

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-e-commerce/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 resource group as 'rg_capestone' in location South India:

The screenshot shows the 'Create a resource group' page in the Azure portal. At the top, there's a breadcrumb navigation: Home > Resource groups > Create a resource group. Below the title, there are three tabs: Basics (which is selected), Tags, and Review + create. The Basics tab contains a descriptive text about what a resource group is, followed by 'Project details' and 'Resource details' sections. In the 'Project details' section, 'Subscription' is set to 'Azure subscription 1' and 'Resource group' is set to 'rg_capestone'. In the 'Resource details' section, 'Region' is set to '(Asia Pacific) South India'. At the bottom of the page are three buttons: 'Review + create', '< Previous', and 'Next : Tags >'.

Home >

Resource groups

Default Directory

+ Create Manage view Refresh Export to CSV Open query Assign tags

Filter for any field... Subscription == all Location == all Add filter

Unsecure resources 0

Name ↑	Subscription ↑	Location ↑
<input type="checkbox"/> rg_capestone	Azure subscription 1	South India

No grouping List view

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

Home > Storage accounts >

Create a storage account

Basics Advanced Networking Data protection Encryption Tags Review + create

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription * Azure subscription 1

Resource group * rg_capestone Create new

Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name * sacapestone

Region * (Asia Pacific) South India

Performance * Standard: Recommended for most scenarios (general-purpose v2 account)
Premium: Recommended for scenarios that require low latency.

Redundancy * Locally-redundant storage (LRS)

Review + create < Previous Next : Advanced >

Home > Storage accounts >

Create a storage account

Basics Advanced Networking Data protection Encryption Tags Review + create

Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace

Blob storage

Enable SFTP (preview)
Enable network file system v3
Allow cross-tenant replication
 ⓘ Cross-tenant replication and hierarchical namespace cannot be enabled simultaneously.

Access tier * Hot: Frequently accessed data and day-to-day usage scenarios
Cool: Infrequently accessed data and backup scenarios

Azure Files

Enable large file shares

Review + create < Previous Next : Networking >

Home > sacapestone_1650524435072 >

sacapestone ...
Storage account

Search (Ctrl+)

Upload Open in Explorer Delete Move Refresh Mobile Feedback

Overview Essentials

Resource group (move) : rg_capestone
Location : South India
Subscription (move) : Azure subscription 1
Subscription ID : fc92392b-57f7-4d09-84f8-1e36b82658c0
Disk state : Available
Tags (edit) : Click here to add tags

Performance : Standard
Replication : Locally-redundant storage (LRS)
Account kind : StorageV2 (general purpose v2)
Provisioning state : Succeeded
Created : 4/21/2022, 12:30:40 PM

Properties Monitoring Capabilities (5) Recommendations Tutorials Developer Tools

Data storage

Data Lake Storage Security

Hierarchical namespace Enabled
Default access tier Hot
Require secure transfer for REST API operations Enabled

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

Home > sacapestone

sacapestone | Containers ...

Search (Ctrl+)

+ Container Change access level Restore containers Refresh Delete

Show deleted containers

Name	Last modified	Public access level	Lease state	...
Logs	4/12/2022, 5:03:50 PM	Private	Available	...
dataset	4/12/2022, 5:04:08 PM	Private	Available	...
validdata	4/12/2022, 5:05:56 PM	Private	Available	...

Overview Activity log Tags Diagnose and solve problems Access Control (IAM) Data migration Events Storage browser (preview)

Containers File shares

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

Home > sacapestone >

dataset ...
Container

Search (Ctrl+)

Upload Add Directory Refresh Rename Delete Change tier Acquire lease Break lease

Authentication method: Access key (Switch to Azure AD User Account)
Location: dataset / ecom

Show deleted objects

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state	...
[...]							...
olist_customers_dataset.csv	4/12/2022, 5:04:59 PM	Hot (Inferred)		Block blob	8.62 MiB	Available	...
olist_geolocation_dataset.csv	4/12/2022, 5:05:10 PM	Hot (Inferred)		Block blob	58.44 MiB	Available	...
olist_order_items_dataset.csv	4/12/2022, 5:05:16 PM	Hot (Inferred)		Block blob	14.72 MiB	Available	...
olist_order_payments_dataset.csv	4/12/2022, 5:04:47 PM	Hot (Inferred)		Block blob	5.51 MiB	Available	...
olist_order_reviews_dataset.csv	4/12/2022, 5:04:57 PM	Hot (Inferred)		Block blob	13.78 MiB	Available	...
olist_orders_dataset.csv	4/12/2022, 5:05:11 PM	Hot (Inferred)		Block blob	16.84 MiB	Available	...
olist_products_dataset.csv	4/12/2022, 5:04:43 PM	Hot (Inferred)		Block blob	2.27 MiB	Available	...
olist_sellers_dataset.csv	4/12/2022, 5:04:43 PM	Hot (Inferred)		Block blob	170.61 KiB	Available	...
product_category_name_translation.csv	4/12/2022, 5:04:42 PM	Hot (Inferred)		Block blob	2.55 KiB	Available	...

Overview Diagnose and solve problems Access Control (IAM)

Settings Shared access tokens Manage ACL Access policy Properties Metadata

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

Home > Azure Databricks >

Create an Azure Databricks workspace ...

Basics Networking Advanced Tags Review + create

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure subscription 1
Resource group * ⓘ rg_capestone [Create new](#)

Instance Details

Workspace name * ecomAnalysis
Region * South India
Pricing Tier * Trial (Premium - 14-Days Free DBUs)

[Review + create](#) [< Previous](#) [Next : Networking >](#)

Home > Azure Databricks >

ecomAnalysis ⌂ ...

Azure Databricks Service

Search (Ctrl+ /) Delete JSON View

Overview

Status : Active Managed Resource Group : [databricks-rg-ecomAnalysis-uta7sykujdzc](#)
Resource group : [rg_capestone](#) URL : <https://adb-1318388297379029.azuredatabricks.net>
Location : South India Pricing Tier : Trial (Premium - 14-Days Free DBUs)
Subscription : [Azure subscription 1](#)
Subscription ID : fc92392b-57f7-4d09-84f8-1e36b82658c0
Tags (edit) : [Click here to add tags](#)

Activity log

Access control (IAM)

Tags

Settings

Virtual Network Peerings

Encryption

Properties

Locks

Automation

Tasks (preview)

Export template

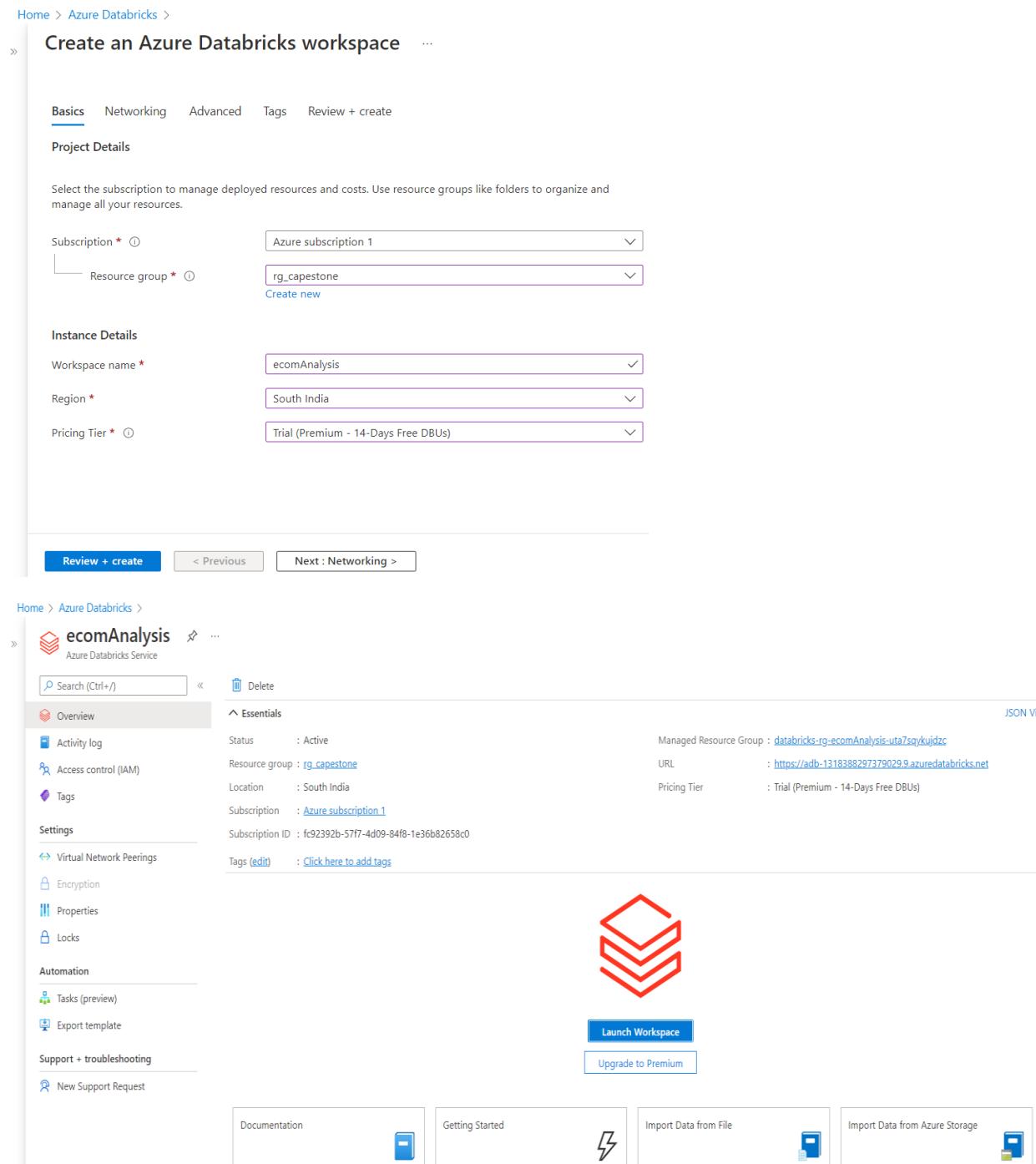
Support + troubleshooting

New Support Request

Launch Workspace

Upgrade to Premium

Documentation Getting Started Import Data from File Import Data from Azure Storage



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):

Microsoft Azure | Databricks

Create Cluster

New Cluster | [Cancel](#) | [Create Cluster](#) | DBU / hour: 0.75 - 1.5 | 0-1 Worker:0-14 GB Memory, 0-4 Cores | 1 Driver:14 GB Memory, 4 Cores

Cluster name: mycap

Cluster mode: Standard

Databricks runtime version: Runtime: 7.3 LTS (Scala 2.12, Spark 3.0.1)

Autopilot options:

- Enable autoscaling
- Terminate after 60 minutes of inactivity

Worker type: Standard_DS3_v2 (14 GB Memory, 4 Cores) | Min workers: 0 | Max workers: 1 | Spot instances

Driver type: Same as worker (14 GB Memory, 4 Cores)

DBU / hour: 0.75 - 1.5 | Standard_DS3_v2

[Advanced options](#)

7. Create an App registration named “capestone”:

Home > App registrations > capestone

[Overview](#) | [Delete](#) | [Endpoints](#) | [Preview features](#)

Essentials

Display name: capestone	Client credentials: Add a certificate or secret
Application (client) ID: 6922fb24-5b84-47c1-8d7e-4b7c4f6d6b30	Redirect URLs: Add a Redirect URI
Object ID: c15a1230-1c3a-45ef-8e70-36b383c351db	Application ID URI: Add an Application ID URI
Directory (tenant) ID: 5b98d9af-6151-464a-ac87-344743cacd35	Managed application in ...: capestone
Supported account types: My organization only	

Get Started | [Documentation](#)

Build your application with the Microsoft identity platform

The Microsoft identity platform is an authentication service, open-source libraries, and application management tools. You can create modern, standards-based authentication solutions, access and protect APIs, and add sign-in for your users and customers. [Learn more](#)

App registrations

[+ New registration](#) | [Endpoints](#) | [Troubleshooting](#) | [Refresh](#) | [Download](#) | [Preview features](#) | [Got feedback?](#)

Owned applications

Display name	Application (client) ID	Created on	Certificates & secrets
capestone	bc6ae2a3-32a9-43ff-aa24-b5973588cd7a	4/12/2022	Current

8. Create a key vault named “ecom1”:

The screenshot shows the Azure Key Vaults list page. At the top, there are navigation links for Home, Key vaults, and a search bar. Below the header are filter options for Subscription, Resource group, Location, and Add filter. A table lists the key vault 'ecom1'. The columns include Name, Type, Resource group, Location, Subscription, and Tags. The 'ecom1' entry has a location of 'South India' and is associated with 'Azure subscription 1'.

Name	Type	Resource group	Location	Subscription	Tags
ecom1	Key vault	rg_capestone	South India	Azure subscription 1	...

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

The screenshot shows the 'ecom1' Key Vault Secrets page. On the left is a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings, Keys, Secrets (which is selected), and Certificates. The main area displays a table of secrets. The columns are Name, Type, Status, and Expiration date. Three secrets are listed: 'clientid' (Type: String, Status: Enabled), 'clientSecret' (Type: String, Status: Enabled), and 'tenantid' (Type: String, Status: Enabled).

Name	Type	Status	Expiration date
clientid	String	✓ Enabled	
clientSecret	String	✓ Enabled	
tenantid	String	✓ Enabled	

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:

The screenshot shows the 'Create Secret Scope' page in the Microsoft Azure Databricks interface. The page title is 'Create Secret Scope' with 'Cancel' and 'Create' buttons. A description explains that it's a store for secrets identified by a name and backed by a specific store type. The 'Scope Name' field contains 'ecomScope'. The 'Manage Principal' dropdown is set to 'Creator'. Under 'Azure Key Vault', the 'DNS Name' is 'https://ecom1.vault.azure.net/' and the 'Resource ID' is '/subscriptions/fc92392b-57f7-4d09-84f8-1e36b82658c0/resourceGroups/rg_capeste'. The left side features a vertical sidebar with various icons for navigation.

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

1 # Define the variables used for creating connection strings
2 adlsAccountName = "sacapestone"
3 adlsContainerName = "dataset"
4 adlsFolderName = "ecom"
5 mountPoint = "/mnt/Files/dataset"
6
7 # Application (Client) ID
8 applicationId = dbutils.secrets.get(scope="ecomScope",key="clientid")
9
10 # Application (Client) Secret Key
11 authenticationKey = dbutils.secrets.get(scope="ecomScope",key="clientSecret")
12
13 # Directory (Tenant) ID
14 tenandId = dbutils.secrets.get(scope="ecomScope",key="tenantid")
15
16 endpoint = "https://login.microsoftonline.com/" + tenandId + "/oauth2/token"
17 source = "abfss://" + adlsContainerName + "@" + adlsAccountName + ".dfs.core.windows.net/" + adlsFolderName
18
19 # Connecting using Service Principal secrets and OAuth
20 configs = {"fs.azure.account.auth.type": "OAuth",
21             "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
22             "fs.azure.account.oauth2.client.id": applicationId,
23             "fs.azure.account.oauth2.client.secret": authenticationKey,
24             "fs.azure.account.oauth2.client.endpoint": endpoint}
25
26 # Mounting ADLS Storage to DBFS
27 # Mount only if the directory is not already mounted
28 if not any(mount.mountPoint == mountPoint for mount in dbutils.fs.mounts()):
29     dbutils.fs.mount(
30         source = source,
31         mount_point = mountPoint,
32         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:

1 %fs
2 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.

grid icon, chart icon, dropdown icon, download icon

Code for creating a log file:

```

1 import logging
2 import time
3 import datetime
4 """
5 This program using for custom logging in Pyspark and it will be created a log file.
6 """
7 file_date = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d-%H-%M-%S')
8 p_dir = '/tmp/'
9 p_filename = 'custom_log'+file_date+'.log'
10 p_logfile = p_dir + p_filename
11 print(p_logfile)
12 # create logger with 'Custom_log'
13 logger = logging.getLogger('ecom_log')
14 logger.setLevel(logging.DEBUG)
15 # create file handler which logs even debug messages
16 fh = logging.FileHandler(p_logfile,mode='a')
17 # create console handler with a higher log level
18 ch = logging.StreamHandler()
19 ch.setLevel(logging.DEBUG)
20 # create formatter and add it to the handlers
21 formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
22 #setting for ignoring frequent log information
23 #logging.getLogger("py4j").setLevel(logging.ERROR)
24 # tell the handler to use this format
25 #fh (file Handler)
26 fh.setFormatter(formatter)
27 #ch (console handler)
28 ch.setFormatter(formatter)
29 #Clearing old frequent log information to ignore that.
30 if (logger.hasHandlers()):
31     logger.handlers.clear()
32 # add the handlers to the logger
33 logger.addHandler(fh)
34 logger.addHandler(ch)

```

Starting the required loggers:

```
1 logger.info("*****START INFO LOGGER HERE*****")
2 logger.debug("*****START DEBUG LOGGER HERE*****")
3 logger.error("*****START ERROR LOGGER HERE*****")
4 logger.warning("*****START WARNING LOGGER HERE*****")
5 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

```
1 from pyspark import SparkConf, SparkContext
2 from pyspark.sql.functions import *
3 from pyspark.sql import SparkSession
4 import ipywidgets as widgets
5 from ipywidgets import interact, interactive
6 import pygal
7 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

```
1 df_customerdata = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_customers_dataset.csv")
2 df_cus=df_customerdata.dropDuplicates()
3 df_cus.printSchema()
4 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.

```
1 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

```
1 df_orderitem = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_items_dataset.csv")
2 df_ordi = df_orderitem.dropDuplicates()
3 df_ordi.printSchema()
4 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’.

```
1 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

```
1 df_ordi=df_ordi.withColumn('shipping_limit_date',to_timestamp('shipping_limit_date')).withColumn('order_item_id',df_ordi.order_item_id.cast('int'))
2 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
00143d0f86d6fb9f...	2 e95ee6822b66ac605...	a17ff621c590ea0fab...	2017-10-20 16:07:52	21.33	15.10	
0015ebb40f117286b...	1 50fd2b788d166edd...	b321bb669392f516...	2018-01-18 09:11:24	21.90	15.10	
00125cb692d048878...	1 lc0c0093a48f13ba7...	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 d54c5bb81fc2b38707...	0241d4d5d36f10f80...	2018-07-04 17:11:12	249.90	34.23	
00259na44fcad3fc04...	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)
```

```
1 df_payment = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_payments_dataset.csv")
2 df_pay = df_payment.dropDuplicates()
3 df_pay.printSchema()
4 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|
+-----+-----+-----+-----+
|4d3bffd4c7d6578ea...|           1| credit_card|                 1|    77.29|
|12e5cfe0e4716b59a...|           1| credit_card|                10| 157.45|
|298cdf1f73eb413e...|           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|
|4214cda550ce8ee6...|           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.

```
1 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

```
1 df_geoloc = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_geolocation_dataset.csv")
2 df_geo = df_geoloc.dropDuplicates()
3 df_geo.printSchema()
4 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.

```
1 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

1 df_review = spark.read.format("csv").option("header", "true").load("dbfs:/mnt/Files/dataset/olist_order_reviews_dataset.csv")
2 df_rev = df_review.dropDuplicates()
3 df_rev.printSchema()
4 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.

```
1 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

1 df_rev = df_rev.na.drop(subset=['review_creation_date','review_answer_timestamp'])
2 df_rev = df_rev.withColumn('review_creation_date',to_timestamp('review_creation_date')).withColumn('review_answer_timestamp',to_timestamp('review_answer_timestamp'))
3 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|
|cb2fc3e571b5ae85...|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|
|caf7d6d688909fedf0...|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|
|8670d52e15e00943a...|b9bf720beb4ab3728...|        4| recomend...| 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_a
nswer_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
|---+-----+-----+-----+-----+-----+-----+-----+-----+
| 3e0f398b664b20e32...|    esporte_lazer|          57|         900|           4|        400|          16|           4|
|1| 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_length	product_description_length	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

End 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|  
+-----+-----+-----+-----+  
|cm|  
+-----+-----+-----+-----+  
--+  
|3e0f398b664b26e32...|        esporte_lazer|            57|         9  
11|  
|96bd76ec8810374ed...|        esporte_lazer|            46|         2  
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_ap  
proved_at')).withColumn('order_delivered_customer_date',to_timestamp('order_delivered_customer_date  
'')).withColumn('order_estimated_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_estimated_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
1	d650b663c3b5f6fb3...	06713	cotia	SP
2	4cf490a58259286ad...	14910	tabatinga	SP
3	b7ba853e9551f4558...	95500	santo antonio da ...	RS
4	276677b5d08786d5d...	31730	belo horizonte	MG
5	51a04a8a6bdbcb23de...	12914	braganca paulista	SP
6	8cb7c5ddf41f4d506...	75110	anapolis	GO
7	5c030029b5916fed0...	88075	sao jose	SC
8	13c2ed7698b3ca92d...	84010	ponta grossa	PR
9	41ab63a91b8b264e8...	31555	belo horizonte	MG
10	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

:md 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]
```

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)
```

```

1 df_geo = df_sell.join(df_geo).where(df_sell['seller_zip_code_prefix']==df_geo['geolocation_zip_code_prefix']).drop(*
2 ('seller_id','seller_zip_code_prefix','seller_state','geolocation_city')).withColumnRenamed('seller_city','geolocation_city')
3 df_geo = df_geo.select('geolocation_zip_code_prefix','geolocation_lat','geolocation_lng','geolocation_city','geolocation_state')
4 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'}).withColumnRenamed('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 occurred while finding top seller*****")
Top selling products

```

Cmd 36

▶ (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_accessories	20
auto	20
furniture_decor	15
garden_tools	15
telephony	14

Saving the output into Datalake:

```
1 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*****")
```

Total Revenue generated between dd/mm/yyyy to dd/mm/yyyy

Cmd 39

```
1 df_ord.createOrReplaceTempView("orders")
2 df_pay.createOrReplaceTempView("payments")
```

Command took 0.07 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:11 PM on mycap

Cmd 40

```
1 try:
2     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'")
3     Totalrev.show()
4     logger.info("*****Finding total revenue generated*****")
5 except Exception as e:
6     print(e)
7     logger.error("*****Error occured while finding total revenue generated*****")
```

```
▶ (4) Spark Jobs
▶ Totalrev: pyspark.sql.dataframe.DataFrame = [total_revenue_generated: double]
+-----+
|total_revenue_generated|
+-----+
| 1.542183142999988E7|
+-----+
```

2022-04-19 13:57:52,820 - ecom_log - INFO - *****Finding total revenue generated*****

Saving the output into Datalake:

```
1 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
1 try:
2     product_category = df_ordi.join(df_prod,['product_id'])
3     product_category = product_category.join(df_tran,['product_category_name'])
4     product_category =
5         product_category.groupBy('product_category_name_english').count().withColumnRenamed('count','Total_item').withColumnRenamed('product_category_name_english','Product_Category')
6     product_category.show(100)
7     logger.info("*****Finding total orders by product category*****")
8 except Exception as e:
9     print(e)
10    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_clothing	132
kitchen_dining_living	281
small_appliances	679
la_cuisine	14
bed_bath_table	11115

Saving the output into Datalake:

```
1 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

```
1 try:
2     Customer_by_region_state = df_cus.groupBy('customer_state').count().withColumnRenamed('count','No_of_customers')
3     Customer_by_region_state.show(100)
4     logger.info("*****Finding total number of customers by region-state*****")
5 except Exception as e:
6     print(e)
7     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:

```
1 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

```
1 try:  
2     Customer_by_region_city = df_cus.groupBy('customer_city').count().orderBy('count', ascending=False).withColumnRenamed('count', 'No_of_customers')  
3     Customer_by_region_city.show(100)  
4     logger.info("*****Finding total number of customers by region-city*****")  
5 except Exception as e:  
6     print(e)  
7     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:

```
1 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 generated annually*****")  
except Exception as e:  
    print(e)  
    logger.error("*****Error occured while finding revenue generated annually*****")
```

Revenue generated annually

```
Cmd 54 Python ►▼▼-
```

```
1 try:
2     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)")
3     revenue_annual = revenue_annual.where(revenue_annual.annual_year.isNotNull())
4     revenue_annual.show()
5     logger.info("*****Finding revenue generated annually*****")
6 except Exception as e:
7     print(e)
8     logger.error("*****Error occurred while finding revenue generated 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 generated annually*****

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

```
Cmd 55
```

```
1 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 occurred while finding most valued customer*****")
```

Most valued customers and salesman

```
Cmd 58 Python ►▼▼-
```

```
1 try:
2     orders_df_filter = df_ord.filter(df_ord.order_status=='delivered')
3     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'))
4     Most_fav_customer = Most_fav.groupBy('Customer_id').count().withColumnRenamed('count','total_product_buy')
5     Most_fav_customer = Most_fav_customer.orderBy('total_product_buy',ascending=False)
6     Most_fav_customer.show()
7     logger.info("*****Finding most valued customer*****")
8 except Exception as e:
9     print(e)
10    logger.error("*****Error occurred 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
18de381f8a8d23ff... 15
adb32467ecc74b535... 15

Saving the output into Datalake:

```
1 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*****")
```

```
1 try:  
2     Most_fav_seller = Most_fav.groupBy('seller_id').count().withColumnRenamed('count','total_product_sold')  
3     Most_fav_seller = Most_fav_seller.orderBy('total_product_sold',ascending=False)  
4     Most_fav_seller.show()  
5     logger.info("*****Finding most valued salesman*****")  
6 except Exception as e:  
7     print(e)  
8     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:

```
1 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 occurred while finding total orders by regions-city*****")
Total orders by regions/city

```

Cmd 65

```

1 try:
2     Order_by_region = df_cus.join(orders_df_filter,['customer_id'],'leftsemi')
3     Customer_by_region_city = Order_by_region.groupBy('customer_city').count()
4     Customer_by_region_city.orderBy('count',ascending=False).withColumnRenamed('count','Total_orders').show()
5     Customer_by_region_city = Customer_by_region_city.sample(0.01)
6     Customer_by_region_city.show()
7     logger.info("*****Finding total orders by regions-city*****")
8 except Exception as e:
9     print(e)
10    logger.error("*****Error occurred 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:

```
1 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 occurred while finding total reviews of products*****")

```

Total reviews of products

```
Cmd 68
```

```
1 try:
2     Reviewed_product = df_ordi.join(orders_df_filter,['order_id'],'leftsemi').join(df_prod,['product_id']).join(df_rev,['order_id']).join(df_tran,
3     ['product_category_name']).select('review_id','order_id','product_id','product_category_name_english','review_comment_title','review_comment_message').dropna()
4     Reviewed_product = Reviewed_product.groupby('product_category_name_english','review_comment_title').count().groupBy('product_category_name_english').agg({'count':'sum'}).withColumnRenamed('sum(
5     count)','Total_review')
6     Reviewed_product = Reviewed_product.orderBy('Total_review',ascending = False)
7     Reviewed_product.show()
8     logger.info("*****Finding total reviews of products*****")
9 except Exception as e:
10     print(e)
11     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:

```
1 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'}).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*****")
```

Minimum and Maximum priced products

```
Cmd 71
1 try:
2     minmax = df_ordi.groupBy('product_id').agg({'price':'min'})
3     product_priced = minmax.join(df_prod,['product_id'])
4     product_priced = product_priced.join(df_tran,['product_category_name'])
5     product_priced =
6     product_priced.groupBy('product_category_name_english').agg({'min(price)':'min'}).withColumnRenamed('min(min(price))','Price').withColumnRenamed('product_category_name_english','Product_Name')
7     product_priced.orderBy('Price',ascending=True).show()
8
9     minmax = df_ordi.groupBy('product_id').agg({'price':'max'})
10    product_priced = minmax.join(df_prod,['product_id'])
11    product_priced = product_priced.join(df_tran,['product_category_name'])
12    product_priced =
13    product_priced.groupBy('product_category_name_english').agg({'max(price)':'max'}).withColumnRenamed('max(max(price))','Price').withColumnRenamed('product_category_name_english','Product_Name')
14    product_priced.orderBy('Price',ascending=False).show()
15    logger.info("*****Finding minimum and maximum priced products*****")
16 except Exception as e:
17     print(e)
18     logger.error("*****Error occurred 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 occurred while finding minimum and maximum priced product*****")
```

```
1 df_ordi.createOrReplaceTempView("order_item")
2 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
1 try:
2     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 ")
3     min_max_pro.show()
4     logger.info("*****Finding minimum and maximum priced product*****")
5 except Exception as e:
6     print(e)
7     logger.error("*****Error occurred 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 e95ee682b66ac605...	
999.99	0.85 96f608a5fe010cb34...	
999.99	0.85 422879e10f4668299...	
999.99	0.85 6ed0ed10d62b45f3d...	
999.99	0.85 f9b0ee95ce81ef8b...	
999.99	0.85 77f217636750a5d2e...	
999.99	0.85 009af1277432f1a05...	
999.99	0.85 22c5ddc22ea8cae84...	
999.99	0.85 472ee464088101bf2...	

Saving the output into Datalake:

```
1 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_ord,['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 occurred while finding returning customers****")  
Returning Customers to understand Customer loyalty
```

```
1 try:  
2     Return = df_ord.filter(df_ord.order_status=='delivered')  
3     Loyalty = Return.join(df_ord,['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'))  
4     Loyalty = Loyalty.groupBy('Customer_id').count().withColumnRenamed('count','total_product_buy')  
5     print("The Most Valuable Customer is: ",Loyalty.orderBy('total_product_buy').tail(1))  
6     Loyalty = Loyalty.orderBy('total_product_buy',ascending=False)  
7     Loyalty.show()  
8     logger.info("*****Finding returning customers*****")  
9 except Exception as e:  
10    print(e)  
11    logger.error("****Error occurred while finding returning customers****")  
  
▶ (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='fc3d1daec319d62d49fb5e1f83123e9', total_product_buy=21)]  
+-----+  
| Customer_id|total_product_buy|  
+-----+  
|fc3d1daec319d62d4...| 21|  
|be1b70680b9f9694d...| 20|  
|bd5d39761aa56689a...| 20|  
|10de381f8a8d23ffff...| 15|  
|adb32467ecc74b535...| 15|
```

Saving the output into Datalake:

```
1 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_ord,['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).withColumnRenamed('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
1 try:
2     no_orders = df_ord.filter((df_ord.order_status !='canceled') & (df_ord.order_status !='unavailable'))
3     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'))
4     no_orders = no_orders.groupBy('customer_id').count()
5     no_orders = no_orders.groupby('customer_id').sum('count').orderBy('sum(count)',ascending=False).withColumnRenamed('sum(count)',"Total_orders")
6     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")
7     no_orders_no_customers.show()
8     logger.info("*****Finding total number of orders by total number of customers*****")
9 except Exception as e:
10     print(e)
11     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:

```

1 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
1 month_yr = df_ord.withColumn('Month_year',date_format('order_purchase_timestamp','M')).withColumn('Year',year('order_purchase_timestamp'))
2 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'))
3 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
1 Analysis = month_yr.drop('order_item_id').withColumn('Total_Price',month_yr['price']+month_yr['freight_value'])
2 Analysis = Analysis.groupBy('MY').agg({'order_id':'count','price':'sum','freight_value':'sum','Total_Price':'sum'})
3 Analysis =
Analysis.withColumn('price_per_product',Analysis['sum(Total_price)']/Analysis['count(order_id)']).withColumn('freight_per_product',Analysis['sum(freight_value)']/Analysis['co
unt(order_id)']).withColumnRenamed('sum(Total_price)','Total_price').withColumnRenamed('count(order_id)','Order_id').withColumnRenamed('sum(freight_value)','freight_value').w
ithColumnRenamed('sum(price)','price').orderBy('MY',ascending=True)
4 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
1 !pip install plotly

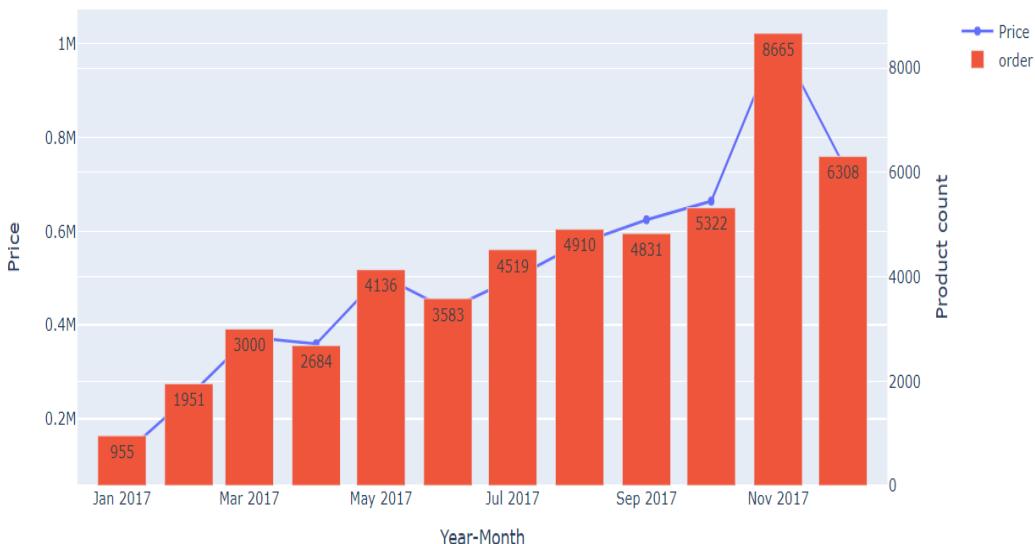
```

```

1 from plotly.subplots import make_subplots
2 import plotly.graph_objects as go
3
4 Analysis_80_20 = Analysis.filter(Analysis['Year']==2017)
5 list_date_a = Analysis_80_20.select('MY').rdd.flatMap(lambda x:x).collect()
6 list_date_count_a = Analysis_80_20.select('price').rdd.flatMap(lambda x:x).collect()
7 list_orders = Analysis_80_20.select('order_id').rdd.flatMap(lambda x:x).collect()
8
9
10 fig = go.Figure()
11 fig = make_subplots(specs=[[{"secondary_y":True}]])
12 fig.add_trace(
13     go.Scatter(
14         x = [str(x) for x in list_date_a],
15         y = [x for x in list_date_count_a],
16         name='Price',
17         text = [x for x in list_date_count_a],
18     ),secondary_y=False,
19 )
20 fig.add_trace(
21     go.Bar(
22         x = [str(x) for x in list_date_a],
23         y = [x for x in list_orders],
24         name='order',
25         text = [x for x in list_orders]
26     ),
27     secondary_y=True,
28 )
29 )
30 fig.update_layout(
31     autosize=False,
32     width = 1000,
33     height = 500,
34     title = 'Orders and Price',
35     xaxis_title = 'Year-Month',
36     yaxis_title = 'Price'
37 )
38 )
39 fig.update_yaxes(automargin=True,title_text='Product count',secondary_y=True)
40 fig.show()

```

Orders and Price



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 occurred while finding 80-20 analysis of products v/s sales*****")  
  
1  try:  
2      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")  
3      analysis80_20.show()  
4      logger.info("*****Finding 80-20 analysis of products v/s sales*****")  
5  except Exception as e:  
6      print(e)  
7      logger.error("*****Error occurred 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	1712553.669999998
beleza_saude	1657373.119999999
informatica_acess...	1585330.449999993
moveis_decoracao	1430176.389999992
relogios_presentes	1429216.680000002
esporte_lazer	1392127.559999998
utilidades_domest...	1094758.130000004
automotivo	852294.330000001
ferramentas_jardim	838280.750000002
cool_stuff	779697.999999999
moveis_escritorio	646826.490000001
brinquedos	619037.689999997
bebés	537884.659999999
perfumaria	506738.659999998

Saving the output into Datalake:

```
1  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 occurred while finding shipment age*****")  
    logging.shutdown()
```

Calculate Shipment Aging for each order

```
Cmd 89
```

```
try:
    Shipment_Aging = orders_df_filter.join(df_ordi,['order_id']).join(df_prod,['product_id']).join(df_tran,['product_category_name'])
    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 occurred 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_accessories 4969ccdf1cd3a57ea... a45263aab483c922e... 7ddcb64b5bc1ef36...				2
sports_leisure 3b29577134414b4e9... a92da5f6ab925f1b4... 218d46b86c1881d02...				4
perfumery 6ac15d09391be5079... 7850017ab0acb2ba... fe2032dab1a61af87...				3
sports_leisure 88977dfcf102823ad... 211ac32c2e3b19201... 80e6699fe29150b37...				1
stationery 863a4a0cdd7648886... e4f89f57953265051... 2c9e548be18521d1c...				1

Saving the output into Datalake:

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

Output files saved into Datalake:

The screenshot shows the Microsoft Azure Storage Explorer interface. The left sidebar displays the 'validdata' container under 'Container'. The main area shows a list of CSV files with their details. The columns include Name, Modified, Access tier, Archive status, Blob type, Size, and Lease state. The files listed are: [...] (modified 2023-09-01), analysis80_20.csv (modified 2023-09-01), cus_by_region_city.csv (modified 2023-09-01), cus_by_region_state.csv (modified 2023-09-01), loyalty.csv (modified 2023-09-01), minmax_product.csv (modified 2023-09-01), Most_fav_customer.csv (modified 2023-09-01), Most_fav_seller.csv (modified 2023-09-01), no_of_ord_cus.csv (modified 2023-09-01), product_category.csv (modified 2023-09-01), revenue_annual.csv (modified 2023-09-01), Reviewed_product.csv (modified 2023-09-01), ship.age.csv (modified 2023-09-01), top_selling.csv (modified 2023-09-01), total_orders_by_city.csv (modified 2023-09-01), and total_revenue.csv (modified 2023-09-01).

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
[...]	2023-09-01				-	---
analysis80_20.csv	2023-09-01				-	---
cus_by_region_city.csv	2023-09-01				-	---
cus_by_region_state.csv	2023-09-01				-	---
loyalty.csv	2023-09-01				-	---
minmax_product.csv	2023-09-01				-	---
Most_fav_customer.csv	2023-09-01				-	---
Most_fav_seller.csv	2023-09-01				-	---
no_of_ord_cus.csv	2023-09-01				-	---
product_category.csv	2023-09-01				-	---
revenue_annual.csv	2023-09-01				-	---
Reviewed_product.csv	2023-09-01				-	---
ship.age.csv	2023-09-01				-	---
top_selling.csv	2023-09-01				-	---
total_orders_by_city.csv	2023-09-01				-	---
total_revenue.csv	2023-09-01				-	---

Saving the logger file into Datalake:

Saving the logger file into the datalake

```
Cmd 92
1 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
1 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
1 %sql
2 DROP TABLE IF EXISTS CUSTOM_LOGGING;
3 CREATE TABLE IF NOT EXISTS CUSTOM_LOGGING
4 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
1 %sql
2 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:

The screenshot shows the Microsoft Azure Storage Explorer interface. At the top, there's a navigation bar with 'Microsoft Azure', a search bar, and several icons. Below the navigation bar, the path 'Home > sacapestone >' is shown. The main area displays a 'Container' named 'validdata'. On the left, there's a sidebar with various management options like 'Overview', 'Diagnose and solve problems', 'Access Control (IAM)', 'Settings', 'Shared access tokens', 'Manage ACL', 'Access policy', 'Properties', and 'Metadata'. The 'Overview' tab is selected. In the center, there's a table listing blobs. The table has columns for 'Name', 'Modified', 'Access tier', 'Archive status', and 'Blob type'. Two blobs are listed: 'custom_log' and 'output'. Both blobs have a yellow folder icon next to their names, indicating they are directories.

Name	Modified	Access tier	Archive status	Blob type
custom_log				
output				

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 generated 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 generated 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)
```

Python ▶▼▼-

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...	00143d0f86d6fb9f...	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...
f8d3e4dff67f24dc...	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	fffe9eff12fcdb74...	ffffe41c64501cc87c...	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 |  
+-----+
```

```

1 from pyspark.sql import functions
2 df_pro_ordi.groupBy('Category_Name').agg(functions.min('price'),
3                                         functions.max('price'),
4                                         functions.sum('price'),
5                                         functions.mean('price'),
6                                         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	29393.40999999996	57.63413725490195	57.63413725490195
alimentos_bebidas	10.00	99.99	15179.47999999998	54.60244604316546	54.60244604316546
artes	109.60	99.99	24202.64000000003	115.80210526315791	115.80210526315791
artes_e_artesanato	11.90	9.80	1814.01000000004	75.58375000000002	75.58375000000002
artigos_de_festas	109.00	90.00	4485.18	104.30651162790699	104.30651162790699
artigos_de_natal	11.62	95.00	8800.82000000002	57.52169934640524	57.52169934640524
audio	108.90	98.90	50688.4999999999	139.25412087912085	139.25412087912085
automotivo	10.00	99.99	592720.1100000007	139.95752302243227	139.95752302243227
bebés	10.00	999.00	409830.8900000106	133.75681788511784	133.75681788511784
bebidas	10.00	94.06	22428.6999999993	59.178627968337715	59.178627968337715
beleza_saude	1.20	990.00	1258681.339999943	130.16353050672123	130.16353050672123
brinquedos	10.75	99.99	483946.6000000097	117.54836045664342	117.54836045664342
cama_mesa_banho	10.50	999.99	1036988.6799999914	93.29632748537935	93.29632748537935
casa_comforto	104.99	99.90	58572.0399999998	134.95861751152069	134.95861751152069
casa_comforto_2	109.99	59.90	760.27	25.34233333333332	25.34233333333332
casa_construcao	10.50	98.80	83088.1199999998	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
ntile1 = df_pro_ordi.withColumn("ntile",ntile(2).over(windowSpec))
ntile1.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

```

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

```

▶ (5) Spark Jobs

▶ 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

```

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

```

▶ (5) Spark Jobs

▶ 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

```

1 #percent_rank
2 percent_rank1 = df_pro_ordi.withColumn("percent_rank",percent_rank().over(windowSpec))
3 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

```

1 #ntile
2 ntile1 = df_pro_ordi.withColumn("ntile",ntile(2).over(windowSpec))
3 ntile1.show()

```

▶ (5) Spark Jobs

▶ ntile1: 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

```

1 #cume_dist
2 cume_dist1 = df_pro_ordi.withColumn("cume_dist",cume_dist().over(windowSpec))
3 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...	75facf271f6fael1a5...	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

```

1 #lag
2 lag1 = df_pro_ordi.withColumn("lag",lag("price",2).over(windowSpec))
3 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...	75facf271f6fael1a5...	119.00	artes	112.00
4fe644d766c7566db...	c40f686ce1817e9bf...	119.00	artes	119.00
4fe644d766c7566db...	2eff76a325fc7035e...	119.00	artes	119.00
50a64d6b013980391...	dd621d7d942344da7...	119.90	artes	119.00
50a64d6b013980391...	22cbc77349f8317cd...	119.90	artes	119.00

```

1 #lead
2 lead1 = df_pro_ordi.withColumn("lead",lead("price",2).over(windowSpec))
3 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
1	49ab5384de586d3e4...	10ed5499d1623638e...	109.60	artes	110.00
2	986c47683f9e701e1...	816712dda4284eeda...	110.00	artes	110.99
3	986c47683f9e701e1...	dc37184f657b0a29c...	110.00	artes	110.99
4	4fe644d766c7566db...	1cde9001045df2307...	110.99	artes	110.99
5	4fe644d766c7566db...	a890088df39d4c062...	110.99	artes	110.99
6	4fe644d766c7566db...	8c9e6ed4aa9e011a3...	110.99	artes	110.99
7	4fe644d766c7566db...	b26c483e11283be69...	110.99	artes	110.99
8	4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	110.99
9	4fe644d766c7566db...	40a6a2ccfc8d35aa...	110.99	artes	110.99
10	4fe644d766c7566db...	bb924cd9637a232ef...	110.99	artes	110.99
11	4fe644d766c7566db...	00dc6ad47477b3b62...	110.99	artes	112.00
12	4fe644d766c7566db...	99687fd750781ecf4...	110.99	artes	119.00
13	c7ef7595086f66548...	e54709b9ea953fc92...	112.00	artes	119.00
14	4fe644d766c7566db...	75facf271f6fae1a5...	119.00	artes	119.00
15	4fe644d766c7566db...	c40f686ce1817e9bf...	119.00	artes	119.90
16	4fe644d766c7566db...	2eff76a325fc7035e...	119.00	artes	119.90
17	50a64d6b013980391...	dd621d7d942344da7...	119.90	artes	119.90
18	50a64d6b013980391...	22cbc77349f8317cd...	119.90	artes	12.00

```
1 #Aggregate Functions
```

```

2 windowSpecAgg = Window.partitionBy("Category_Name")
3 aggDF = df_pro_ordi.withColumn("row",row_number().over(windowSpec)).withColumn("avg", avg(col("price")).over(windowSpecAgg)).withColumn("sum",
4 sum(col("price")).over(windowSpecAgg)).withColumn("min", min(col("price")).over(windowSpecAgg)).withColumn("max",
5 max(col("price")).over(windowSpecAgg)).where(col("row")==1).select("Category_Name","avg","sum","min","max")
6 aggDF.show()

```

▶ (4) Spark Jobs

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

	Category_Name	avg	sum	min	max
1	agro_industria_e...	342.12485849056617	72530.47000000003	109.90	99.00
2	alimentos	57.63413725490217	29393.41000000011	100.00	99.00
3	alimentos_bebidas	54.60244604316538	15179.47999999976	10.00	99.99
4	artes	115.80210526315834	24202.64000000094	109.60	99.99
5	artes_e_artesanato	75.5837500000002	1814.01000000004	11.90	9.80
6	artigos_de_festas	104.30651162790699	4485.18	109.00	90.00
7	artigos_de_natal	57.52169934640513	8800.81999999985	11.62	95.00
8	audio	139.25412087912093	50688.5000000002	108.90	98.90
9	automotivo	139.95752302243486	592720.1100000116	10.00	99.99
10	bebés	133.75681788511963	409830.890000066	10.00	999.00
11	bebidas	59.178627968337764	22428.7000000001	10.00	94.06
12	beleza_saude	130.16353050672035	1258681.3399999856	1.20	990.00
13	brinquedos	117.54836045664574	483946.600000105	10.75	99.99
14	cama_mesa_banho	93.29632748538664	1036988.6800000725	10.50	999.99
15	casa_conforto	134.9586175115215	58572.04000000336	104.99	99.90
16	casa_conforto_2	25.3423333333322	760.269999999996	109.99	59.90
17	casa_construcao	137.56311258278075	83088.11999999957	10.50	98.80
18	cds dvds musicais	52.1428571428571461	730.01	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

Cmd 119

Python ▶ v - x

```
1 #Joining using normal join
2 norm_join = df_tran.join(df_prod,['product_category_name'])
3 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 pr	duct_height_cm product_width_cm
cama_mesa_banho	bed_bath_table a5bc1334f1762ce0a...	48
16		645
utilidades_domest...	housewares 53f92b0474f91fcb5...	54
51		2952
51		3
		30000
		76

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

Markdown ▶ v - x

```
1 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[dfs:/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_p
      hotos_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_phot
      os_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#149
      2, 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_w
      eight_g#1492))
```

Performing Broadcast Join:

```
1 #Now lets join using broadcast join
2 broad1 = broadcast(df_tran)
3 broad2 = df_prod.join(broadcast(broad1),df_prod.product_category_name==broad1.product_category_name)
4 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_english
a5bc1334f1762ce0a...	cama_mesa_banho	48	645	1	1200	16	10		
16 cama_mesa_banho	bed_bath_table								
53f92b0474f91fc...	utilidades_domest...	54	2952	3	30000	76	51		
51 utilidades_domest...	housewares								
45852eb1811c96c2...	cama_mesa_banho	56	393	2	1100	35	9		
35 cama_mesa_banho	bed_bath_table								
bc7496ccc90a64c0...	utilidades_domest...	28	238	1	150	26	16		
16 utilidades_domest...	housewares								
fdddfbd2585f9007f...	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								
02fcfaaba3a4b4882...	cama mesa banhol	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 shuffling 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 the PySpark application.

It is a join operation of a large dataframe with a smaller dataframe in PySpark Join model. It reduces data shuffling 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 the PySpark application.

```
Cmd 124
1 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 isnullobject(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: [AtLeastNNNulls(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)], 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 isnullobject(product_category_name#2609)
```

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

Demonstration of OOPs concept and parameterization

Cmd 126

```
1 class ecomAnalysis:
2     def __init__(self, storage_account_name, storage_account_key, container):
3         self.storage_container_name = storage_account_name
4         self.storage_account_key = storage_account_key
5         self.container = container
6
7     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'"):
8         try:
9             return spark.sql(query)
10        except:
11            print("error in query")
12
```

Python ▶

Command took 0.02 seconds -- by ananth.kamath@outlook.com at 4/19/2022, 7:22:31 PM on mycap

Cmd 127

```
1 if __name__ == '__main__':
2     storage_account_name = "sacapestone"
3     storage_account_key = "q6WpJ3arLAz1nmSMBlu4AMHAu6tIc366kSIp9x1qLcr43/UfwT4Zyi/J2mAcdHYqucnrIVhe4KXIArOTZcQK2g=="
4     container = "dataset"
5     df_ord.createOrReplaceTempView("orders")
6     df_pay.createOrReplaceTempView("payments")
7     b = ecomAnalysis(storage_account_name, storage_account_key, container)
8     print("Revenue between particular year 2016-01-01 to 2020-01-01")
9     total_rev_between = b.total_revenue_bet_years('2016-01-01', '2020-01-01')
10    total_rev_between.show()
```

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/J2mAcdHYqucnrIVhe4KXIArOTZcQ
K2g=="
    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.assertEqual("Test Failed!!",True)
AssertionError: 'Test Failed!!' is not equal to True

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

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