ANALYZING DATA WITH APACHE SPARK

- MICROSOFT FABRIC - **Bamidele Ajamu**

The notebook below shows how you to use Apache Spark in Microsoft Fabric

- Load csv data into a dataframe in spark
- Confirming header false and header true function
- Since the data has no header, you can create a new header using spark sql types
- Explore the dataframe
  - Filter a dataframe
  - Group
  - Data Manipulation
- Work with tables and SQL
- Visualize the data

Sales Order data exploration

In [1]:
```
df = spark.read.format("csv").option("header","true").load("Files/orders/2019.csv")
# df now is a Spark DataFrame containing CSV data from "Files/orders/2019.csv".
display(df)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 3, Finished, Available)
SynapseWidget(Synapse.DataFrame, 56dcd7ce-1a7a-41df-aec4-cd33913dabe6)
```

In [2]:
```
df = spark.read.format("csv").option("header","false").load("Files/orders/2019.csv")
# df now is a Spark DataFrame containing CSV data from "Files/orders/2019.csv".
display(df)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 4, Finished, Available)
SynapseWidget(Synapse.DataFrame, 954ab7e2-496c-4ebc-9e20-98822484efcb)
```

In [3]:
```
# You can create a new header for the dataframe
from pyspark.sql.types import *

orderSchema = StructType([
    StructField("SalesOrderNumber", StringType()),
    StructField("SalesOrderLineNumber", IntegerType()),
    StructField("OrderDate", DateType()),
```

```
    StructField("CustomerName", StringType()),
    StructField("Email", StringType()),
    StructField("Item", StringType()),
    StructField("Quantity", IntegerType()),
    StructField("UnitPrice", FloatType()),
    StructField("Tax", FloatType())
    ])

df = spark.read.format("csv").schema(orderSchema).load("Files/orders/2019.csv")
display(df)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 5, Finished, Available)
SynapseWidget(Synapse.DataFrame, 1e4719a3-cd9a-4775-86c6-0274e4bba40e)
```

In [ ]:

Modify the code so that the file path uses a * wildcard to read the sales order data from all of the files in the orders folder:

In [4]:

```
from pyspark.sql.types import *

orderSchema = StructType([
    StructField("SalesOrderNumber", StringType()),
    StructField("SalesOrderLineNumber", IntegerType()),
    StructField("OrderDate", DateType()),
    StructField("CustomerName", StringType()),
    StructField("Email", StringType()),
    StructField("Item", StringType()),
    StructField("Quantity", IntegerType()),
    StructField("UnitPrice", FloatType()),
    StructField("Tax", FloatType())
    ])

df = spark.read.format("csv").schema(orderSchema).load("Files/orders/*.csv")
display(df)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 6, Finished, Available)
SynapseWidget(Synapse.DataFrame, d1819bfb-cb89-4a6b-ba7c-4ef69d101683)
```

Explore data in a dataframe

- Filter a dataframe
- Group
- Data Manipulation

1. Filter

In [5]:

```
customers = df['CustomerName', 'Email']
print(customers.count())
print(customers.distinct().count())
display(customers.distinct())
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 7, Finished, Available)
32718
12427
```

```
SynapseWidget(Synapse.DataFrame, 5982f9e3-df06-4bbd-b73a-f915962a1044)
```

In [6]:
```
# Use the where clause
customers = df.select("CustomerName", "Email").where(df['Item']=='Road-250 Red, 52')
print(customers.count())
print(customers.distinct().count())
display(customers.distinct())
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 8, Finished, Available)
133
133
SynapseWidget(Synapse.DataFrame, 59741d99-f5d2-4bf7-9d92-55fbef8313ef)
```

2. Aggregate and group data in a dataframe

In [7]:
```
productSales = df.select("Item", "Quantity").groupBy("Item").sum()
display(productSales)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 9, Finished, Available)
SynapseWidget(Synapse.DataFrame, a6b2a245-c1ea-41ab-912f-f4d7c7de0304)
```

In [9]:
```
# year aggregate of oder counts
from pyspark.sql.functions import *

yearlySales = df.select(year("OrderDate").alias("Year")).groupBy("Year").count().orderB
display(yearlySales)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 11, Finished, Available)
SynapseWidget(Synapse.DataFrame, cc4ac63f-88fd-4066-83db-3ec15ba6bea0)
```

Data Transformation with Spark

In [11]:
```
from pyspark.sql.functions import *

## Create Year and Month columns
transformed_df = df.withColumn("Year", year(col("OrderDate"))).withColumn("Month", mont

# Create the new FirstName and LastName fields
transformed_df = transformed_df.withColumn("FirstName", split(col("CustomerName"), " ")

# Filter and reorder columns
transformed_df = transformed_df["SalesOrderNumber", "SalesOrderLineNumber", "OrderDate"

# Display the first ten orders
display(transformed_df.limit(10))
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 13, Finished, Available)
SynapseWidget(Synapse.DataFrame, f24dff6a-92cc-4479-a4b1-083346da9a8d)
```

Save the transformed data

In [13]:
```
transformed_df.write.mode("overwrite").parquet('Files/transformed_data/orders')
print ("Transformed data has now been saved! Thanks")
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 15, Finished, Available)
```

```
Transformed data has now been saved! Thanks
```

In [14]:
```
# load a new dataframe from the parquet files in the transformed_orders/orders folder
orders_df = spark.read.format("parquet").load("Files/transformed_data/orders")
display(orders_df)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 16, Finished, Available)
SynapseWidget(Synapse.DataFrame, e526884b-8e78-4a3a-b76f-f34d21085bba)
```

Save data in partition files

In [16]:
```
# To partition data by Year and Month use the below code

orders_df.write.partitionBy("Year","Month").mode("overwrite").parquet("Files/partitione
print ("Transformed and partitioned data has been data saved!")
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 18, Finished, Available)
Transformed and partitioned data has been data saved!
```

In [17]:
```
# You can load a new dataframe from the orders.parquet file using the below code e-g 20
orders_2020_df = spark.read.format("parquet").load("Files/partitioned_data/Year=2020/Mo
display(orders_2020_df)
```

```
StatementMeta(, d8217fed-e3ea-4bfc-af57-3dc511ee4e88, 19, Finished, Available)
SynapseWidget(Synapse.DataFrame, 1962b3ad-158b-420d-b1c2-30aa452c9b7c)
```

Work with tables and SQL

Create a table

In [2]:
```
# Create a new table
df.write.format("delta").saveAsTable("salesorder")

# Get the table description
spark.sql("DESCRIBE EXTENDED salesorder").show(truncate=False)
```

In [16]:
```
%%sql

SELECT * FROM sales
LIMIT 5
```

```
StatementMeta(, 15388063-942e-49d6-a1d4-ee8c2fa33ac9, 18, Finished, Available)
```

Out[16]:
```
<Spark SQL result set with 5 rows and 9 fields>
```

In [30]:
```
df = spark.sql("SELECT * FROM Sales_LakeHouse.sales LIMIT 1000")
display(df)
```

```
StatementMeta(, 15388063-942e-49d6-a1d4-ee8c2fa33ac9, 32, Finished, Available)
SynapseWidget(Synapse.DataFrame, bd69700b-df1a-42ef-92d6-d0720160b8ba)
```

In [25]:
```
%%sql
SELECT YEAR(OrderDate) AS OrderYear,
       SUM((UnitPrice * Quantity) + TaxAmount) AS GrossRevenue
```

```
FROM sales
GROUP BY YEAR(OrderDate)
ORDER BY OrderYear
```

```
StatementMeta(, 15388063-942e-49d6-a1d4-ee8c2fa33ac9, 27, Finished, Available)
```

Out[25]:
```
<Spark SQL result set with 1 rows and 2 fields>
```

Visualize data with Spark

In [26]:
```
%%sql
SELECT * FROM sales
# you can use the %% to convert the code to writing sql queries
```

```
StatementMeta(, 15388063-942e-49d6-a1d4-ee8c2fa33ac9, 28, Finished, Available)
```

Out[26]:
```
<Spark SQL result set with 1000 rows and 9 fields>
```

In [40]:
```
sqlQuery = "SELECT CAST(YEAR(OrderDate) AS CHAR(4)) AS OrderYear, \
                SUM((UnitPrice * Quantity) + TaxAmount) AS GrossRevenue \
            FROM Sales_LakeHouse.sales \
            GROUP BY CAST(YEAR(OrderDate) AS CHAR(4)) \
            ORDER BY OrderYear"
```

```
StatementMeta(, 15388063-942e-49d6-a1d4-ee8c2fa33ac9, 42, Finished, Available)
```

In [42]:
```
df_spark = spark.sql(sqlQuery)
df_spark.show()
```

```
StatementMeta(, 15388063-942e-49d6-a1d4-ee8c2fa33ac9, 44, Finished, Available)
+---------+--------------------+
|OrderYear|        GrossRevenue|
+---------+--------------------+
|     null|2.2602264277594723E7|
+---------+--------------------+
```

In [6]:
```
# conda install pandoc
```

In [5]:
```
# conda update -n base -c defaults conda
```