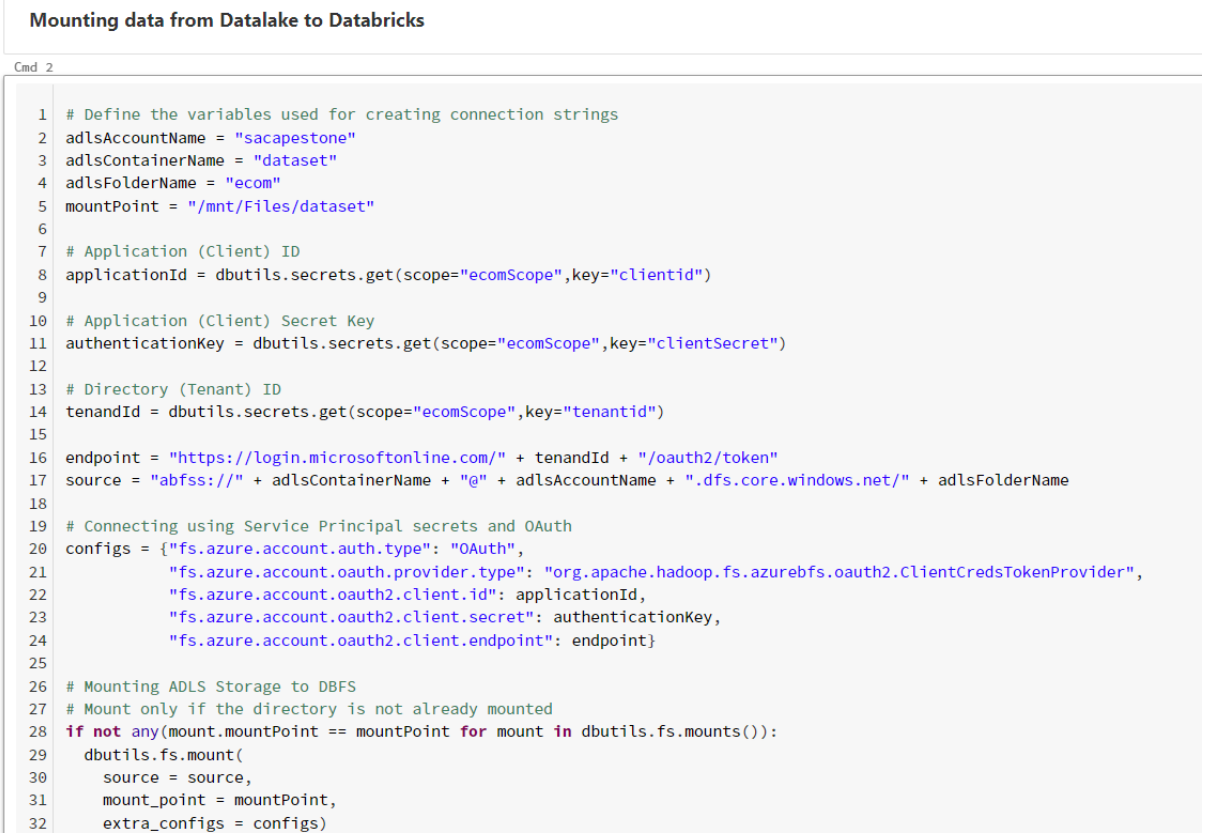


10. Create a Secret Scope by typing the following URL. “ecomScope” is the name of the scope. Get the DNS Name and Resource ID from properties of the created key vault:



Since I didn't have any of the two mentioned IDE's, I have done this whole project in Azure Databricks workspace.

Mounting the data from Datalake into Azure Databricks:



Mounting data from Datalake to Databricks

Cmd 2

```
# Define the variables used for creating connection strings
adlsAccountName = "sacapestone"
adlsContainerName = "dataset"
adlsFolderName = "ecom"
mountPoint = "/mnt/Files/dataset"

# Application (Client) ID
applicationId = dbutils.secrets.get(scope="ecomScope",key="clientid")

# Application (Client) Secret Key
authenticationKey = dbutils.secrets.get(scope="ecomScope",key="clientSecret")

# Directory (Tenant) ID
tenandId = dbutils.secrets.get(scope="ecomScope",key="tenantid")

endpoint = "https://login.microsoftonline.com/" + tenandId + "/oauth2/token"
source = "abfss://" + adlsContainerName + "@" + adlsAccountName + ".dfs.core.windows.net/" + adlsFolderName

# Connecting using Service Principal secrets and OAuth
configs = {"fs.azure.account.auth.type": "OAuth",
           "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
           "fs.azure.account.oauth2.client.id": applicationId,
           "fs.azure.account.oauth2.client.secret": authenticationKey,
           "fs.azure.account.oauth2.client.endpoint": endpoint}

# Mounting ADLS Storage to DBFS
# Mount only if the directory is not already mounted
if not any(mount.mountPoint == mountPoint for mount in dbutils.fs.mounts()):
  dbutils.fs.mount(
    source = source,
    mount_point = mountPoint,
    extra_configs = configs)
```

Code for mounting input data:

```
# Define the variables used for creating connection strings
adlsAccountName = "sacapestone"
adlsContainerName = "dataset"
adlsFolderName = "ecom"
mountPoint = "/mnt/Files/dataset"

# Application (Client) ID
applicationId = dbutils.secrets.get(scope="ecomScope",key="clientid")

# Application (Client) Secret Key
authenticationKey = dbutils.secrets.get(scope="ecomScope",key="clientSecret")

# Directory (Tenant) ID
tenandId = dbutils.secrets.get(scope="ecomScope",key="tenantid")

endpoint = "https://login.microsoftonline.com/" + tenandId + "/oauth2/token"
source = "abfss://" + adlsContainerName + "@" + adlsAccountName + ".dfs.core.windows.net/" +
adlsFolderName

# Connecting using Service Principal secrets and OAuth
configs = {"fs.azure.account.auth.type": "OAuth",
       "fs.azure.account.oauth.provider.type":
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
       "fs.azure.account.oauth2.client.id": applicationId,
```

```
        "fs.azure.account.oauth2.client.secret": authenticationKey,
        "fs.azure.account.oauth2.client.endpoint": endpoint}

# Mounting ADLS Storage to DBFS
# Mount only if the directory is not already mounted
if not any(mount.mountPoint == mountPoint for mount in dbutils.fs.mounts()):
  dbutils.fs.mount(
    source = source,
    mount_point = mountPoint,
    extra_configs = configs)
```

List of files mounted to databricks:

```
%fs
ls dbfs:/mnt/Files/dataset/
```

	path	name	size
1	dbfs:/mnt/Files/dataset/olist_customers_dataset.csv	olist_customers_dataset.csv	9033957
2	dbfs:/mnt/Files/dataset/olist_geolocation_dataset.csv	olist_geolocation_dataset.csv	61273883
3	dbfs:/mnt/Files/dataset/olist_order_items_dataset.csv	olist_order_items_dataset.csv	15438671
4	dbfs:/mnt/Files/dataset/olist_order_payments_dataset.csv	olist_order_payments_dataset.csv	5777138
5	dbfs:/mnt/Files/dataset/olist_order_reviews_dataset.csv	olist_order_reviews_dataset.csv	14451670
6	dbfs:/mnt/Files/dataset/olist_orders_dataset.csv	olist_orders_dataset.csv	17654914
7	dbfs:/mnt/Files/dataset/olist_products_dataset.csv	olist_products_dataset.csv	2379446

Showing all 9 rows.

Code for creating a log file:

```
import logging
import time
import datetime
"""
This program using for custom logging in Pyspark and it will be created a log file.
"""
file_date = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d-%H-%M-%S')
p_dir = '/tmp/'
p_filename = 'custom_log'+file_date+'.log'
p_logfile = p_dir + p_filename
print(p_logfile)
# create logger with 'Custom_log'
logger = logging.getLogger('ecom_log')
logger.setLevel(logging.DEBUG)
# create file handler which logs even debug messages
fh = logging.FileHandler(p_logfile,mode='a')
# create console handler with a higher log level
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
# create formatter and add it to the handlers
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
#setting for ingoring frequest log information
#logging.getLogger("py4j").setLevel(logging.ERROR)
# tell the handler to use this format
#fh (file Handler)
fh.setFormatter(formatter)
#ch (console handler)
ch.setFormatter(formatter)
#Clearing old frequent log information to ignore that.
if (logger.hasHandlers()):
    logger.handlers.clear()
# add the handlers to the logger
logger.addHandler(fh)
logger.addHandler(ch)
```

Starting the required loggers:

```
logger.info("*****START INFO LOGGER HERE*****")
logger.debug("*****START DEBUG LOGGER HERE*****")
logger.error("*****START ERROR LOGGER HERE*****")
logger.warning("*****START WARNING LOGGER HERE*****")
logger.critical("*****START CRITICAL LOGGER HERE*****")
```

```
2022-04-19 13:53:23,778 - ecom_log - INFO - *****START INFO LOGGER HERE*****
2022-04-19 13:53:23,779 - ecom_log - DEBUG - *****START DEBUG LOGGER HERE*****
2022-04-19 13:53:23,779 - ecom_log - ERROR - *****START ERROR LOGGER HERE*****
2022-04-19 13:53:23,779 - ecom_log - WARNING - *****START WARNING LOGGER HERE*****
2022-04-19 13:53:23,779 - ecom_log - CRITICAL - *****START CRITICAL LOGGER HERE*****
```

Extracting the data and performing cleaning operations on each of the nine input files:

Performing extraction and cleaning the dataset

Cmd 12

```
from pyspark import SparkConf, SparkContext
from pyspark.sql.functions import *
from pyspark.sql import SparkSession
import ipywidgets as widgets
from ipywidgets import interact, interactive
import pygal
spark = SparkSession.builder.appName("E-Commerce_Analysis").getOrCreate()
```

```
Command took 0.20 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:06 PM on mycap
```

Cmd 13

```
df_customerdata = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_customers_dataset.csv")
df_cus=df_customerdata.dropDuplicates()
df_cus.printSchema()
df_cus.show(100)
```

```
▸ (3) Spark Jobs
▸ df_customerdata: pyspark.sql.dataframe.DataFrame = [customer_id: string, customer_unique_id: string ... 3 more fields]
▸ df_cus: pyspark.sql.dataframe.DataFrame = [customer_id: string, customer_unique_id: string ... 3 more fields]
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)
```

1. Customer Data:

df_customerdata = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_customers_dataset.csv")
df_cus=df_customerdata.dropDuplicates()
df_cus.printSchema()
df_cus.show(100)

- We observe that there are no null values. We perform only drop duplicates on this data.

```
df_cus.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_cus.columns]).show()
```

(3) Spark Jobs

```
+-----------+------------------+-------------------------+-------------+--------------+
|customer_id|customer_unique_id|customer_zip_code_prefix|customer_city|customer_state|
+-----------+------------------+-------------------------+-------------+--------------+
|          0|                 0|                        0|            0|             0|
+-----------+------------------+-------------------------+-------------+--------------+
```

Command took 3.08 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:06 PM on mycap

Cmd 15

```
df_orderitem = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_items_dataset.csv")
df_ordi = df_orderitem.dropDuplicates()
df_ordi.printSchema()
df_ordi.show(100)
```

(3) Spark Jobs

df_orderitem: pyspark.sql.dataframe.DataFrame = [order_id: string, order_item_id: string ... 5 more fields]

df_ordi: pyspark.sql.dataframe.DataFrame = [order_id: string, order_item_id: string ... 5 more fields]

```
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)
```

2. Order Items Data:

df_orderitem = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_items_dataset.csv")
df_ordi = df_orderitem.dropDuplicates()
df_ordi.printSchema()
df_ordi.show(100)

- We observe that there are no null values. We perform only drop duplicates on this data.

df_ordi=df_ordi.withColumn('shipping_limit_date',to_timestamp('shipping_limit_date')).withColumn('order_item_id',df_ordi.order_item_id.cast('int'))
df_ordi.show(100)

- 'shipping_limit_date' column is converted to proper timestamp format and 'order_item_id' column is casted to 'int'.

```
df_ordi.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_ordi.columns]).show()
```

(3) Spark Jobs

```
+--------+-------------+----------+---------+-------------------+-----+-------------+
|order_id|order_item_id|product_id|seller_id|shipping_limit_date|price|freight_value|
+--------+-------------+----------+---------+-------------------+-----+-------------+
|       0|            0|         0|        0|                  0|    0|            0|
+--------+-------------+----------+---------+-------------------+-----+-------------+
```

Command took 2.65 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:07 PM on mycap

Cmd 17

```
df_ordi=df_ordi.withColumn('shipping_limit_date',to_timestamp('shipping_limit_date')).withColumn('order_item_id',df_ordi.order_item_id.cast('int'))
df_ordi.show(100)
```

(2) Spark Jobs

df_ordi: pyspark.sql.dataframe.DataFrame = [order_id: string, order_item_id: integer ... 5 more fields]

```
+--------------------+-------------+--------------------+--------------------+-------------------+-----+-------------+
|            order_id|order_item_id|          product_id|           seller_id|shipping_limit_date|price|freight_value|
+--------------------+-------------+--------------------+--------------------+-------------------+-----+-------------+
|00143d0f86d6fbd9f...|            2|e95ee6822b66ac605...|a17f621c590ea0fab...|2017-10-20 16:07:52|21.33|        15.10|
|0015ebb40fb17286b...|            1|50fd2b788dc166edd...|8b321bb669392f516...|2018-01-18 09:11:24|21.90|        15.10|
|00125cb692d048878...|            1|1c0c0093a48f13ba7...|41b39e28db005d973...|2017-03-29 13:05:42|109.90|        25.51|
|0011d82c4b53e22e8...|            1|c389f712c4b4510bc...|bfd27a966d91cfaaf...|2018-01-29 21:51:25|289.00|        26.33|
|0005a1a1728c9d785...|            1|310ae3c140ff94b03...|a416b6a846a117243...|2018-03-26 18:31:29|145.95|        11.65|
|002f16b7bc4530031...|            1|d54c5b81fc2b38707...|0241d4d5d36f10f80...|2018-07-04 17:11:12|249.90|        34.23|
|00259a44fcad3fc04...|            1|0c4d0a08f95c7b7dc...|9f505651f4a6abe90...|2018-01-03 17:59:35| 19.99|        14.10|
|00324b3eda39ba5ec...|            1|54e5063e43f27f747...|0dd184061fb0eaa7c...|2018-04-17 09:12:11| 76.00|        34.07|
```

3. Payments Data:

df_payment = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_payments_dataset.csv")
df_pay = df_payment.dropDuplicates()
df_pay.printSchema()
df_pay.show(100)

```
df_payment = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_payments_dataset.csv")
df_pay = df_payment.dropDuplicates()
df_pay.printSchema()
df_pay.show(100)
```

(3) Spark Jobs

df_payment: pyspark.sql.dataframe.DataFrame = [order_id: string, payment_sequential: string ... 3 more fields]

df_pay: pyspark.sql.dataframe.DataFrame = [order_id: string, payment_sequential: string ... 3 more fields]

```
root
 |-- order_id: string (nullable = true)
 |-- payment_sequential: string (nullable = true)
 |-- payment_type: string (nullable = true)
 |-- payment_installments: string (nullable = true)
 |-- payment_value: string (nullable = true)
```

order_id	payment_sequential	payment_type	payment_installments	payment_value
4d3bfdf4c7d6578ea...	1	credit_card	1	77.29
12e5cfe0e4716b59a...	1	credit_card	10	157.45
298fcdf1f73eb413e...	1	credit_card	2	96.12
595f598849d89203c...	1	credit_card	1	26.58
4b2d035932915aa9c...	1	credit_card	3	187.51
aa964845f1de81254...	1	credit_card	2	27.35
25e8ea4e93396b6fa...	1	credit_card	1	65.71
a9810da82917af2d9...	1	credit_card	1	24.39
4214cda550ece8ee6...	1	credit_card	2	170.57
3d7239c394a212faa...	1	credit_card	3	51.84
0573b5e23cbd79800...	1	boleto	1	51.95

- We observe that there are no null values. We perform only drop duplicates on this data.

```
df_pay.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_pay.columns]).show()
```

(3) Spark Jobs

order_id	payment_sequential	payment_type	payment_installments	payment_value
0	0	0	0	0

Command took 1.63 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:07 PM on mycap

Cmd 20

```
df_geoloc = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_geolocation_dataset.csv")
df_geo =df_geoloc.dropDuplicates()
df_geo.printSchema()
df_geo.show(100)
```

(3) Spark Jobs

df_geoloc: pyspark.sql.dataframe.DataFrame = [geolocation_zip_code_prefix: string, geolocation_lat: string ... 3 more fields]

df_geo: pyspark.sql.dataframe.DataFrame = [geolocation_zip_code_prefix: string, geolocation_lat: string ... 3 more fields]

```
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)
```

4. Geolocation Data:

df_geoloc = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_geolocation_dataset.csv")
df_geo =df_geoloc.dropDuplicates()
df_geo.printSchema()
df_geo.show(100)

- We observe that there are no null values. We perform only drop duplicates on this data.

```
df_geo.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_geo.columns]).show()
```

(3) Spark Jobs

```
+--------------------------+---------------+---------------+----------------+-----------------+
|geolocation_zip_code_prefix|geolocation_lat|geolocation_lng|geolocation_city|geolocation_state|
+--------------------------+---------------+---------------+----------------+-----------------+
|                         0|              0|              0|               0|                0|
+--------------------------+---------------+---------------+----------------+-----------------+
```

Command took 4.77 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:07 PM on mycap

md 22

```
df_review = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_reviews_dataset.csv")
df_rev =df_review.dropDuplicates()
df_rev.printSchema()
df_rev.show(100)
```

(3) Spark Jobs

df_review: pyspark.sql.dataframe.DataFrame = [review_id: string, order_id: string ... 5 more fields]

df_rev: pyspark.sql.dataframe.DataFrame = [review_id: string, order_id: string ... 5 more fields]

```
root
 |-- review_id: string (nullable = true)
 |-- order_id: string (nullable = true)
 |-- review_score: string (nullable = true)
 |-- review_comment_title: string (nullable = true)
 |-- review_comment_message: string (nullable = true)
 |-- review_creation_date: string (nullable = true)
 |-- review_answer_timestamp: string (nullable = true)
```

5. Review Data:

df_review = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_reviews_dataset.csv")
df_rev =df_review.dropDuplicates()
df_rev.printSchema()
df_rev.show(100)

- We observe that there are null values. We perform drop duplicates on this data.

```
df_rev.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_rev.columns]).show()
```

(3) Spark Jobs

```
+---------+--------+------------+--------------------+----------------------+--------------------+----------------------+
|review_id|order_id|review_score|review_comment_title|review_comment_message|review_creation_date|review_answer_timestamp|
+---------+--------+------------+--------------------+----------------------+--------------------+----------------------+
|        1|    2181|        2324|               92077|                 62994|                8679|                  8700|
+---------+--------+------------+--------------------+----------------------+--------------------+----------------------+
```

Command took 2.29 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:08 PM on mycap

Cmd 24

Python

```
df_rev = df_rev.na.drop(subset=['review_creation_date','review_answer_timestamp'])
df_rev = df_rev.withColumn('review_creation_date',to_timestamp('review_creation_date')).withColumn('review_answer_timestamp',to_timestamp('review_answer_timestamp'))
df_rev.show(100)
```

(2) Spark Jobs

df_rev: pyspark.sql.dataframe.DataFrame = [review_id: string, order_id: string ... 5 more fields]

```
+--------------------+--------------------+------------+--------------------+--------------------+--------------------+----------------------+
|           review_id|            order_id|review_score|review_comment_title|review_comment_message|review_creation_date|review_answer_timestamp|
+--------------------+--------------------+------------+--------------------+--------------------+--------------------+----------------------+
|34e5d18f3bb64d586...|bf6861804f97bcf76...|           4|                null|                null| 2017-03-31 00:00:00|   2017-04-01 23:43:01|
|ab3056e4fb5a36df4...|aad1dcbe4c9fe2e34...|           2|                null|                null| 2018-05-13 00:00:00|   2018-05-14 16:29:15|
|cb2fc3e5711b5ae85...|34e6d418f368f8079...|           3|                null|  Produto bom, poré...| 2018-01-31 00:00:00|   2018-01-31 23:29:21|
|31b63ea961f9f43c2...|46d2d651f006b9b94...|           5|                null|                null| 2018-08-17 00:00:00|   2018-08-31 03:13:36|
|caf7d6d68090fedf0...|c186ebe3937470a2f...|           5|                null|                null| 2017-12-05 00:00:00|   2017-12-07 23:34:56|
|d23ac17822a812782...|705402bc1d9560673...|           1|                null|                null| 2018-04-22 00:00:00|   2018-04-26 10:25:11|
|8670d52e15e00043a...|b9bf720beb4ab3728...|           4|          recomendo|  aparelho eficient...| 2018-05-22 00:00:00|   2018-05-23 16:45:47|
```

- We also perform drop na and convert to proper timestamp format on ‘review_creation_date’ and ‘review_answer_timestamp’ columns of this data.

df_rev = df_rev.na.drop(subset=['review_creation_date','review_answer_timestamp'])
df_rev =
df_rev.withColumn('review_creation_date',to_timestamp('review_creation_date')).withColumn('review_answer_timestamp',to_timestamp('review_answer_timestamp'))
df_rev.show(100)

6. Products Data:

df_product = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_products_dataset.csv")
df_prod=df_product.dropDuplicates()
df_prod.printSchema()
df_prod.show(100)

```
1 df_product = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_products_dataset.csv")
2 df_prod=df_product.dropDuplicates()
3 df_prod.printSchema()
4 df_prod.show(100)
```

(3) Spark Jobs

df_product: pyspark.sql.dataframe.DataFrame = [product_id: string, product_category_name: string ... 7 more fields]

df_prod: pyspark.sql.dataframe.DataFrame = [product_id: string, product_category_name: string ... 7 more fields]

```
root
 |-- product_id: string (nullable = true)
 |-- product_category_name: string (nullable = true)
 |-- product_name_lenght: string (nullable = true)
 |-- product_description_lenght: string (nullable = true)
 |-- product_photos_qty: string (nullable = true)
 |-- product_weight_g: string (nullable = true)
 |-- product_length_cm: string (nullable = true)
 |-- product_height_cm: string (nullable = true)
 |-- product_width_cm: string (nullable = true)

+--------------------+--------------------+-------------------+--------------------------+------------------+----------------+-----------------+-----------------+-------------
--+
|          product_id|product_category_name|product_name_lenght|product_description_lenght|product_photos_qty|product_weight_g|product_length_cm|product_height_cm|product_width_
cm|
+--------------------+--------------------+-------------------+--------------------------+------------------+----------------+-----------------+-----------------+-------------
--+
|3e0f398b664b26e32...|       esporte_lazer|                57|                       900|                 4|             400|               16|                4|
11|
|96bd76ec8810374ed...|       esporte_lazer|                46|                       250|                 1|             154|               18|                9|
```

- We observe that there are null values. We perform drop duplicates and drop na on this data.

df_prod = df_prod.na.drop()
df_prod.show(100)

```
1 df_prod.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_prod.columns]).show()
```

(3) Spark Jobs

```
+----------+--------------------+-------------------+--------------------------+------------------+----------------+-----------------+-----------------+-----------------+
|product_id|product_category_name|product_name_lenght|product_description_lenght|product_photos_qty|product_weight_g|product_length_cm|product_height_cm|product_width_cm|
+----------+--------------------+-------------------+--------------------------+------------------+----------------+-----------------+-----------------+-----------------+
|         0|                 610|                610|                       610|               610|               2|                2|                2|               2|
+----------+--------------------+-------------------+--------------------------+------------------+----------------+-----------------+-----------------+-----------------+
```

Command took 1.33 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:09 PM on mycap

Cmd 27

```
1 df_prod = df_prod.na.drop()
2 df_prod.show(100)
```

(2) Spark Jobs

df_prod: pyspark.sql.dataframe.DataFrame = [product_id: string, product_category_name: string ... 7 more fields]

```
+--------------------+--------------------+-------------------+--------------------------+------------------+----------------+-----------------+-----------------+-------------
--+
|          product_id|product_category_name|product_name_lenght|product_description_lenght|product_photos_qty|product_weight_g|product_length_cm|product_height_cm|product_width_
cm|
+--------------------+--------------------+-------------------+--------------------------+------------------+----------------+-----------------+-----------------+-------------
--+
|3e0f398b664b26e32...|       esporte_lazer|                57|                       900|                 4|             400|               16|                4|
11|
|96bd76ec8810374ed...|       esporte_lazer|                46|                       250|                 1|             154|               18|                9|
15|
```

7. Orders Data:

df_order = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_orders_dataset.csv")
df_ord=df_order.dropDuplicates()
df_ord.printSchema()
df_ord.show(100)

```
1 df_order = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_orders_dataset.csv")
2 df_ord=df_order.dropDuplicates()
3 df_ord.printSchema()
4 df_ord.show(100)
```

(3) Spark Jobs

df_order: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 6 more fields]

df_ord: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 6 more fields]

```
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)
 |-- order_delivered_customer_date: string (nullable = true)
 |-- order_estimated_delivery_date: string (nullable = true)

+--------------------+--------------------+------------+------------------------+-------------------+----------------------------+-----------------------------+-----------------------------+
|            order_id|         customer_id|order_status|order_purchase_timestamp|  order_approved_at|order_delivered_carrier_date|order_delivered_customer_date|order_estimated_delivery_date|
+--------------------+--------------------+------------+------------------------+-------------------+----------------------------+-----------------------------+-----------------------------+
|6ea2f835b4556291f...|c7340080e39435614...|   delivered|     2017-11-24 21:27:48|2017-11-25 00:21:09|         2017-12-13 21:14:05|          2017-12-28 18:59:23|          2017-12-21 00:00:00|
|6d25592267349b322...|5bb39c890c91b1d26...|   delivered|     2018-08-26 22:04:34|2018-08-28 04:10:18|         2018-08-28 12:56:00|          2018-08-29 12:40:53|          2018-08
```

- We observe that there are null values. We perform drop duplicates on this data.
- We convert the required columns into proper timestamp format.

df_ord = df_ord.withColumn('order_purchase_timestamp', to_timestamp('order_purchase_timestamp')).withColumn('order_delivered_carrier_data', to_timestamp('order_delivered_carrier_date')).withColumn('order_approved_at',to_timestamp('order_approved_at')).withColumn('order_delivered_customer_date',to_timestamp('order_delivered_customer_date')).withColumn('order_estimed_delivery_date',to_timestamp('order_estimated_delivery_date'))
df_ord.printSchema()
df_ord.show(100)

```
1 df_ord.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_ord.columns]).show()
```

(3) Spark Jobs

```
+--------+-----------+------------+------------------------+----------------+----------------------------+-----------------------------+-----------------------------+
|order_id|customer_id|order_status|order_purchase_timestamp|order_approved_at|order_delivered_carrier_date|order_delivered_customer_date|order_estimated_delivery_date|
+--------+-----------+------------+------------------------+----------------+----------------------------+-----------------------------+-----------------------------+
|       0|          0|           0|                       0|             160|                        1783|                         2965|                            0|
+--------+-----------+------------+------------------------+----------------+----------------------------+-----------------------------+-----------------------------+
```

Command took 2.17 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:09 PM on mycap

md 30

```
1 df_ord = df_ord.withColumn('order_purchase_timestamp', to_timestamp('order_purchase_timestamp')).withColumn('order_delivered_carrier_data',
  to_timestamp('order_delivered_carrier_date')).withColumn('order_approved_at',to_timestamp('order_approved_at')).withColumn('order_delivered_customer_date',to_timestamp('order
  _delivered_customer_date')).withColumn('order_estimed_delivery_date',to_timestamp('order_estimated_delivery_date'))
2 df_ord.printSchema()
3 df_ord.show(100)
```

(2) Spark Jobs

df_ord: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 8 more fields]

```
root
 |-- order_id: string (nullable = true)
 |-- customer_id: string (nullable = true)
 |-- order_status: string (nullable = true)
 |-- order_purchase_timestamp: timestamp (nullable = true)
 |-- order_approved_at: timestamp (nullable = true)
 |-- order_delivered_carrier_date: string (nullable = true)
 |-- order_delivered_customer_date: timestamp (nullable = true)
```

8. Sellers Data:

df_seller = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_sellers_dataset.csv")
df_sell = df_seller.dropDuplicates()
df_sell.printSchema()
df_sell.show(100)

```
1 df_seller = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_sellers_dataset.csv")
2 df_sell = df_seller.dropDuplicates()
3 df_sell.printSchema()
4 df_sell.show(100)
```

▸ (3) Spark Jobs

▸ df_seller: pyspark.sql.dataframe.DataFrame = [seller_id: string, seller_zip_code_prefix: string ... 2 more fields]

▸ df_sell: pyspark.sql.dataframe.DataFrame = [seller_id: string, seller_zip_code_prefix: string ... 2 more fields]

```
root
 |-- seller_id: string (nullable = true)
 |-- seller_zip_code_prefix: string (nullable = true)
 |-- seller_city: string (nullable = true)
 |-- seller_state: string (nullable = true)

+--------------------+----------------------+--------------------+------------+
|           seller_id|seller_zip_code_prefix|         seller_city|seller_state|
+--------------------+----------------------+--------------------+------------+
|d650b663c3b5f6fb3...|                 06713|              cotia|          SP|
|4cf490a58259286ad...|                 14910|           tabatinga|          SP|
|b7ba853e9551f4558...|                 95500|santo antonio da ...|          RS|
|276677b5d08786d5d...|                 31730|      belo horizonte|          MG|
|51a04a8a6bdcb23de...|                 12914|   braganca paulista|          SP|
|8cb7c5ddf41f4d506...|                 75110|            anapolis|          GO|
|5c030029b5916fed0...|                 88075|            sao jose|          SC|
|13c2ed7698b3ca92d...|                 84010|        ponta grossa|          PR|
|41ab63a91b8b264e8...|                 31555|      belo horizonte|          MG|
|41e0fa5761c886a63...|                 11446|             guaruja|          SP|
```

- We observe that there are no null values. We perform only drop duplicates on this data.

```
1 df_sell.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_sell.columns]).show()
```

▸ (3) Spark Jobs

```
+---------+----------------------+-----------+------------+
|seller_id|seller_zip_code_prefix|seller_city|seller_state|
+---------+----------------------+-----------+------------+
|        0|                     0|          0|           0|
+---------+----------------------+-----------+------------+
```

Command took 0.71 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:10 PM on mycap

Cmd 33

```
1 df_tran =spark.read.format("csv").option("header","true").load("dbfs:/mnt/Files/dataset/product_category_name_translation.csv")
2 df_tran.printSchema()
3 df_tran.show(100)
```

▸ (2) Spark Jobs

▸ df_tran: pyspark.sql.dataframe.DataFrame = [product_category_name: string, product_category_name_english: string]

```
root
 |-- product_category_name: string (nullable = true)
 |-- product_category_name_english: string (nullable = true)

+----------------------+-----------------------------+
|product_category_name|product_category_name_english|
+----------------------+-----------------------------+
|         beleza_saude|                health_beauty|
| informatica_acess...|         computers_accesso...|
|           automotivo|                         auto|
```

9. Product category name translation Data:

df_tran
=spark.read.format("csv").option("header","true").load("dbfs:/mnt/Files/dataset/product_category_name_translation.csv")
df_tran.printSchema()
df_tran.show(100)

```
df_geo = df_sell.join(df_geo).where(df_sell['seller_zip_code_prefix']==df_geo['geolocation_zip_code_prefix']).drop(*
('seller_id','seller_zip_code_prefix','seller_state','geolocation_city')).withColumnRenamed('seller_city','geolocation_city')
df_geo = df_geo.select('geolocation_zip_code_prefix','geolocation_lat','geolocation_lng','geolocation_city','geolocation_state')
df_geo.show(100)
```

(3) Spark Jobs

df_geo: pyspark.sql.dataframe.DataFrame = [geolocation_zip_code_prefix: string, geolocation_lat: string ... 3 more fields]

geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state
01222	-23.545460503750906	-46.64710252553522	sao paulo	SP
01210	-23.538204848462094	-46.63944608742571	sao paulo	SP
01233	-23.532868152374615	-46.66033569525076	sao paulo	SP
01201	-23.534308277829044	-46.64974682292688	sao paulo	SP
01201	-23.534308277829044	-46.64974682292688	sao paulo	SP
01310	-23.560629789889834	-46.65726094364944	sao paulo	SP
01310	-23.560629789889834	-46.65726094364944	sao paulo	SP

Solution to the given use cases:

1. Top Selling Products:

```
try:
    order_item_grpby = df_ordi.groupBy('product_id').agg({'order_item_id':'max'})
    top_selling = order_item_grpby.join(df_prod,['product_id'])
    top_selling = top_selling.join(df_tran,['product_category_name'])
    top_selling =
top_selling.groupBy('product_category_name_english').agg({'max(order_item_id)':'max'}).withColumnRe
named('max(max(order_item_id))','order_item').withColumnRenamed('product_category_name_english','
Product_Name')
    top_selling = top_selling.orderBy('order_item',ascending=False)
    top_selling.show()
    logger.info("*****Finding top seller*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding top seller*****")
```

Top selling products

Cmd 36

Python

```
try:
    order_item_grpby = df_ordi.groupBy('product_id').agg({'order_item_id':'max'})
    top_selling = order_item_grpby.join(df_prod,['product_id'])
    top_selling = top_selling.join(df_tran,['product_category_name'])
    top_selling =
top_selling.groupBy('product_category_name_english').agg({'max(order_item_id)':'max'}).withColumnRenamed('max(max(order_item_id))','order_item').withColumnRenamed('product_ca
tegory_name_english','Product_Name')
    top_selling = top_selling.orderBy('order_item',ascending=False)
    top_selling.show()
    logger.info("*****Finding top seller*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding top seller*****")
```

(5) Spark Jobs

order_item_grpby: pyspark.sql.dataframe.DataFrame = [product_id: string, max(order_item_id): integer]

top_selling: pyspark.sql.dataframe.DataFrame = [Product_Name: string, order_item: integer]

Product_Name	order_item
health_beauty	21
computers_accesso...	20
auto	20
furniture_decor	15
garden_tools	15
telephony	14

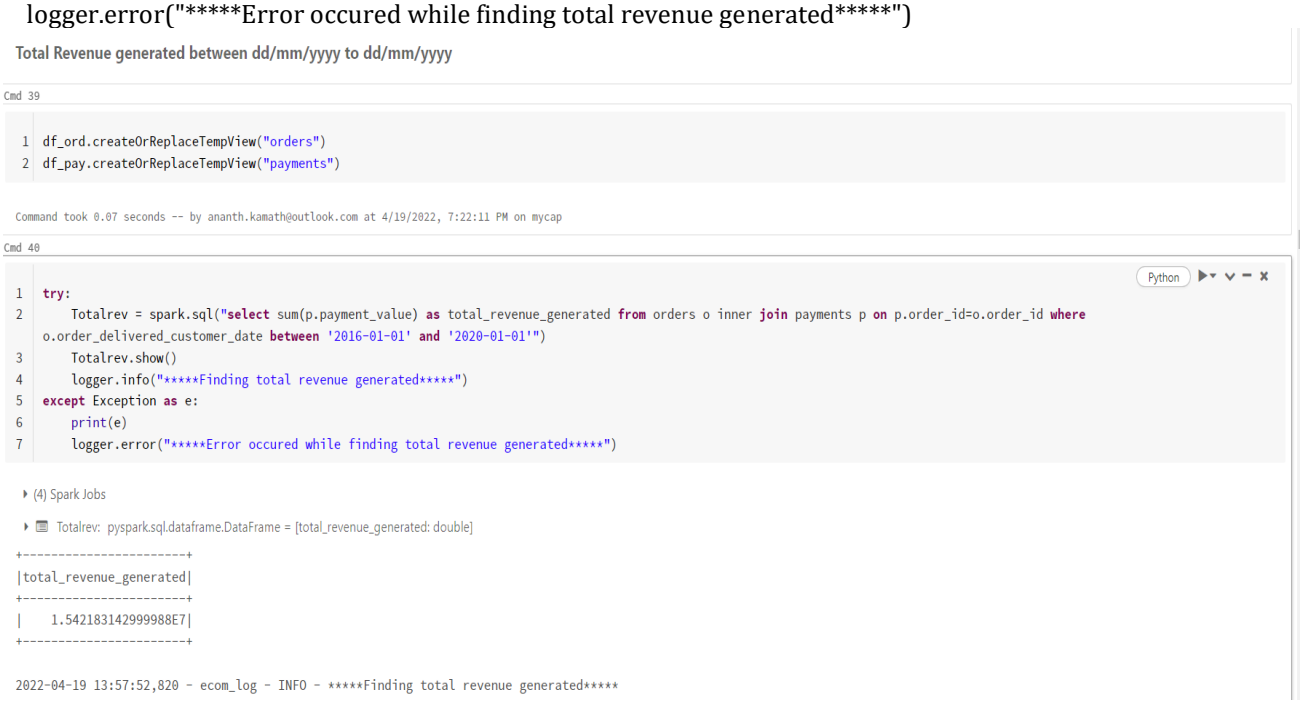
Saving the output into Datalake:

```
top_selling.write.mode("overwrite").csv("/mnt/Files/validdata/output/top_selling.csv")
```

(7) Spark Jobs

2. Total Revenue generated between dd/mm/yyyy to dd/mm/yyyy:

```
df_ord.createOrReplaceTempView("orders")
df_pay.createOrReplaceTempView("payments")
try:
   Totalrev = spark.sql("select sum(p.payment_value) as total_revenue_generated from orders o inner join payments p on p.order_id=o.order_id where o.order_delivered_customer_date between '2016-01-01' and '2020-01-01'")
   Totalrev.show()
   logger.info("*****Finding total revenue generated*****")
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding total revenue generated*****")
```



Saving the output into Datalake:

```
Totalrev.write.mode("overwrite").csv("/mnt/Files/validdata/output/total_revenue.csv")
```

3. Total orders by product category:

```
try:
   product_category = df_ordi.join(df_prod,['product_id'])
   product_category = product_category.join(df_tran,['product_category_name'])
   product_category = product_category.groupBy('product_category_name_english').count().withColumnRenamed('count','Total_item').withColumnRenamed('product_category_name_english','Product_Category')
   product_category.show(100)
   logger.info("*****Finding total orders by product category*****")
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding total orders by product category*****")
```

Total orders by product category

Cmd 45

```
try:
    product_category = df_ordi.join(df_prod,['product_id'])
    product_category = product_category.join(df_tran,['product_category_name'])
    product_category =
product_category.groupBy('product_category_name_english').count().withColumnRenamed('count','Total_item').withColumnRenamed('product_category_name_english','Product_Category'
)
    product_category.show(100)
    logger.info("*****Finding total orders by product category*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding total orders by product category*****")
```

(4) Spark Jobs

product_category: pyspark.sql.dataframe.DataFrame = [Product_Category: string, Total_item: long]

```
+--------------------+----------+
|    Product_Category|Total_item|
+--------------------+----------+
|                 art|       209|
|             flowers|        33|
|   home_construction|       604|
|fashion_male_clot...|       132|
|kitchen_dining_la...|       281|
|    small_appliances|       679|
|          la_cuisine|        14|
|     bed_bath_table|     11115|
```

Saving the output into Datalake:

```
product_category.write.mode("overwrite").csv("/mnt/Files/validdata/output/product_category.csv")
```

(4) Spark Jobs

4. Count total number of customers by regions-state:

try:
  Customer_by_region_state =
df_cus.groupBy('customer_state').count().withColumnRenamed('count','No_of_customers')
  Customer_by_region_state.show(100)
  logger.info("*****Finding total number of customers by region-state*****")
except Exception **as** e:
  print(e)
  logger.error("*****Error occured while finding total number of customers by region-state*****")

Count total number of customers by regions-state

Cmd 48

```
try:
    Customer_by_region_state = df_cus.groupBy('customer_state').count().withColumnRenamed('count','No_of_customers')
    Customer_by_region_state.show(100)
    logger.info("*****Finding total number of customers by region-state*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding total number of customers by region-state*****")
```

(3) Spark Jobs

Customer_by_region_state: pyspark.sql.dataframe.DataFrame = [customer_state: string, No_of_customers: long]

```
+--------------+---------------+
|customer_state|No_of_customers|
+--------------+---------------+
|            SC|           3637|
|            RO|            253|
|            PI|            495|
|            AM|            148|
|            RR|             46|
|            GO|           2020|
|            TO|            280|
|            MT|            907|
|            SP|          41746|
|            ES|           2033|
|            PB|            536|
|            RS|           5466|
|            MS|            715|
```

Saving the output into Datalake:

```
Customer_by_region_state.write.mode("overwrite").csv("/mnt/Files/validdata/output/cus_by_region_state.csv")
```

▸ (3) Spark Jobs

5. Count total number of customers by regions-city:

try:
Customer_by_region_city = df_cus.groupBy('customer_city').count().orderBy('count',ascending=**False**).withColumnRenamed('count','No_of_customers')
Customer_by_region_city.show(100)
logger.info("*****Finding total number of customers by region-city*****")
except Exception **as** e:
print(e)
logger.error("*****Error occured while finding total number of customers by region-city*****")

Count total number of customers by regions-city

Cmd 51

```
try:
    Customer_by_region_city = df_cus.groupBy('customer_city').count().orderBy('count',ascending=False).withColumnRenamed('count','No_of_customers')
    Customer_by_region_city.show(100)
    logger.info("*****Finding total number of customers by region-city*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding total number of customers by region-city*****")
```

▸ (3) Spark Jobs

▸ Customer_by_region_city: pyspark.sql.dataframe.DataFrame = [customer_city: string, No_of_customers: long]

```
+--------------------+---------------+
|       customer_city|No_of_customers|
+--------------------+---------------+
|           sao paulo|          15540|
|      rio de janeiro|           6882|
|      belo horizonte|           2773|
|            brasilia|           2131|
|            curitiba|           1521|
|            campinas|           1444|
|        porto alegre|           1379|
|            salvador|           1245|
|           guarulhos|           1189|
|sao bernardo do c...|            938|
|             niteroi|            849|
|         santo andre|            797|
```

Saving the output into Datalake:

```
Customer_by_region_city.write.mode("overwrite").csv("/mnt/Files/validdata/output/cus_by_region_city.csv")
```

▸ (5) Spark Jobs

6. Revenue generated annually:

try:
revenue_annual = spark.sql("select sum(p.payment_value) as revenue_generated,year(o.order_delivered_customer_date) as annual_year from orders o inner join payments p on o.order_id=p.order_id where o.order_status='delivered' group by year(o.order_delivered_customer_date)")
revenue_annual = revenue_annual.where(revenue_annual.annual_year.isNotNull())
revenue_annual.show()
logger.info("*****Finding revenue genrated annually*****")
except Exception **as** e:
print(e)
logger.error("*****Error occured while finding revenue genrated annually*****")

Revenue generated annually

Cmd 54

```
try:
    revenue_annual = spark.sql("select sum(p.payment_value) as revenue_generated,year(o.order_delivered_customer_date) as annual_year from orders o inner join payments p on o.order_id=p.order_id where o.order_status='delivered'  group by year(o.order_delivered_customer_date)")
    revenue_annual = revenue_annual.where(revenue_annual.annual_year.isNotNull())
    revenue_annual.show()
    logger.info("*****Finding revenue genrated annually*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding revenue genrated annually*****")
```

(4) Spark Jobs

revenue_annual: pyspark.sql.dataframe.DataFrame = [revenue_generated: double, annual_year: integer]

```
+------------------+-----------+
| revenue_generated|annual_year|
+------------------+-----------+
| 8863677.519999979|       2018|
|46586.329999999994|       2016|
| 6510819.000000014|       2017|
+------------------+-----------+

2022-04-19 13:58:08,797 - ecom_log - INFO - *****Finding revenue genrated annually*****
```

Command took 2.34 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:14 PM on mycap

Cmd 55

```
revenue_annual.write.mode("overwrite").csv("/mnt/Files/validdata/output/revenue_annual.csv")
```

7. Most valued customers and salesman:

```
try:
   orders_df_filter = df_ord.filter(df_ord.order_status=='delivered')
   Most_fav = orders_df_filter.join(df_ordi,['order_id']).drop(*('order_status','order_purchase_timestamp','order_approved_at','order_delivered_carrier_date','order_delivered_customer_date','order_estimated_delivery_date','shipping_limit_date','price','freight_value'))
   Most_fav_customer = Most_fav.groupBy('Customer_id').count().withColumnRenamed('count','total_product_buy')
   Most_fav_customer = Most_fav_customer.orderBy('total_product_buy',ascending=False)
   Most_fav_customer.show()
   logger.info("*****Finding most valued customer*****")
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding most valued customer*****")
```

Most valued customers and salesman

Cmd 58

```
try:
    orders_df_filter = df_ord.filter(df_ord.order_status=='delivered')
    Most_fav = orders_df_filter.join(df_ordi,['order_id']).drop(*('order_status','order_purchase_timestamp','order_approved_at','order_delivered_carrier_date','order_delivered_customer_date','order_estimated_delivery_date','shipping_limit_date','price','freight_value'))
    Most_fav_customer = Most_fav.groupBy('Customer_id').count().withColumnRenamed('count','total_product_buy')
    Most_fav_customer = Most_fav_customer.orderBy('total_product_buy',ascending=False)
    Most_fav_customer.show()
    logger.info("*****Finding most valued customer*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding most valued customer*****")
```

(4) Spark Jobs

orders_df_filter: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 8 more fields]

Most_fav: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 5 more fields]

Most_fav_customer: pyspark.sql.dataframe.DataFrame = [Customer_id: string, total_product_buy: long]

```
+--------------------+-----------------+
|         Customer_id|total_product_buy|
+--------------------+-----------------+
|fc3d1daec319d62d4...|               21|
|bd5d39761aa56689a...|               20|
|be1b70680b9f9694d...|               20|
|10de381f8a8d23fff...|               15|
|adb32467ecc74b535...|               15|
```

Saving the output into Datalake:

```
Most_fav_customer.write.mode("overwrite").csv("/mnt/Files/validdata/output/Most_fav_customer.csv")
```

```
try:
    Most_fav_seller = Most_fav.groupBy('seller_id').count().withColumnRenamed('count','total_product_sold')
    Most_fav_seller = Most_fav_seller.orderBy('total_product_sold',ascending=False)
    Most_fav_seller.show()
    logger.info("*****Finding most valued salesman*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding most valued salesman*****")
```

```
try:
    Most_fav_seller = Most_fav.groupBy('seller_id').count().withColumnRenamed('count','total_product_sold')
    Most_fav_seller = Most_fav_seller.orderBy('total_product_sold',ascending=False)
    Most_fav_seller.show()
    logger.info("*****Finding most valued salesman*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding most valued salesman*****")
```

▸ (4) Spark Jobs

▸ Most_fav_seller: pyspark.sql.dataframe.DataFrame = [seller_id: string, total_product_sold: long]

```
+--------------------+------------------+
|           seller_id|total_product_sold|
+--------------------+------------------+
|6560211a19b47992c...|              1996|
|4a3ca9315b744ce9f...|              1949|
|1f50f920176fa81da...|              1926|
|cc419e0650a3c5ba7...|              1719|
|da8622b14eb17ae28...|              1548|
|955fee9216a65b617...|              1472|
|1025f0e2d44d7041d...|              1420|
|7c67e1448b00f6e96...|              1355|
|ea8482cd71df3c196...|              1188|
|7a67c85e85bb2ce85...|              1155|
|4869f7a5dfa277a7d...|              1148|
|3d871de0142ce09b7...|              1131|
|8b321bb669392f516...|              1005|
|cca3071e3e9bb7d12...|               817|
|620c87c171fb2a6dd...|               778|
```

Saving the output into Datalake:

```
Most_fav_seller.write.mode("overwrite").csv("/mnt/Files/validdata/output/Most_fav_seller.csv")
```

8. Total orders by regions/city:

```
try:
    Order_by_region = df_cus.join(orders_df_filter,['customer_id'],'leftsemi')
    Customer_by_region_city = Order_by_region.groupBy('customer_city').count()

Customer_by_region_city.orderBy('count',ascending=False).withColumnRenamed('count','Total_orders').
show()
    Customer_by_region_city = Customer_by_region_city.sample(0.01)
    Customer_by_region_city.show()
    logger.info("*****Finding total orders by regions-city*****")
except Exception as e:
```

print(e)
logger.error("*****Error occured while finding total orders by regions-city*****")

Total orders by regions/city

Cmd 65

```
try:
    Order_by_region = df_cus.join(orders_df_filter,['customer_id'],'leftsemi')
    Customer_by_region_city = Order_by_region.groupBy('customer_city').count()
    Customer_by_region_city.orderBy('count',ascending=False).withColumnRenamed('count','Total_orders').show()
    Customer_by_region_city = Customer_by_region_city.sample(0.01)
    Customer_by_region_city.show()
    logger.info("*****Finding total orders by regions-city*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding total orders by regions-city*****")
```

▸ (8) Spark Jobs

▸ Order_by_region: pyspark.sql.dataframe.DataFrame = [customer_id: string, customer_unique_id: string ... 3 more fields]

▸ Customer_by_region_city: pyspark.sql.dataframe.DataFrame = [customer_city: string, count: long]

```
+--------------------+------------+
|       customer_city|Total_orders|
+--------------------+------------+
|           sao paulo|       15045|
|      rio de janeiro|        6601|
|      belo horizonte|        2697|
|            brasilia|        2071|
|            curitiba|        1489|
|            campinas|        1406|
|        porto alegre|        1342|
|            salvador|        1188|
```

Saving the output into Datalake:

```
Customer_by_region_city.write.mode("overwrite").csv("/mnt/Files/validdata/output/total_orders_by_city.csv")
```

9. Total reviews of products:

try:
Reviewed_product = df_ordi.join(orders_df_filter,['order_id'],'leftsemi').join(df_prod,['product_id']).join(df_rev,['order_id']).join(df_tran,['product_category_name']).select('review_id','order_id','product_id','product_category_name_english','review_comment_title','review_comment_message').dropna()
Reviewed_product = Reviewed_product.groupby('product_category_name_english','review_comment_title').count().groupBy('product_category_name_english').agg({'count':'sum'}).withColumnRenamed('sum(count)','Total_review')
Reviewed_product = Reviewed_product.orderBy('Total_review',ascending = **False**)
Reviewed_product.show()
logger.info("*****Finding total reviews of products*****")
except Exception **as** e:
print(e)
logger.error("*****Error occured while finding total reviews of products*****")

Total reviews of products

Cmd 68

```
try:
    Reviewed_product = df_ordi.join(orders_df_filter,['order_id'],'leftsemi').join(df_prod,['product_id']).join(df_rev,['order_id']).join(df_tran,
['product_category_name']).select('review_id','order_id','product_id','product_category_name_english','review_comment_title','review_comment_message').dropna()
    Reviewed_product =
Reviewed_product.groupby('product_category_name_english','review_comment_title').count().groupBy('product_category_name_english').agg({'count':'sum'}).withColumnRenamed('sum(
count)','Total_review')
    Reviewed_product = Reviewed_product.orderBy('Total_review',ascending = False)
    Reviewed_product.show()
    logger.info("*****Finding total reviews of products*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding total reviews of products*****")
```

(7) Spark Jobs

Reviewed_product: pyspark.sql.dataframe.DataFrame = [product_category_name_english: string, Total_review: long]

```
+---------------------------+------------+
|product_category_name_english|Total_review|
+---------------------------+------------+
|              bed_bath_table|        1027|
|               health_beauty|         983|
|                 housewares|         824|
|              watches_gifts|         808|
|             sports_leisure|         651|
|            furniture_decor|         626|
|        computers_accesso...|         610|
|                       auto|         492|
```

Saving the output into Datalake:

```
Reviewed_product.write.mode("overwrite").csv("/mnt/Files/validdata/output/Reviewed_product.csv")
```

10. Minimum and Maximum priced products:

```
try:
   minmax = df_ordi.groupBy('product_id').agg({'price':'min'})
   product_priced = minmax.join(df_prod,['product_id'])
   product_priced = product_priced.join(df_tran,['product_category_name'])
   product_priced =
product_priced.groupBy('product_category_name_english').agg({'min(price)':'min'}).withColumnRename
d('min(min(price))','Price').withColumnRenamed('product_category_name_english','Product_Name')
   product_priced.orderBy('Price',ascending=True).show()


   minmax = df_ordi.groupBy('product_id').agg({'price':'max'})
   product_priced = minmax.join(df_prod,['product_id'])
   product_priced = product_priced.join(df_tran,['product_category_name'])
   product_priced =
product_priced.groupBy('product_category_name_english').agg({'max(price)':'max'}).withColumnRenam
ed('max(max(price))','Price').withColumnRenamed('product_category_name_english','Product_Name')
   product_priced.orderBy('Price',ascending=False).show()
   logger.info("*****Finding minimum and maximum priced products*****")
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding minimum and maximum priced products*****")
```

Minimum and Maximum priced products

Cmd 71

```python
try:
    minmax = df_ordi.groupBy('product_id').agg({'price':'min'})
    product_priced = minmax.join(df_prod,['product_id'])
    product_priced = product_priced.join(df_tran,['product_category_name'])
    product_priced = product_priced.groupBy('product_category_name_english').agg({'min(price)':'min'}).withColumnRenamed('min(min(price))','Price').withColumnRenamed('product_category_name_english','Product_Name')
    product_priced.orderBy('Price',ascending=True).show()


    minmax = df_ordi.groupBy('product_id').agg({'price':'max'})
    product_priced = minmax.join(df_prod,['product_id'])
    product_priced = product_priced.join(df_tran,['product_category_name'])
    product_priced = product_priced.groupBy('product_category_name_english').agg({'max(price)':'max'}).withColumnRenamed('max(max(price))','Price').withColumnRenamed('product_category_name_english','Product_Name')
    product_priced.orderBy('Price',ascending=False).show()
    logger.info("*****Finding minimum and maximum priced products*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding minimum and maximum priced products*****")
```

(10) Spark Jobs

minmax: pyspark.sql.dataframe.DataFrame = [product_id: string, max(price): string]

product_priced: pyspark.sql.dataframe.DataFrame = [Product_Name: string, Price: string]

Finding both Minimum and Maximum priced products together:

```
df_ordi.createOrReplaceTempView("order_item")
df_prod.createOrReplaceTempView("product")
try:
   min_max_pro = spark.sql("select max(i.price) OVER () as max_price,min(i.price) OVER () as min_price,p.product_id from product p inner join order_item i on i.product_id=p.product_id ")
   min_max_pro.show()
   logger.info("*****Finding minimum and maximum priced product*****")
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding minimum and maximum priced product*****")
```

```python
df_ordi.createOrReplaceTempView("order_item")
df_prod.createOrReplaceTempView("product")
```

Command took 0.04 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:17 PM on mycap

Cmd 73

```python
try:
    min_max_pro = spark.sql("select max(i.price) OVER () as max_price,min(i.price) OVER () as min_price,p.product_id from product p inner join order_item i on i.product_id=p.product_id ")
    min_max_pro.show()
    logger.info("*****Finding minimum and maximum priced product*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding minimum and maximum priced product*****")
```

(4) Spark Jobs

min_max_pro: pyspark.sql.dataframe.DataFrame = [max_price: string, min_price: string ... 1 more field]

```
+---------+---------+--------------------+
|max_price|min_price|          product_id|
+---------+---------+--------------------+
|   999.99|     0.85|e95ee6822b66ac605...|
|   999.99|     0.85|96f608a5fe010cb34...|
|   999.99|     0.85|422879e10f4668299...|
|   999.99|     0.85|6ed0ed10d62b45f3d...|
|   999.99|     0.85|f9b0ee95ce8b1ef8b...|
|   999.99|     0.85|77f217636750a5d2e...|
|   999.99|     0.85|009af1277432f1a05...|
|   999.99|     0.85|22c5ddc22ea8cae84...|
|   999.99|     0.85|472e6464088101bf2...|
```

Saving the output into Datalake:

```python
min_max_pro.write.mode("overwrite").csv("/mnt/Files/validdata/output/minmax_product.csv")
```

11. Returning Customers to understand Customer loyalty:

```
try:
   Return = df_ord.filter(df_ord.order_status=='delivered')
   Loyalty =
Return.join(df_ordi,['order_id']).drop(*('order_status','order_purchase_timestamp','order_approved_at','o
rder_delivered_carrier_date','order_delivered_customer_date','order_estimated_delivery_date','shipping_li
mit_date','price','freight_value'))
   Loyalty = Loyalty.groupBy('Customer_id').count().withColumnRenamed('count','total_product_buy')
   print("The Most Valuable Customer is: ",Loyalty.orderBy('total_product_buy').tail(1))
   Loyalty = Loyalty.orderBy('total_product_buy',ascending=False)
   Loyalty.show()
   logger.info("*****Finding returning customers*****")
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding returning customers*****")
```

Returning Customers to understand Customer loyalty

Cmd 76

```
try:
    Return = df_ord.filter(df_ord.order_status=='delivered')
    Loyalty = Return.join(df_ordi,['order_id']).drop(*
('order_status','order_purchase_timestamp','order_approved_at','order_delivered_carrier_date','order_delivered_customer_date','order_estimated_delivery_date','shipping_limit_
date','price','freight_value'))
    Loyalty = Loyalty.groupBy('Customer_id').count().withColumnRenamed('count','total_product_buy')
    print("The Most Valuable Customer is: ",Loyalty.orderBy('total_product_buy').tail(1))
    Loyalty = Loyalty.orderBy('total_product_buy',ascending=False)
    Loyalty.show()
    logger.info("*****Finding returning customers*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding returning customers*****")
```

Python

(10) Spark Jobs

Return: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 8 more fields]

Loyalty: pyspark.sql.dataframe.DataFrame = [Customer_id: string, total_product_buy: long]

```
The Most Valuable Customer is:  [Row(Customer_id='fc3d1daec319d62d49bfb5e1f83123e9', total_product_buy=21)]
+--------------------+-----------------+
|         Customer_id|total_product_buy|
+--------------------+-----------------+
|fc3d1daec319d62d4...|               21|
|be1b70680b9f9694d...|               20|
|bd5d39761aa56689a...|               20|
|10de381f8a8d23fff...|               15|
|adb32467ecc74b535...|               15|
```

Saving the output into Datalake:

```
Loyalty.write.mode("overwrite").csv("/mnt/Files/validdata/output/loyalty.csv")
```

12. Total number of orders by total number of customers:

```
try:
   no_orders = df_ord.filter((df_ord.order_status !='canceled') & (df_ord.order_status !='unavailable'))
   no_orders =
no_orders.join(df_ordi,['order_id']).drop(*('order_status','order_purchase_timestamp','order_approved_at
','order_delivered_carrier_date','order_delivered_customer_date','order_estimated_delivery_date','shippin
g_limit_date','price','freight_value'))
   no_orders = no_orders.groupBy('customer_id').count()
   no_orders =
no_orders.groupby('customer_id').sum('count').orderBy('sum(count)',ascending=False).withColumnRen
amed('sum(count)',"Total_orders")
   no_orders_no_customers =
no_orders.agg({'customer_id':'count','Total_orders':'sum'}).withColumnRenamed('sum(Total_orders)','To
tal_orders').withColumnRenamed("count(customer_id)",'Total_customer')
   no_orders_no_customers.show()
   logger.info("*****Finding total number of orders by total number of customers*****")
```

```
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding total number of orders by total number of customers*****")
```

Total number of orders by total number of customers

Cmd 79

```
try:
    no_orders = df_ord.filter((df_ord.order_status !='canceled') & (df_ord.order_status !='unavailable'))
    no_orders = no_orders.join(df_ordi,['order_id']).drop(*('order_status','order_purchase_timestamp','order_approved_at','order_delivered_carrier_date','order_delivered_customer_date','order_estimated_delivery_date','shipping_limit_date','price','freight_value'))
    no_orders = no_orders.groupBy('customer_id').count()
    no_orders = no_orders.groupby('customer_id').sum('count').orderBy('sum(count)',ascending=False).withColumnRenamed('sum(count)',"Total_orders")
    no_orders_no_customers = no_orders.agg({'customer_id':'count','Total_orders':'sum'}).withColumnRenamed('sum(Total_orders)','Total_orders').withColumnRenamed("count(customer_id)",'Total_customer')
    no_orders_no_customers.show()
    logger.info("*****Finding total number of orders by total number of customers*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding total number of orders by total number of customers*****")
```

(5) Spark Jobs

no_orders: pyspark.sql.dataframe.DataFrame = [customer_id: string, Total_orders: long]

no_orders_no_customers: pyspark.sql.dataframe.DataFrame = [Total_orders: long, Total_customer: long]

```
+------------+--------------+
|Total_orders|Total_customer|
+------------+--------------+
|      112101|         98199|
+------------+--------------+
```

```
2022-04-19 13:58:54,214 - ecom_log - INFO - *****Finding total number of orders by total number of customers*****
```

Saving the output into Datalake:

```
no_orders_no_customers.write.mode("overwrite").csv("/mnt/Files/validdata/output/no_of_ord_cus.csv")
```

13. Show 80-20 Analysis of Products vs Sales/Profit:

Code for plotting 80-20 Analysis of Products vs Sales/Profit:

Show 80-20 Analysis of Products vs Sales/Profit

Cmd 82

```
month_yr = df_ord.withColumn('Month_year',date_format('order_purchase_timestamp','M')).withColumn('Year',year('order_purchase_timestamp'))
month_yr = month_yr.select('order_id','customer_id','Month_year','Year',concat(col('year'),lit('-'),trim(col('Month_year')))).alias('MY')).select('order_id','MY').withColumn('MY',to_date('MY'))
month_yr = month_yr.join(df_ordi,['order_id'])
```

month_yr: pyspark.sql.dataframe.DataFrame = [order_id: string, MY: date ... 6 more fields]

```
Command took 0.11 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:19 PM on mycap
```

Cmd 83

```
Analysis = month_yr.drop('order_item_id').withColumn('Total_Price',month_yr['price']+month_yr['freight_value'])
Analysis = Analysis.groupBy('MY').agg({'order_id':'count','price':'sum','freight_value':'sum','Total_Price':'sum'})
Analysis = Analysis.withColumn('price_per_product',Analysis['sum(Total_price)']/Analysis['count(order_id)']).withColumn('freight_per_product',Analysis['sum(freight_value)']/Analysis['count(order_id)']).withColumnRenamed('sum(Total_price)','Total_price').withColumnRenamed('count(order_id)','Order_id').withColumnRenamed('sum(freight_value)','freight_value').withColumnRenamed('sum(price)','price').orderBy('MY',ascending=True)
Analysis = Analysis.withColumn("Year",year('MY')).withColumn('price',Analysis.price.cast('int'))
```

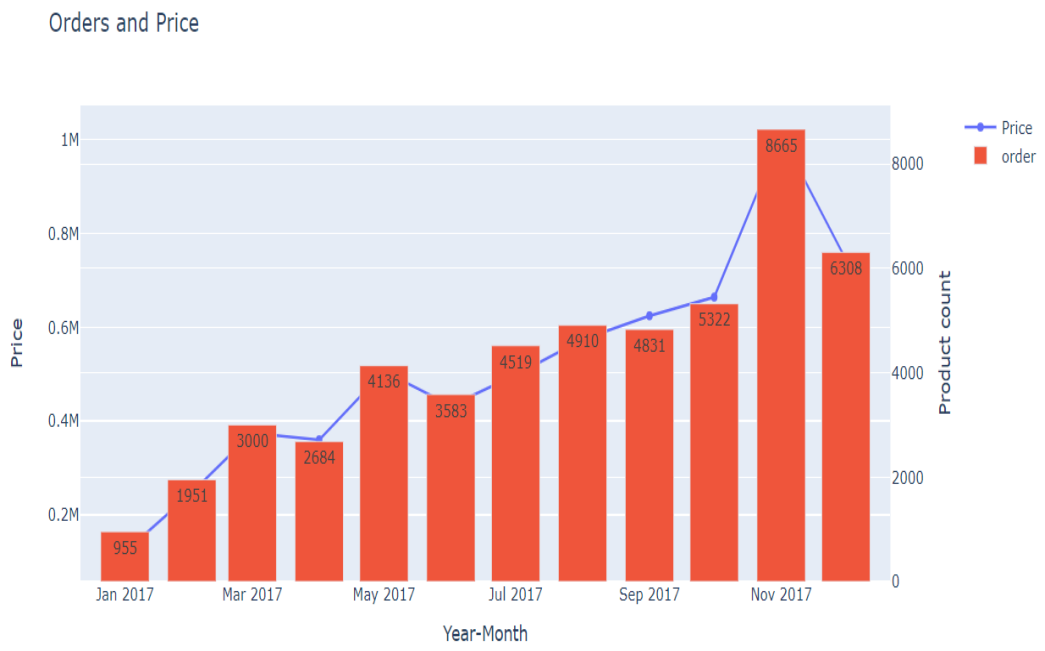
Analysis: pyspark.sql.dataframe.DataFrame = [MY: date, price: integer ... 6 more fields]

```
Command took 0.10 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:19 PM on mycap
```

Cmd 84

```
!pip install plotly
```

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

Analysis_80_20 = Analysis.filter(Analysis['Year']==2017)
list_date_a = Analysis_80_20.select('MY').rdd.flatMap(lambda x:x).collect()
list_date_count_a = Analysis_80_20.select('price').rdd.flatMap(lambda x:x).collect()
list_orders = Analysis_80_20.select('order_id').rdd.flatMap(lambda x:x).collect()


fig = go.Figure()
fig = make_subplots(specs=[[{'secondary_y':True}]])
fig.add_trace(
    go.Scatter(
        x = [str(x) for x in list_date_a],
        y = [x for x in list_date_count_a],
        name='Price',
        text = [x for x in list_date_count_a],

    ),secondary_y=False,)
fig.add_trace(
    go.Bar(
        x = [str(x) for x in list_date_a],
        y = [x for x in list_orders],
        name='order',
        text = [x for x in list_orders]

    ),
    secondary_y=True,
)
fig.update_layout(
    autosize=False,
    width = 1000,
    height = 500,
    title = 'Orders and Price',
    xaxis_title = ' Year-Month',
    yaxis_title = 'Price'

)
fig.update_yaxes(automargin=True,title_text='Product count',secondary_y=True)
fig.show()
```



Code to find 80-20 Analysis of Products vs Sales:

```
try:
   analysis80_20 = spark.sql("select p.product_category_name,sum(py.payment_value) as product_sales from product p inner join order_item i on i.product_id=p.product_id inner join orders o on o.order_id=i.order_id inner join payments py on py.order_id=o.order_id group by p.product_category_name order by product_sales desc")
   analysis80_20.show()
   logger.info("*****Finding 80-20 analysis of products v/s sales*****")
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding 80-20 analysis of products v/s sales*****")
```

```
try:
    analysis80_20 = spark.sql("select p.product_category_name,sum(py.payment_value) as product_sales from product p inner join order_item i on i.product_id=p.product_id inner join orders o on o.order_id=i.order_id inner join payments py on py.order_id=o.order_id group by p.product_category_name order by product_sales desc")
    analysis80_20.show()
    logger.info("*****Finding 80-20 analysis of products v/s sales*****")
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding 80-20 analysis of products v/s sales*****")
```

(6) Spark Jobs

analysis80_20: pyspark.sql.dataframe.DataFrame = [product_category_name: string, product_sales: double]

product_category_name	product_sales
cama_mesa_banho	1712553.6699999988
beleza_saude	1657373.1199999999
informatica_acess...	1585330.4499999993
moveis_decoracao	1430176.3899999992
relogios_presentes	1429216.6800000002
esporte_lazer	1392127.5599999998
utilidades_domest...	1094758.1300000004
automotivo	852294.3300000001
ferramentas_jardim	838280.7500000002
cool_stuff	779697.9999999999
moveis_escritorio	646826.4900000001
brinquedos	619037.6899999997
bebes	537884.6599999999
perfumaria	506738.6599999998

Saving the output into Datalake:

```
analysis80_20.write.mode("overwrite").csv("/mnt/Files/validdata/output/analysis80_20.csv")
```

14. Calculate Shipment Aging for each order:

```
try:
   Shipment_Aging = orders_df_filter.join(df_ordi,['order_id']).join(df_prod,['product_id']).join(df_tran,['product_category_name'])
   Shipment_Aging = Shipment_Aging.withColumn('Shipment_Aging_days',datediff('order_delivered_carrier_date','order_approved_at')).select('product_category_name_english','order_id','customer_id','seller_id','Shipment_Aging_days')
   Shipment_Aging = Shipment_Aging.sample(0.001)
   Shipment_Aging.show()
   logger.info("*****Finding shipment age for each order*****")
   logging.shutdown()
except Exception as e:
   print(e)
   logger.error("*****Error occured while finding shipment age*****")
   logging.shutdown()
```

Calculate Shipment Aging for each order

Cmd 89

```python
try:
    Shipment_Aging = orders_df_filter.join(df_ordi,['order_id']).join(df_prod,['product_id']).join(df_tran,['product_category_name'])
    Shipment_Aging =
Shipment_Aging.withColumn('Shipment_Aging_days',datediff('order_delivered_carrier_date','order_approved_at')).select('product_category_name_english','order_id','customer_id',
'seller_id','Shipment_Aging_days')
    Shipment_Aging = Shipment_Aging.sample(0.001)
    Shipment_Aging.show()
    logger.info("*****Finding shipment age for each order*****")
    logging.shutdown()
except Exception as e:
    print(e)
    logger.error("*****Error occured while finding shipment age*****")
    logging.shutdown()
```

(6) Spark Jobs

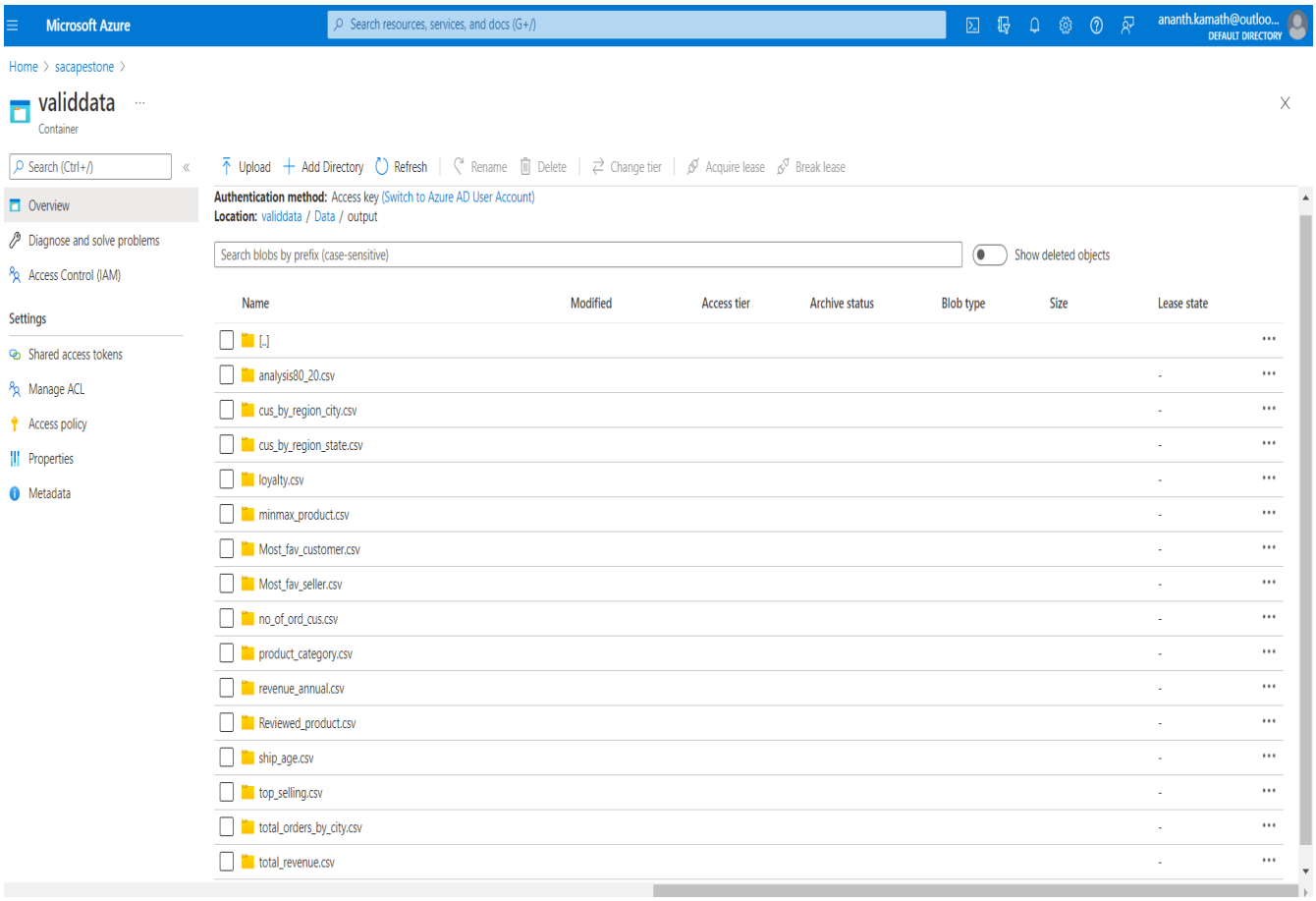
Shipment_Aging: pyspark.sql.dataframe.DataFrame = [product_category_name_english: string, order_id: string ... 3 more fields]

product_category_name_english	order_id	customer_id	seller_id	Shipment_Aging_days
sports_leisure	36aca7635fb851ae5...	a0c1cb74483d60b2e...	71039d19d4303bf90...	1
health_beauty	163679b19841bb6d7...	f06e0c84401adbe76...	01fdefa7697d26ad9...	-1
computers_accesso...	4969cddf1cd3a57ea...	a45263aab483c922e...	7ddcbb64b5bc1ef36...	2
sports_leisure	3b29577134414b4e9...	a92da5f6ab925f1b4...	218d46b86c1881d02...	4
perfumery	6ac15d09391be5079...	78500017ab0acb2ba...	fe2032dab1a61af87...	3
sports_leisure	88977dfcf102823ad...	211ac32c2e3b19201...	80e6699fe29150b37...	1
stationery	863a4a0cdd7648886...	e4f89f57953265051...	2c9e548be18521d1c...	1

Saving the output into Datalake:

```python
Shipment_Aging.write.mode("overwrite").csv("/mnt/Files/validdata/output/ship_age.csv")
```

Output files saved into Datalake:



Saving the logger file into Datalake:

Saving the logger file into the datalake

Cmd 92

```
dbutils.fs.mv("file:"+p_logfile,"dbfs:/mnt/Files/validdata/custom_log/"+p_filename)
```

```
Out[73]: True
```

```
Command took 0.12 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:21 PM on mycap
```

Cmd 93

```
dbutils.fs.ls('/mnt/Files/validdata/custom_log/')
```

```
Out[74]: [FileInfo(path='dbfs:/mnt/Files/validdata/custom_log/custom_log2022-04-19-05-27-05.log', name='custom_log2022-04-19-05-27-05.log', size=1877),
 FileInfo(path='dbfs:/mnt/Files/validdata/custom_log/custom_log2022-04-19-13-53-23.log', name='custom_log2022-04-19-13-53-23.log', size=1877)]
```

```
Command took 0.04 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:21 PM on mycap
```

Cmd 94

```
%sql
DROP TABLE IF EXISTS CUSTOM_LOGGING;
CREATE TABLE IF NOT EXISTS CUSTOM_LOGGING
USING TEXT OPTIONS(path'/mnt/Files/validdata/custom_log/*',header=true)
```

```
OK
```

```
Command took 1.67 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:21 PM on mycap
```

Cmd 95

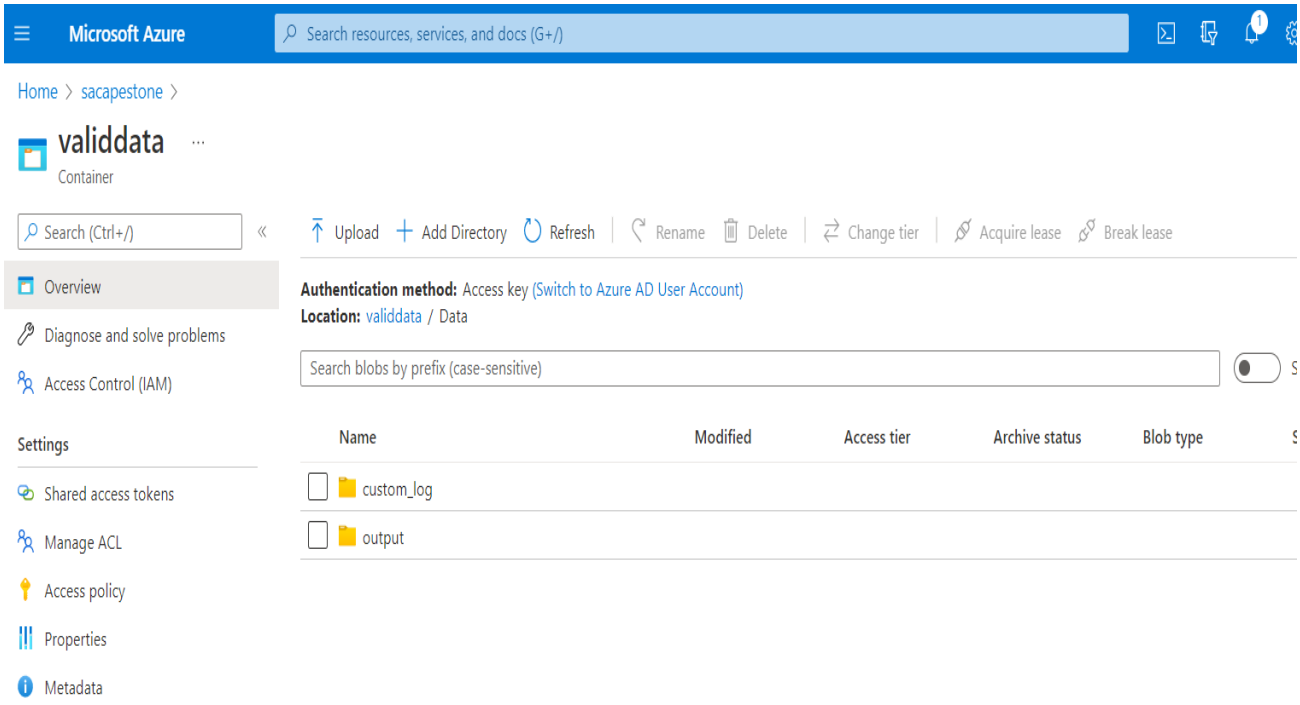
```
%sql
select * from custom_logging
```

▸ (2) Spark Jobs

Table | Data Profile

	value
1	2022-04-19 05:27:05,995 - ecom_log - INFO - *****START INFO LOGGER HERE*****
2	2022-04-19 05:27:05,996 - ecom_log - DEBUG - *****START DEBUG LOGGER HERE*****

Log file saved into Datalake:



Content of the log file:

	value
6	2022-04-19 05:31:08,099 - ecom_log - INFO - *****Finding top seller*****
7	2022-04-19 05:31:12,467 - ecom_log - INFO - *****Finding total revenue generated*****
8	2022-04-19 05:31:17,369 - ecom_log - INFO - *****Finding total orders by product category*****
9	2022-04-19 05:31:19,677 - ecom_log - INFO - *****Finding total number of customers by region-state*****
10	2022-04-19 05:31:21,418 - ecom_log - INFO - *****Finding total number of customers by region-city*****
11	2022-04-19 05:31:24,406 - ecom_log - INFO - *****Finding revenue genrated annually*****

Contents of the log file downloaded from datalake:

custom_log2022-04-18-07-33-30 - Notepad

File Edit Format View Help

```
2022-04-18 07:33:30,269 - ecom_log - INFO - *****START INFO LOGGER HERE*****
2022-04-18 07:33:30,269 - ecom_log - DEBUG - *****START DEBUG LOGGER HERE*****
2022-04-18 07:33:30,269 - ecom_log - ERROR - *****START ERROR LOGGER HERE*****
2022-04-18 07:33:30,269 - ecom_log - WARNING - *****START WARNING LOGGER HERE*****
2022-04-18 07:33:30,270 - ecom_log - CRITICAL - *****START CRITICAL LOGGER HERE*****
2022-04-18 07:37:43,053 - ecom_log - INFO - *****Finding top seller*****
2022-04-18 07:37:48,241 - ecom_log - INFO - *****Finding total revenue generated*****
2022-04-18 07:37:52,973 - ecom_log - INFO - *****Finding total orders by product category*****
2022-04-18 07:37:56,072 - ecom_log - INFO - *****Finding total number of customers by region-state*****
2022-04-18 07:37:58,278 - ecom_log - INFO - *****Finding total number of customers by region-city*****
2022-04-18 07:38:01,959 - ecom_log - INFO - *****Finding revenue genrated annually*****
2022-04-18 07:38:06,103 - ecom_log - INFO - *****Finding most valued customer*****
2022-04-18 07:38:10,733 - ecom_log - INFO - *****Finding most valued salesman*****
2022-04-18 07:38:18,263 - ecom_log - INFO - *****Finding total orders by regions-city*****
2022-04-18 07:38:24,347 - ecom_log - INFO - *****Finding total reviews of products*****
2022-04-18 07:38:31,418 - ecom_log - INFO - *****Finding minimum and maximum priced products*****
2022-04-18 07:38:33,205 - ecom_log - INFO - *****Finding minimum and maximum priced product*****
2022-04-18 07:38:38,723 - ecom_log - INFO - *****Finding returning customers*****
2022-04-18 07:38:43,493 - ecom_log - INFO - *****Finding total number of orders by total number of customers*****
2022-04-18 07:39:20,809 - ecom_log - INFO - *****Finding 80-20 analysis of products v/s sales*****
2022-04-18 07:40:28,776 - ecom_log - INFO - *****Finding shipment age for each order*****
```

Solution to KO's:

Performing SQL Joins, Aggregation and Window functions:

PERFORMING SQL JOINS,AGGREGATION AND WINDOW FUNCTIONS

Cmd 97

```
1 df_ordi.createOrReplaceTempView("order_item")
```

Command took 0.04 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:22 PM on mycap

Cmd 98

```
1 df_prod.createOrReplaceTempView("product")
```

Command took 0.05 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:22 PM on mycap

Cmd 99

```
1 from pyspark.sql import SQLContext
2 sc = SQLContext(sc)
```

Command took 0.04 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:22 PM on mycap

Cmd 100

```
1 df_prod_orderi = spark.sql("SELECT p.product_id as Product_ID,o.order_id as Order_ID,o.price,p.product_category_name as Category_Name FROM product p JOIN order_item o ON
  p.product_id==o.product_id")
2 df_prod_orderi.na.drop()
3 df_pro_ordi = df_prod_orderi.where(df_prod_orderi.Category_Name.isNotNull())
4 df_pro_ordi.show()
```

Here products and order item tables are considered to perform the Join Operation:

```
+--------------------+--------------------+------+--------------------+
|          Product_ID|            Order_ID| price|       Category_Name|
+--------------------+--------------------+------+--------------------+
|e95ee6822b66ac605...|00143d0f86d6fbd9f...| 21.33|       esporte_lazer|
|96f608a5fe010cb34...|010078e0913f48d01...| 49.89|     cama_mesa_banho|
|422879e10f4668299...|012b3f6ab7776a8ab...| 53.90|  ferramentas_jardim|
|6ed0ed10d62b45f3d...|015dd4d6681d1d15c...|109.90|               bebes|
|f9b0ee95ce8b1ef8b...|01b4dfef1c8848e8c...| 25.89|       esporte_lazer|
|77f217636750a5d2e...|020f04b3da1d597b9...|125.00|          brinquedos|
|009af1277432f1a05...|02e827a9f1e829b50...| 59.90|          brinquedos|
|22c5ddc22ea8cae84...|031a9aee4d307a1ae...| 56.99|          perfumaria|
|472e6464088101bf2...|0322e0a7d6d283584...| 69.90|informatica_acess...|
|ff92ca9bb0b3f4ec0...|03b70c61372aebf25...|858.90|portateis_casa_fo...|
|b791d5d05f0c974d1...|058b47cafcb628456...| 25.99|     cama_mesa_banho|
|8ebb1bc470f9bfd40...|05f5cfec67106e59d...| 49.90|     cama_mesa_banho|
|25e4d8fec8c2188a8...|0752235c99010e0a4...|229.90|          automotivo|
|d1c427060a0f73f6b...|084aeb6e79318fd38...|129.00|informatica_acess...|
|f8d3e4ddff67f24dc...|08f616b964002ad1b...| 39.00|utilidades_domest...|
|a096400f2b6fbe9c8...|097c6926697b318e2...| 43.20|       esporte_lazer|
|99f5b2240b82bf938...|09bfed54145e573d4...| 95.00|informatica_acess...|
|caf540627b7914ab7...|09c681e9069e20d7f...|250.00|    moveis_decoracao|
```

```
1 df_pro_ordi.describe().show()
```

▸ (4) Spark Jobs

```
+-------+--------------------+--------------------+------------------+--------------------+
|summary|          Product_ID|            Order_ID|             price|       Category_Name|
+-------+--------------------+--------------------+------------------+--------------------+
|  count|              111046|              111046|            111046|              111046|
|   mean|                null|                null|120.76233650918414|                null|
| stddev|                null|                null|183.33979015840185|                null|
|    min|00066f42aeeb9f300...|00010242fe8c5a6d1...|              0.85|agro_industria_e_...|
|    max|fffe9eeff12fcbd74...|fffe41c64501cc87c...|            999.99|utilidades_domest...|
+-------+--------------------+--------------------+------------------+--------------------+
```

Command took 1.92 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:23 PM on mycap

Cmd 102

```
1 #approx_count_distinct()
2 pro_count=df_pro_ordi.select(approx_count_distinct("price"))
3 pro_count.show()
```

▸ (4) Spark Jobs

▸ pro_count: pyspark.sql.dataframe.DataFrame = [approx_count_distinct(price): long]

```
+----------------------------+
|approx_count_distinct(price)|
+----------------------------+
|                        5912|
+----------------------------+
```

Aggregate functions like approximate count, average, count, maximum and minimum is performed:

Code:

```
#approx_count_distinct()
pro_count=df_pro_ordi.select(approx_count_distinct("price"))
pro_count.show()

#average
avg_pro = df_pro_ordi.select(avg("price"))
avg_pro.show()

#count
count_pro = df_pro_ordi.select(count("price"))
count_pro.show()

#maximum
max_pro = df_pro_ordi.select(max("price"))
max_pro.show()

#minimum
min_pro = df_pro_ordi.select(min("price"))
min_pro.show()

#Finding minimum,maximum,sum,mean,average
from pyspark.sql import functions
df_pro_ordi.groupBy('Category_Name').agg(functions.min('price'),
```

functions.max('price'),
functions.sum('price'),
functions.mean('price'),
functions.avg('price')).show()

```
1  #average
2  avg_pro = df_pro_ordi.select(avg("price"))
3  avg_pro.show()
```

▸ (4) Spark Jobs

▸ avg_pro: pyspark.sql.dataframe.DataFrame = [avg(price): double]

```
+------------------+
|        avg(price)|
+------------------+
|120.76233650918414|
+------------------+
```

Command took 1.33 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:23 PM on mycap

md 104

```
1  #count
2  count_pro = df_pro_ordi.select(count("price"))
3  count_pro.show()
```

▸ (4) Spark Jobs

▸ count_pro: pyspark.sql.dataframe.DataFrame = [count(price): long]

```
+-------------+
|count(price)|
+-------------+
|       111046|
+-------------+
```

```
1  #max
2  max_pro = df_pro_ordi.select(max("price"))
3  max_pro.show()
```

▸ (4) Spark Jobs

▸ max_pro: pyspark.sql.dataframe.DataFrame = [max(price): string]

```
+-----------+
|max(price)|
+-----------+
|     999.99|
+-----------+
```

Command took 0.95 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:24 PM on mycap

md 106

```
1  #min
2  min_pro = df_pro_ordi.select(min("price"))
3  min_pro.show()
```

▸ (4) Spark Jobs

▸ min_pro: pyspark.sql.dataframe.DataFrame = [min(price): string]

```
+-----------+
|min(price)|
+-----------+
|       0.85|
+-----------+
```

```
from pyspark.sql import functions
df_pro_ordi.groupBy('Category_Name').agg(functions.min('price'),
                                         functions.max('price'),
                                         functions.sum('price'),
                                         functions.mean('price'),
                                         functions.avg('price')).show()
```

▸ (4) Spark Jobs

```
+--------------------+----------+----------+------------------+------------------+------------------+
|       Category_Name|min(price)|max(price)|        sum(price)|        avg(price)|        avg(price)|
+--------------------+----------+----------+------------------+------------------+------------------+
|agro_industria_e_...|    109.90|     99.00|          72530.47|342.12485849056606|342.12485849056606|
|           alimentos|    100.00|     99.00|29393.409999999996| 57.63413725490195| 57.63413725490195|
|   alimentos_bebidas|     10.00|     99.99|15179.479999999998| 54.60244604316546| 54.60244604316546|
|              artes|    109.60|     99.99|24202.640000000003|115.80210526315791|115.80210526315791|
|  artes_e_artesanato|     11.90|      9.80|1814.0100000000004| 75.58375000000002| 75.58375000000002|
|   artigos_de_festas|    109.00|     90.00|           4485.18|104.30651162790699|104.30651162790699|
|    artigos_de_natal|     11.62|     95.00| 8800.820000000002| 57.52169934640524| 57.52169934640524|
|              audio|    108.90|     98.90| 50688.49999999999|139.25412087912085|139.25412087912085|
|          automotivo|     10.00|     99.99| 592720.1100000007|139.95752302243227|139.95752302243227|
|              bebes|     10.00|    999.00|409830.89000000106|133.75681788511784|133.75681788511784|
|             bebidas|     10.00|     94.06|22428.699999999993|59.178627968337715|59.178627968337715|
|        beleza_saude|      1.20|    990.00|1258681.3399999943|130.16353050672123|130.16353050672123|
|          brinquedos|     10.75|     99.99|483946.60000000097|117.54836045664342|117.54836045664342|
|     cama_mesa_banho|     10.50|    999.99|1036988.6799999914| 93.29632748537935| 93.29632748537935|
|       casa_conforto|    104.99|     99.90| 58572.03999999998|134.95861751152069|134.95861751152069|
|     casa_conforto_2|    109.99|     59.90|            760.27|25.342333333333332|25.342333333333332|
|     casa_construcao|     10.50|     98.80| 83088.11999999998|137.56311258278143|137.56311258278143|
|   cds_dvds_musicais|     45.00|     65.00|             730.0|52.142857142857146|52.142857142857146|
```

Windowing functions are performed on the resultant table:

Code:

```
from pyspark.sql.window import *
#row_number
windowSpec  = Window.partitionBy("Category_Name").orderBy("price")
row_num = df_pro_ordi.withColumn("row_number",row_number().over(windowSpec))
row_num.show()

#rank
rank1 = df_pro_ordi.withColumn("rank",rank().over(windowSpec))
rank1.show()

#dens_rank
dens_rank1 = df_pro_ordi.withColumn("dense_rank",dense_rank().over(windowSpec))
dens_rank1.show()

#percent_rank
percent_rank1 = df_pro_ordi.withColumn("percent_rank",percent_rank().over(windowSpec))
percent_rank1.show()

#ntile
ntitle1 = df_pro_ordi.withColumn("ntile",ntile(2).over(windowSpec))
ntitle1.show()
```

```
#cume_dist
cume_dist1 = df_pro_ordi.withColumn("cume_dist",cume_dist().over(windowSpec))
cume_dist1.show()
#lag
lag1 = df_pro_ordi.withColumn("lag",lag("price",2).over(windowSpec))
lag1.show()

#lead
lead1 = df_pro_ordi.withColumn("lead",lead("price",2).over(windowSpec))
lead1.show()

#Aggregate Functions
windowSpecAgg  = Window.partitionBy("Category_Name")
aggDF = df_pro_ordi.withColumn("row",row_number().over(windowSpec)).withColumn("avg",
avg(col("price")).over(windowSpecAgg)).withColumn("sum",
sum(col("price")).over(windowSpecAgg)).withColumn("min",
min(col("price")).over(windowSpecAgg)).withColumn("max",
max(col("price")).over(windowSpecAgg)).where(col("row")==1).select("Category_Name","avg","sum","m
in","max")
aggDF.show()
```

```
1  from pyspark.sql.window import *
```

Command took 0.02 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:25 PM on mycap

Cmd 109

```
1  #row_number
2  windowSpec  = Window.partitionBy("Category_Name").orderBy("price")
3  row_num = df_pro_ordi.withColumn("row_number",row_number().over(windowSpec))
4  row_num.show()
```

▸ (5) Spark Jobs

▸ row_num: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

Product_ID	Order_ID	price	Category_Name	row_number
49ab5384de586d3e4...	10ed5499d1623638e...	109.60	artes	1
986c47683f9e701e1...	816712dda4284eeda...	110.00	artes	2
986c47683f9e701e1...	dc37184f657b0a29c...	110.00	artes	3
4fe644d766c7566db...	1cde9001045df2307...	110.99	artes	4
4fe644d766c7566db...	a890088df39d4c062...	110.99	artes	5
4fe644d766c7566db...	8c9e6ed4aa9e011a3...	110.99	artes	6
4fe644d766c7566db...	b26c483e11283be69...	110.99	artes	7
4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	8
4fe644d766c7566db...	40a6a2ccfc8d35aaf...	110.99	artes	9
4fe644d766c7566db...	bb924cd9637a232ef...	110.99	artes	10
4fe644d766c7566db...	00dc6ad47477b3b62...	110.99	artes	11
4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	12

```
#rank
rank1 = df_pro_ordi.withColumn("rank",rank().over(windowSpec))
rank1.show()
```

▸ (5) Spark Jobs

▸ rank1: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

```
+--------------------+--------------------+------+-------------+----+
|          Product_ID|            Order_ID| price|Category_Name|rank|
+--------------------+--------------------+------+-------------+----+
|49ab5384de586d3e4...|10ed5499d1623638e...|109.60|        artes|   1|
|986c47683f9e701e1...|816712dda4284eeda...|110.00|        artes|   2|
|986c47683f9e701e1...|dc37184f657b0a29c...|110.00|        artes|   2|
|4fe644d766c7566db...|1cde9001045df2307...|110.99|        artes|   4|
|4fe644d766c7566db...|a890088df39d4c062...|110.99|        artes|   4|
|4fe644d766c7566db...|8c9e6ed4aa9e011a3...|110.99|        artes|   4|
|4fe644d766c7566db...|b26c483e11283be69...|110.99|        artes|   4|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|   4|
|4fe644d766c7566db...|40a6a2ccfc8d35aaf...|110.99|        artes|   4|
|4fe644d766c7566db...|bb924cd9637a232ef...|110.99|        artes|   4|
|4fe644d766c7566db...|00dc6ad47477b3b62...|110.99|        artes|   4|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|   4|
|c7ef7595086f66548...|e54709b9ea953fc92...|112.00|        artes|  13|
|4fe644d766c7566db...|75facf271f6fae1a5...|119.00|        artes|  14|
|4fe644d766c7566db...|c40f686ce1817e9bf...|119.00|        artes|  14|
|4fe644d766c7566db...|2eff76a325fc7035e...|119.00|        artes|  14|
|50a64d6b013980391...|dd621d7d942344da7...|119.90|        artes|  17|
```

```
#dens_rank
dens_rank1 = df_pro_ordi.withColumn("dense_rank",dense_rank().over(windowSpec))
dens_rank1.show()
```

▸ (5) Spark Jobs

▸ dens_rank1: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

```
+--------------------+--------------------+------+-------------+----------+
|          Product_ID|            Order_ID| price|Category_Name|dense_rank|
+--------------------+--------------------+------+-------------+----------+
|49ab5384de586d3e4...|10ed5499d1623638e...|109.60|        artes|         1|
|986c47683f9e701e1...|816712dda4284eeda...|110.00|        artes|         2|
|986c47683f9e701e1...|dc37184f657b0a29c...|110.00|        artes|         2|
|4fe644d766c7566db...|1cde9001045df2307...|110.99|        artes|         3|
|4fe644d766c7566db...|a890088df39d4c062...|110.99|        artes|         3|
|4fe644d766c7566db...|8c9e6ed4aa9e011a3...|110.99|        artes|         3|
|4fe644d766c7566db...|b26c483e11283be69...|110.99|        artes|         3|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|         3|
|4fe644d766c7566db...|40a6a2ccfc8d35aaf...|110.99|        artes|         3|
|4fe644d766c7566db...|bb924cd9637a232ef...|110.99|        artes|         3|
|4fe644d766c7566db...|00dc6ad47477b3b62...|110.99|        artes|         3|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|         3|
|c7ef7595086f66548...|e54709b9ea953fc92...|112.00|        artes|         4|
|4fe644d766c7566db...|75facf271f6fae1a5...|119.00|        artes|         5|
|4fe644d766c7566db...|c40f686ce1817e9bf...|119.00|        artes|         5|
|4fe644d766c7566db...|2eff76a325fc7035e...|119.00|        artes|         5|
|50a64d6b013980391...|dd621d7d942344da7...|119.90|        artes|         6|
|50a64d6b013980391...|22cbc77349f8317cd...|119.90|        artes|         6|
```

```
#percent_rank
percent_rank1 = df_pro_ordi.withColumn("percent_rank",percent_rank().over(windowSpec))
percent_rank1.show()
```

▸ (5) Spark Jobs

▸ percent_rank1: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

```
+--------------------+--------------------+------+-------------+--------------------+
|          Product_ID|            Order_ID| price|Category_Name|        percent_rank|
+--------------------+--------------------+------+-------------+--------------------+
|49ab5384de586d3e4...|10ed5499d1623638e...|109.60|        artes|                 0.0|
|986c47683f9e701e1...|816712dda4284eeda...|110.00|        artes|0.004807692307692308|
|986c47683f9e701e1...|dc37184f657b0a29c...|110.00|        artes|0.004807692307692308|
|4fe644d766c7566db...|1cde9001045df2307...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|a890088df39d4c062...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|8c9e6ed4aa9e011a3...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|b26c483e11283be69...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|40a6a2ccfc8d35aaf...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|bb924cd9637a232ef...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|00dc6ad47477b3b62...|110.99|        artes|0.014423076923076924|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|0.014423076923076924|
|c7ef7595086f66548...|e54709b9ea953fc92...|112.00|        artes|0.057692307692307696|
|4fe644d766c7566db...|75facf271f6fae1a5...|119.00|        artes|              0.0625|
|4fe644d766c7566db...|c40f686ce1817e9bf...|119.00|        artes|              0.0625|
|4fe644d766c7566db...|2eff76a325fc7035e...|119.00|        artes|              0.0625|
|50a64d6b013980391...|dd621d7d942344da7...|119.90|        artes| 0.07692307692307693|
|50a64d6b013980391...|22cbc77349f8317cd...|119.90|        artes| 0.07692307692307693|
```

```
#ntile
ntitle1 = df_pro_ordi.withColumn("ntile",ntile(2).over(windowSpec))
ntitle1.show()
```

▸ (5) Spark Jobs

▸ ntitle1: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

```
+--------------------+--------------------+------+-------------+-----+
|          Product_ID|            Order_ID| price|Category_Name|ntile|
+--------------------+--------------------+------+-------------+-----+
|49ab5384de586d3e4...|10ed5499d1623638e...|109.60|        artes|    1|
|986c47683f9e701e1...|816712dda4284eeda...|110.00|        artes|    1|
|986c47683f9e701e1...|dc37184f657b0a29c...|110.00|        artes|    1|
|4fe644d766c7566db...|1cde9001045df2307...|110.99|        artes|    1|
|4fe644d766c7566db...|a890088df39d4c062...|110.99|        artes|    1|
|4fe644d766c7566db...|8c9e6ed4aa9e011a3...|110.99|        artes|    1|
|4fe644d766c7566db...|b26c483e11283be69...|110.99|        artes|    1|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|    1|
|4fe644d766c7566db...|40a6a2ccfc8d35aaf...|110.99|        artes|    1|
|4fe644d766c7566db...|bb924cd9637a232ef...|110.99|        artes|    1|
|4fe644d766c7566db...|00dc6ad47477b3b62...|110.99|        artes|    1|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|    1|
|c7ef7595086f66548...|e54709b9ea953fc92...|112.00|        artes|    1|
|4fe644d766c7566db...|75facf271f6fae1a5...|119.00|        artes|    1|
|4fe644d766c7566db...|c40f686ce1817e9bf...|119.00|        artes|    1|
|4fe644d766c7566db...|2eff76a325fc7035e...|119.00|        artes|    1|
|50a64d6b013980391...|dd621d7d942344da7...|119.90|        artes|    1|
```

```
#cume_dist
cume_dist1 = df_pro_ordi.withColumn("cume_dist",cume_dist().over(windowSpec))
cume_dist1.show()
```

▸ (5) Spark Jobs

▸ cume_dist1: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

Product_ID	Order_ID	price	Category_Name	cume_dist
49ab5384de586d3e4...	10ed5499d1623638e...	109.60	artes	0.004784688995215311
986c47683f9e701e1...	816712dda4284eeda...	110.00	artes	0.014354066985645933
986c47683f9e701e1...	dc37184f657b0a29c...	110.00	artes	0.014354066985645933
4fe644d766c7566db...	1cde9001045df2307...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	a890088df39d4c062...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	8c9e6ed4aa9e011a3...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	b26c483e11283be69...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	40a6a2ccfc8d35aaf...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	bb924cd9637a232ef...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	00dc6ad47477b3b62...	110.99	artes	0.05741626794258373
4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	0.05741626794258373
c7ef7595086f66548...	e54709b9ea953fc92...	112.00	artes	0.06220095693779904
4fe644d766c7566db...	75facf271f6fae1a5...	119.00	artes	0.07655502392344497
4fe644d766c7566db...	c40f686ce1817e9bf...	119.00	artes	0.07655502392344497
4fe644d766c7566db...	2eff76a325fc7035e...	119.00	artes	0.07655502392344497
50a64d6b013980391...	dd621d7d942344da7...	119.90	artes	0.09090909090909091
50a64d6b013980391...	22cbc77349f8317cd...	119.90	artes	0.09090909090909091

```
#lag
lag1 = df_pro_ordi.withColumn("lag",lag("price",2).over(windowSpec))
lag1.show()
```

▸ (5) Spark Jobs

▸ lag1: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

Product_ID	Order_ID	price	Category_Name	lag
49ab5384de586d3e4...	10ed5499d1623638e...	109.60	artes	null
986c47683f9e701e1...	816712dda4284eeda...	110.00	artes	null
986c47683f9e701e1...	dc37184f657b0a29c...	110.00	artes	109.60
4fe644d766c7566db...	1cde9001045df2307...	110.99	artes	110.00
4fe644d766c7566db...	a890088df39d4c062...	110.99	artes	110.00
4fe644d766c7566db...	8c9e6ed4aa9e011a3...	110.99	artes	110.99
4fe644d766c7566db...	b26c483e11283be69...	110.99	artes	110.99
4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	110.99
4fe644d766c7566db...	40a6a2ccfc8d35aaf...	110.99	artes	110.99
4fe644d766c7566db...	bb924cd9637a232ef...	110.99	artes	110.99
4fe644d766c7566db...	00dc6ad47477b3b62...	110.99	artes	110.99
4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	110.99
c7ef7595086f66548...	e54709b9ea953fc92...	112.00	artes	110.99
4fe644d766c7566db...	75facf271f6fae1a5...	119.00	artes	110.99
4fe644d766c7566db...	c40f686ce1817e9bf...	119.00	artes	112.00
4fe644d766c7566db...	2eff76a325fc7035e...	119.00	artes	119.00
50a64d6b013980391...	dd621d7d942344da7...	119.90	artes	119.00
50a64d6b013980391...	22cbc77349f8317cd...	119.90	artes	119.00

```
#lead
lead1 = df_pro_ordi.withColumn("lead",lead("price",2).over(windowSpec))
lead1.show()
```

▸ (5) Spark Jobs

▸ lead1: pyspark.sql.dataframe.DataFrame = [Product_ID: string, Order_ID: string ... 3 more fields]

```
+--------------------+--------------------+------+-------------+------+
|          Product_ID|            Order_ID| price|Category_Name|  lead|
+--------------------+--------------------+------+-------------+------+
|49ab5384de586d3e4...|10ed5499d1623638e...|109.60|        artes|110.00|
|986c47683f9e701e1...|816712dda4284eeda...|110.00|        artes|110.99|
|986c47683f9e701e1...|dc37184f657b0a29c...|110.00|        artes|110.99|
|4fe644d766c7566db...|1cde9001045df2307...|110.99|        artes|110.99|
|4fe644d766c7566db...|a890088df39d4c062...|110.99|        artes|110.99|
|4fe644d766c7566db...|8c9e6ed4aa9e011a3...|110.99|        artes|110.99|
|4fe644d766c7566db...|b26c483e11283be69...|110.99|        artes|110.99|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|110.99|
|4fe644d766c7566db...|40a6a2ccfc8d35aaf...|110.99|        artes|110.99|
|4fe644d766c7566db...|bb924cd9637a232ef...|110.99|        artes|110.99|
|4fe644d766c7566db...|00dc6ad47477b3b62...|110.99|        artes|112.00|
|4fe644d766c7566db...|99687fd750781ecf4...|110.99|        artes|119.00|
|c7ef7595086f66548...|e54709b9ea953fc92...|112.00|        artes|119.00|
|4fe644d766c7566db...|75facf271f6fae1a5...|119.00|        artes|119.00|
|4fe644d766c7566db...|c40f686ce1817e9bf...|119.00|        artes|119.90|
|4fe644d766c7566db...|2eff76a325fc7035e...|119.00|        artes|119.90|
|50a64d6b013980391...|dd621d7d942344da7...|119.90|        artes|119.90|
|50a64d6b013980391...|22cbc77349f8317cd...|119.90|        artes| 12.00|
```

```
#Aggregate Functions
windowSpecAgg  = Window.partitionBy("Category_Name")
aggDF = df_pro_ordi.withColumn("row",row_number().over(windowSpec)).withColumn("avg", avg(col("price")).over(windowSpecAgg)).withColumn("sum",
sum(col("price")).over(windowSpecAgg)).withColumn("min", min(col("price")).over(windowSpecAgg)).withColumn("max",
max(col("price")).over(windowSpecAgg)).where(col("row")==1).select("Category_Name","avg","sum","min","max")
aggDF.show()
```

▸ (4) Spark Jobs

▸ aggDF: pyspark.sql.dataframe.DataFrame = [Category_Name: string, avg: double ... 3 more fields]

```
+--------------------+------------------+------------------+------+------+
|       Category_Name|               avg|               sum|   min|   max|
+--------------------+------------------+------------------+------+------+
|agro_industria_e_...|342.12485849056617| 72530.47000000003|109.90| 99.00|
|           alimentos| 57.63413725490217| 29393.41000000011|100.00| 99.00|
|    alimentos_bebidas| 54.60244604316538|15179.479999999976| 10.00| 99.99|
|              artes|115.80210526315834|24202.640000000094|109.60| 99.99|
|  artes_e_artesanato| 75.58375000000002|1814.0100000000004| 11.90|  9.80|
|    artigos_de_festas|104.30651162790699|           4485.18|109.00| 90.00|
|     artigos_de_natal| 57.52169934640513| 8800.819999999985| 11.62| 95.00|
|              audio|139.25412087912093| 50688.50000000002|108.90| 98.90|
|          automotivo|139.95752302243486| 592720.1100000116| 10.00| 99.99|
|              bebes|133.75681788511963| 409830.8900000066| 10.00|999.00|
|             bebidas|59.178627968337764| 22428.70000000001| 10.00| 94.06|
|        beleza_saude|130.16353050672035|1258681.3399999856|  1.20|990.00|
|          brinquedos|117.54836045664574| 483946.6000000105| 10.75| 99.99|
|     cama_mesa_banho| 93.29632748538664|1036988.6800000725| 10.50|999.99|
|       casa_conforto| 134.9586175115215|58572.040000000336|104.99| 99.90|
|     casa_conforto_2|25.342333333333322| 760.2699999999996|109.99| 59.90|
|     casa_construcao|137.56311258278075| 83088.11999999957| 10.50| 98.80|
|   cds_dvds_musicais|52.142857142857146|             730.0| 45.00| 65.00|
```

Performing Normal Join and Broadcast Join:

Performing Normal Join. Products and Product category name translation tables are considered:

JOINS AND BROADCAST JOINS

d 119

Python

```
#Joining using normal join
norm_join = df_tran.join(df_prod,['product_category_name'])
norm_join.show()
```

(2) Spark Jobs

norm_join: pyspark.sql.dataframe.DataFrame

```
product_category_name: string
product_category_name_english: string
product_id: string
product_name_lenght: string
product_description_lenght: string
product_photos_qty: string
product_weight_g: string
product_length_cm: string
product_height_cm: string
product_width_cm: string
```

```
+--------------------+----------------------------+--------------------+-----------------+----------------------+----------------+---------------+----------------+---------------+
|product_category_name|product_category_name_english|          product_id|product_name_lenght|product_description_lenght|product_photos_qty|product_weight_g|product_length_cm|product_height_cm|product_width_cm|
+--------------------+----------------------------+--------------------+-----------------+----------------------+----------------+---------------+----------------+---------------+----------------+
|      cama_mesa_banho|                bed_bath_table|a5bc1334f1762ce0a...|               48|                   645|               1|           1200|              16|             10|              16|
| utilidades_domest...|                  housewares|53f92b0474f91fcb5...|               54|                  2952|               3|          30000|              76|             51|              51|
```

Spark SQL Joins comes with more optimization by default. Here by default we are using inner join.

Spark SQL Joins comes with more optimization by default. Here by default we are using inner join.

Cmd 121

Python

```
norm_join.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Current Plan ==
   Project [product_category_name#2609, product_category_name_english#2610, product_id#1487, product_name_lenght#1489, product_description_lenght#1490, product_photos_qty#1491, product_weight_g#1492, product_length_cm#1493, product_height_cm#1494, product_width_cm#1495]
   +- BroadcastHashJoin [product_category_name#2609], [product_category_name#1488], Inner, BuildLeft
      :- BroadcastExchange HashedRelationBroadcastMode(ArrayBuffer(input[0, string, true])), [id=#63959]
      :  +- Project [product_category_name#2609, product_category_name_english#2610]
      :     +- Filter isnotnull(product_category_name#2609)
      :        +- FileScan csv [product_category_name#2609,product_category_name_english#2610] Batched: false, DataFilters: [isnotnull(product_category_name#2609)], Format: CSV, Location: InMemoryFileIndex[dbfs:/mnt/Files/dataset/product_category_name_translation.csv], PartitionFilters: [], PushedFilters: [IsNotNull(product_category_name)], ReadSchema: struct<product_category_name:string,product_category_name_english:string>
      +- HashAggregate(keys=[product_width_cm#1495, product_name_lenght#1489, product_height_cm#1494, product_description_lenght#1490, product_length_cm#1493, product_photos_qty#1491, product_id#1487, product_category_name#1488, product_weight_g#1492], functions=[])
         +- Exchange hashpartitioning(product_width_cm#1495, product_name_lenght#1489, product_height_cm#1494, product_description_lenght#1490, product_length_cm#1493, product_photos_qty#1491, product_id#1487, product_category_name#1488, product_weight_g#1492, 200), true, [id=#63956]
            +- HashAggregate(keys=[product_width_cm#1495, product_name_lenght#1489, product_height_cm#1494, product_description_lenght#1490, product_length_cm#1493, product_photos_qty#1491, product_id#1487, product_category_name#1488, product_weight_g#1492], functions=[])
               +- Project [product_id#1487, product_category_name#1488, product_name_lenght#1489, product_description_lenght#1490, product_photos_qty#1491, product_weight_g#1492, product_length_cm#1493, product_height_cm#1494, product_width_cm#1495]
                  +- Filter (AtLeastNNulls(n, product_id#1487,product_category_name#1488,product_name_lenght#1489,product_description_lenght#1490,product_photos_qty#1491,product
```

Performing Broadcast Join:

```
#Now lets join using broadcast join
broad1 = broadcast(df_tran)
broad2 = df_prod.join(broadcast(broad1),df_prod.product_category_name==broad1.product_category_name)
broad2.show()
```

(2) Spark Jobs

broad1: pyspark.sql.dataframe.DataFrame = [product_category_name: string, product_category_name_english: string]

broad2: pyspark.sql.dataframe.DataFrame = [product_id: string, product_category_name: string ... 9 more fields]

```
+--------------------+--------------------+------------------+----------------------+-----------------+---------------+---------------+----------------+-------------------+-----------------------+
|          product_id|product_category_name|product_name_lenght|product_description_lenght|product_photos_qty|product_weight_g|product_length_cm|product_height_cm|product_width_cm|product_category_name|product_category_name_english|
+--------------------+--------------------+------------------+----------------------+-----------------+---------------+---------------+----------------+-------------------+-----------------------+
|a5bc1334f1762ce0a...|      cama_mesa_banho|                48|                   645|                1|           1200|             16|              10|                16|      cama_mesa_banho|              bed_bath_table|
|53f92b0474f91fcb5...| utilidades_domest...|                54|                  2952|                3|          30000|             76|              51|                51| utilidades_domest...|                  housewares|
|45852eb1811c96c22...|      cama_mesa_banho|                56|                   393|                2|           1100|             35|               9|                35|      cama_mesa_banho|              bed_bath_table|
|bc7496ccc90a64c02...| utilidades_domest...|                28|                   238|                1|            150|             26|              16|                16| utilidades_domest...|                  housewares|
|feddfbd2585f9007f...|        beleza_saude|                54|                  2627|                1|            700|             16|              16|                15|        beleza_saude|               health_beauty|
|aaea62612b84679dd...| utilidades_domest...|                36|                  2600|                1|            850|             43|               5|                24| utilidades_domest...|                  housewares|
|485c67ced40028609...| utilidades_domest...|                47|                   469|                1|           9650|             22|              52|                42| utilidades_domest...|                  housewares|
|02fcaaaba3a4b4882...|      cama_mesa_banho|                55|                   139|                3|            950|             36|               6|
```

It is a join operation of a large dataframe with a smaller dataframe in PySpark Join model. It reduces data shuffllling by broadcasting the smaller dataframe in the nodes of PySpark cluster. The data is sent and broadcasted to all nodes in the cluster. This is an optimal and cost-efficient join model that can be used in thePySpark application.

It is a join operation of a large dataframe with a smaller dataframe in PySpark Join model. It reduces data shuffllling by broadcasting the smaller dataframe in the nodes of PySpark cluster. The data is sent and broadcasted to all nodes in the cluster. This is an optimal and cost-efficient join model that can be used in thePySpark application.

Cmd 124

```
broad2.explain()
```

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Current Plan ==
   BroadcastHashJoin [product_category_name#1488], [product_category_name#2609], Inner, BuildRight
   :- HashAggregate(keys=[product_width_cm#1495, product_name_lenght#1489, product_height_cm#1494, product_description_lenght#1490, product_length_cm#1493, product_photos_qty#1491, product_id#1487, product_category_name#1488, product_weight_g#1492], functions=[])
   :  +- Exchange hashpartitioning(product_width_cm#1495, product_name_lenght#1489, product_height_cm#1494, product_description_lenght#1490, product_length_cm#1493, product_photos_qty#1491, product_id#1487, product_category_name#1488, product_weight_g#1492, 200), true, [id=#64600]
   :     +- HashAggregate(keys=[product_width_cm#1495, product_name_lenght#1489, product_height_cm#1494, product_description_lenght#1490, product_length_cm#1493, product_photos_qty#1491, product_id#1487, product_category_name#1488, product_weight_g#1492], functions=[])
   :        +- Project [product_id#1487, product_category_name#1488, product_name_lenght#1489, product_description_lenght#1490, product_photos_qty#1491, product_weight_g#1492, product_length_cm#1493, product_height_cm#1494, product_width_cm#1495]
   :           +- Filter (AtLeastNNulls(n, product_id#1487,product_category_name#1488,product_name_lenght#1489,product_description_lenght#1490,product_photos_qty#1491,product_weight_g#1492,product_length_cm#1493,product_height_cm#1494,product_width_cm#1495) AND isnotnull(product_category_name#1488))
   :              +- FileScan csv [product_id#1487,product_category_name#1488,product_name_lenght#1489,product_description_lenght#1490,product_photos_qty#1491,product_weight_g#1492,product_length_cm#1493,product_height_cm#1494,product_width_cm#1495] Batched: false, DataFilters: [AtLeastNNulls(n, product_id#1487,product_category_name#1488,product_name_lenght#1489,product_des..., Format: CSV, Location: InMemoryFileIndex[dbfs:/mnt/Files/dataset/olist_products_dataset.csv], PartitionFilters: [], PushedFilters: [IsNotNull(product_category_name)], ReadSchema: struct<product_id:string,product_category_name:string,product_name_lenght:string,product_descript...
   +- BroadcastExchange HashedRelationBroadcastMode(ArrayBuffer(input[0, string, true])), [id=#64603]
      +- Project [product_category_name#2609, product_category_name_english#2610]
         +- Filter isnotnull(product_category_name#2609)
```

Demonstrating OOPs concept and parameterization on one of the use cases:

Code:

```
class ecomAnalysis:
  def __init__(self, storage_account_name, storage_account_key, container):
    self.storage_container_name = storage_account_name
    self.storage_account_key = storage_account_key
    self.container = container

  def total_revenue_bet_years(self, star_year="", end_year="", query="select sum(p.payment_value) as total_revenue_generated from orders o inner join payments p on p.order_id=o.order_id where o.order_delivered_customer_date between '2016-01-01' and '2020-01-01'"):
    try:
      return spark.sql(query)
    except:
      print("error in query")


if __name__ == '__main__':
  storage_account_name = "sacapestone"
  storage_account_key = "q6WpJ3arLAz1nmSMBlu4AMHAu6tIc366kSIp9x1qLcr43/UfwT4Zyi/J2mAcdHYqucnrIVhe4KXIArOTZcQK2g=="
  container = "dataset"
  df_ord.createOrReplaceTempView("orders")
  df_pay.createOrReplaceTempView("payments")
  b = ecomAnalysis(storage_account_name, storage_account_key, container)
  print("Revenue between particular year 2016-01-01 to 2020-01-01")
```

```
total_rev_between = b.total_revenue_bet_years('2016-01-01', '2020-01-01')
total_rev_between.show()
```

```
Revenue between particular year 2016-01-01 to 2020-01-01
+-----------------------+
|total_revenue_generated|
+-----------------------+
|   1.5421831429999886E7|
+-----------------------+
```

Performing Unit Testing using unittest:

Code:

```
import unittest
testcase_date = ['2016-01-01','2018-01-01','2021-08-29','2017-01-31','2022-01-18','2015-01-01','2019-12-07']
years = ["2016","2017","2018","2019","2020"]
print("Testing years")
y1= testcase_date[0][:4]
y2= testcase_date[1][:4]
y3= testcase_date[2][:4]
y4= testcase_date[3][:4]
y5= testcase_date[4][:4]
y6= testcase_date[5][:4]
y7= testcase_date[6][:4]
class addTest(unittest.TestCase):
  def test_fun1(self):
    if y1 in years:
      self.assertTrue("Test Successfull!!",True)
    else:
      self.assertFalse("Test Failed!!",True)
  def test_fun2(self):
    if y2 in years:
      self.assertTrue("Test Successfull!!",True)
    else:
      self.assertFalse("Test Failed!!",True)
  def test_fun3(self):
    if y3 in years:
      self.assertTrue("Test Successfull!!",True)
    else:
      self.assertFalse("Test Failed!!",True)
  def test_fun4(self):
    if y4 in years:
      self.assertTrue("Test Successfull!!",True)
    else:
      self.assertFalse("Test Failed!!",True)
  def test_fun5(self):
    if y5 in years:
      self.assertTrue("Test Successfull!!",True)
    else:
      self.assertFalse("Test Failed!!",True)
  def test_fun6(self):
    if y6 in years:
      self.assertTrue("Test Successfull!!",True)
    else:
```

```
        self.assertFalse("Test Failed!!",True)
    def test_fun7(self):
        if y7 in years:
            self.assertTrue("Test Successfull!!",True)
        else:
            self.assertFalse("Test Failed!!",True)

suite=unittest.TestLoader().loadTestsFromTestCase(addTest)
unittest.TextTestRunner(verbosity=2).run(suite)
```

Unit testing using unittest

Cmd 129

```
1 import unittest
```

Command took 0.05 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:33:29 PM on mycap

Cmd 130

```
1 testcase_date = ['2016-01-01','2018-01-01','2021-08-29','2017-01-31','2022-01-18','2015-01-01','2019-12-07']
2 years = ["2016","2017","2018","2019","2020"]
```

```
1  print("Testing years")
2  y1= testcase_date[0][:4]
3  y2= testcase_date[1][:4]
4  y3= testcase_date[2][:4]
5  y4= testcase_date[3][:4]
6  y5= testcase_date[4][:4]
7  y6= testcase_date[5][:4]
8  y7= testcase_date[6][:4]
9  class addTest(unittest.TestCase):
10     def test_fun1(self):
11         if y1 in years:
12             self.assertTrue("Test Successfull!!",True)
13         else:
14             self.assertFalse("Test Failed!!",True)
15     def test_fun2(self):
16         if y2 in years:
17             self.assertTrue("Test Successfull!!",True)
18         else:
19             self.assertFalse("Test Failed!!",True)
20     def test_fun3(self):
21         if y3 in years:
22             self.assertTrue("Test Successfull!!",True)
23         else:
24             self.assertFalse("Test Failed!!",True)
25     def test_fun4(self):
26         if y4 in years:
27             self.assertTrue("Test Successfull!!",True)
28         else:
29             self.assertFalse("Test Failed!!",True)
30     def test_fun5(self):
31         if y5 in years:
32             self.assertTrue("Test Successfull!!",True)
33         else:
34             self.assertFalse("Test Failed!!",True)
35     def test_fun6(self):
36         if y6 in years:
37             self.assertTrue("Test Successfull!!",True)
38         else:
39             self.assertFalse("Test Failed!!",True)
40     def test_fun7(self):
41         if y7 in years:
42             self.assertTrue("Test Successfull!!",True)
43         else:
44             self.assertFalse("Test Failed!!",True)
45 suite=unittest.TestLoader().loadTestsFromTestCase(addTest)
46 unittest.TextTestRunner(verbosity=2).run(suite)
```

Output of unittesting. Out off the 7 testcases written 4 have passed and 3 have failed:

```
Testing years
test_fun1 (__main__.addTest) ... ok
test_fun2 (__main__.addTest) ... ok
test_fun3 (__main__.addTest) ... FAIL
test_fun4 (__main__.addTest) ... ok
test_fun5 (__main__.addTest) ... FAIL
test_fun6 (__main__.addTest) ... FAIL
test_fun7 (__main__.addTest) ... ok

======================================================================
FAIL: test_fun3 (__main__.addTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<command-1277975875779314>", line 24, in test_fun3
    self.assertFalse("Test Failed!!",True)
AssertionError: 'Test Failed!!' is not false : True

======================================================================
FAIL: test_fun5 (__main__.addTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<command-1277975875779314>", line 34, in test_fun5
    self.assertFalse("Test Failed!!",True)
AssertionError: 'Test Failed!!' is not false : True

======================================================================
FAIL: test_fun6 (__main__.addTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<command-1277975875779314>", line 39, in test_fun6
    self.assertFalse("Test Failed!!",True)
AssertionError: 'Test Failed!!' is not false : True

----------------------------------------------------------------------
Ran 7 tests in 0.001s

FAILED (failures=3)
Out[110]: <unittest.runner.TextTestResult run=7 errors=0 failures=3>
```

Databricks file as HTML File:

ecom_analysis.html