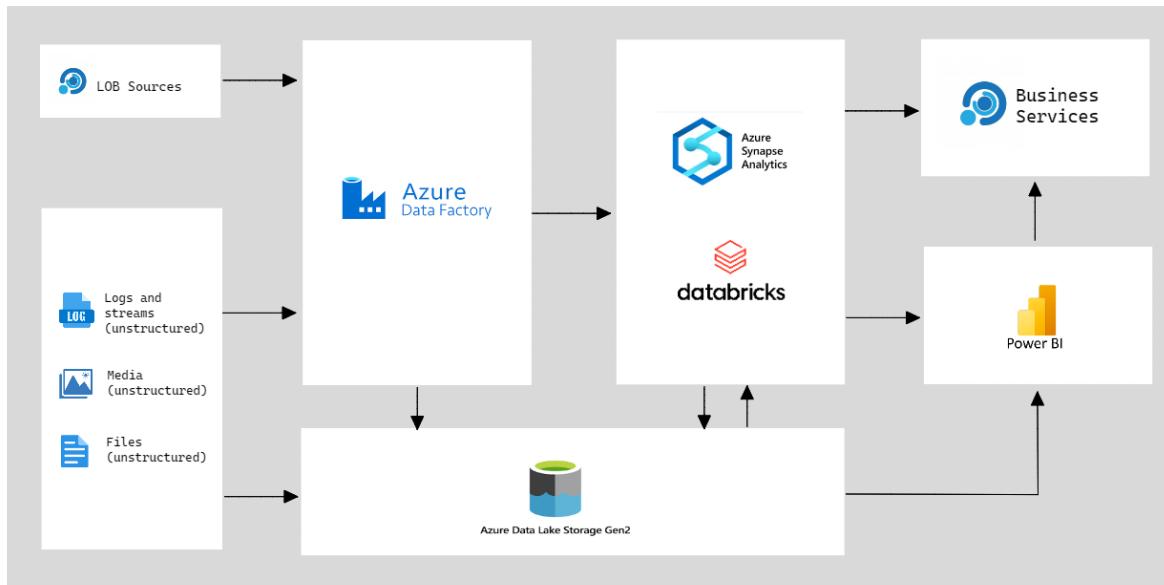


# RETAIL SALES ANALYSIS

Group 06: Roshan Rawat, Gangothri Gadige, Chandan Kumar, Narendra Reddy, Faryar Memon

## Modern Data Warehouse Architecture

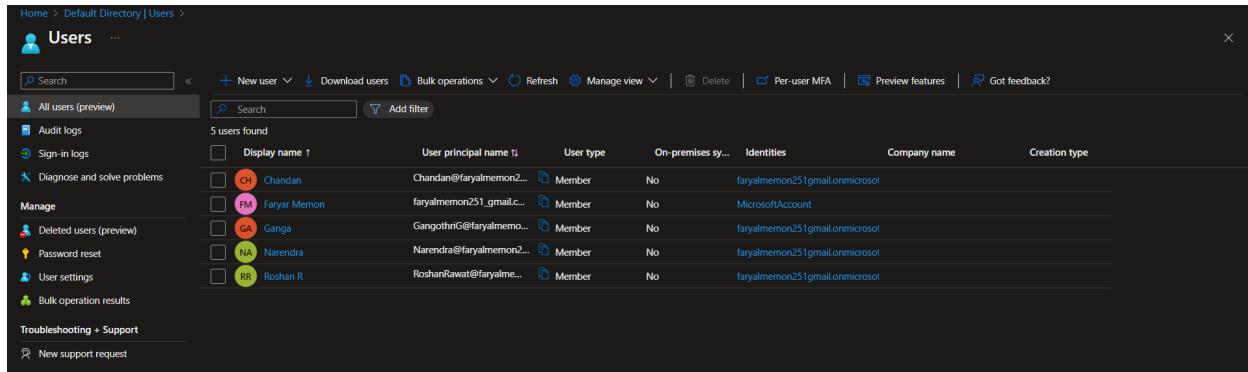


## Tasks

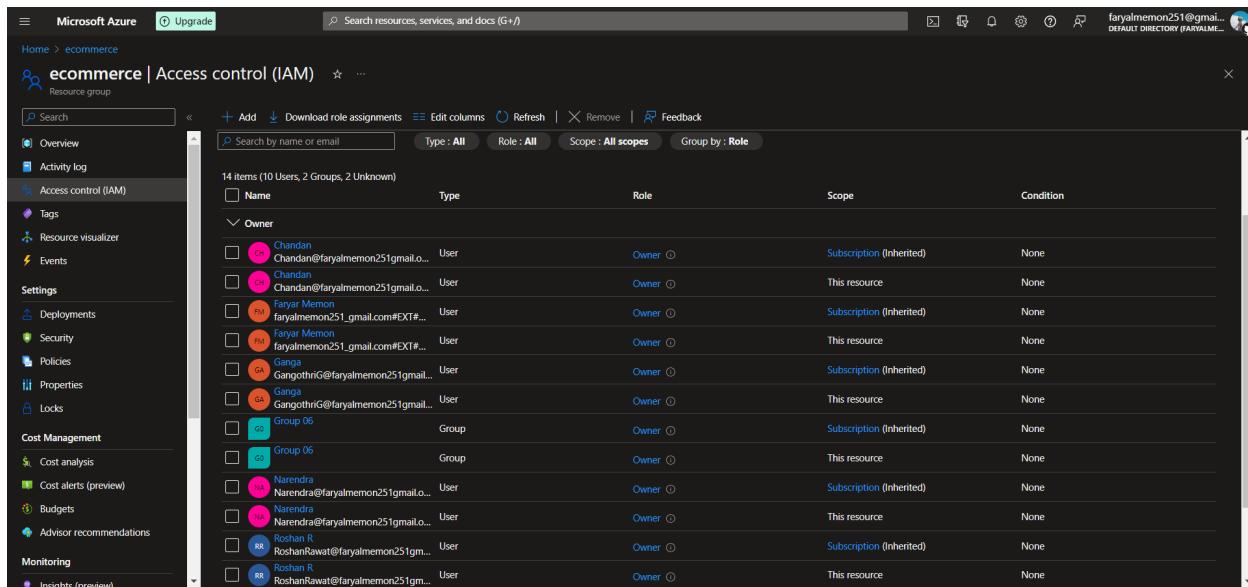
Source	Azure SQL	<ul style="list-style-type: none"><li>• Collecting the data from the client source</li></ul>
ETL	Azure Data Factory	<ul style="list-style-type: none"><li>• Using ADF to ingest data from client source</li></ul> <p><b>Azure SQL Database to Azure Data Lake Storage</b></p>
Analysis	Azure Synapse Analytics databricks Spark	<ul style="list-style-type: none"><li>• Integrating ADLS with Synapse using Linked Services &amp; mounting ADLS to Azure Databricks</li></ul>
Visualization	Power BI	<ul style="list-style-type: none"><li>• Integrating the analysis results with Power BI for impactful visualizations and reporting</li></ul>

# Data Governance

## 1. Create multiple users to work under the same resource group



### Give privileges to the users in Resource Group and Subscriptions



## 2. Create Storage Account and Database

# Data Ingestion

## Client Source Azure SQL Database config

```
create table retail(region varchar(1000), country varchar(1000), item_type varchar(1000),  
sales_channel varchar(1000), order_priority varchar(1000), order_date varchar(2000), order_id  
int, ship_date varchar(1000), units_sold int, unit_price float, unit_cost float, total_revenue float,  
total_cost float, total_profit float)
```

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD ='Roshan@3003'
```

```
create database scoped credential MyAzureBlobStorageCredential  
WITH IDENTITY ='SHARED ACCESS SIGNATURE',  
SECRET = '<SAS token>'
```

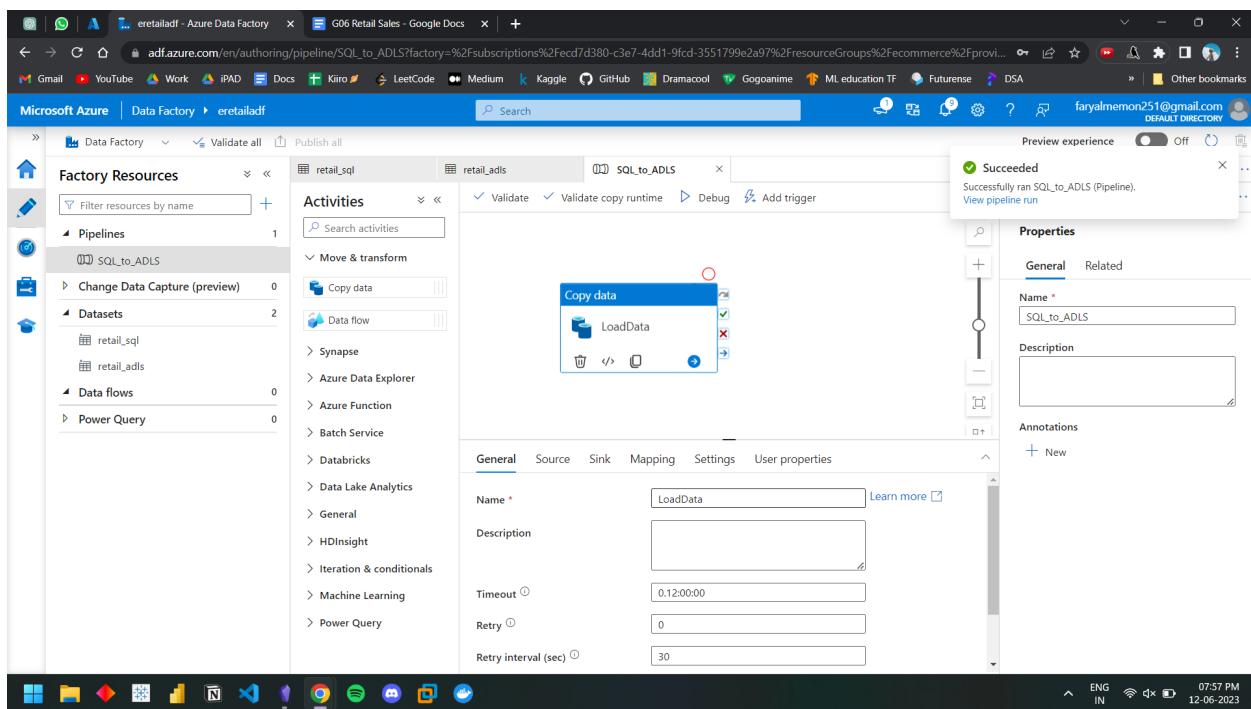
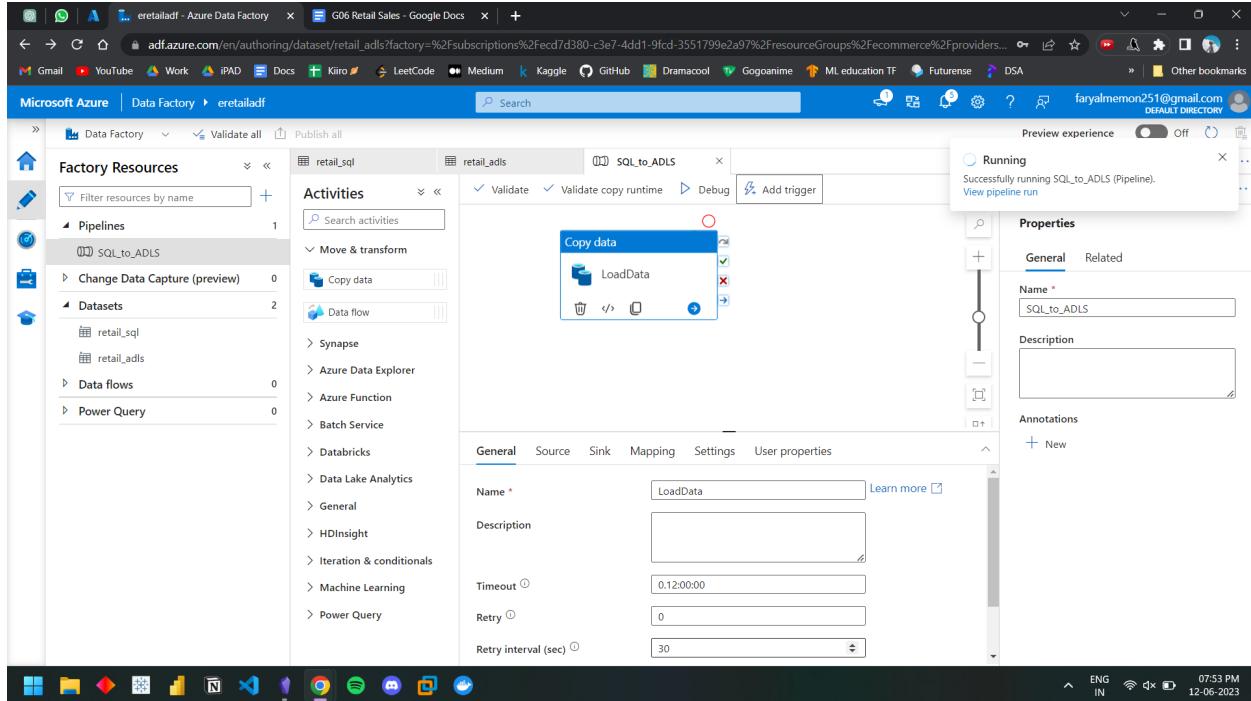
```
CREATE EXTERNAL DATA SOURCE MyAzureBlobStorage  
WITH ( TYPE =BLOB_STORAGE,  
LOCATION = 'https://ecomfiles1.blob.core.windows.net/csvfiles/',  
credential = MyAzureBlobStorageCredential)
```

```
BULK INSERT dbo.retail from 'sales_data.csv'  
with (check_constraints,
```

```
    DATA_SOURCE = 'MyAzureBlobStorage',  
    datafiletype ='char',  
    fieldterminator =',',  
    rowterminator='0x0a',  
    firstrow=2,  
    keepidentity,  
    tablock)
```

Rule name	Start IPv4 address	End IPv4 address	
ClientIPAddress_2023-6-4_14-48-11	122.171.16.101	122.171.16.101	
Faryar	192.168.159.1	192.168.159.1	
Ganga	192.168.1.20	192.168.1.20	
Narendra	192.168.40.1	192.168.40.1	
Roshan	192.168.56.1	192.168.56.1	

# Transferring Data from Client Source (Azure SQL Database) to ADLS using ADF



Home > eretailadls\_1686578727026 | Overview > eretailadls | Containers >

**clientdata** Container

Search < Upload + Add Directory ⏪ Refresh | ⏷ Rename ⏷ Delete ⏷ Change tier ⏷ Acquire lease ⏷ Break lease ⏪ Give feedback

Authentication method: Access key [Switch to Azure AD User Account](#)

Location: clientdata

Search blobs by prefix (case-sensitive)  Show deleted objects

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
retail.csv	6/12/2023, 7:56:57 PM	Hot (Inferred)		Block blob	868.34 KiB	Available

Settings

- Shared access tokens
- Manage ACL
- Access policy
- Properties
- Metadata

# Analysis

## Synapse Studio

1. Create Spark Pool and give each user the appropriate permissions of the Synapse Workspace

- 2.

New Apache Spark pool

Basics Additional settings Tags Review + create

Create an Synapse Analytics Apache Spark pool with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize.

**Apache Spark pool details**

Name your Apache Spark pool and choose its initial settings.

Apache Spark pool name \* ecomsparkpool

Isolated compute \*  Enabled  Disabled

Node size family \* Memory Optimized

Node size \* Small (4 vCores / 32 GB)

Autoscale \*  Enabled  Disabled

Number of nodes \* 3

Estimated price  Enabled  Disabled  
Est. cost per hour 130.05 to 130.05 INR [View pricing details](#)

Review + create Next: Additional settings > Cancel

New Apache Spark pool

Review + Create to provision with smart defaults, or visit each tab to customize.

**Apache Spark pool details**

Name your Apache Spark pool and choose its initial settings.

Apache Spark pool name \* ecomsparkpool

Isolated compute \*  Enabled  Disabled

Node size family \* Memory Optimized

Node size \* Small (4 vCores / 32 GB)

Autoscale \*  Enabled  Disabled

Number of nodes \* 3

Estimated price  Enabled  Disabled  
Est. cost per hour 130.05 to 130.05 INR [View pricing details](#)

Dynamically allocate executors \*  Enabled  Disabled

Default number of executors 1

Review + create Next: Additional settings > Cancel

New Apache Spark pool

**Basics**   Additional settings   Tags   Review + create

Customize additional parameters including pause settings and component versions.

**Automatic pausing**

Configure the pause settings for the Apache Spark pool.

Automatic pausing \*  Enabled  Disabled

Number of minutes idle \* 15

**Component version**

Select the Spark version for your Apache Spark pool.

Apache Spark *	3.3
Python	3.10
Scala	2.12.15
Java	1.8.0_282
Delta Lake	2.2

**Review + create**   < Previous   Next: Tags >   Cancel

New Apache Spark pool

**Apache Spark configuration**  Use default configuration  View configurations

**Packages**

Configure settings related to packages and how they can be installed onto your Spark pool.

Allow session level packages \*  Enabled  Disabled

**Intelligent cache**

Reserve space for Synapse storage cache. [Learn more](#)

Intelligent cache size \* 10%

**Review + create**   < Previous   Next: Tags >   Cancel

### 3. Analyzing Queries

#### # A.7. Display the number of sales year wise .(Using pyspark)

```
from operator import add
year_and_revenue = rdd_df.map(lambda l: (l[5].year, l[11]))
yearly_sales = year_and_revenue.reduceByKey(add).sortBy(lambda x: x[0])
yearly_sales.collect()
```

#### # A.8. Display the number of orders for each item.(Using pyspark)

```
from operator import add
each_item = rdd_df.map(lambda l: (l[2], 1))
item_ocount = each_item.reduceByKey(add).sortBy(lambda x: x[1], ascending=False)
item_ocount.collect()
```

```
# A.9. Display the country with highest sale (Using pyspark)
from operator import add
country_order = rdd_df.map(lambda l: ([l[1], l[11]]))
country_revenue = country_order.reduceByKey(add).sortBy(lambda x: x[1], ascending=False)
country_revenue.take(10)
```

#### **# B.4 Yearly sum for specific country**

```
from operator import add
iyear = 2013
year_and_revenue = rdd_df.map(lambda l: ([l[5].year, l[11]]))
yearly_sales = year_and_revenue.reduceByKey(add).sortBy(lambda x: x[0])
yearly_sales.filter(lambda l: l[0]==iyear).map(lambda x: x[1]).first()
```

#### **# B.6. monthly sales report of a particular year**

```
from operator import add

iyear = 2014
each_month = rdd_df.map(lambda l: ((l[5].year, l[5].month), l[11]))
yearlymonth_sales = each_month.reduceByKey(add).sortBy(lambda x: x[0])
yearlymonth_sales.filter(lambda l: l[0][0]==iyear).map(lambda x: (x[0][1], x[1])).collect()
```

#### **# B. 7) Quarterly sales report of a particular year**

```
import math
def quarter(date):
    return math.ceil(date.month/3)

iyear = 2014
each_quarterly = rdd_df.map(lambda l: ((l[5].year, quarter(l[5])), l[11]))
yearlyquarter_sales = each_quarterly.reduceByKey(add).sortBy(lambda x: x[0])
yearlyquarter_sales.filter(lambda l: l[0][0]==iyear).map(lambda x: (x[0][1], x[1])).collect()
```

#### **# B. 8, 11) For each quarter report for each year Write udf function for that. And for all year**

```
import math
def quarter(date):
    return math.ceil(date.month/3)

each_quarter = rdd_df.map(lambda l: ((l[5].year, quarter(l[5])), l[11]))
yearquarter_sales = each_quarter.reduceByKey(add).sortBy(lambda x: x[0])
yearquarter_sales.collect()
```

#### **# B.15. Compare the quarterly sales of each country of the particular year.**

```
import math
def quarter(date):
    return math.ceil(date.month/3)

iyear = 2014
country_quarterly = rdd_df.map(lambda l: ((l[5].year, l[1], quarter(l[5])), l[11]))
cyearlyquarter_sales = country_quarterly.reduceByKey(add).sortBy(lambda x: x[0])
cyearlyquarter_sales.filter(lambda l: l[0][0]==iyear).map(lambda x: ((x[0][1], x[0][2]),
x[1])).take(20)
```

---

```
1   df.printSchema()
✓ <1 sec - Command executed in 177 ms by gangothrigadige985 on 12:38:25 AM, 6/12/23

root
|-- Region: string (nullable = true)
|-- Country: string (nullable = true)
|-- Item Type: string (nullable = true)
|-- Sales Channel: string (nullable = true)
|-- Order Priority: string (nullable = true)
|-- Order Date: string (nullable = true)
|-- Order ID: integer (nullable = true)
|-- Ship Date: string (nullable = true)
|-- Units Sold: integer (nullable = true)
|-- Unit Price: double (nullable = true)
|-- Unit Cost: double (nullable = true)
|-- Total Revenue: double (nullable = true)
|-- Total Cost: double (nullable = true)
|-- Total Profit: double (nullable = true)
```

---

```
1   from pyspark.sql.functions import *
✓ <1 sec - Command executed in 192 ms by gangothrigadige985 on 12:38:25 AM, 6/12/23
```

---

```
1   spark.sql("set spark.sql.legacy.timeParserPolicy=LEGACY")
✓ <1 sec - Command executed in 172 ms by gangothrigadige985 on 12:38:26 AM, 6/12/23

DataFrame[key: string, value: string]

1   retail_df = df.withColumn('order date', to_date(col('Order Date'), 'MM/dd/yyyy')).withColumn('ship date', to_date(col('Ship Date'), 'MM/dd/yyyy'))
```

```
1   retail_df.printschema()
✓ <1 sec - Command executed in 199 ms by gangothrigac
```

```
root
 |-- Region: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Item Type: string (nullable = true)
 |-- Sales Channel: string (nullable = true)
 |-- Order Priority: string (nullable = true)
 |-- order date: date (nullable = true)
 |-- Order ID: integer (nullable = true)
 |-- ship date: date (nullable = true)
 |-- Units Sold: integer (nullable = true)
 |-- Unit Price: double (nullable = true)
 |-- Unit Cost: double (nullable = true)
 |-- Total Revenue: double (nullable = true)
 |-- Total Cost: double (nullable = true)
 |-- Total Profit: double (nullable = true)
```

```
1   display(retail_df.limit(10))
✓ <1 sec - Command executed in 1 sec 108 ms by gangothrigadige985 on 12:38:28 AM, 6/12/23
```

> Job execution Succeeded Spark 2 executors 8 cores

[View in monitoring](#) [Open](#)

View [Table](#) [Chart](#) [Export results](#)

Region	Country	Item Type	Sales Channel	Order Priority	order date
Central America and the Caribbean	Antigua and Barbuda	Baby Food	Online	M	2013-12-20
Central America and the Caribbean	Panama	Snacks	Offline	C	2010-07-05
Europe	Czech Republic	Beverages	Offline	C	2011-09-12
Asia	North Korea	Cereal	Offline	L	2010-05-13
Asia	Sri Lanka	Snacks	Offline	C	2015-07-20
Middle East and North Africa	Morocco	Personal Care	Offline	L	2010-11-08
Australia and Oceania	Federated States of Micronesia	Clothes	Offline	H	2011-03-28
Europe	Bosnia and Herzegovina	Clothes	Online	M	2013-10-14
Middle East and North Africa	Afghanistan	Clothes	Offline	M	2016-08-27

```
1   retail_df.createOrReplaceTempView("retail_table")
✓ <1 sec - Command executed in 183 ms by gangothrigadige985 on 12:38:28 AM, 6/12/23
```

```
1   products_with_a_occurrences=spark.sql("SELECT distinct `item type` FROM retail_table WHERE LOWER(`item type`) LIKE '%a%a%' ")
✓ <1 sec - Command executed in 170 ms by gangothrigadige985 on 12:38:28 AM, 6/12/23
```

```
1   products_with_a_occurrences.show()
✓ <1 sec - Command executed in 600 ms by gangothrigadige985 on 1:16:14 AM, 6/12/23
```

> Job execution Succeeded Spark 2 executors 8 cores

```
+-----+
| item type|
+-----+
|Personal Care|
+-----+
```

```
yearly_sales = retail_df.groupBy(year("Order Date").alias("Year")).agg(sum("Total Revenue").alias("Sales")).orderBy("Year")
<1 sec - Command executed in 171 ms by gangothrigadige985 on 1:16:24 AM, 6/12/23
```

```
adls_account_name = "retailsales123"
adls_container = "source"
adls_folder = "output"
# adls_output_path = f"abfs://{adls_account_name}.blob.core.windows.net/{adls_container}/{adls_folder}/products_with_a_occurrences.csv"
adls_output_path = r'abfs://source@retailsales123.dfs.core.windows.net/output/yearly_sales'

# r'abfs://retailsales123.blob.core.windows.net/source/output/products_with_a_occurrences.csv'

# Save the DataFrame to ADLS
yearly_sales.write \
....mode("overwrite") \
....csv(adls_output_path)
```

#### A.9. Display the country with highest sale (Using pyspark)

```
1  from operator import add
2  country_order = rdd_df.map(lambda l: (l[1], l[11]))
3  country_revenue = country_order.reduceByKey(add).sortBy(lambda x: x[1], ascending=False)
4  countrywise_sales = spark.createDataFrame(country_revenue, ['country', 'sales'])
5  countrywise_sales.show()
✓ 1 sec - Command executed in 1 sec 161 ms by gangothrigadige985 on 12:46:41 AM, 6/12/23
```

> Job execution Succeeded Spark 2 executors 8 cores

```
+-----+-----+
|      country|      sales|
+-----+-----+
|      Rwanda| 6.03987395899999E7|
|      Myanmar| 5.883846785E7|
| South Korea| 5.74343554299999E7|
|      Ghana| 5.627138265000001E7|
|      Niger| 5.529821127999998E7|
|     Grenada| 5.498818455000001E7|
|Republic of the C...| 5.437980838999998E7|
```

```
adls_account_name = "retailsales123"
adls_container = "source"
adls_folder = "output"
adls_output_path = r'abfs://source@retailsales123.dfs.core.windows.net/output/countrywise_sales'

# Save the DataFrame to ADLS
countrywise_sales.write \
    .mode("overwrite") \
    .csv(adls_output_path)
```

```
%%pyspark
df = spark.read.load('abfss://retaildb@ecomfiles1.dfs.core.windows.net/dbo.retail.csv',
format='csv'
## If header exists uncomment line below
, header=True,inferSchema=True
)
display(df.limit(10))
```

```
from pyspark.sql.functions import *

spark.sql("set spark.sql.legacy.timeParserPolicy=LEGACY")
retail_df = df.withColumn('order_date', to_date(col('order_date'),
'MM/dd/yyyy')).withColumn('ship_date', to_date(col('ship_date'), 'MM/dd/yyyy'))
```

```
retail_df.printSchema()

root
 |-- region: string (nullable = true)
 |-- country: string (nullable = true)
 |-- item_type: string (nullable = true)
 |-- sales_channel: string (nullable = true)
 |-- order_priority: string (nullable = true)
 |-- order_date: date (nullable = true)
 |-- order_id: integer (nullable = true)
 |-- ship_date: date (nullable = true)
 |-- units_sold: integer (nullable = true)
 |-- unit_price: double (nullable = true)
 |-- unit_cost: double (nullable = true)
 |-- total_revenue: double (nullable = true)
 |-- total_cost: double (nullable = true)
 |-- total_profit: double (nullable = true)
```

```
1   retail_df.printSchema()
✓ - Command executed in 155 ms on 8:38:58 PM, 6/04/23
```

---

```
root
 |-- region: string (nullable = true)
 |-- country: string (nullable = true)
 |-- item_type: string (nullable = true)
 |-- sales_channel: string (nullable = true)
 |-- order_priority: string (nullable = true)
 |-- order_date: date (nullable = true)
 |-- order_id: integer (nullable = true)
 |-- ship_date: date (nullable = true)
 |-- units_sold: integer (nullable = true)
 |-- unit_price: double (nullable = true)
 |-- unit_cost: double (nullable = true)
 |-- total_revenue: double (nullable = true)
 |-- total_cost: double (nullable = true)
 |-- total_profit: double (nullable = true)
```

```
retail_df.createOrReplaceTempView("retail_table")
products_with_a_occurrences=spark.sql("SELECT distinct item_type FROM retail_table WHERE
LOWER(item_type) LIKE '%a%a%' ")
```

```
1   products_with_a_occurrences.show()  
✓ - Command executed in 548 ms on 8:53:59 PM, 6/04/23
```

```
+-----+  
| item_type |  
+-----+  
|Personal Care|  
+-----+
```

# Databricks Workspace

Untitled Notebook 2023-06-10 11:09:24 Python

```
dbutils.fs.mount(  
    source='wasbs://mycontainer151@mystorageaccount151160.blob.core.windows.net/',  
    mount_point='/mnt/capstun',  
    extra_configs={'fs.azure.account.key.mystorageaccount151160.blob.core.windows.net':'y33+zi2yumQufcCd+G5fbU5iHt/73uCPQ91wZpCpViqDwClst4qu0E/  
    lonBQNC4WJPkYEwlala1+AstJ9vKhg=='}  
)
```

Out[2]: True

Command took 11.73 seconds -- by narendraredd1200@gmail.com at 6/10/2023, 12:29:16 PM on Narendra Reddy's Cluster

```
dbutils.fs.ls('/mnt/capstun')
```

Out[3]: [FileInfo(path='dbfs:/mnt/practice/5000 Sales Records (1).csv', name='5000 Sales Records (1).csv', size=623134, modificationTime=1686374675000)]

Command took 0.26 seconds -- by narendraredd1200@gmail.com at 6/10/2023, 12:31:16 PM on Narendra Reddy's Cluster

Shift+Enter to run  
Shift+Ctrl+Enter to run selected text

Untitled Notebook 2023-06-10 22:48:34 Python

```
sales_report=df.groupby('Order Date').agg({'Total Cost':'sum','Total Revenue':'sum','Total Profit':'sum'})
```

sales\_report: pyspark.sql.dataframe.DataFrame = [year(Order Date): integer, sum(Total Cost): double ... 2 more fields]

Command took 0.15 seconds -- by narendraredd1200@gmail.com at 6/10/2023, 11:05:55 PM on Narendra Reddy's Cluster

```
sales_report.show()
```

(2) Spark jobs

year(Order Date)	sum(Total Cost)	sum(Total Revenue)	sum(Total Profit)
2015	7.1175791873E8	9.84503786199993E8	2.72745875469999E8
2013	6.378836150199995E8	9.028807039600003E8	2.64997088940001E8
2014	5.999882564799997E8	8.563097745599985E8	2.564094890799977E8
2012	6.279867956099996E8	9.03647422800004E8	2.756606271000002E8
2016	6.0137735788E8	8.553197613500007E8	2.539424034699998E8
2010	5.762016424799999E8	8.177653681700003E8	2.415637256899994E8
2011	5.793084282600002E8	8.3762222024E8	2.583137919799994E8
2017	3.309619729000004E8	4.705522082599993E8	1.3959022736000013E8

Command took 0.83 seconds -- by narendraredd1200@gmail.com at 6/10/2023, 11:05:58 PM on Narendra Reddy's Cluster

Microsoft Azure databricks Q Search data, notebooks, recents, and more... CTRL + P myworkspace1 narendrareddi2000@gmail.com

Untitled Notebook 2023-06-10 22:48:34 Python v Last edit was 2 minutes ago Provide feedback

File Edit View Run Help Narendra Reddy's Cluster Share

Cmd 8

```
1 country_report=df.groupby([year(df['Order Date']).alias('year'),df['Country']]).agg({'Total Cost':'sum','Total Revenue':'sum','Total Profit':'sum'})
```

country\_report: pyspark.sql.dataframe.DataFrame = [year: integer, Country: string ... 3 more fields]

Command took 0.09 seconds -- by narendrareddi2000@gmail.com at 6/10/2023, 11:20:48 PM on Narendra Reddy's Cluster

Cmd 9

```
1 country_report.orderBy(country_report.year.asc(),country_report.Country.asc()).show()
```

(2) Spark Jobs

year	Country	sum(Total Cost)	sum(Total Revenue)	sum(Total Profit)
2010	Albania	5798207.049999999	8531301.81	2733094.76
2010	Algeria	631933.79	1131473.33	499539.5399999999
2010	Andorra	7228385.570000001	1.046352798E7	324322.41
2010	Angola	344365.35	577656.8	233298.65
2010	Antigua and Barbuda	9653361.49	1.35583839E7	4505022.41
2010	Armenia	5741959.42	7672185.09	1930234.67
2010	Australia	2517154.85	4133990.789999996	1616835.9400000002
2010	Austria	8142984.28	1.1632719089999987	3489734.81
2010	Azerbaijan	74388.6	111033.0	36644.4
2010	Bahrain	4016249.559999996	5940846.98	1924591.42
2010	Bangladesh	1163331.22	1557794.74	394463.52
2010	Barbados	1823108.59	1639914.55	606805.96
2010	Belarus	1306786.99	2080164.1800000002	773377.19
2010	Bolivia	830578.66	130001.07	170012.36

In [0]: double\_a=df.filter(df['Item Type'].like('%a%a%'))

In [0]: double\_aa=double\_a.select(double\_a['Item Type'])

In [0]: double\_aa.distinct().show()

```
+-----+
| Item Type|
+-----+
|Personal Care|
+-----+
```

# Data Storage

The output of the analysis queries were written into a container inside the ADLS itself

```
adls_account_name = "retailsales123"
adls_container = "source"
adls_folder = "output"
adls_output_path = r'abfs://source@retailsales123.dfs.core.windows.net/output/countrywise_sales'

# Save the DataFrame to ADLS
countrywise_sales.write \
    .mode("overwrite") \
    .csv(adls_output_path)
```

Home > [retailsales123](#) | Containers >

 **source** ...  
Container

Search  << Upload + Add Directory Refresh

Name
[...]
countrywise_sales
countrywise_yearly_sales
half_yearly_sales
item_type_orders_count
monthly_report_2014
monthly_sales
products_with_a_occurrences
quarterly_report_2014
quarterly_sales_total_revenue
specific_country_yearly_sales
yearly_sales

# Data Visualization and Reporting

## ADLS to Power BI Connection

The screenshot shows the Power BI desktop interface with two windows open.

The top window is a connection dialog titled "Azure Data Lake Storage Gen2". It contains fields for "Account key" (with a redacted value) and "Back" and "Connect" buttons. The URL <https://mydatalakegen2.dfs.core.windows.net/> is displayed above the dialog.

The bottom window is titled "Untitled - Power BI". It shows a preview of data from the URL <https://retailsales123.dfs.core.windows.net/>. The data is presented in a table with columns: Content, Name, Extension, Date accessed, Date modified, Date created, and Attributes. The table lists numerous CSV files, mostly named "\_SUCCESS". A note at the bottom of the preview states: "The data in the preview has been truncated due to size limits." Below the preview are buttons for "Combine", "Load", "Transform Data", and "Cancel".

# Power BI Dashboard

COUNTRY  
All

