

Mini Project – Big Data Applications ---- shshaik@iu.edu

Introduction –

The following extensive analysis is based on the UK-based online retail dataset that comprises 541,909 transaction records from December 2010 to September 2011. The subject retailer specializes in unique, all-occasion gifts, and hence can be used as an ideal case study for any ecommerce analytics. The dataset encompasses major business metrics ranging from transaction information to product and customer data across various geographies.

Project Objectives -

Implement a scalable big data processing pipeline using AWS services
Develop machine learning-based models for High value purchase prediction. Create interactive visualizations that underpin business insights. Analyze Country sales geographically and sales patterns.

Dataset -

Source -

<https://archive.ics.uci.edu/dataset/352/online+retail>

This is a transactional dataset containing 541,909 records of purchases made between 01/12/2010 and 09/12/2011 for an online retailer based in the UK, specializing in unique, all-occasion gifts. The main features of the dataset are transaction details: InvoiceNo, InvoiceDate, Quantity, UnitPrice; product

identifiers: StockCode, Description; customer information: CustomerID, Country; and no missing values.

The dataset can support classification and clustering tasks, therefore providing ample opportunities for customer segmentation, sales forecasting, and product performance analysis. Being multivariate, sequential, and time-series in nature, it is ideal for exploring consumer trends and informing business strategies.

Methodology –

Setting Up Data Infrastructure

- AWS S3 Setup
- Created specific buckets for storing raw and processed data
- Appropriately configured the directory structure in the data storage
- Configured access using AWS CLI

Computing Environment Setup

- Provisioned the EC2 instance with adequate specifications
- Configured Linux with the required dependencies
- Installed PySpark and verified that it works
- Development Tools Setup
- Jupyter Lab for interactive development
- VS Code configuration with remote SSH capabilities
- AWS CLI to provide easy interaction with S3

Data Processing Pipeline

Initial Data Processing

- Ingestion of data from S3 using AWS CLI
- Schema validation and verification of data types
- Data cleaning procedure implementation

Transformation Steps

- DateTime Processing
- Converted InvoiceDate to timestamp format
- Created InvoiceYearMonth for temporal analysis
- Implemented date-based aggregations

Financial Calculations

- Generated TotalPrice column: Quantity × UnitPrice
- Computed revenue metrics per transaction
- Implemented currency standardization

Feature Engineering

- Created HighValue_Purchase indicator
- Developed customer segmentation metrics
- Built a ML model in sagemaker and Generated geographical aggregations using PowerBI

SageMaker Implementation -

Amazon SageMaker played a crucial role in developing machine learning solution. The implementation process involved:

- Model Configuration

Created a new binary classification model for
HighValue_Purchase prediction.

Processed 456,829 rows and 1,827,316 total cells

The automated ML of SageMaker was used to derive a very accurate model with a lot of speed that is appropriate for production deployment. With integrated tools for model evaluation and feature importance, the platform allowed various business insights to be uncovered.

Results –

- Performance Achievements

Model achieved exceptional metrics:

99.994% balanced accuracy

1.790 log loss score

0.101 seconds inference latency

Perfect classification with zero false positives/negatives

- Feature Analysis

The model identified critical predictive features:

Quantity emerged as the primary driver (59.67%)

UnitPrice showed significant influence (37.24%)

CustomerID and Country contributed smaller but meaningful impacts

PowerBI – Visualization Insights

Temporal Analysis

- Interactive time slider (12/1/2011 to 12/20/2015), Monthly revenue trends by country, Seasonal pattern identification

Geographic Performance

- Country-wise revenue distribution, Regional performance comparisons

Product Analysis

- Product performance, Quantity sold distributions, Cross-country product preferences

Conclusions –

Through this comprehensive analysis of the online retail dataset, I gained valuable insights that demonstrate the effectiveness of big data analytics in e-commerce applications.

Key Findings -

My project delivered several significant results:

- Successfully processed and analyzed 541,909 transaction records using an AWS-based data pipeline
- The machine learning model achieved 99.994% accuracy with perfect F1 scores
- Geographic analysis revealed the UK as the dominant market
- Feature importance analysis showed Quantity (59.67%) and UnitPrice (37.24%) as key drivers

Technical Achievements

I successfully implemented several technical components:

- Built an integrated AWS infrastructure using S3, EC2, and SageMaker
- Developed a PySpark-based data processing pipeline
- Created interactive PowerBI dashboards
- Deployed a machine learning model with 0.101 seconds inference latency

Future Recommendations

- Technical Enhancements – Optimization
- Deploy real-time data streaming using AWS Kinesis
- Develop API endpoints for integration
- Establish sales forecasting system

This project demonstrates how combining cloud infrastructure, machine learning, and business intelligence tools can provide valuable insights for retail operations

References –

UCI Machine Learning Repository - Online Retail Dataset

AWS Documentation (S3, EC2, SageMaker)

PySpark Documentation

PowerBI Documentation for Visualization

Tasks ---

1) Dataset Selection –

Source -

<https://archive.ics.uci.edu/dataset/352/online+retail>

This is a transactional dataset containing 541,909 records of purchases made between 01/12/2010 and 09/12/2011 for an online retailer based in the UK, specializing in unique, all-occasion gifts. The main features of the dataset are transaction details: InvoiceNo, InvoiceDate, Quantity, UnitPrice; product identifiers: StockCode, Description; customer information: CustomerID, Country; and no missing values.

The dataset can support classification and clustering tasks, therefore providing ample opportunities for customer segmentation, sales forecasting, and product performance analysis. Being multivariate, sequential, and time-series in nature, it is ideal for exploring consumer trends and informing business strategies.

2) Environment Setup –

1. AWS S3 for Data Storage –

a. Step 1 –

Creating bucket instance

The screenshot shows the AWS S3 console interface. At the top, there's a banner for 'Account snapshot - updated every 24 hours' and a link to 'View Storage Lens dashboard'. Below this, there are tabs for 'General purpose buckets' (selected) and 'Directory buckets'. A search bar labeled 'Find buckets by name' is present. The main table lists one 'General purpose buckets' entry:

| Name | AWS Region | IAM Access Analyzer | Creation date |
|-------------------------|---------------------------------|---|--|
| customer-storage-bucket | US East (N. Virginia) us-east-1 | View analyzer for us-east-1 | December 7, 2024, 11:14:26 (UTC-05:00) |

Created directories – (raw and processed data)

The screenshot shows the 'customer-storage-bucket' details page. At the top, there's a success message: 'Successfully created folder "processed_data".' Below this, the bucket name is displayed with a 'Info' link. The 'Objects' tab is selected, showing two objects in the 'Objects' table:

| Name | Type | Last modified | Size | Storage class |
|-----------------|--------|---------------|------|---------------|
| processed_data/ | Folder | - | - | - |
| raw_data/ | Folder | - | - | - |

b. Step 2 – Upload the raw dataset to the S3 bucket

The screenshot shows the AWS S3 'Upload' interface. At the top, there's a message about uploading files larger than 160GB. Below is a drag-and-drop area and a table for managing files. The table shows one file: 'Online_Retail.xlsx' (22.6 MB). The destination is set to 's3://customer-storage-bucket/raw_data/'. The 'Upload' button is highlighted in yellow at the bottom right.

| Name | Folder | Type | Size | Status | Error |
|--------------------|--------|--|---------|-----------|-------|
| Online_Retail.xlsx | - | application/vnd.openxmlformats-office... | 22.6 MB | Succeeded | - |

Upload: status

After you navigate away from this page, the following information is no longer available.

Summary

| Destination | Succeeded | Failed |
|--|---------------------------|-------------------|
| s3://customer-storage-bucket/raw_data/ | 1 file, 22.6 MB (100.00%) | 0 files, 0 B (0%) |

Files and folders

2. Linux Environment with PySpark –

a. Step 1 – SETUP ENVIRONMENT EC2 INSTANCE

- Go to aws cloud, search for ec2 instance
 - o Create an Aws EC2 instance –
 - o By providing name tag, and choosing appropriate AMI

The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The user has selected the 'Amazon Machine Image (AMI)' tab. They have chosen the 'Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type' AMI (AMI ID: ami-0166fe664262f64c). The instance type is set to 't2.micro'. The 'Free tier' information is displayed, indicating 750 hours of t2.micro usage included. The 'Launch instance' button is visible at the bottom right.

- While creating the instance generate the .pem file (save in local directory, to connect with instance locally later)

The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The user is prompted to select a key pair name, which is required. A dropdown menu shows 'BDA' selected. An alternative option 'Proceed without a key pair (Not recommended)' is available. A 'Create new key pair' button is also present. The 'Subnet' section is partially visible below.

-After successful deployment

The screenshot shows the AWS EC2 Instances Launch log page. At the top, there's a green success message: "Successfully initiated launch of instance (i-05948133b3df32809)". Below this, there's a "Launch log" button. A "Next Steps" section follows, containing several cards:

- Create billing and free tier usage alerts**: To manage costs and avoid surprise bills, set up email notifications for billing and free tier usage thresholds. Includes a "Create billing alerts" button.
- Connect to your instance**: Once your instance is running, log into it from your local computer. Includes a "Connect to instance" button and a "Learn more" link.
- Connect an RDS database**: Configure the connection between an EC2 instance and a database to allow traffic flow between them. Includes a "Connect an RDS database" button and a "Learn more" link.
- Create EBS snapshot policy**: Create a policy that automates the creation, retention, and deletion of EBS snapshots. Includes a "Create EBS snapshot policy" button.
- Manage detailed monitoring**: Enable or disable detailed monitoring for the instance. If you enable detailed monitoring, the system will collect and store metrics for the instance. Includes a "Learn more" link.
- Create Load Balancer**: Create a application, network gateway or classic Elastic Load Balancer. Includes a "Create a new RDS database" button and a "Learn more" link.
- Create AWS budget**: AWS Budgets allows you to create budgets, forecast spend, and take action on your costs and usage from the instance. Includes a "Create a new RDS database" button and a "Learn more" link.
- Manage CloudWatch alarms**: Create or update Amazon CloudWatch alarms for the instance. Includes a "Create a new RDS database" button and a "Learn more" link.

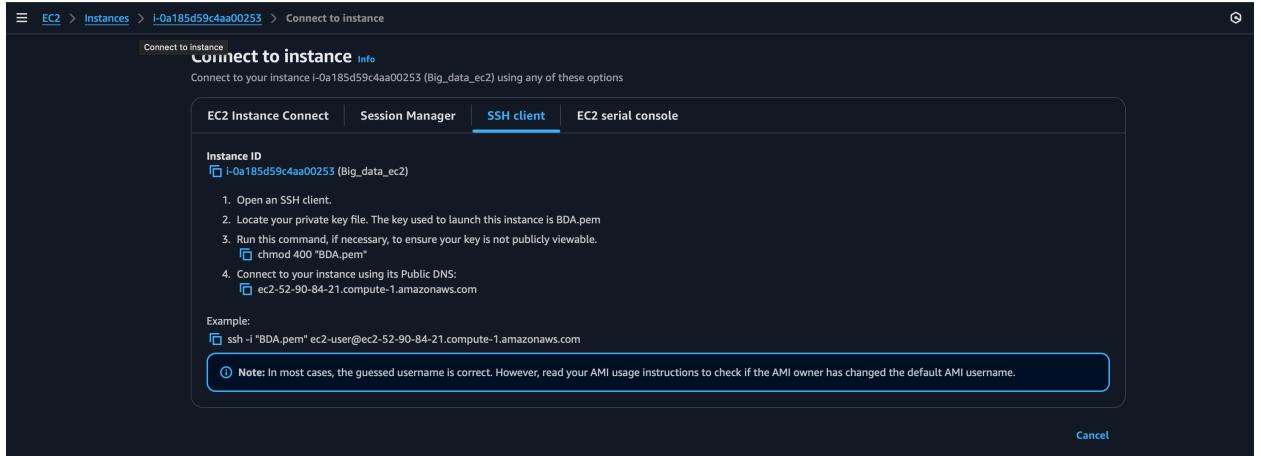
At the bottom right of the "Next Steps" section, there are navigation arrows for pages 1 through 6.

- Go to the folder where project is organized or pem file is located

The screenshot shows the VS Code terminal tab. The terminal window displays the following file list:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● (base) shoukath@Mac Mini Project % LS
BDA.pem          Report.docx    rawdata      ~$Report.docx
○ (base) shoukath@Mac Mini Project %
```

- Connect to EC2 instance on VS code – (Connect to Host method)



By ssh cmd -

```
(base) shoukath@Mac Mini Project % ssh -i "BDA.pem" ec2-user@ec2-52-90-84-21.compute-1.amazonaws.com
The authenticity of host 'ec2-52-90-84-21.compute-1.amazonaws.com (52.90.84.21)' can't be established.
ED25519 key fingerprint is SHA256:zv8BY1gSCJpoxQWP6wvST9UuxMFxiC+jxQ+/dk+Lofo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-52-90-84-21.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

      #_
  ~\_ #####_          Amazon Linux 2
  ~~ \#####\
  ~~  \###|          AL2 End of Life is 2025-06-30.
  ~~   \#/__-
  ~~    \~'`-->
  ~~~   /          A newer version of Amazon Linux is available!
  ~~ .-`/`-`/
  _/m/`          Amazon Linux 2023, GA and supported until 2028-03-15.
                           https://aws.amazon.com/linux/amazon-linux-2023/
```

By following above steps – Connected Successfully

b. Step 2 – Pyspark Installation –

Established connection

```
(base) shoukath@Mac Mini Project % ssh -i "BDA.pem" ec2-user@ec2-52-90-84-21.compute-1.amazonaws.com
The authenticity of host 'ec2-52-90-84-21.compute-1.amazonaws.com (52.90.84.21)' can't be established.
ED25519 key fingerprint is SHA256:zv8BY1gSCJpoxQWP6wvST9UuxMFxiC+jxQ+/dk+Lofo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-52-90-84-21.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

      #_
  ~\_ #####_          Amazon Linux 2
  ~~ \#####\
  ~~  \###|          AL2 End of Life is 2025-06-30.
  ~~   \#/__-
  ~~    \~'`-->
  ~~~   /          A newer version of Amazon Linux is available!
  ~~ .-`/`-`/
  _/m/`          Amazon Linux 2023, GA and supported until 2028-03-15.
                           https://aws.amazon.com/linux/amazon-linux-2023/
```

Installing and updating required modules – (Java, pip3, jupyter)

```
[ec2-user@ip-172-31-86-248 ~]$ sudo yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
[ec2-user@ip-172-31-86-248 ~]$ sudo yum install python3 python3-pip -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Package python3-3.7.16-1.amzn2.0.8.x86_64 already installed and latest version
Package python3-pip-20.2.2-1.amzn2.0.7.noarch already installed and latest version
Nothing to do
[ec2-user@ip-172-31-86-248 ~]$ sudo yum install java-1.8.0-openjdk -y
```

```
Complete!
[ec2-user@ip-172-31-86-248 ~]$ wget https://downloads.lightbend.com/scala/2.13.6/scala-2.13.6.rpm
--2024-12-07 03:18:43-- https://downloads.lightbend.com/scala/2.13.6/scala-2.13.6.rpm
Resolving downloads.lightbend.com (downloads.lightbend.com)... 3.167.56.60, 3.167.56.66, 3.167.56.52, ...
Connecting to downloads.lightbend.com (downloads.lightbend.com)|3.167.56.60|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 138124551 (132M) [application/octet-stream]
Saving to: 'scala-2.13.6.rpm'

100%[=====] 138,124,551 66.9MB/s   in 2.0s
2024-12-07 03:18:46 (66.9 MB/s) - 'scala-2.13.6.rpm' saved [138124551/138124551]
```

Installing Jupyter –

```
[ec2-user@ip-172-31-86-248 ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ec2-user/.local/bin:/home/ec2-user/bin:/home/ec2-user/.local/bin
[ec2-user@ip-172-31-86-248 ~]$ find ~/.local -name jupyter
[ec2-user@ip-172-31-86-248 ~]$ 
[ec2-user@ip-172-31-86-248 ~]$ pip3 install jupyter --user
Collecting jupyter
  Downloading jupyter-1.1.1-py2.py3-none-any.whl (2.7 kB)
```

Installing pyspark –

```
[ec2-user@ip-172-31-86-248 ~]$ pip3 install pyspark --user
Collecting pyspark
  Downloading pyspark-3.4.4.tar.gz (311.4 MB)
    |██████████| 311.4 MB 24.7 MB/s eta 0:00:01Killed
```

```
[ec2-user@ip-172-31-86-248 ~]$ pip3 install pyspark --user --no-cache-dir
Collecting pyspark
  Downloading pyspark-3.4.4.tar.gz (311.4 MB)
    |██████████| 311.4 MB 82.4 MB/s
Collecting py4j==0.10.9.7
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
    |██████████| 200 kB 61.2 MB/s
Using legacy 'setup.py install' for pyspark, since package 'wheel' is not installed.
Installing collected packages: py4j, pyspark
  Running setup.py install for pyspark ... done
Successfully installed py4j-0.10.9.7 pyspark-3.4.4
[ec2-user@ip-172-31-86-248 ~]$
```

Testing –

```
[ec2-user@ip-172-31-86-248 ~]$ pyspark
Python 3.7.16 (default, Oct 30 2024, 20:44:12)
[GCC 7.3.1 20180712 (Red Hat 7.3.1-17)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/07 03:25:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
/home/ec2-user/.local/lib/python3.7/site-packages/pyspark/context.py:317: FutureWarning: Python 3.7 support is deprecated in Spark 3.4.
warnings.warn("Python 3.7 support is deprecated in Spark 3.4.", FutureWarning)
Welcome to
    _____
   / \   \
  /   \  /
 /     \/
 \     /
  \   /
   \ /
    \_/
    / \
   /   \
  /     \
 /       \
 \     /
  \   /
   \_/
version 3.4.4

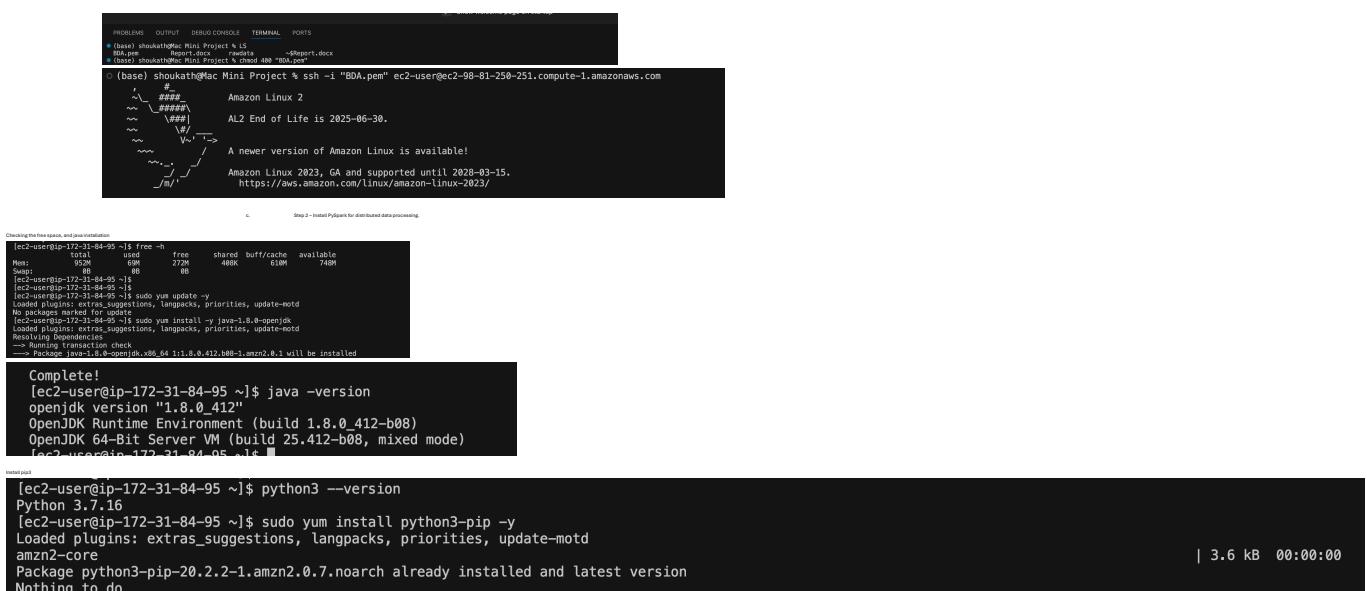
Using Python version 3.7.16 (default, Oct 30 2024 20:44:12)
Spark context Web UI available at http://ip-172-31-86-248.ec2.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1733541936222).
SparkSession available as 'spark'.
>>> 
```

Successfully installed pyspark using this approach!!

----- Challenges -----

- The installation gets killed, to solve this introduced a swap memory method, but failed! (Version doesn't match)
- Approached by manual installation of pyspark with suitable python version, again failed to detect the Java.
- Later to solve this problem, installed using `pip3 install pyspark --user --no-cache-dir` cmd, instead of [pip3 install pyspark]

(below are the screenshots of the challenges)



```

-- Challenge --
Now install the pypark
Loaded plugins: extras suggestions, langpacks, priorities, update-motd
Package python3-pip-20.2.2-1.amzn2.0.7.noarch already installed and latest version
Nothing to do
[ec2-user@ip-172-31-85-70 ~]$ pip3 install pypark
Defaulting to user installation because normal site-packages is not writeable
Collecting pypark
  Downloading pypark-3.4.4.tar.gz (311.4 MB)
[ec2-user@ip-172-31-85-70 ~]$ 

Piping generated code...
Checking the file size...
[ec2-user@ip-172-31-85-70 ~]$ pip3 install pypark
Defaulting to user installation because normal site-packages is not writeable
Collecting pypark
  Downloading pypark-3.4.4.tar.gz (311.4 MB)
[ec2-user@ip-172-31-85-70 ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        468M   0    468M  0% /dev
tmpfs          477M   0    477M  0% /dev/shm
tmpfs          477M  468K  476M  1% /run
tmpfs          477M   0    477M  0% /sys/fs/cgroup
/dev/xvda1     8.8G  2.8G  5.3G  33% /run/user/1000
tmpfs          96M   0    96M  0% /run/user/0
tmpfs          96M   0    96M  0% /run/user/0
[ec2-user@ip-172-31-85-70 ~]$ 
List of options available...
Checking the file size...
[ec2-user@ip-172-31-85-70 project]$ pip3 install pypark
Defaulting to user installation because normal site-packages is not writeable
Collecting pypark
  Downloading pypark-3.4.4.tar.gz (311.4 MB)
[ec2-user@ip-172-31-85-70 project]$ free -h
total       used       free     shared  buff/cache   available
Mem:      952M       62M     860M       468K      29M      895M
Swap:      0B       0B       0B
[ec2-user@ip-172-31-85-70 project]$ 
Warning: temporary path '/tmp' is not implemented yet. It will be removed in the next release.
While creating swap file of size 2GB, no implemented smaller block size of 2GB with increase in count...
[ec2-user@ip-172-31-85-70 project]$ sudo dd if=/dev/zero of=/swapfile bs=1G count=2
dd: memory exhausted by input buffer of size 1073741824 bytes (1.0 GiB)
[ec2-user@ip-172-31-85-70 project]$ sudo dd if=/dev/zero of=/swapfile bs=1M count=2048
2048+0 records in
2048+0 records out
2147483648 bytes (2.1 GB) copied, 14.0191 s, 153 MB/s
[ec2-user@ip-172-31-85-70 project]$ sudo swapon /swapfile
2147483648 bytes (2.1 GB) copied, 14.0191 s, 153 MB/s
[ec2-user@ip-172-31-85-70 project]$ sudo chmod 600 /swapfile
[ec2-user@ip-172-31-85-70 project]$ sudo mkswap /swapfile
Setting up swapspace version 1, size = 2 GiB (2147479552 bytes)
no label, UUID=0008ad0-f06c-4841-9e07-ebf8d1333259
[ec2-user@ip-172-31-85-70 project]$ sudo swapon /swapfile
[ec2-user@ip-172-31-85-70 project]$ swapon --show
NAME      TYPE SIZE USED PRIQ
/swapfile file  2G  0B   -2
[ec2-user@ip-172-31-85-70 project]$ free -h
total       used       free     shared  buff/cache   available
Mem:      952M       65M     69M       468K      817M      752M
Swap:      2.0G       0B      2.0G
[ec2-user@ip-172-31-85-70 project]$ 
Verifying the swap space...
Mounting swap space...
[ec2-user@ip-172-31-85-70 project]$ echo '/swapfile swap swap defaults 0 0' | sudo tee -a /etc/fstab
/swapfile swap swap defaults 0 0
Now finally installed pypark
[ec2-user@ip-172-31-85-70 project]$ pip3 install pypark
Defaulting to user installation because normal site-packages is not writeable
Collecting pypark
  Downloading pypark-3.4.4.tar.gz (311.4 MB)
[ec2-user@ip-172-31-85-70 ~]$ 
Collecting py4j==0.10.9.7
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
[ec2-user@ip-172-31-85-70 ~]$ Using legacy 'setup.py install' for pypark, since package 'wheel' is not installed.
Installing collected packages: py4j, pypark
  Running setup.py install for pypark ... done
Successfully installed py4j-0.10.9.7 pypark-3.4.4
[ec2-user@ip-172-31-85-70 project]$ python3 -c "import pypark; print(pypark.__version__)"
3.4.4
[ec2-user@ip-172-31-85-70 project]$ 
Now setting up Environment variables...
[ec2-user@ip-172-31-85-70 project]$ nano ~/.bashrc
GNU nano 2.9.8                               /home/ec2-user
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
  . /etc/bashrc

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
export SPARK_HOME=/usr/local/spark
export PATH=$SPARK_HOME/bin:$PATH
export PYSPARK_PYTHON=python3
export PYSPARK_DRIVER_PYTHON=python3

[ec2-user@ip-172-31-85-70 project]$ source ~/.bashrc
[ec2-user@ip-172-31-85-70 project]$ 
Temporary file '/tmp/tmp-172-31-85-70-project-1' deleted.
[ec2-user@ip-172-31-85-70 project]$ 
[ec2-user@ip-172-31-85-70 project]$ ls -la /etc/fstab
total 0
[ec2-user@ip-172-31-85-70 project]$ ls -la /etc/swapfile
ls: /etc/swapfile: No such file or directory
[ec2-user@ip-172-31-85-70 project]$ 
[ec2-user@ip-172-31-85-70 project]$ swapoff /swapfile
[ec2-user@ip-172-31-85-70 project]$ sudo nano /etc/fstab
[ec2-user@ip-172-31-85-70 project]$ sudo rm /swapfile
[ec2-user@ip-172-31-85-70 project]$ swapon --show

```

```

[ec2-user@ip-172-31-85-78 project]$ free -h
total used     free   shared  buff/cache available
Mem:      952M     68M    489M     468K    394M    750M
Swap:        0B       0B       0B       0B       0B       0B

Now creating payload - (Created a new instance and followed the steps above except challenge block)
[ec2-user@ip-172-31-84-95 ~]$ wget https://www.apache.org/dist/spark/3.4.4/spark-3.4.4-bin-hadoop3.tgz
--2024-12-06 22:31:53.189... 100% [=====
Resolving www.apache.org [www.apache.org]... 3.6.254.131, 3.6.254.132, 3.6.254.133, ...
Connecting to www.apache.org (www.apache.org)|3.6.254.131:443| (夭)...
HTTP request sent, awaiting response... 200 OK (http://www.apache.org)
Length: 388,988,563 (371MiB) [application/x-gzip]
Saving to: 'spark-3.4.4-bin-hadoop3.tgz'

100%[=====] 388,988,563 14.79GB/s  in 3.6s

[ec2-user@ip-172-31-84-95 ~]$ ls
spark-3.4.4-bin-hadoop3  spark-3.4.4-bin-hadoop3.tgz

[ec2-user@ip-172-31-84-95 ~]$ sudo mv spark-3.4.4-bin-hadoop3 /opt/spark

Set ENV VARS
[ec2-user@ip-172-31-84-95 ~]$ echo "export SPARK_HOME=/opt/spark" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ echo "export PATH=\$PATH:$SPARK_HOME/bin" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ echo "export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ echo "export PATH=\$PATH:$JAVA_HOME/bin" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ source ~/.bashrc

Install packages
[ec2-user@ip-172-31-84-95 ~]$ wget https://dldcdn.apache.org/spark/spark-3.4.4/spark-3.4.4-bin-hadoop3.tgz
--2024-12-06 22:31:53.189... 100% [=====
Resolving dldcdn.apache.org [dldcdn.apache.org]... 151.101.2.132, 2a04:46d2:644
Connecting to dldcdn.apache.org (dldcdn.apache.org)|151.101.2.132|:443...
HTTP request sent, awaiting response... 200 OK
Length: 388,988,563 (371MiB) [application/x-gzip]
Saving to: 'spark-3.4.4-bin-hadoop3.tgz'

100%[=====] 388,988,563 100MB/s  in 3.6s

2024-12-06 22:32:11 104 MB/s) - 'spark-3.4.4-bin-hadoop3.tgz' saved [388988563/388988563]

Mounting spark-3.4.4-bin-hadoop3.tgz
[ec2-user@ip-172-31-84-95 ~]$ tar -xvf spark-3.4.4-bin-hadoop3.tgz --strip-components=1
[ec2-user@ip-172-31-84-95 ~]$ cd ./spark-3.4.4
[ec2-user@ip-172-31-84-95 ~]$ echo "export SPARK_HOME=/opt/spark" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ echo "export PATH=\$SPARK_HOME/bin" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ echo "export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ echo "export PATH=\$PATH:$JAVA_HOME/bin" >> ~/.bashrc
[ec2-user@ip-172-31-84-95 ~]$ source ~/.bashrc

New python
[ec2-user@ip-172-31-84-95 ~]$ pip3 install pyspark==3.5.3.tar.gz
Requirement to user installation because normal site-packages is not writeable
Processing /tmp/pip-req-build-5zj-qq-20/pip3-3.5.3.tar.gz
ERROR: Package 'pyspark==3.5.3.tar.gz' required a different Python 3.7.10 not in '>=3.8'

Downloaded from the S3 bucket
[ec2-user@ip-172-31-84-95 ~]$ rm pyspark-3.5.3.tar.gz
[ec2-user@ip-172-31-84-95 ~]$ wget https://files.pythonhosted.org/packages/2c/0a/4103a64a5e23a36946463213a8090f7c32a310946480327512084656272706074297407950/pyspark-3.4.4.tar.gz
--2024-12-06 22:32:27.000... 100% [=====
Resolving files.pythonhosted.org [files.pythonhosted.org]... 146.75.32.223, 2a04:46d2:78::273
Connecting to files.pythonhosted.org (files.pythonhosted.org)|146.75.32.223|:443...
HTTP request sent, awaiting response... 200 OK
Length: 311,480,584 (290MB) [application/x-tar]
Saving to: 'pyspark-3.4.4.tar.gz'

100%[=====] 311,480,584 100MB/s  in 2.8s

2024-12-06 22:32:30.000... 100% [=====
[ec2-user@ip-172-31-84-95 ~]$ pip3 install pyspark==3.4.4
Defaulting to user installation because normal site-packages is not writeable
Processing /tmp/pip-req-build-5zj-qq-20/pip3-3.4.4.tar.gz
Collecting pyspark==3.4.4
  Downloading pyspark-3.4.4.tar.gz (290 MB)
    ERROR: Could not find a valid wheel for wheel.
    Installing collected packages: pyspark
      Found existing wheel for wheel at /tmp/pip-req-build-5zj-qq-20/pip3-3.4.4/pyspark-3.4.4.dist-info/wheel.
      Successfully installed pyspark==3.4.4
[ec2-user@ip-172-31-84-95 ~]$ pip3 install pyspark
[ec2-user@ip-172-31-84-95 ~]$ nano /opt/spark/conf/spark-env.sh
[ec2-user@ip-172-31-84-95 ~]$ export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
[ec2-user@ip-172-31-84-95 ~]$ exit
[ec2-user@ip-172-31-84-95 ~]$
```

c. Step 3 –

AWS CLI INSTALLATION AND CONFIGURATION - By default Ec2 instances have aws cli installed, all we need to do is configure as shown below, keep your Aws Access key ID, AWS Secret Access key, default region, default output format(optional, by default Json) ready.

```
[ec2-user@ip-172-31-86-248 ~]$ aws --version
aws-cli/1.18.147 Python/2.7.18 Linux/5.10.228-219.884.amzn2.x86_64 botocore/1.18.6
[ec2-user@ip-172-31-86-248 ~]$ aws configure
AWS Access Key ID [None]: AKIA2RP6IB772QZCCHG2
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-172-31-86-248 ~]$ █
```

CHECK – By running simple commands like ls and credentials inside the files

```
[ec2-user@ip-172-31-86-248 ~]$ cd .aws
[ec2-user@ip-172-31-86-248 .aws]$ ls
ls: no input files
[ec2-user@ip-172-31-86-248 .aws]$ ls config credentials
[ec2-user@ip-172-31-86-248 .aws]$ cat credentials
[default]
aws_access_key_id = AKIA2RP6IB772QZCCHG2
aws_secret_access_key = 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
[ec2-user@ip-172-31-86-248 .aws]$ █
```

SET ENV VAR –

```
[ec2-user@ip-172-31-86-248 .aws]$ export AWS_ACCESS_KEY_ID=AKIA2RP6IB772QZCCHG2
[ec2-user@ip-172-31-86-248 .aws]$ cd
[ec2-user@ip-172-31-86-248 ~]$ export AWS_SECRET_ACCESS_KEY=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
[ec2-user@ip-172-31-86-248 ~]$ printenv | grep AWS
AWS_SECRET_ACCESS_KEY=fJzXsou3orFJ6X1n40kqwLW3lgNvm05pRVKJlKJg
AWS_ACCESS_KEY_ID=AKIA2RP6IB772QZCCHG2
```

Now connect to S3 - using aws s3 commands as shown below.

```
[ec2-user@ip-172-31-86-248 ~]$ aws s3 ls
2024-12-07 16:14:26 customer-storage-bucket
[ec2-user@ip-172-31-86-248 ~]$ aws s3 ls s3://customer-storage-bucket
      PRE processed_data/
      PRE raw_data/
[ec2-user@ip-172-31-86-248 ~]$ aws s3 ls s3://customer-storage-bucket/raw_data/
2024-12-07 16:15:55      0
2024-12-07 16:19:44  23715344 Online_Retail.xlsx
[ec2-user@ip-172-31-86-248 ~]$ █
```

Successfully interacted with s3 and can retrieve files

3) Data Pipeline Tasks –

Firstly, I was thinking of setting up jupyter lab for pyspark environment
(for better readability and flexibility)

Go to directory or pass the pem file to initiate ec2 instance at local host and run the jupyter lab command -

```
ssh -L 8888:localhost:8888 -i BDA.pem ec2-user@ec2-44-211-155-23.compute-1.amazonaws.com
```

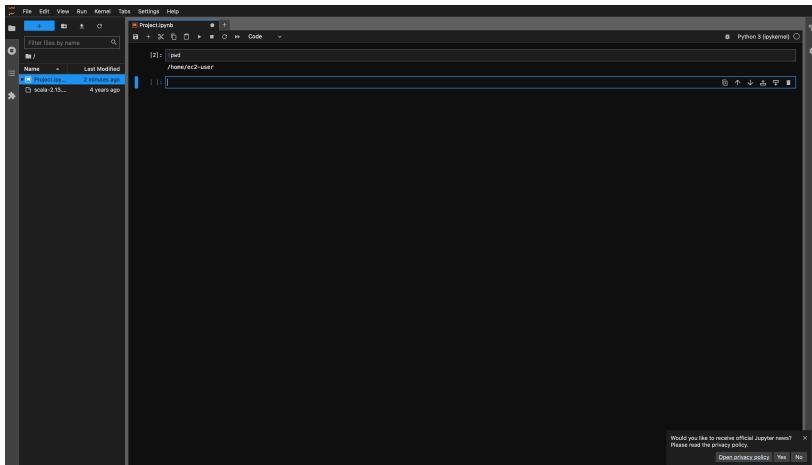
As shown below -

```
● (base) shoukath@Mac Mini Project % ls -al
total 3104
drwxr-xr-x  6 shoukath  staff      192 Dec  6 15:34 .
drwxr-x---+ 71 shoukath  staff     2272 Dec  6 15:38 ..
-rw-r--r--@  1 shoukath  staff     1678 Dec  6 09:09 BDA.pem
-rw-r--r--@  1 shoukath  staff   1580907 Dec  6 10:13 Report.docx
drwxr-xr-x  3 shoukath  staff      96 Dec  6 09:22 rawdata
-rw-r--r--@  1 shoukath  staff     162 Nov 30 12:40 ~$Report.docx
● (base) shoukath@Mac Mini Project % chmod 400 BDA.pem
● (base) shoukath@Mac Mini Project % ls -al
total 3104
drwxr-xr-x  6 shoukath  staff      192 Dec  6 15:34 .
drwxr-x---+ 71 shoukath  staff     2272 Dec  6 15:38 ..
-r-----@  1 shoukath  staff     1678 Dec  6 09:09 BDA.pem
-rw-r--r--@  1 shoukath  staff   1580907 Dec  6 10:13 Report.docx
drwxr-xr-x  3 shoukath  staff      96 Dec  6 09:22 rawdata
-rw-r--r--@  1 shoukath  staff     162 Nov 30 12:40 ~$Report.docx
○ (base) shoukath@Mac Mini Project % ssh -L 8888:localhost:8888 -i BDA.pem ec2-user@ec2-44-211-155-23.compute-1.amazonaws.com
Last login: Sat Dec  7 16:08:44 2024 from 129.79.197.51
,
  #_
  \_ #####
  ~~ \_\#\#\#\#
  ~~ \#\#|
  ~~  \#/ 
  ~~   \~\__>
  ~~    /
  ~~  _.
  ~~ /_/
  _m/'  Amazon Linux 2
  ~~ \#\#\#\#
  ~~ \#\#\#|      AL2 End of Life is 2025-06-30.
  ~~  \#/ 
  ~~   \~\__>
  ~~    /
  ~~  _.
  ~~ /_/
  _m/'  A newer version of Amazon Linux is available!
  ~~ \#\#\#\#
  ~~ \#\#\#|      Amazon Linux 2023, GA and supported until 2028-03-15.
  ~~  \#/ 
  ~~   \~\__>
  ~~    /
  ~~  _.
  ~~ /_/
  _m/'  https://aws.amazon.com/linux/amazon-linux-2023/
```

```
[ec2-user@ip-172-31-86-248 ~]$ ~/.local/bin/jupyter lab --ip=0.0.0.0 --port=8888 --no-browser --allow-root
```

Check –

Successfully running at localhost (at specified port in terminal)

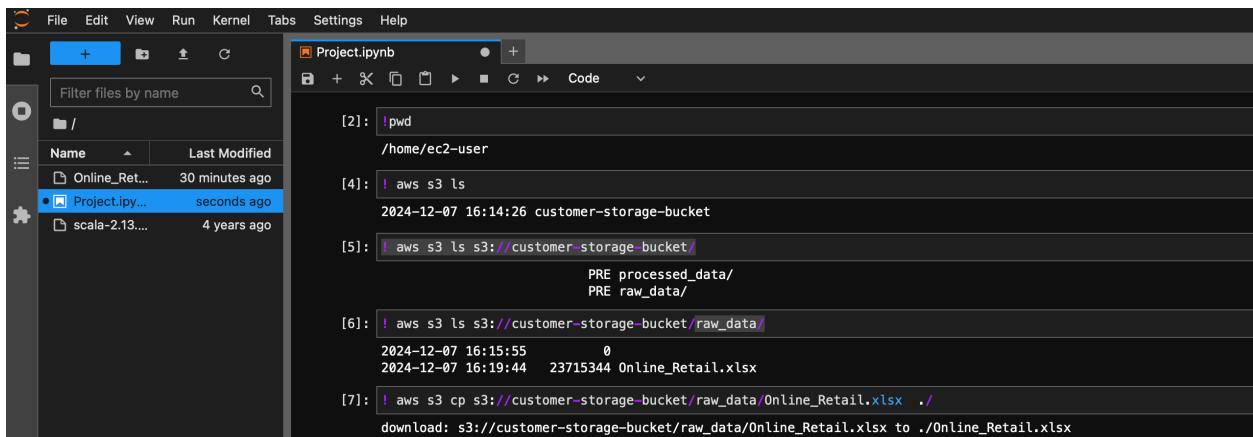


```
[2]: ls  
/home/ec2-user
```

Let's start the actual tasks

1. Task 1 - Data Ingestion from S3 –

- Step 1 – Use AWS CLI or PySpark's built-in S3 support to load the dataset directly



```
[2]: !pwd  
/home/ec2-user  
[4]: ! aws s3 ls  
2024-12-07 16:14:26 customer-storage-bucket  
[5]: ! aws s3 ls s3://customer-storage-bucket/  
    PRE processed_data/  
    PRE raw_data/  
[6]: ! aws s3 ls s3://customer-storage-bucket/raw_data/  
2024-12-07 16:15:55      0  
2024-12-07 16:19:44  23715344 Online_Retail.xlsx  
[7]: ! aws s3 cp s3://customer-storage-bucket/raw_data/Online_Retail.xlsx ./  
download: s3://customer-storage-bucket/raw_data/Online_Retail.xlsx to ./Online_Retail.xlsx
```

- Step 2 –

The screenshot shows a Jupyter Notebook interface. On the left is a file browser with a search bar and a list of files: Online_Ret... (30 minutes ago), Project.ipynb (seconds ago, selected), and scala-2.13.... (4 years ago). The main area is a terminal window with the following command history:

```
[2]: !pwd  
/home/ec2-user  
[4]: ! aws s3 ls  
2024-12-07 16:14:26 customer-storage-bucket  
[5]: ! aws s3 ls s3://customer-storage-bucket/  
    PRE processed_data/  
    PRE raw_data/  
[6]: ! aws s3 ls s3://customer-storage-bucket/raw_data/  
2024-12-07 16:15:55      0  
2024-12-07 16:19:44  23715344 Online_Retail.xlsx  
[7]: ! aws s3 cp s3://customer-storage-bucket/raw_data/Online_Retail.xlsx ./  
download: s3://customer-storage-bucket/raw_data/Online_Retail.xlsx to ./Online_Retail.xlsx  
[8]: !ls  
Online_Retail.xlsx  Project.ipynb  scala-2.13.6.rpm  
[ ]:
```

2. Task 2: Data Processing with PySpark -

Check pyspark -

The screenshot shows a Jupyter Notebook cell with the following code and output:

```
[9]: from pyspark.sql import SparkSession  
  
# Create Spark session  
spark = SparkSession.builder \  
    .appName("MiniProject_Retail") \  
    .getOrCreate()  
  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
24/12/07 16:57:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
/home/ec2-user/.local/lib/python3.7/site-packages/pyspark/context.py:317: FutureWarning: Python 3.7 support is deprecated in Spark 3.4.  
warnings.warn("Python 3.7 support is deprecated in Spark 3.4.", FutureWarning)  
[10]: spark  
[10]: SparkSession - in-memory  
SparkContext  
Spark UI  
Version          v3.4.4  
Master           local[*]  
AppName         MiniProject_Retail
```

Cleaned data and verified schema before transformations -

```

[21]: print(f"Total Records: {df.count()}")
Total Records: 541909

[22]: ### Checking null values if any (as per source there are no null values)
from pyspark.sql.functions import col, sum

# Check if any column has null values
df.select([sum(col(c).isNull().cast("int")).alias(c) for c in df.columns]).show()

[Stage 9:> (0 + 1) / 1]
+|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---+-----+-----+-----+-----+-----+-----+-----+
| 0 | 0 | 1454 | 0 | 0 | 0 | 135080 | 0 |
+---+-----+-----+-----+-----+-----+-----+-----+


[23]: ### droped null values
df = df.dropna(subset=["CustomerID", "Description"])

[24]: df.select([sum(col(c).isNull().cast("int")).alias(c) for c in df.columns]).show()

[Stage 12:> (0 + 1) / 1]
+|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---+-----+-----+-----+-----+-----+-----+-----+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+-----+-----+-----+-----+-----+-----+-----+


[25]: print(f"Total Records: {df.count()}")
[Stage 15:> (0 + 1) / 1]
Total Records: 406829

```

a. Data Transformation:

Transformed a column – InvoiceDate into a timestamp format.

Created a new column TotalPrice, from existing columns Quantity x UnitPrice. This helps to reduce the complexity of computing revenue at the later stage.

For our analytics, Implemented a new column called, InvoiceYearMonth, where it extracts the year and month from the existing column.

All the transformations performed are in the provided screenshot.

1. Data Transformations

```
[29]: from pyspark.sql.functions import col, to_timestamp, year, month, concat_ws
# Convert InvoiceDate to timestamp type
df = df.withColumn("InvoiceDate", to_timestamp(col("InvoiceDate")))

[27]: # Add TotalPrice column
df = df.withColumn("TotalPrice", col("Quantity") * col("UnitPrice"))

[30]: # Add InvoiceYearMonth column
df = df.withColumn("InvoiceYearMonth", concat_ws("-", year(col("InvoiceDate")), month(col("InvoiceDate"))))

[31]: df.show(10)
```

| InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | Country | TotalPrice | InvoiceYearMonth |
|-----------|-----------|-----------------------|----------|---------------------|-----------|------------|----------------|-------------------|------------------|
| 536365 | 85123A | WHITE HANGING HEA... | 6 | 2010-12-01 08:26:00 | 2.55 | 17850.0 | United Kingdom | 15.29999999999999 | 2010-12 |
| 536365 | 71053 | WHITE METAL LANTERN | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom | 20.34 | 2010-12 |
| 536365 | 84406B | CREAM CUPID HEART... | 8 | 2010-12-01 08:26:00 | 2.75 | 17850.0 | United Kingdom | 22.0 | 2010-12 |
| 536365 | 84029G | KNITTED UNION FLAG... | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom | 20.34 | 2010-12 |
| 536365 | 84029E | RED WOOLLY HOTTIE... | 6 | 2010-12-01 08:26:00 | 3.39 | 17850.0 | United Kingdom | 20.34 | 2010-12 |
| 536365 | 22752 | SET 7 BABUSHKA NE... | 2 | 2010-12-01 08:26:00 | 7.65 | 17850.0 | United Kingdom | 15.3 | 2010-12 |
| 536365 | 21730 | GLASS STAR FROSTED... | 6 | 2010-12-01 08:26:00 | 4.25 | 17850.0 | United Kingdom | 25.5 | 2010-12 |
| 536366 | 22633 | HAND WARMER UNION... | 6 | 2010-12-01 08:28:00 | 1.85 | 17850.0 | United Kingdom | 11.10000000000001 | 2010-12 |
| 536366 | 22632 | HAND WARMER RED P... | 6 | 2010-12-01 08:28:00 | 1.85 | 17850.0 | United Kingdom | 11.10000000000001 | 2010-12 |
| 536367 | 84879 | ASSORTED COLOUR B... | 32 | 2010-12-01 08:34:00 | 1.69 | 13047.0 | United Kingdom | 54.08 | 2010-12 |

only showing top 10 rows

b. Data Aggregation:

All details are in notebook (attached for your reference)

Found the list of countries data we have and calculated the total revenue collected by each country.

2. Data Aggregation

```
[40]: df.select("Country").distinct().show()
```

| Country |
|--------------------|
| Sweden |
| Singapore |
| Germany |
| RSA |
| France |
| Greece |
| European Community |
| Belgium |
| Finland |
| Malta |
| Unspecified |
| Italy |
| EIRE |
| Lithuania |
| Norway |
| Spain |
| Denmark |
| Iceland |
| Israel |
| Channel Islands |

only showing top 20 rows


```
[42]: from pyspark.sql import functions as F
# Compute Total Revenue by Country
df = df.withColumn("Revenue", F.col("Quantity") * F.col("UnitPrice")) # Create a column for revenue

# Total Revenue by Country
totalRevenueByCountry = df.groupby("Country").agg(F.sum("Revenue").alias("Total_Revenue")).orderBy(F.desc("Total_Revenue"))

totalRevenueByCountry.show(10)
```

| Country | Total_Revenue |
|----------------|--------------------|
| United Kingdom | 6767873.394002574 |
| Netherlands | 284661.54000000015 |
| EIRE | 250285.21999999872 |
| Germany | 221698.20999999862 |
| France | 196712.8399999999 |
| Australia | 137077.2699999973 |
| Switzerland | 55739.4000000004 |
| Spain | 54774.5799999997 |
| Belgium | 40910.9599999998 |

Would you like to receive official Jupyter news?
Please read the privacy policy.
[Open privacy policy](#)

Highest customer transactions made and amount of quantity sold / ordered per country

```
[43]: customerTransactionValue = df.groupBy("CustomerID").agg(  
    F.sum("Revenue").alias("Total_Transaction_Value"))  
.orderBy(F.desc("Total_Transaction_Value"))  
  
customerTransactionValue.show(10)
```

```
[Stage 25:> (0 + 1) / 1]  
+-----+  
|CustomerID|Total_Transaction_Value|  
+-----+  
| 14646.0 | 279489.0199999999 |  
| 18102.0 | 256438.49000000005 |  
| 17450.0 | 187482.17000000013 |  
| 14911.0 | 132572.619999998 |  
| 12415.0 | 123725.4499999987 |  
| 14156.0 | 113384.1399999985 |  
| 17511.0 | 88125.3799999996 |  
| 16684.0 | 65892.0799999999 |  
| 13694.0 | 62653.1000000003 |  
| 15311.0 | 59419.3400000011 |  
+-----+  
only showing top 10 rows
```

```
[44]: # Compute Total Quantity Sold by Region**  
totalQuantityCountry = df.groupBy("Country").agg(  
    F.sum("Quantity").alias("Total_Quantity_Sold"))  
.orderBy(F.desc("Total_Quantity_Sold"))  
  
totalQuantityCountry.show(10)
```

```
[Stage 28:> (0 + 1) / 1]  
+-----+  
| Country|Total_Quantity_Sold|  
+-----+  
|United Kingdom| 4008533 |  
| Netherlands| 200128 |  
| EIRE| 136329 |  
| Germany| 117448 |  
| France| 109848 |  
| Australia| 83653 |  
| Sweden| 35637 |  
| Switzerland| 29778 |  
| Spain| 26824 |  
| Japan| 25218 |  
+-----+  
only showing top 10 rows
```

Aggregated the data for monthly spending trend from the transformed column invoice year month and the revenue.

```
[45]: # Monthly Spending Trends**
monthly_spending_trends = df.groupBy("InvoiceYearMonth").agg(
    F.sum("Revenue").alias("Total_Revenue")
).orderBy("InvoiceYearMonth")

monthly_spending_trends.show(12)
```

[Stage 31:> (0 + 1) / 1]

| InvoiceYearMonth | Total_Revenue |
|------------------|--------------------|
| 2010-12 | 554604.020000018 |
| 2011-1 | 475074.38000001636 |
| 2011-10 | 974603.5899999909 |
| 2011-11 | 1132407.7399999578 |
| 2011-12 | 342506.3800000034 |
| 2011-2 | 436546.1500000147 |
| 2011-3 | 579964.6100000151 |
| 2011-4 | 426047.8510000125 |
| 2011-5 | 648251.080000003 |
| 2011-6 | 608013.1600000106 |
| 2011-7 | 574238.481000012 |
| 2011-8 | 616368.0000000092 |

only showing top 12 rows

```
[46]: # Compute Average Transaction Value per Customer
averageTransactionPerCustomer = df.groupBy("CustomerID").agg(
    F.avg("Revenue").alias("Average_Transaction_Value")
).orderBy(F.desc("Average_Transaction_Value"))

averageTransactionPerCustomer.show(10)
```

[Stage 34:> (0 + 1) / 1]

| CustomerID | Average_Transaction_Value |
|------------|---------------------------|
| 15195.0 | 3861.0 |
| 13135.0 | 3096.0 |
| 17846.0 | 2033.1 |
| 16532.0 | 1687.2 |
| 15749.0 | 1435.7266666666667 |
| 16000.0 | 1377.0777777777778 |
| 16754.0 | 1001.2 |
| 12798.0 | 872.129999999999 |
| 17553.0 | 743.8 |
| 17949.0 | 667.7321518987343 |

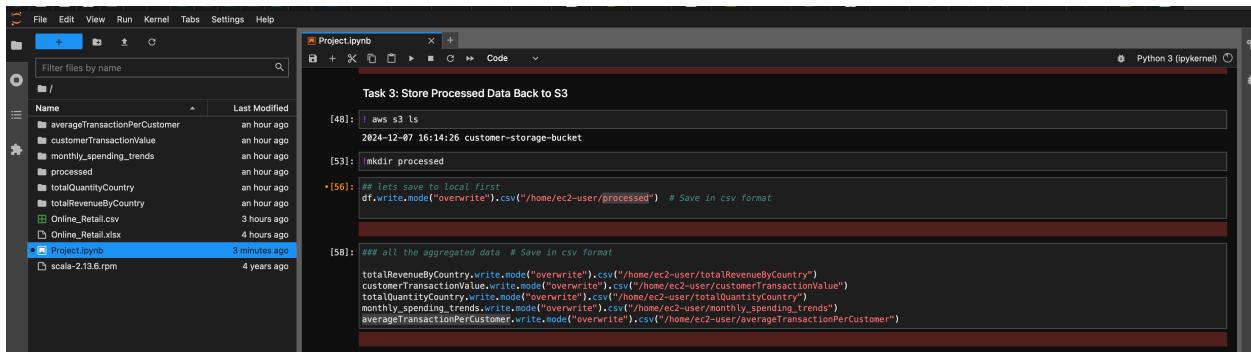
only showing top 10 rows

Displayed average transactional value made by each customer.

3. Task 3: Store Processed Data Back to S3 –

a. Step 1 – Export data in CSV or Parquet format.

Exported all the data in csv format, as we know that spark stores or exports the data in directory format and not the files, we need to provide the path for empty output directories. (As shown below)



The screenshot shows a Jupyter Notebook environment. On the left, there is a file browser window titled 'File' showing local files like 'averageTransactionPerCustomer', 'customerTransactionValue', 'monthly_spending_trends', 'processed', 'totalQuantityCountry', 'totalRevenueByCountry', 'Online_Retail.csv', 'Online_Retail.xlsx', and 'Project.ipynb'. The main area is a code editor with the title 'Task 3: Store Processed Data Back to S3'. The code is as follows:

```
[48]: ! aws s3 ls  
2024-12-07 16:14:26 customer-storage-bucket  
[53]: mkdir processed  
*(56): ## lets save to local first  
df.write.mode("overwrite").csv("/home/ec2-user/processed") # Save in csv format  
  
[58]: ## all the aggregated data # Save in csv format  
totalRevenueByCountry.write.mode("overwrite").csv("/home/ec2-user/totalRevenueByCountry")  
customerTransactionValue.write.mode("overwrite").csv("/home/ec2-user/customerTransactionValue")  
totalQuantityCountry.write.mode("overwrite").csv("/home/ec2-user/totalQuantityCountry")  
monthly_spending_trends.write.mode("overwrite").csv("/home/ec2-user/monthly_spending_trends")  
averageTransactionPerCustomer.write.mode("overwrite").csv("/home/ec2-user/averageTransactionPerCustomer")
```

b. Step 2 – Upload the processed data to a designated S3 location for easy access

Now to export data to S3 we can perform with the help of AWS CLI capabilities, as we know that CLI is readily available in EC2.

```
[2]: ### Now lets copy all of them to S3 using AWS CLI
aws s3 ls
2024-12-07 16:14:26 customer-storage-bucket

[6]: ! aws s3 cp /home/ec2-user/processed/ s3://customer-storage-bucket/processed_data --recursive
upload: processed/_SUCCESS.crc to s3://customer-storage-bucket/processed_data/_SUCCESS
upload: processed/_SUCCESS to s3://customer-storage-bucket/processed_data/_SUCCESS
upload: processed/.part-00000-0cc0c2cf-8386-4f5a-98ae-c6609780d9e1-c000.csv.crc to s3://customer-storage-bucket/processed_data/.part-00000-0cc0c2cf-8386-4f5a-98ae-c6609780d9e1-c000.csv
upload: processed/part-00000-0cc0c2cf-8386-4f5a-98ae-c6609780d9e1-c000.csv to s3://customer-storage-bucket/processed_data/part-00000-0cc0c2cf-8386-4f5a-98ae-c6609780d9e1-c000.csv

[11]: # creating folders in aws s3
aws s3 cp /dev/null s3://customer-storage-bucket/totalRevenueByCountry
aws s3 cp /dev/null s3://customer-storage-bucket/customerTransactionValue
aws s3 cp /dev/null s3://customer-storage-bucket/totalQuantityCountry
aws s3 cp /dev/null s3://customer-storage-bucket/monthly_spending_trends
aws s3 cp /dev/null s3://customer-storage-bucket/averageTransactionPerCustomer

warning: Skipping file /dev/null. File is character special device, block special device, FIFO, or socket.
warning: Skipping file /dev/null. File is character special device, block special device, FIFO, or socket.
warning: Skipping file /dev/null. File is character special device, block special device, FIFO, or socket.
warning: Skipping file /dev/null. File is character special device, block special device, FIFO, or socket.
warning: Skipping file /dev/null. File is character special device, block special device, FIFO, or socket.

[12]: #copying
! aws s3 cp /home/ec2-user/totalRevenueByCountry/ s3://customer-storage-bucket/totalRevenueByCountry --recursive
! aws s3 cp /home/ec2-user/customerTransactionValue/ s3://customer-storage-bucket/customerTransactionValue --recursive
! aws s3 cp /home/ec2-user/totalQuantityCountry/ s3://customer-storage-bucket/totalQuantityCountry --recursive
! aws s3 cp /home/ec2-user/monthly_spending_trends/ s3://customer-storage-bucket/monthly_spending_trends --recursive
! aws s3 cp /home/ec2-user/averageTransactionPerCustomer/ s3://customer-storage-bucket/averageTransactionPerCustomer --recursive

upload: totalRevenueByCountry/_SUCCESS.crc to s3://customer-storage-bucket/totalRevenueByCountry/_SUCCESS
upload: totalRevenueByCountry/.part-00000-612d133a-33af-487e-bb22-bb170cf78e68-c000.csv.crc to s3://customer-storage-bucket/totalRevenueByCountry/.part-00000-612d133a-33af-487e-bb22-bb170cf78e68-c000.csv
upload: totalRevenueByCountry/part-00000-612d133a-33af-487e-bb22-bb170cf78e68-c000.csv to s3://customer-storage-bucket/totalRevenueByCountry/part-00000-612d133a-33af-487e-bb22-bb170cf78e68-c000.csv
upload: totalRevenueByCountry/_SUCCESS to s3://customer-storage-bucket/totalRevenueByCountry/_SUCCESS
upload: customerTransactionValue/_SUCCESS.crc to s3://customer-storage-bucket/customerTransactionValue/_SUCCESS
upload: customerTransactionValue/_SUCCESS to s3://customer-storage-bucket/customerTransactionValue/_SUCCESS
upload: customerTransactionValue/.part-00000-30ea851b-8259-440e-b0c4-11ea1bdf76fd-c000.csv.crc to s3://customer-storage-bucket/customerTransactionValue/.part-00000-30ea851b-8259-440e-b0c4-11ea1bdf76fd-c000.csv
upload: customerTransactionValue/part-00000-30ea851b-8259-440e-b0c4-11ea1bdf76fd-c000.csv to s3://customer-storage-bucket/customerTransactionValue/part-00000-30ea851b-8259-440e-b0c4-11ea1bdf76fd-c000.csv
upload: customerTransactionValue/_SUCCESS to s3://customer-storage-bucket/customerTransactionValue/_SUCCESS
```

Successfully Uploaded – (Verification in aws S3)

The screenshot shows the AWS S3 console interface. The left sidebar lists various AWS services like General purpose buckets, Storage Lens, and IAM Access Analyzer. The main area shows the 'customer-storage-bucket' with its objects. The 'Objects' tab is selected, displaying a table with columns: Name, Type, Last modified, Size, and Storage class. The table shows the structure of the uploaded files and folders.

| Name | Type | Last modified | Size | Storage class |
|-------------------------------|--------|---------------|------|---------------|
| _SUCCESS | Folder | | - | - |
| averageTransactionPerCustomer | Folder | | - | - |
| customerTransactionValue | Folder | | - | - |
| monthly_spending_trends | Folder | | - | - |
| processed_data | Folder | | - | - |
| raw_data | Folder | | - | - |
| totalQuantityCountry | Folder | | - | - |
| totalRevenueByCountry | Folder | | - | - |

4. Task 4: Data Analysis Using Spark SQL –

- Total Revenue by Country: This summarizes the total revenue by country to show which region contributes the most revenue.

```
[42]: df.createOrReplaceTempView("Retail_data")

[45]: # Total Revenue by Country
spark.sql("""
SELECT
    Country,
    SUM(Revenue) AS TotalRevenue
FROM Retail_data
GROUP BY Country
ORDER BY TotalRevenue DESC
""").show()

[Stage 41:>                                     (0 + 1) / 1]
+-----+-----+
|   Country|  TotalRevenue|
+-----+-----+
| United Kingdom| 6767873.394002574|
| Netherlands| 284661.54000000015|
| EIRE| 250285.21999999872|
| Germany| 221698.20999999862|
| France| 196712.8399999999|
| Australia| 137077.26999999973|
| Switzerland| 55739.4000000004|
```

- Monthly Spend Trending: The spending trends are aggregated monthly using InvoiceYearMonth for the analysis of spending.

```
[47]: # Monthly Spending Trends
spark.sql("""
SELECT
    InvoiceYearMonth,
    SUM(Revenue) AS MonthlyRevenue
FROM Retail_data
GROUP BY InvoiceYearMonth
ORDER BY InvoiceYearMonth
""").show()

[Stage 44:>
+-----+-----+
|InvoiceYearMonth|  MonthlyRevenue|
+-----+-----+
| 2010-12| 554604.020000018|
| 2011-1| 475074.38000001636|
| 2011-10| 974603.5899999909|
| 2011-11| 1132407.7399999578|
| 2011-12| 342506.3800000034|
| 2011-2| 436546.1500000147|
| 2011-3| 57964.6100000151|
| 2011-4| 426047.8510000125|
| 2011-5| 648251.080000003|
| 2011-6| 608013.1600000106|
| 2011-7| 574238.481000012|
| 2011-8| 616368.0000000092|
| 2011-9| 931440.371999959|
```

c. Top 10 Customers by Revenue: Identifies the big spenders to target them as key clients or a customer segment.

```
[48]: #Top 10 Customers by Revenue
spark.sql("""
SELECT
    CustomerID,
    SUM(Revenue) AS TotalCustomerRevenue
FROM Retail_data
GROUP BY CustomerID
ORDER BY TotalCustomerRevenue DESC
LIMIT 10
""").show()
```

[Stage 47:>

| CustomerID | TotalCustomerRevenue |
|------------|----------------------|
| 14646.0 | 279489.0199999999 |
| 18102.0 | 256438.4900000005 |
| 17450.0 | 187482.1700000013 |
| 14911.0 | 132572.6199999998 |
| 12415.0 | 123725.4499999987 |
| 14156.0 | 113384.1399999985 |
| 17511.0 | 88125.3799999996 |
| 16684.0 | 65892.0799999999 |
| 13694.0 | 62653.1000000003 |
| 15311.0 | 59419.3400000011 |

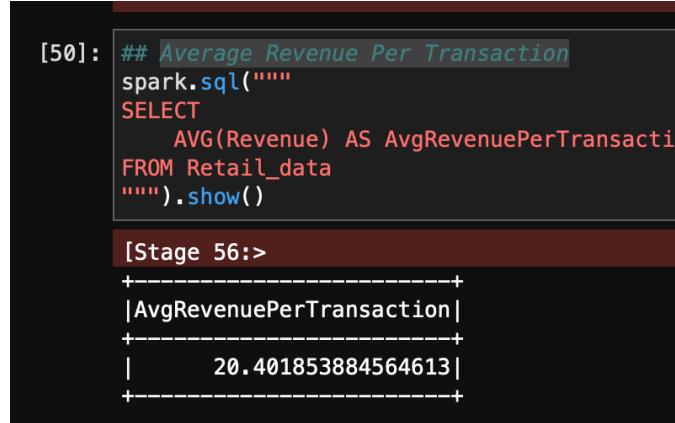
d. Count of Unique Transactions: Counts distinct transaction IDs (InvoiceNo) to review how many unique transactions exist within the dataset.

```
[49]: # Count of Unique Transactions
spark.sql("""
SELECT
    COUNT(DISTINCT InvoiceNo) AS UniqueTransactions
FROM Retail_data
""").show()
```

[Stage 50:>

| UniqueTransactionCount |
|------------------------|
| 22190 |

e. Average Revenue Per Transaction: This brings into view how much return is usually obtained per transaction.

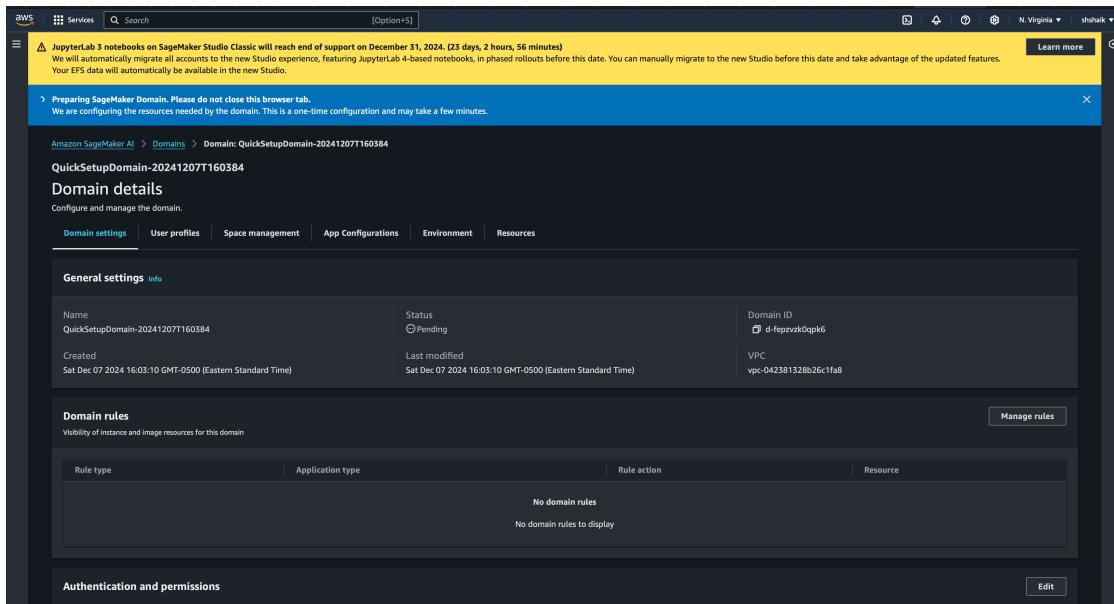


```
[50]: ## Average Revenue Per Transaction
spark.sql("""
SELECT
    AVG(Revenue) AS AvgRevenuePerTransaction
FROM Retail_data
""").show()
```

[Stage 56:>

| AvgRevenuePerTransaction |
|--------------------------|
| 20.401853884564613 |

5. Task 5: Machine Learning with AWS SageMaker Autopilot – Created domain for sagemaker



JupyterLab 3 notebooks on SageMaker Studio Classic will reach end of support on December 31, 2024. (23 days, 2 hours, 56 minutes)
We will automatically migrate all accounts to the new Studio experience, featuring JupyterLab 4-based notebooks, in phased rollouts before this date. You can manually migrate to the new Studio before this date and take advantage of the updated features.
Your EFS data will automatically be available in the new Studio.

Preparing SageMaker Domain. Please do not close this browser tab.
We are configuring the resources needed by the domain. This is a one-time configuration and may take a few minutes.

Amazon SageMaker AI > Domains > Domain: QuickSetupDomain-20241207T160384

QuickSetupDomain-20241207T160384

Domain details

Configure and manage the domain.

Domain settings User profiles Space management App Configurations Environment Resources

General settings info

| Name | Status | Domain ID |
|---|---|-----------------------|
| QuickSetupDomain-20241207T160384 | Pending | d-fepzvzk0qpk6 |
| Created | Last modified | VPC |
| Sat Dec 07 2024 16:03:10 GMT-0500 (Eastern Standard Time) | Sat Dec 07 2024 16:03:10 GMT-0500 (Eastern Standard Time) | vpc-042381328b26c1fa8 |

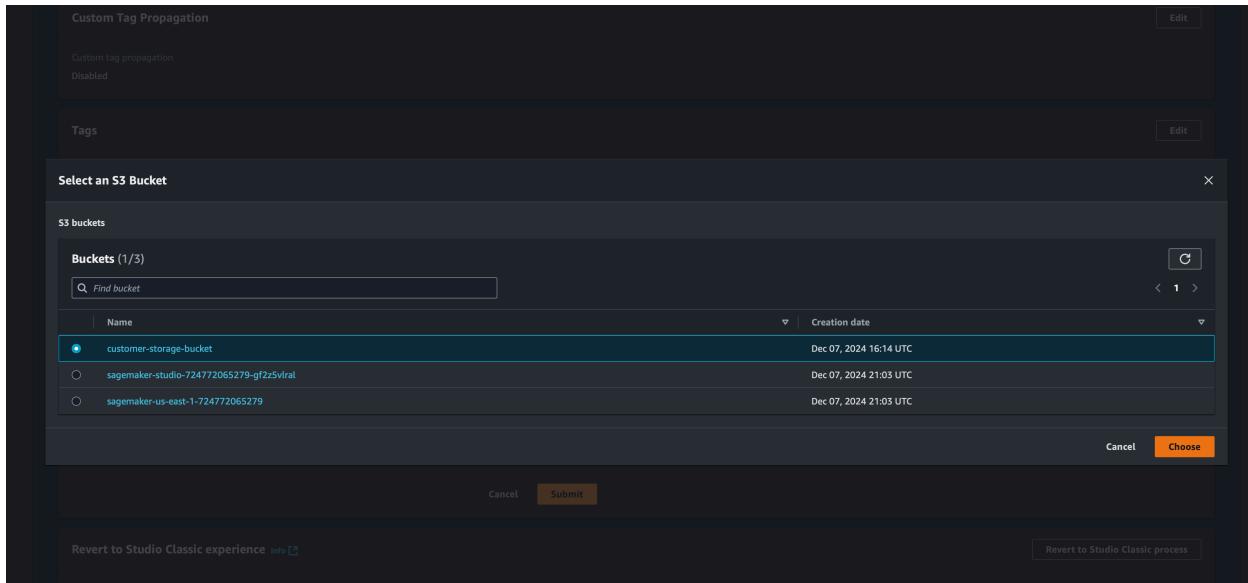
Domain rules

No domain rules

No domain rules to display

Edit

Select the S3 bucket while configuring



Launch SageMaker-

Select the AutoML, from the column in the left and Start the canvas -

The screenshot shows the SageMaker Studio interface for the Auto ML workspace. On the left, there's a sidebar with various application icons like JupyterLab, RStudio, and Canvas. The main content area is titled 'Autopilot' and describes it as a no-code workspace for building generative AI and ML solutions. It features a 'Run in Canvas' button and a status indicator showing 'Stopped'. Below this, there are sections for 'How Autopilot works' (Train models, Predict outcomes, Automate workflows), 'Tutorials and blogs' (Automatically create machine learning models, Automate batch predictions in Amazon SageMaker Canvas using updated datasets), and 'Learn more' (Get Started, Documentation, What's new). A purple banner at the top states: 'Autopilot is now in SageMaker Canvas with integrated data preparation, multi-modality support, built-in visualizations, what-if analysis, and automation support for predictions.'

Now, its time to Open in Canvas,

This screenshot is similar to the one above, but the 'Open in Canvas' button is highlighted in blue, indicating it has been clicked. A green success message at the bottom center says 'Canvas is now running' with a 'Open Canvas' button. The rest of the interface and banner are identical to the first screenshot.

AWS CANVAS LAUNCH –

New! Amazon SageMaker Canvas supports comprehensive data preparation including a conversational interface for data transformation, Gen AI capabilities to access, evaluate, and fine-tune LLMs, configuration of parameters to build models, and the ability to directly deploy models to real-time endpoints. [Learn more](#)

My models

[Grid](#) [List](#)

Build model → Analyze → Predict → Deploy

A new way to build Autopilot models in Canvas

Build, analyze, generate predictions and deploy your Autopilot models.

[+ Create new model](#)

Learn more

Resources

- Tutorials
 - Prepare training data for machine learning with minimal code
 - Automatically create machine learning models
- Documentation
 - Pricing in Canvas
 - User guide

What's new

- Automate batch predictions in Amazon SageMaker Canvas using updated datasets
- Operationalize ML models built in Amazon SageMaker Canvas to production using the Amazon SageMaker Model Registry
- Support for Custom Text and Image Classification Models in Amazon SageMaker Canvas

Click, create new model

Create new model

Model name

Retail_project

Use only letters, numbers, and underscores, up to 32 characters.

Problem type

Select the problem type you want the model to solve.

Predictive analysis

Build models using tabular datasets to predict single or multiple categories as well as regression and time-series forecast problems.

Image analysis

Build models using image datasets to predict single or multiple categories for image classification problems.

Text analysis

Build models using tabular datasets to predict single or multiple categories for text classification problems.

Fine-tune foundation model

Customize a foundation model on your data to improve its performance for a specific task or domain.

[Cancel](#) [Create](#)

a. Import Processed Data:

DATAWRANGLER -> IMPORT AND PREPARE

Data Wrangler

Search data flows Import dat

Data flows Jobs

Amazon Q

Data Wrangler

Datasets

My Models

ML Ops

Ready-to-use

Gen AI

Import data → Prepare data → Scale data operations → Build models

You haven't imported and prepared any data

Import data from over 50 sources, join, transform, and analyze your data using 300+ built-in operators or chat in Data Wrangle flows. Export your prepared data with a single click to bu

Import and prepare

Select s3 –

Import tabular data

Select a data source: **Canvas Datasets**

Search datasets in

| Name |
|--|
| <input checked="" type="radio"/> canvas-sample-hous |
| <input type="radio"/> canvas-sample-retai |
| <input type="radio"/> canvas-sample-loan |
| <input type="radio"/> canvas-sample-shipp |
| <input type="radio"/> canvas-sample-databricks-dolly-15k.csv |
| <input type="radio"/> canvas-sample-diabetic-readmission.csv |
| <input type="radio"/> canvas-sample-loans-part-2.csv |
| <input type="radio"/> canvas-sample-product-descriptions.csv |
| <input type="radio"/> canvas-sample-maintenance.csv |

Search data source Filter by: All (56) Frequently used

| | | | |
|-----------------|--------------|------------|-----------------------|
| Canvas Datasets | Local upload | Amazon S3 | Snowflake |
| Redshift | Athena | Databricks | Salesforce Data Cloud |
| | | SQLServer | |

Learn more about data sources

| File Name | Version | Rows | Columns | Last Updated | Status | |
|--|---------|------|---------|--------------|--------------------|-------|
| canvas-sample-databricks-dolly-15k.csv | V1 | 2 | 10,879 | 21,758 | 12/07/2024 4:17 PM | Ready |
| canvas-sample-diabetic-readmission.csv | V1 | 16 | 1,000 | 16,000 | 12/07/2024 4:17 PM | Ready |
| canvas-sample-loans-part-2.csv | V1 | 5 | 1,000 | 5,000 | 12/07/2024 4:17 PM | Ready |
| canvas-sample-product-descriptions.csv | V1 | 5 | 120 | 600 | 12/07/2024 4:17 PM | Ready |
| canvas-sample-maintenance.csv | V1 | 9 | 1,000 | 9,000 | 12/07/2024 4:17 PM | Ready |

Select the processed file –

Import tabular data

Select a data source:  Amazon S3 ▾

▼ Input S3 endpoint

Provide the ARN, URI, or alias

Aliases should have the format: "s3://<alias prefix>-<s3alias>"; URIs should have the format: "s3://<bucket>/<key>"; ARNs should have ARN standard format. [Learn More](#)

Amazon S3 / customer-storage-bucket / **processed_data**

| <input type="checkbox"/> | Name | Size | Last updated |
|-------------------------------------|--|--------|--------------------|
| | _SUCCESS.crc | 8 B | 12/07/2024 2:52 PM |
| | .part-00000-0cc0c2cf-8386-4f5a-98ae-c6609780d9e1-c000.csv.cr | 373 kB | 12/07/2024 2:52 PM |
| <input type="checkbox"/> | _SUCCESS | 0 B | 12/07/2024 2:52 PM |
| <input checked="" type="checkbox"/> | part-00000-0cc0c2cf-8386-4f5a-98ae-c6609780d9e1-c000.csv | 47 MB | 12/07/2024 2:52 PM |

Data Preview -

Import tabular data

← Previewing 1 file Showing the first 100 rows

X

[Import settings](#)

If your data has special character delimiters, use the advanced import settings to specify a custom delimiter. [Learn More](#)

part-00000-f046cc7f-87ee-4b39-b6cb-d2aedf34dde9-c000.csv ▾ 

| InvoiceNo | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | City |
|-----------|------------------------------|----------|--------------------------|-----------|------------|------|
| 536365 | WHITE HANGING HEART T-LIG... | 6 | 2010-12-01T08:26:00.000Z | 2.55 | 17850 | Ur |
| 536365 | WHITE METAL LANTERN | 6 | 2010-12-01T08:26:00.000Z | 3.39 | 17850 | Ur |
| 536365 | CREAM CUPID HEARTS COAT H... | 8 | 2010-12-01T08:26:00.000Z | 2.75 | 17850 | Ur |
| 536365 | KNITTED UNION FLAG HOT WA... | 6 | 2010-12-01T08:26:00.000Z | 3.39 | 17850 | Ur |
| 536365 | RED WOOLLY HOTTIE WHITE H... | 6 | 2010-12-01T08:26:00.000Z | 3.39 | 17850 | Ur |
| 536365 | SET 7 BABUSHKA NESTING BO... | 2 | 2010-12-01T08:26:00.000Z | 7.65 | 17850 | Ur |

Import settings
Settings apply to all imported files. [Learn more](#) ▾

Dataset name *

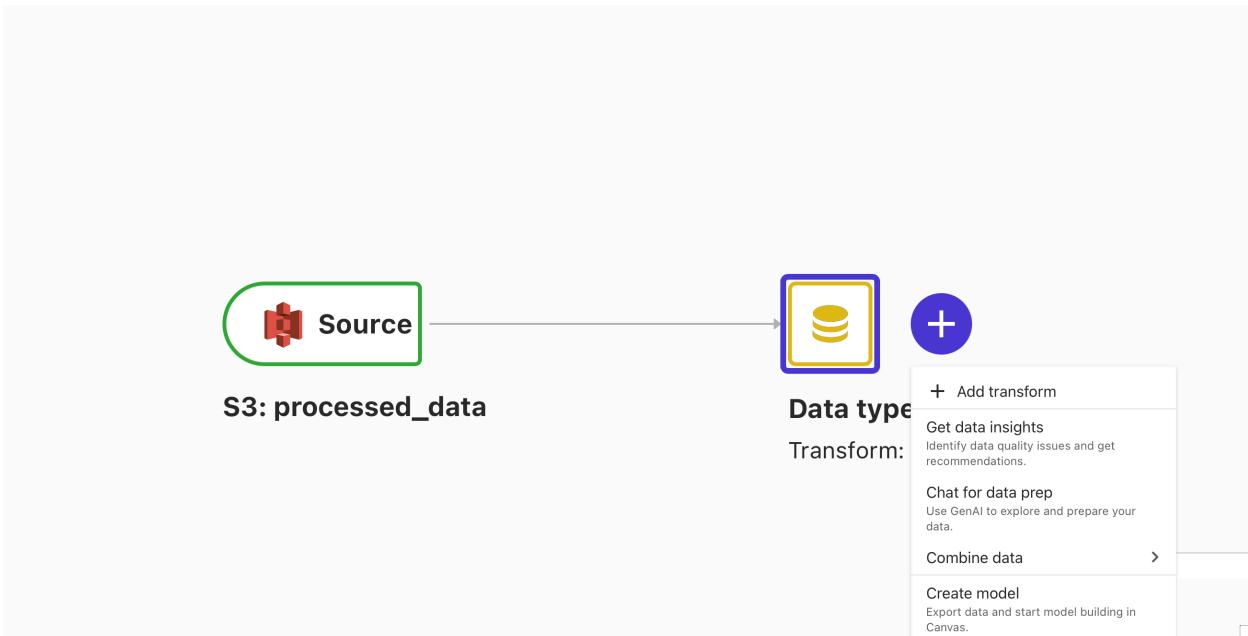
Sampling
Sample your dataset for faster exploration. Your full dataset will be used for data export or model build. [Learn more](#) ▾

Sampling method *  Random
Random sampling ensures that each row has an equal probability of being chosen.

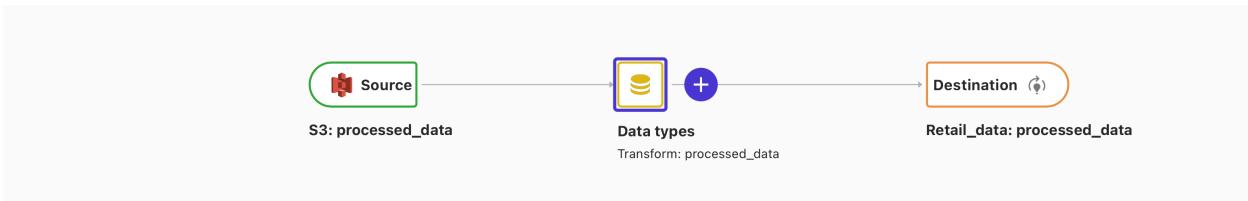
Sample size  50000
1 50k 100k 150k 200k (Recommended)

> Advanced

Data Imported Successfully, its time to create model (click on create model as shown below)



b. Run Autopilot Experiment:



This page opens up when we start creating model -

Select **Build** Analyze Predict Deploy

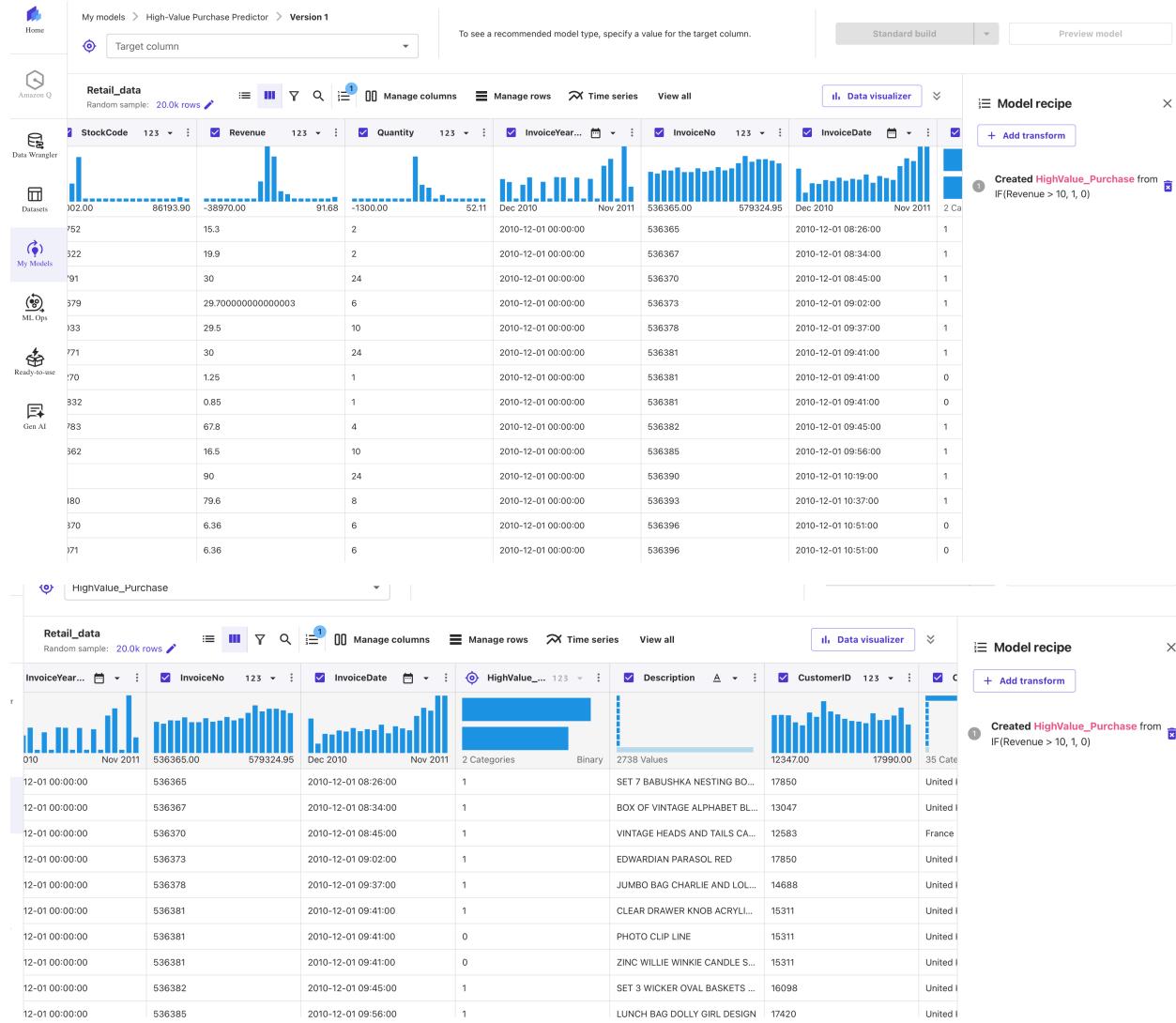
Select a column to predict
Choose the target column. The model that you build predicts values for the column that you select.
Target column: Revenue

Model type
SageMaker Canvas automatically recommends the appropriate model type for your analysis.
Time series forecasting
Your model will forecast Revenue by using past data values to predict future data values.
Configure model

Retail_data
Random sample: 20.00k rows

| Column name | Data type | Feature type | Missing | Mismatched | Unique | Mode |
|--|-------------|--------------|--------------|------------|--------|------------------------|
| UnitPrice | 123 Numeric | - | 0.00% (0) | 0.00% (0) | 141 | 1.25 |
| StockCode | 123 Numeric | - | 0.00% (2356) | 0.00% (0) | 1840 | 22,423 |
| Revenue <small>(Target)</small> | 123 Numeric | - | 0.00% (0) | 0.00% (0) | 1005 | 15 |
| Quantity | 123 Numeric | - | 0.00% (0) | 0.00% (0) | 134 | 1 |
| InvoiceYearMonth | Datetime | Binary | 0.00% (0) | 0.00% (0) | 2 | 2010-12-01T00:00:00... |
| InvoiceNo | 123 Numeric | - | 0.00% (542) | 0.00% (0) | 1883 | 540,372 |
| InvoiceDate | Datetime | - | 0.00% (0) | 0.00% (0) | 2029 | 2011-01-06T16:41:00... |
| Description | Text | - | 0.00% (0) | 0.00% (0) | 2285 | WHITE HANGING HEA... |
| CustomerID | 123 Numeric | - | 0.00% (0) | 0.00% (0) | 1166 | 12,748 |
| Country | Text | Categorical | 0.00% (0) | 0.00% (0) | 23 | United Kingdom |

Created a new column called HighValue_Purchase with condition on Revenue.



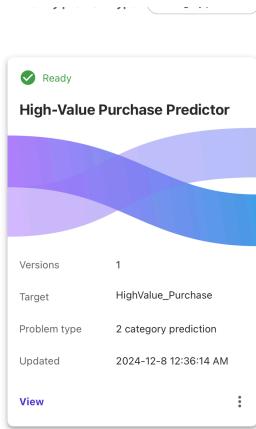
As we are planning to predict HighValue_Purchase, which is a classification problem, i.e 1 or 0. (2 category model)
 Lets configure ml model accordingly as shown below,

The screenshot shows the 'Configure model' dialog box in the Amazon SageMaker Canvas interface. The 'Model type' section is active, displaying the '2 category model' option as selected. Other model types like 'Time series forecasting' and 'Numeric model type' are also listed with their respective descriptions. The background shows a preview of the dataset and some configuration tabs at the top.

Now click on Standard build (It takes almost 1:30 hr to build the best model)

The screenshot shows the 'Build' tab in the Amazon SageMaker Canvas interface. The 'Standard build' button is highlighted in blue. To the right, there's a 'Model recipe' panel showing a list of steps: 1. Created HighValue_Purchase from IF(Revenue > 10, 1, 0), 2. Drop column InvoiceNo, 3. Drop column StockCode, 4. Drop column Description, 5. Drop column InvoiceDate, 6. Drop column Revenue, and 7. Drop column InvoiceYearMonth. The main area shows a table of dataset statistics and a 'Data visualizer' section.

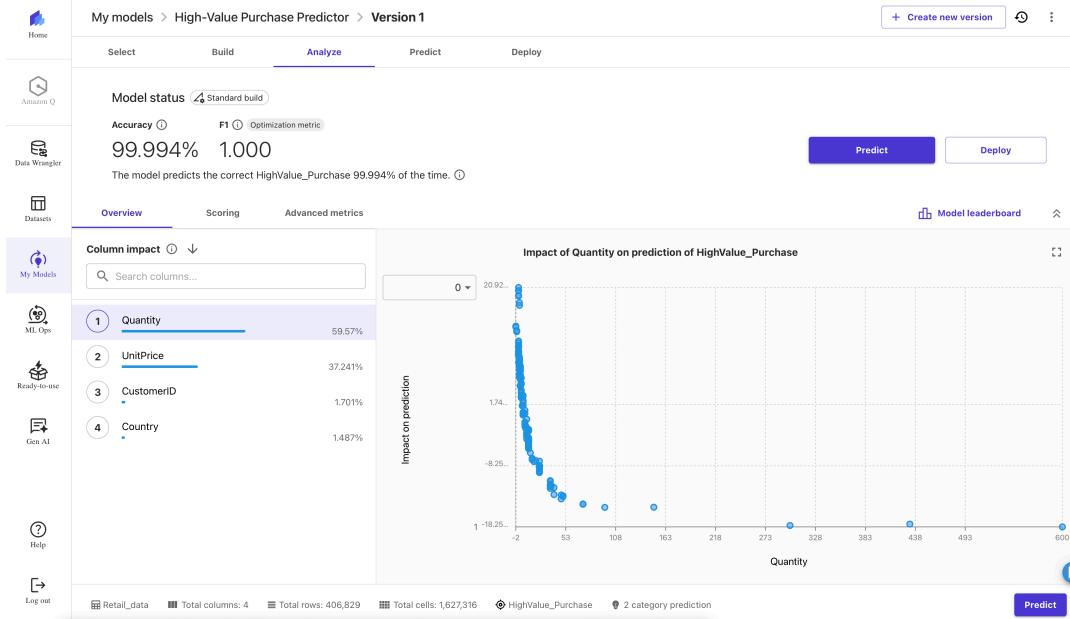
c. Screenshots: when we go through My models section -

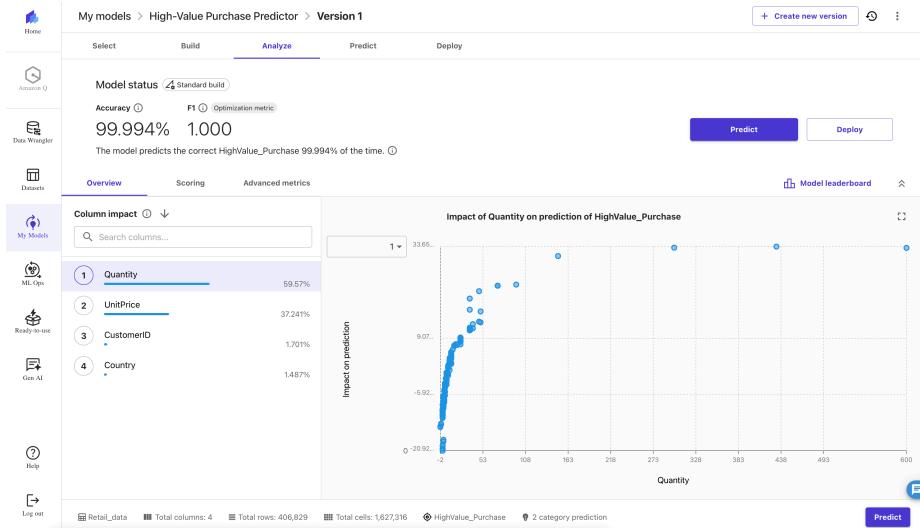


d. Review Results: (After successfully building the model)

Model Performance Metrics -

- Achieved an outstanding accuracy of 99.994%
- Perfect F1 score of 1.000, indicating optimal balance between precision and recall





Feature Importance

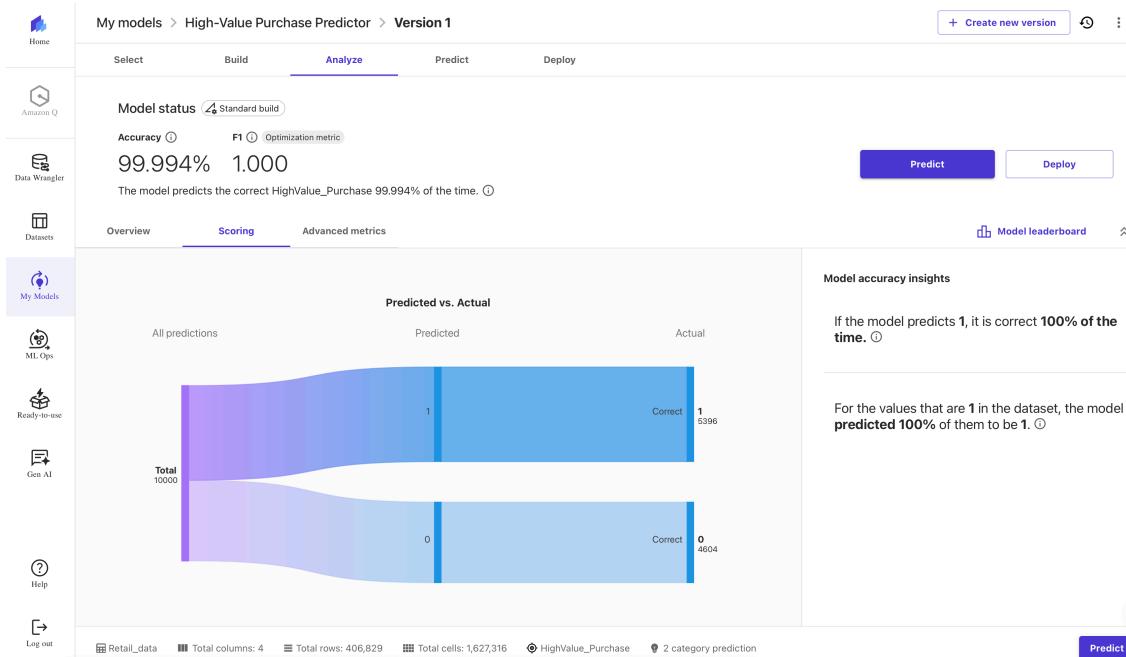
The model's predictive power relies on four main features, ranked by impact:

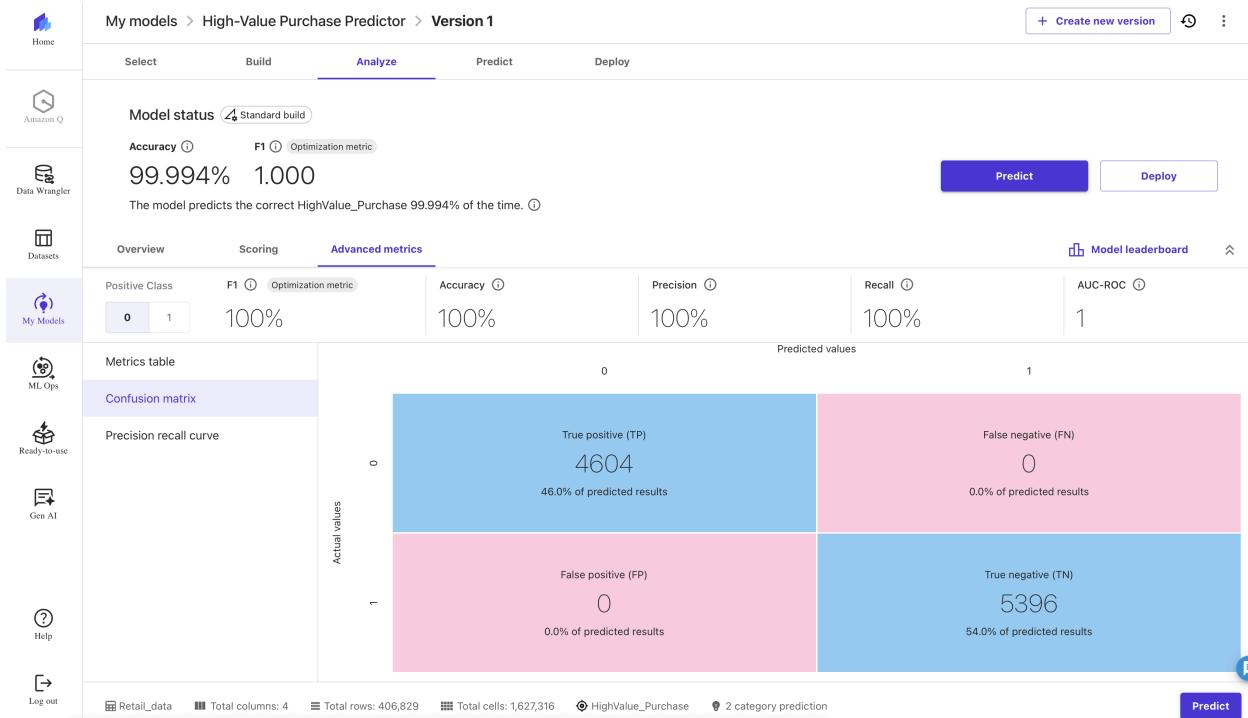
Quantity (59.67% impact)

UnitPrice (37.24% impact)

CustomerID (1.70% impact)

Country (1.457% impact)





Prediction Analysis –

- The model correctly classified 4,604 true positives and 5,396 true negatives.
- Zero false positives and false negatives, indicating perfect classification
- For all values labeled as 1 in the dataset, the model achieved 100% prediction accuracy.

Impact Analysis -

The quantity feature shows a significant non-linear relationship with high-value purchase predictions, as evidenced by the impact curve in the analysis graphs. This suggests that quantity is the most crucial factor in determining high-value purchases.

100% precision, recall, and AUC-ROC scores

My models > High-Value Purchase Predictor > Version 1

+ Create new version ⚙️ ⋮

Select Build Analyze Predict Deploy

Model status Standard build

Accuracy ⓘ F1 ⓘ Optimization metric
99.994% 1.000

The model predicts the correct HighValue_Purchase 99.994% of the time. ⓘ

Predict Deploy

Overview Scoring Advanced metrics

Positive Class F1 ⓘ Optimization metric Accuracy ⓘ Precision ⓘ Recall ⓘ AUC-ROC ⓘ

| Positive Class | F1 ⓘ Optimization metric | Accuracy ⓘ | Precision ⓘ | Recall ⓘ | AUC-ROC ⓘ |
|----------------|--------------------------|------------|-------------|----------|-----------|
| 0 1 | 100% | 100% | 100% | 100% | 1 |

Metrics table Confusion matrix Precision recall curve

Precision vs. recall

- Precision-recall (AUPRC = 1)

Precision Recall

Retail_data Total columns: 4 Total rows: 406,829 Total cells: 1,627,316 ⚙️ HighValue_Purchase 2 category prediction

My models > High-Value Purchase Predictor > Version 1

+ Create new version ⚙️ ⋮

Select Build Analyze Predict Deploy

Model status Standard build

Accuracy ⓘ F1 ⓘ Optimization metric
99.994% 1.000

The model predicts the correct HighValue_Purchase 99.994% of the time. ⓘ

Predict Deploy

Overview Scoring Advanced metrics

Positive Class F1 ⓘ Optimization metric Accuracy ⓘ Precision ⓘ Recall ⓘ AUC-ROC ⓘ

| Positive Class | F1 ⓘ Optimization metric | Accuracy ⓘ | Precision ⓘ | Recall ⓘ | AUC-ROC ⓘ |
|----------------|--------------------------|------------|-------------|----------|-----------|
| 0 1 | 100% | 100% | 100% | 100% | 1 |

Metrics table Confusion matrix Precision recall curve

Metrics table ⓘ

| Metric name | Value |
|-------------|-------|
| precision | 1.000 |
| recall | 1.000 |
| accuracy | 1.000 |
| f1 | 1.000 |
| auc | 1.000 |

Retail_data Total columns: 4 Total rows: 406,829 Total cells: 1,627,316 ⚙️ HighValue_Purchase 2 category prediction

The screenshot shows the Amazon SageMaker Studio interface. On the left, there's a sidebar with various icons for Home, Amazon Q, Data Wrangler, Datasets, My Models (selected), ML Ops, Ready-to-use, Gen AI, Help, and Log out. The main content area is titled "My models > High-Value Purchase Predictor > Version 1". It has tabs for Select, Build, Analyze (which is selected), Predict, and Deploy. Below this is a "Model leaderboard" section with a search bar. The table lists multiple model versions with columns for Model name, F1 Optimization, Accuracy, AUC, Balanced Accur..., Precision, Recall, Log Loss, and Inference latency (seconds). Most models show 100% accuracy and low log loss.

| Model name | F1 Optimization | Accuracy | AUC | Balanced Accur... | Precision | Recall | Log Loss | Inference latency (seconds) | |
|---|-----------------|----------|----------|-------------------|-----------|----------|----------|-----------------------------|-------|
| FULL-t5724772065279Canvas1733629628069 | Default model | 99.993% | 99.994% | 1.000 | 99.994% | 99.989% | 99.997% | 1.790 | 0.101 |
| FULL-t8724772065279Canvas1733629628069 | | 100.000% | 100.000% | 1.000 | 100.000% | 100.000% | 100.000% | 2.686 | 0.156 |
| FULL-t7724772065279Canvas1733629628069 | | 100.000% | 100.000% | 1.000 | 100.000% | 100.000% | 100.000% | 2.686 | 0.169 |
| FULL-t6724772065279Canvas1733629628069 | | 99.996% | 99.996% | 1.000 | 99.996% | 100.000% | 99.992% | 2.250 | 0.107 |
| FULL-t4724772065279Canvas1733629628069 | | 100.000% | 100.000% | 1.000 | 100.000% | 100.000% | 100.000% | 2.686 | 0.156 |
| FULL-t3724772065279Canvas1733629628069 | | 100.000% | 100.000% | 1.000 | 100.000% | 100.000% | 100.000% | 3.176 | 0.104 |
| FULL-t2724772065279Canvas1733629628069 | | 100.000% | 100.000% | 1.000 | 100.000% | 100.000% | 100.000% | 2.686 | 0.151 |
| FULL-t1724772065279Canvas1733629628069 | | 100.000% | 100.000% | 1.000 | 100.000% | 100.000% | 100.000% | 2.686 | 0.148 |
| -L1-FULL-t9724772065279Canvas1733629628069 | | 99.997% | 99.998% | 1.000 | 99.997% | 100.000% | 99.995% | 30.001 | 0.135 |
| -L1-FULL-t10724772065279Canvas1733629628069 | | 99.997% | 99.998% | 1.000 | 99.997% | 100.000% | 99.995% | 30.001 | 0.140 |

Model Leaderboard -

The leaderboard shows multiple model iterations, with most versions achieving 100% or near-perfect scores across various metrics.

The default model shows:

Balanced Accuracy: 99.994%

Log Loss: 1.790

Inference latency: 0.101 seconds

The model processes a substantial dataset with:

Total rows: 456,829

Total cells: 1,827,316

2-category prediction for HighValue_Purchase

4) Visualization –

Gathered all files and exported to local – (As PowerBi doesn't support S3)

```
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" -r ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/averageTransactionPerCustomer ./  
part-00000-21780367-3d60-4e5c-81e8-70aefc690b1-c000.csv  
.part-00000-21780367-3d60-4e5c-81e8-70aefc690b1-c000.csv.crc  
.SUCCESS.crc  
(base) shoukath@Mac Mini Project % ls  
BDA.pem           averageTransactionPerCustomer    rawdata  
Report.docx        ec2-user                         ~$Report.docx  
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" -r ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/customerTransactionValue ./  
part-00000-97f63484-aid3-42da-b0e0-0dd69bc314eb-c000.csv  
.part-00000-97f63484-aid3-42da-b0e0-0dd69bc314eb-c000.csv.crc  
.SUCCESS.crc  
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" -r ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/monthly_spending_trends ./  
part-00000-010364c0-0a71-4c10-8112-3230a0fae36cc-c000.csv  
.part-00000-010364c0-0a71-4c10-8112-3230a0fae36cc-c000.csv.crc  
.SUCCESS.crc  
part-00000-010364c0-0a71-4c10-8112-3230a0fae36cc-c000-checkpoint.csv  
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" -r ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/processed ./  
part-00000-f046cc7f-87ee-4b39-b6cb-d2aedf34dde9-c000.csv  
.part-00000-f046cc7f-87ee-4b39-b6cb-d2aedf34dde9-c000.csv.crc  
.SUCCESS.crc  
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" -r ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/totalQuantityCountry ./  
part-00000-9f6650c7-c4c8-4dcc-a483-d4b38f51ab95-c000.csv  
.part-00000-9f6650c7-c4c8-4dcc-a483-d4b38f51ab95-c000.csv.crc  
.SUCCESS.crc  
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" -r ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/totalRevenueByCountry ./  
part-00000-e67377ec-eb59-40ab-85d4-47b2c1b61305-c000.csv  
.part-00000-e67377ec-eb59-40ab-85d4-47b2c1b61305-c000.csv.crc  
.SUCCESS.crc  
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/Online_Retail.csv ./  
Online_Retail.csv  
(base) shoukath@Mac Mini Project % scp -i "BDA.pem" ec2-user@ec2-98-81-251-159.compute-1.amazonaws.com:/home/ec2-user/Project.ipynb ./  
Project.ipynb  
(base) shoukath@Mac Mini Project % ls  
BDA.pem           Report.docx          ec2-user          rawdata          ~$Report.docx  
Online_Retail.csv averageTransactionPerCustomer monthly_spending_trends totalQuantityCountry  
Project.ipynb      customerTransactionValue processed          totalRevenueByCountry  
(base) shoukath@Mac Mini Project %
```

Task: Create Dashboards

The screenshot shows the Microsoft Power BI desktop application. On the left, there's a vertical sidebar with various navigation options like Home, Create, Browse, etc. The main area has a red header bar with the Power BI logo and 'My workspace'. Below the header, there's a large callout box titled 'Build your first report' with three steps: 'Add and prepare your data', 'Generate a premade report', and 'Customize to suit your needs'. To the right of this, there's a section titled 'Add data to start building a report' with four options: 'Excel (Preview)', 'CSV (Preview)', 'Paste or manually enter data', and 'Pick a published semantic model'. At the bottom, it says 'Don't see the source you're looking for? Download the desktop app.' and 'See all'.

Other items you can create with Microsoft Fabric

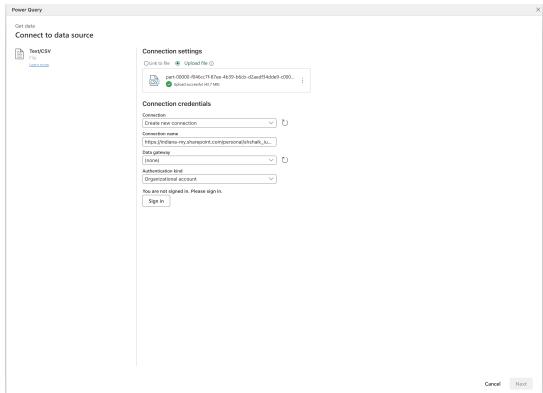
Current workspace: My workspace

Items will be saved to this workspace.

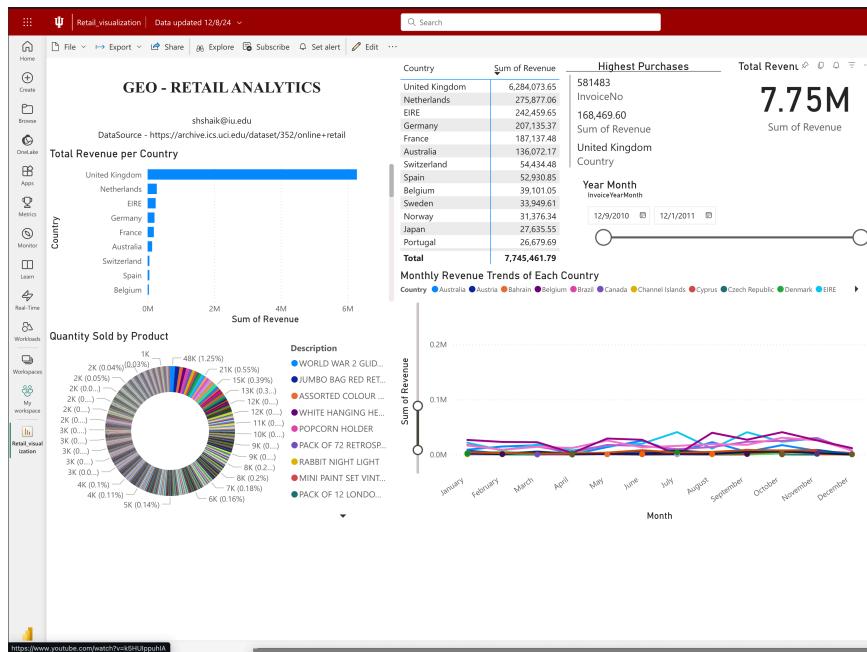
Lakehouse Store big data for cleaning, querying, reporting, and sharing.

Notebook Explore data and build machine learning solutions with Apache Spark applications.

1. Connect QuickSight / PowerBI to the processed data in S3.



2. Design a dashboard with at least 4 insightful visualizations.



YearMonth – (Interactive time slider covering period from 12/1/2011 to 12/20/2015, flexible to modify in real time, the entire visualization get updated)



Highest Purchases – (This displays the highest invoice billed ever)

The screenshot shows the Power BI Data view interface. At the top, there are tabs for Buttons, Visual interactions, Refresh, Save, Pin to a dashboard, Chat in Teams, and Update. Below these are sections for Filters, Visualizations, and Data.

Filters:

- Filters on this visual:
 - Country (is All)
 - InvoiceNo (top 1 by Sum of Revenue)
 - Sum of Revenue (is All)
- Filters on this page:
 - Add data fields here
- Filters on all pages:
 - Add data fields here

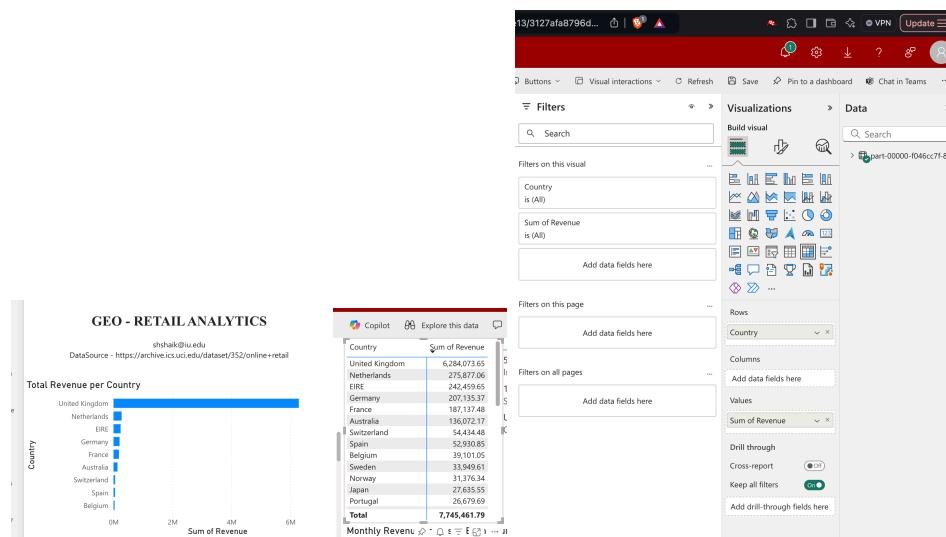
Data:

- Search bar: Search > part-00000-1046cc71-8...
- Fields section:
 - InvoiceNo
 - Sum of Revenue
 - Country
- Drill through, Cross-report, Keep all filters, Add drill-through fields here options.

Country and their total revenue –

Bar chart clearly shows United Kingdom as the dominant market with highest revenue

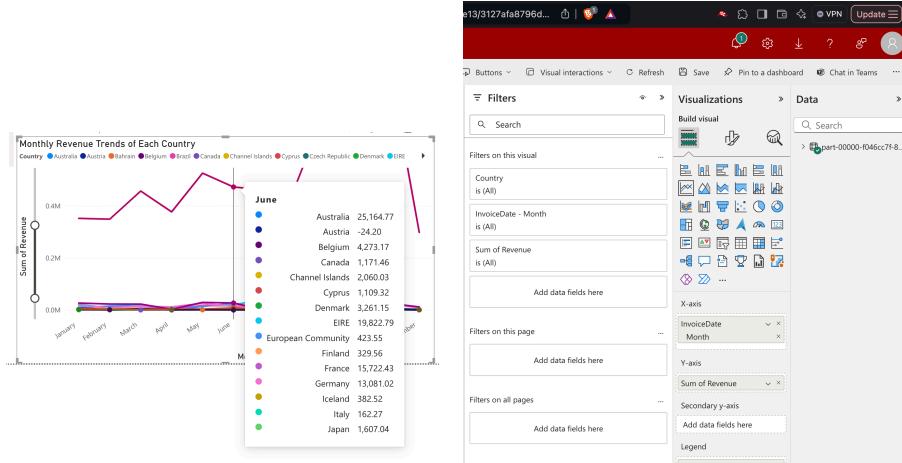
Total revenue of 7.75M across all markets displayed prominently



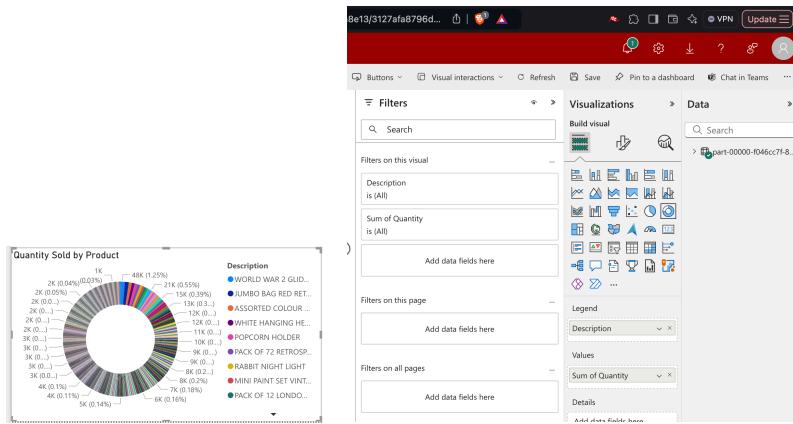
Monthly Revenue trends for each country –

Monthly revenue trends displayed through multi-line graph

Ability to track performance changes over time for each country. With Tooltip of each country for a particular month as shown below.



Quantity sold per Product – Amount of products sold in bulk for a country or Quantity purchased distribution within a country.



Note – All the visualizations can be modified with the help of slider (year month) , country, product and many more.

-----XXX-----