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.

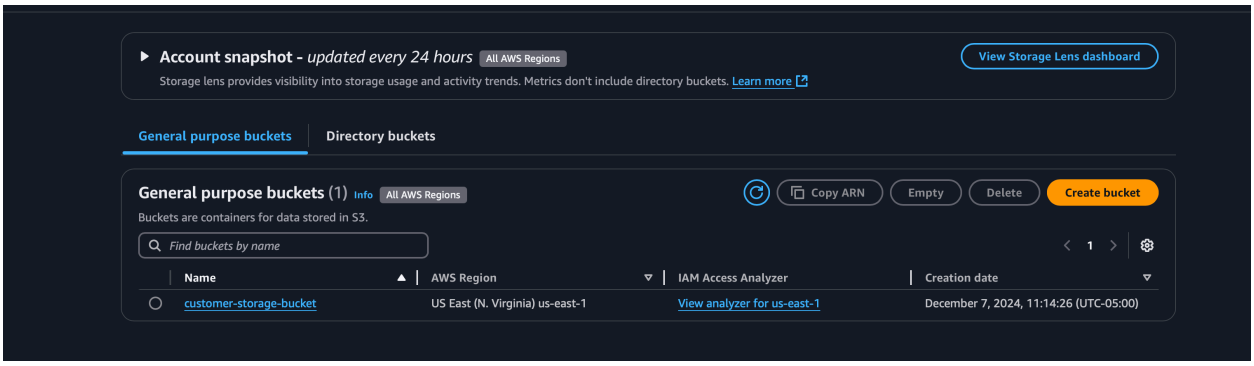
The dataset can support classification and clustering tasks, therefore providing ample opportunities for customer segmentation, sales forecasting, and product performance analysis. Being multivariate, sequential, and time-series in nature, it is ideal for exploring consumer trends and informing business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

2) Environment Setup –

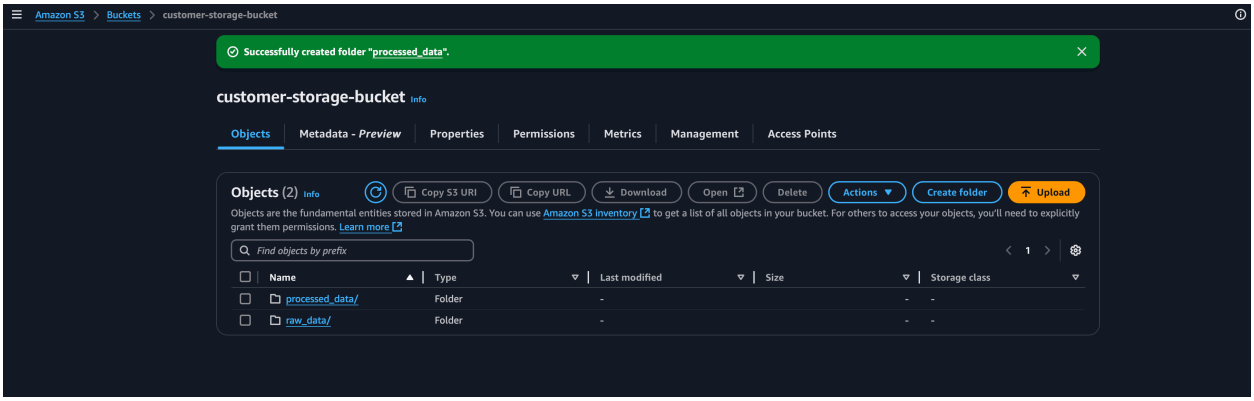
1. AWS S3 for Data Storage –

a. Step 1 –

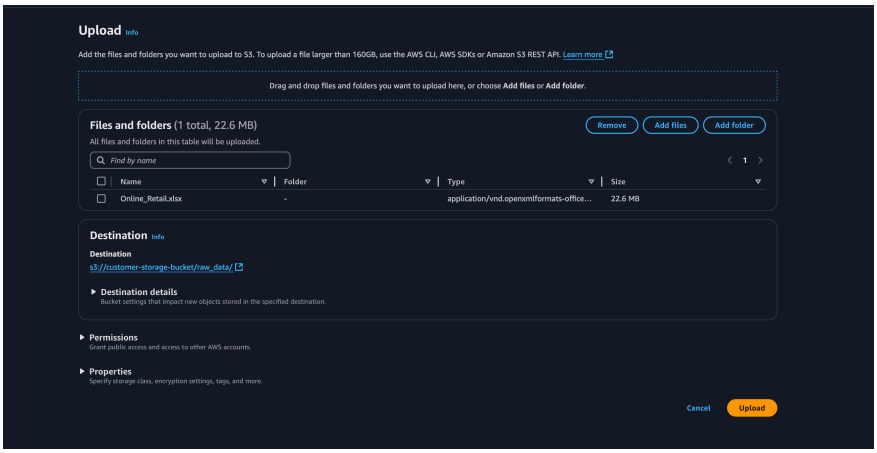
Creating bucket instance

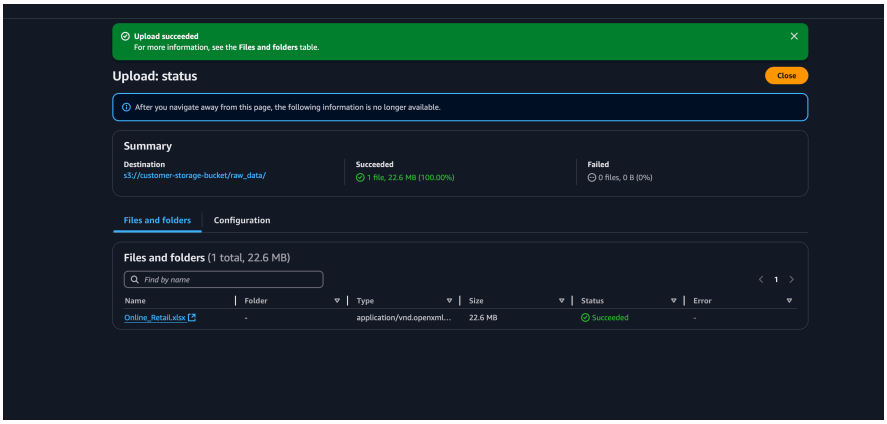


Created directories – (raw and processed data)



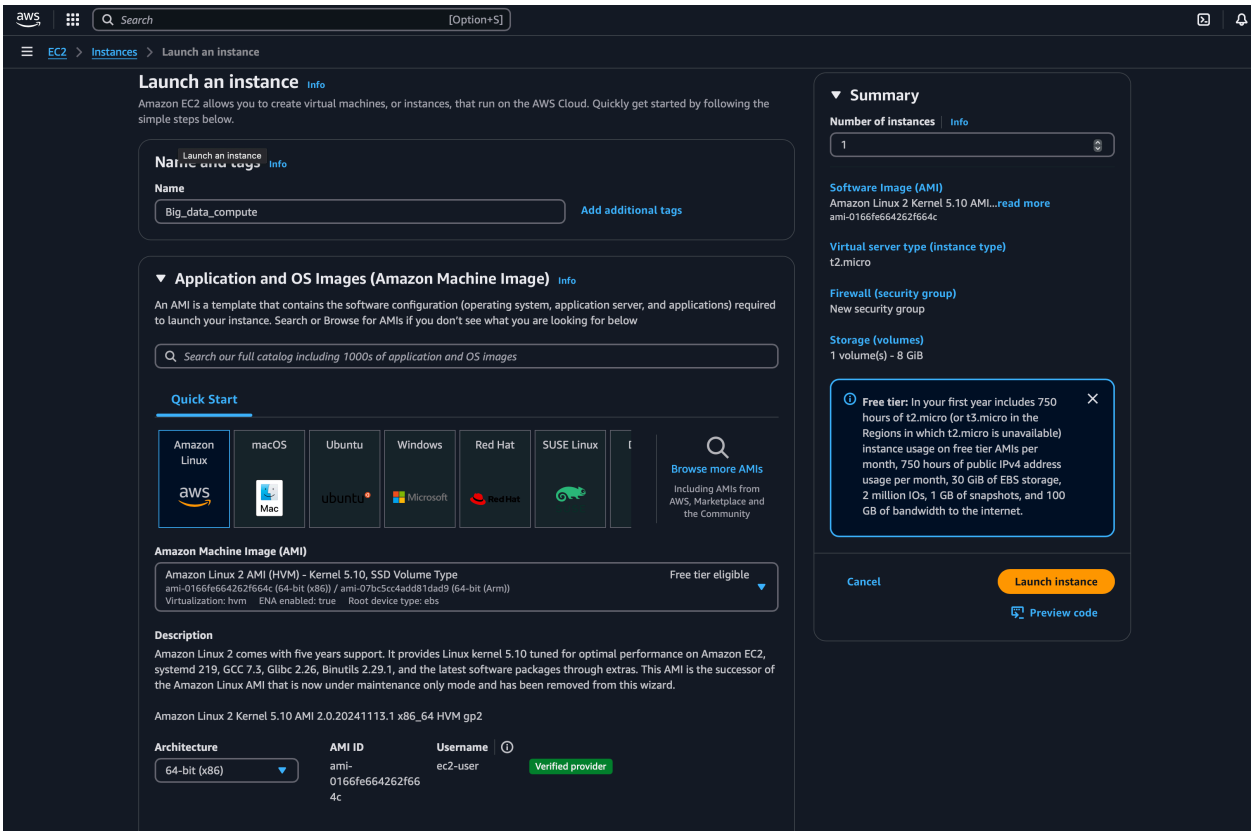
b. Step 2 – Upload the raw dataset to the S3 bucket



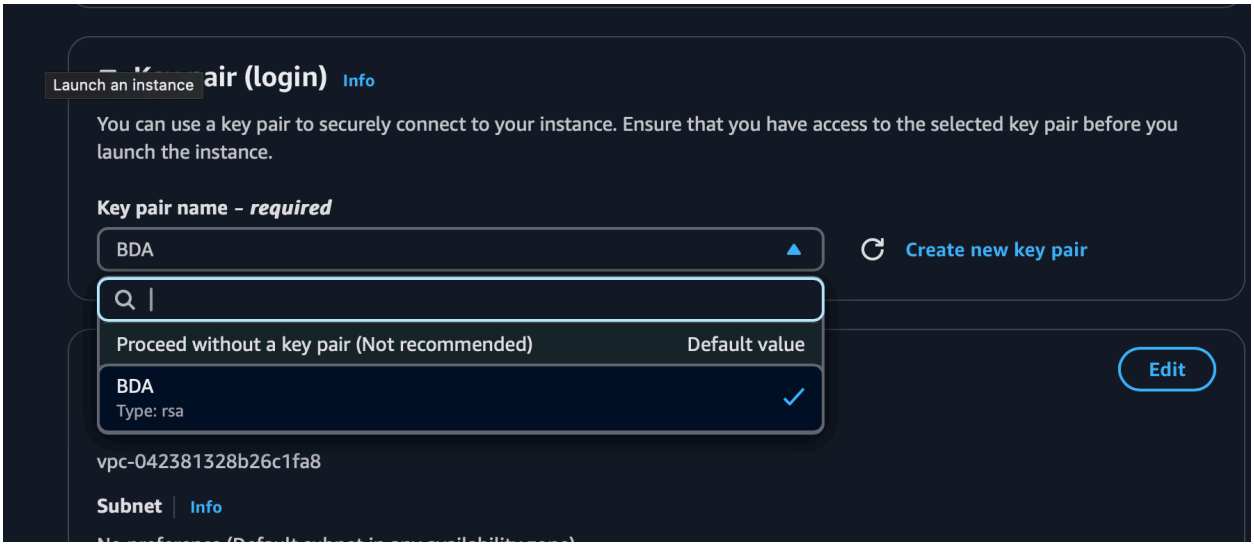


2. Linux Environment with PySpark –
    a. Step 1 – SETUP ENVIRONMENT EC2 INSTANCE

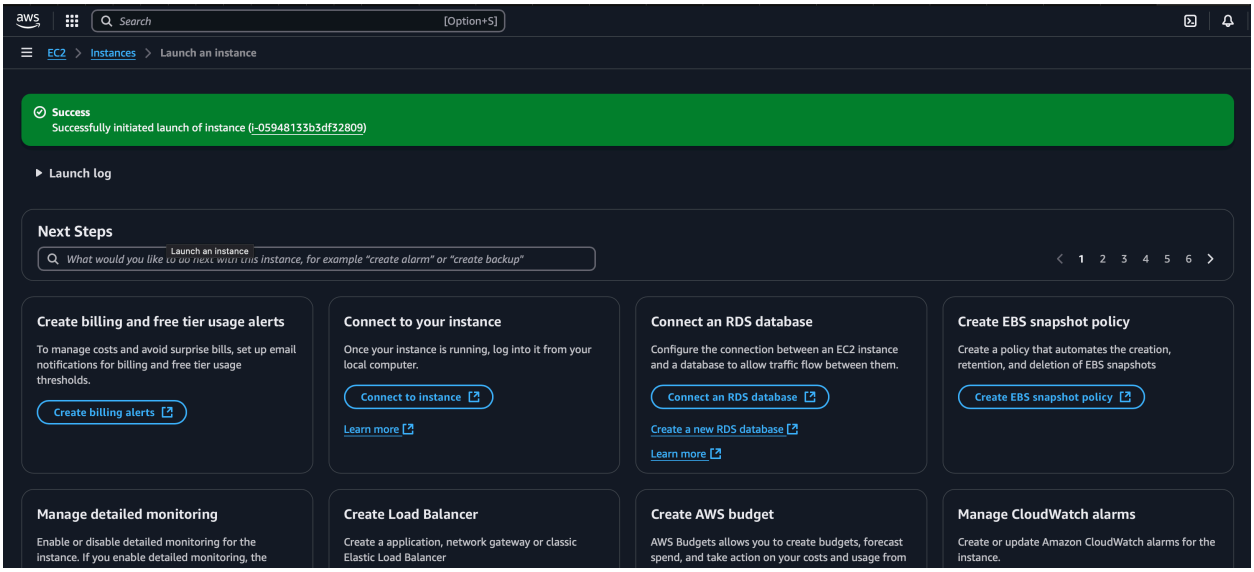
- Go to aws cloud, search for ec2 instance
    - Create an Aws EC2 instance –
    - By providing name tag, and choosing appropriate AMI



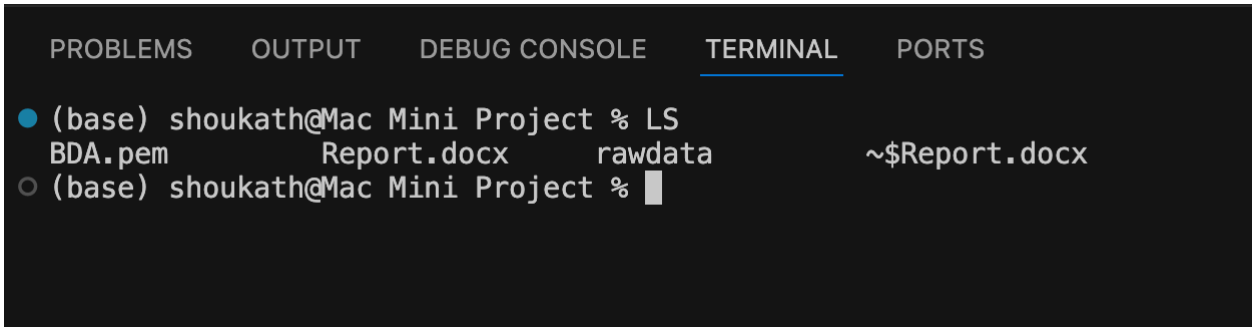
- While creating the instance generate the .pem file (save in local directory, to connect with instance locally later)



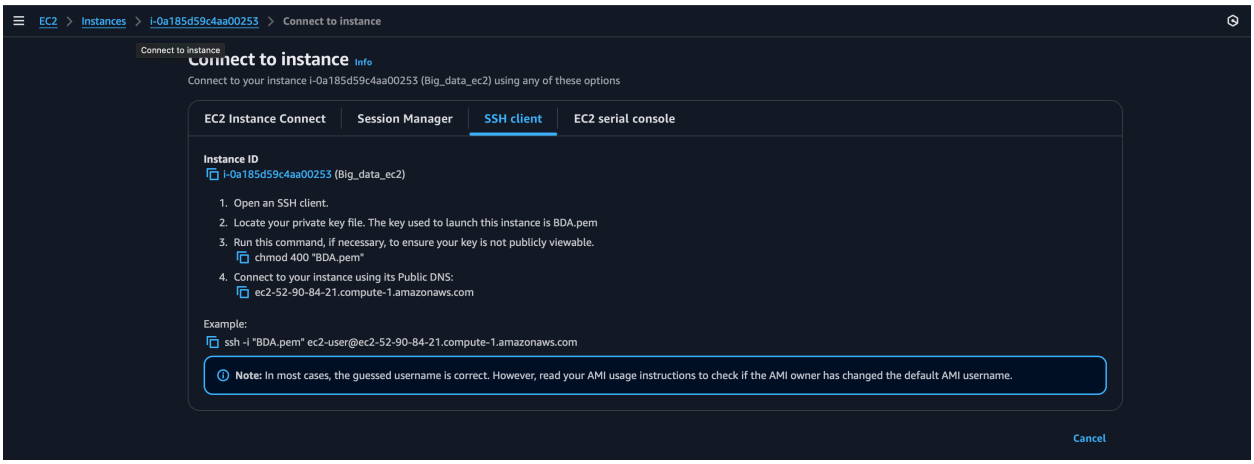
-After successful deployment



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



- 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.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    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/
```

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.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    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/
```

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 ~]$
[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 ~]$
[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
```

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)

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
(base) shoukath@Mac Mini Project % LS
BDA.pem        Report.docx     rawdata         ~$Report.docx
(base) shoukath@Mac Mini Project % chmod 400 "BDA.pem"
```

```
(base) shoukath@Mac Mini Project % ssh -i "BDA.pem" ec2-user@ec2-98-81-250-251.compute-1.amazonaws.com
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    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/
```

c. Step 2 – Install PySpark for distributed data processing.

Checking the free space, and java installation

```
[ec2-user@ip-172-31-84-95 ~]$ free -h
              total        used        free      shared  buff/cache   available
Mem:           952M         69M        272M        408K        610M        748M
Swap:            0B          0B          0B
[ec2-user@ip-172-31-84-95 ~]$
[ec2-user@ip-172-31-84-95 ~]$
[ec2-user@ip-172-31-84-95 ~]$ sudo yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
No packages marked for update
[ec2-user@ip-172-31-84-95 ~]$ sudo yum install -y java-1.8.0-openjdk
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
---> Package java-1.8.0-openjdk.x86_64 1:1.8.0.412.b08-1.amzn2.0.1 will be installed
```

```
Complete!
[ec2-user@ip-172-31-84-95 ~]$ java -version
openjdk version "1.8.0_412"
OpenJDK Runtime Environment (build 1.8.0_412-b08)
OpenJDK 64-Bit Server VM (build 25.412-b08, mixed mode)
```

Install pip3

```
[ec2-user@ip-172-31-84-95 ~]$ python3 --version
Python 3.7.16
[ec2-user@ip-172-31-84-95 ~]$ sudo yum install python3-pip -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core                                                              | 3.6 kB  00:00:00
Package python3-pip-20.2.2-1.amzn2.0.7.noarch already installed and latest version
Nothing to do
```

---------- Challenges ----------

Now install the pyspark

```
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 pyspark
Defaulting to user installation because normal site-packages is not writeable
Collecting pyspark
  Downloading pyspark-3.4.4.tar.gz (311.4 MB)
     |████████████████████████████████| 311.4 MB 65.0 MB/s eta 0:00:01Killed
[ec2-user@ip-172-31-85-70 ~]$
```

It clearly getting killed--

Check the disk usage

```
[ec2-user@ip-172-31-85-70 ~]$ pip3 install pyspark
Defaulting to user installation because normal site-packages is not writeable
Collecting pyspark
  Downloading pyspark-3.4.4.tar.gz (311.4 MB)
     |████████████████████████████████| 311.4 MB 90.8 MB/s eta 0:00:01Killed
[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      8.0G  2.8G  5.3G  35% /
tmpfs            96M     0   96M   0% /run/user/1000
tmpfs            96M     0   96M   0% /run/user/0
```

Lot of space available

Checking the ram usage--

```
[ec2-user@ip-172-31-85-70 project]$ pip3 install pyspark
Defaulting to user installation because normal site-packages is not writeable
Collecting pyspark
  Downloading pyspark-3.4.4.tar.gz (311.4 MB)
     |████████████████████████████████| 311.4 MB 45.2 MB/s eta 0:00:01Killed
[ec2-user@ip-172-31-85-70 project]$ free -h
              total        used        free      shared  buff/cache   available
Mem:           952M         62M        860M        468K         29M        805M
Swap:            0B          0B          0B
```

Lot of free memory available

To solve this problem lets implement the swap method (to stay in free trail limit)--

While creating swap block size of 1gb its throwing error, so implemenanted smaller block size of 2 gb 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
```

Set Permission, enable and format the swap block--

```
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=0b08a0d0-f06c-4841-9e07-ebf8d1333259
[ec2-user@ip-172-31-85-70 project]$ sudo swapon /swapfile
```

```
[ec2-user@ip-172-31-85-70 project]$
[ec2-user@ip-172-31-85-70 project]$ swapon --show
NAME      TYPE SIZE USED PRIO
/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
```

Verifying the swap space--

Making swap permanent--

```
[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 pyspark--

```
[ec2-user@ip-172-31-85-70 project]$ pip3 install pyspark
Defaulting to user installation because normal site-packages is not writeable
Collecting pyspark
  Downloading pyspark-3.4.4.tar.gz (311.4 MB)
     |████████████████████████████████| 311.4 MB 28 kB/s
Collecting py4j==0.10.9.7
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
     |████████████████████████████████| 200 kB 17.4 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-172-31-85-70 project]$ python3 -c "import pyspark; print(pyspark.__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/ec

# .bashrc

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

# 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
```

Reload config--

```
[ec2-user@ip-172-31-85-70 project]$ source ~/.bashrc
```

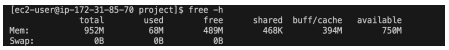
Test pyspark--

```
[ec2-user@ip-172-31-85-70 project]$ pyspark
/home/ec2-user/.local/bin/pyspark: line 24: /usr/local/spark/bin/load-spark-env.sh: No such file or directory
/home/ec2-user/.local/bin/pyspark: line 68: /usr/local/spark/bin/spark-submit: No such file or directory
[ec2-user@ip-172-31-85-70 project]$ print(sc.version)
-bash: syntax error near unexpected token `sc.version'
```

Pyspark is not installed correct, so i'll remove the swap and install manually instead of pip3

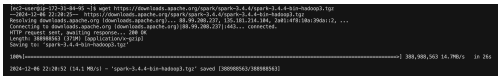
Removing swap--(helps to improve performance further)

```
[ec2-user@ip-172-31-85-70 project]$ sudo 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
```

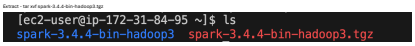


Now installing pyspark – (Created a new instance and followed the steps above except challenge block)

Download –



Extract - tar xvf spark-3.4.4-bin-hadoop3.tgz

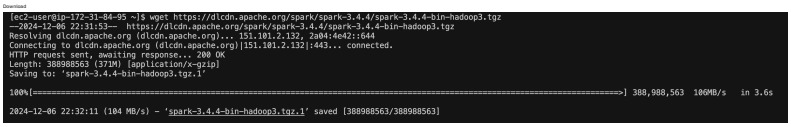


Move to opt dir –

SET ENV VAR –

Install pyspark –

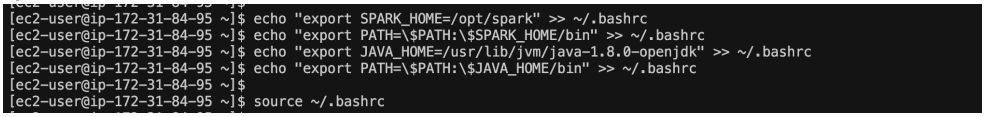
Download



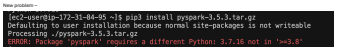
Extract - tar -xvzf spark-3.4.4-bin-hadoop3.tgz

Moving –

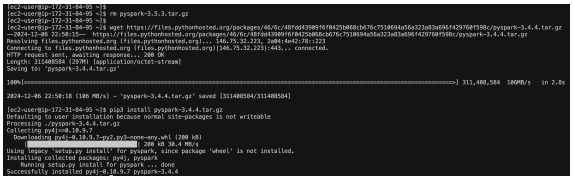
Setting ENV VAR and reloadd–



New problem –



Downloaded the suitable version and finally installed –



Test –

Not identifying JAVA –

SETTING ENV VARS PROPERLY –

---------- Still error ---- (Starting a fresh)

---

c. Step 3 –

AWS CLI INSTALLATION AND CONFIGUARATION - 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]$ ld
ld: 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=fJzXsou3orFJ6X1n4OkqwLW3lgNvm05pRVKJlKJg
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
```

Successfully interacted with s3 and can retrieve files

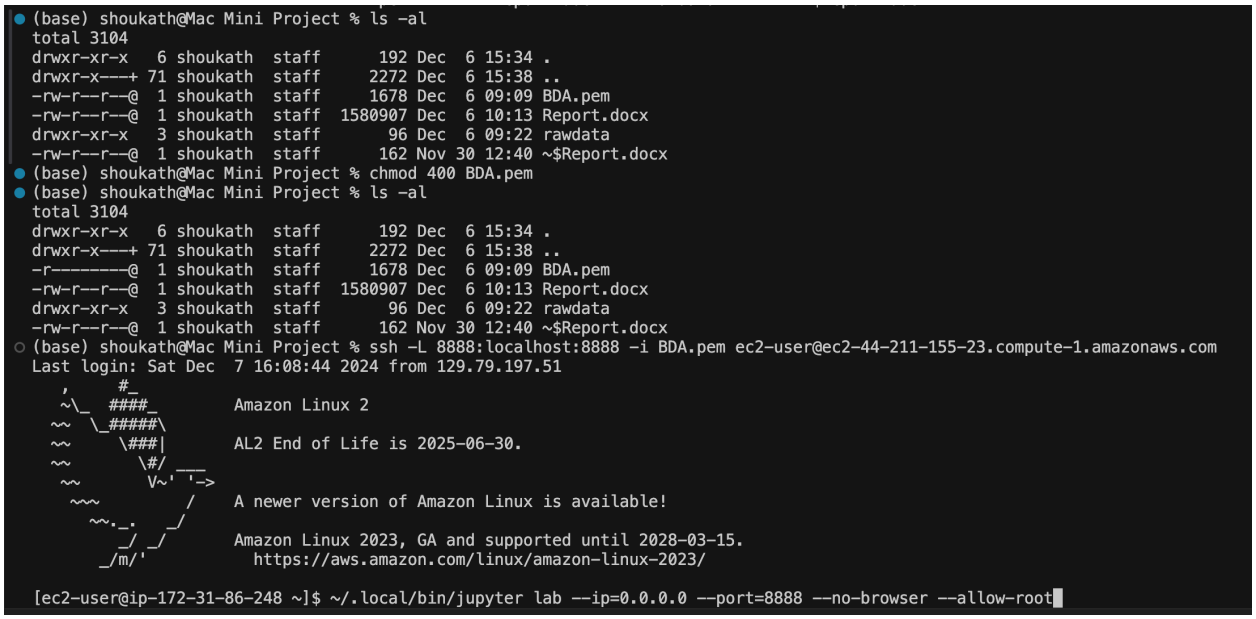
3) Data Pipeline Tasks –

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 -

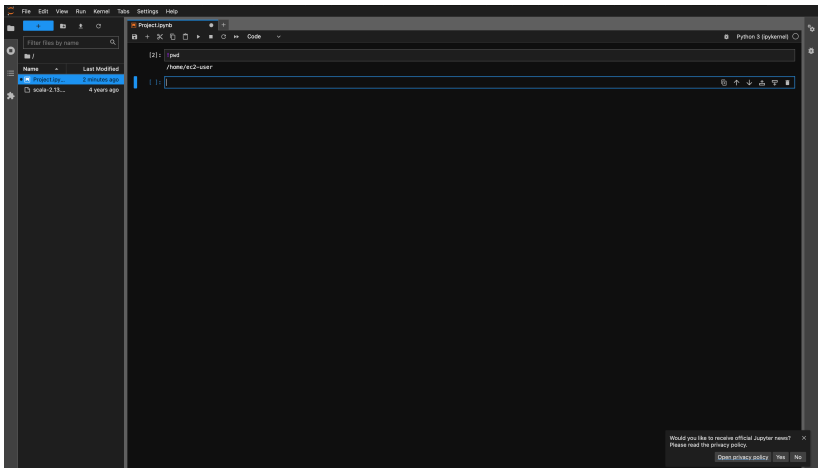
```
ssh -L 8888:localhost:8888 -i BDA.pem ec2-user@ec2-44-211-155-23.compute-1.amazonaws.com
```

As shown below -



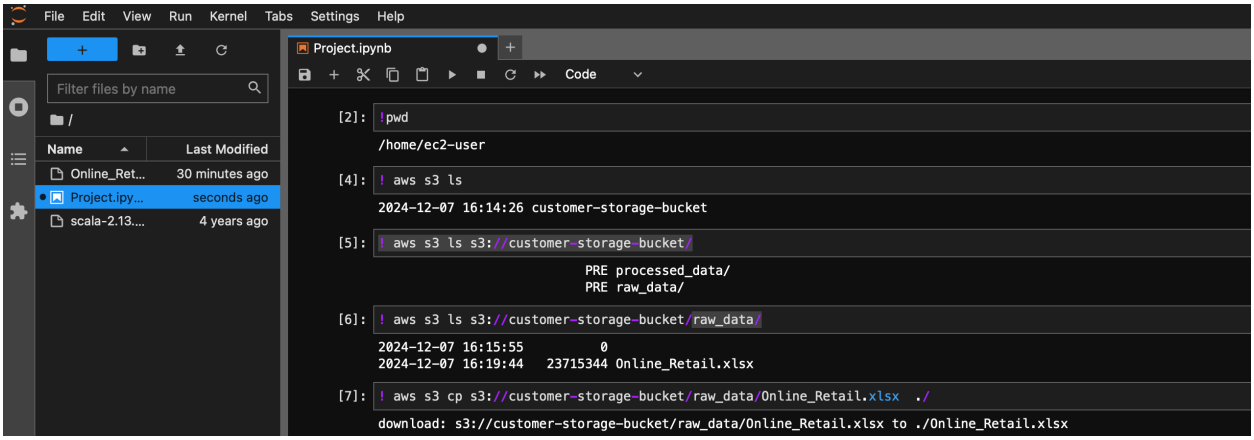
Check –

Successfully running at localhost (at specified port in terminal)

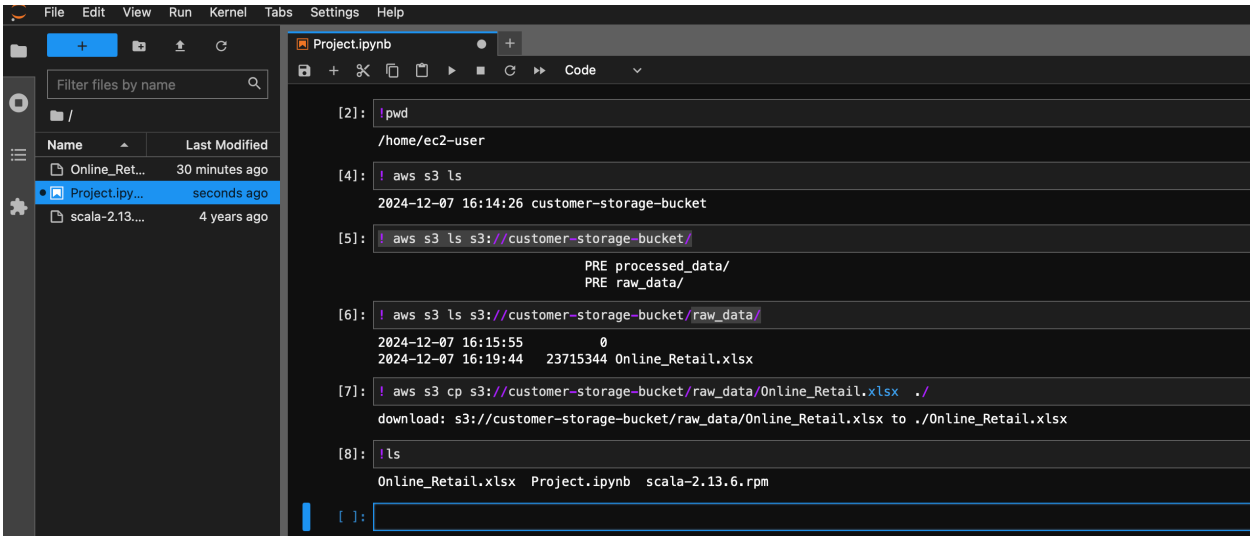


Let’s start the actual tasks

1. Task 1 - Data Ingestion from S3 –
    a. Step 1 – Use AWS CLI or PySpark’s built-in S3 support to load the dataset directly

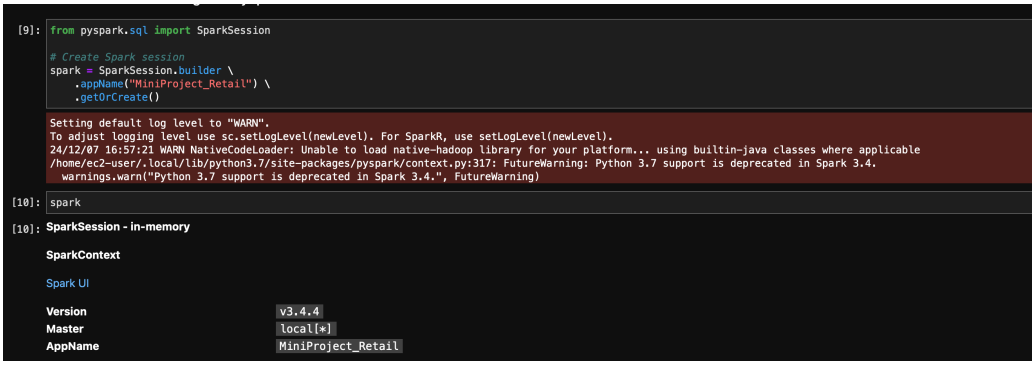


    b. Step 2 –



2. Task 2: Data Processing with PySpark -

   Check pyspark -



   Cleaned data and verified schema before transformations –

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 od 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|15.299999999999999|        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 FLA...|       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 FROSTE...|       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|11.100000000000001|        2010-12|
|   536366|    22632|HAND WARMER RED P...|       6|2010-12-01 08:28:00|     1.85|   17850.0|United Kingdom|11.100000000000001|        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()
```

```
[Stage 19:>                                                         (0 + 1) / 1]
+------------------+
|           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)
```

```
[Stage 22:>                                                         (0 + 1) / 1]
+--------------+------------------+
|       Country|     Total_Revenue|
+--------------+------------------+
|United Kingdom| 6767873.394002574|
|   Netherlands|284661.54000000015|
|          EIRE|250285.21999999872|
|       Germany|221698.20999999862|
|        France| 196712.8399999999|
|     Australia|137077.26999999973|
|   Switzerland| 55739.40000000004|
|         Spain| 54774.57999999997|
|       Belgium| 40910.95999999998|
```

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|      132572.6199999998|
|   12415.0|     123725.44999999987|
|   14156.0|     113384.13999999985|
|   17511.0|      88125.37999999996|
|   16684.0|      65892.07999999999|
|   13694.0|      62653.10000000003|
|   15311.0|      59419.34000000011|
+----------+----------------------+
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
```

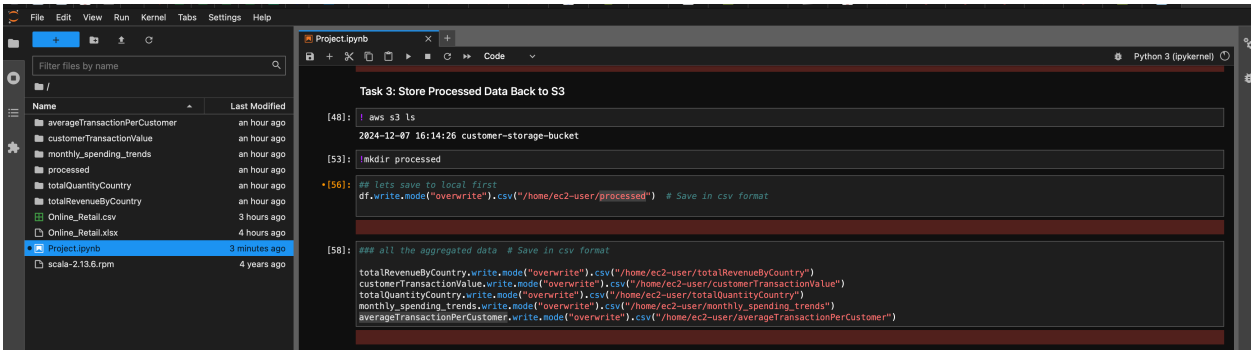
[46]:
```
# Compute Average Transaction Value per Customer
averageTransactionPerCustomer = df.groupBy("CustomerID").agg(
    F.avg("Revenue").alias("Average_Transaction_Value")
).orderBy(F.desc("Average_Transaction_Value"))

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|       872.1299999999999|
|   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 k now 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)

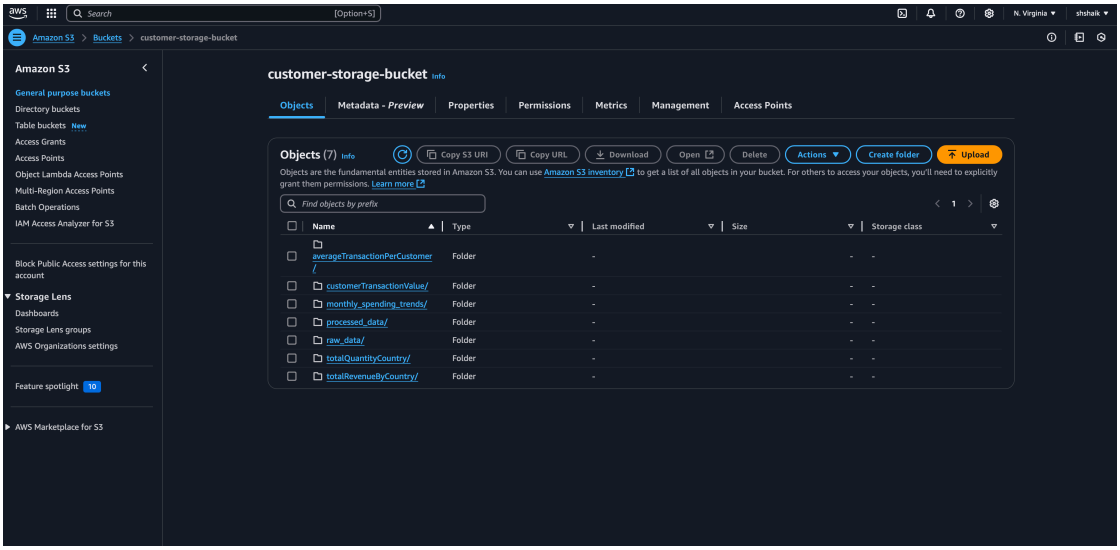


    b. Step 2 – Upload the processed data to a designated S3 location for easy access

        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.crc
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.crc
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.crc
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.crc
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.crc
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.crc
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
```

Successfully Uploaded – (Verification in aws S3)



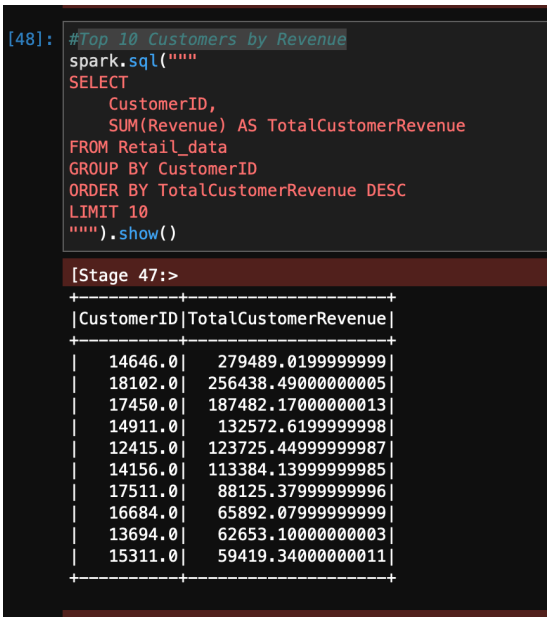
4. Task 4: Data Analysis Using Spark SQL –
    a. Total Revenue by Country: This summarizes the total revenue by country to show which region contributes the most revenue.

```
[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| 55739.40000000004|
```

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

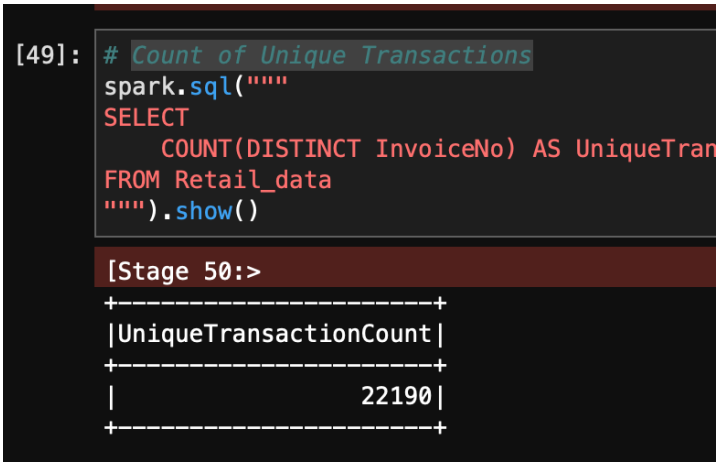
```
[47]: # Monthly Spending Trends

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| 579964.6100000151|
|          2011-4| 426047.8510000125|
|          2011-5|  648251.080000003|
|          2011-6| 608013.1600000106|
|          2011-7|  574238.481000012|
|          2011-8| 616368.0000000092|
|          2011-9| 931440.3719999959|
+---------------+------------------+
```

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



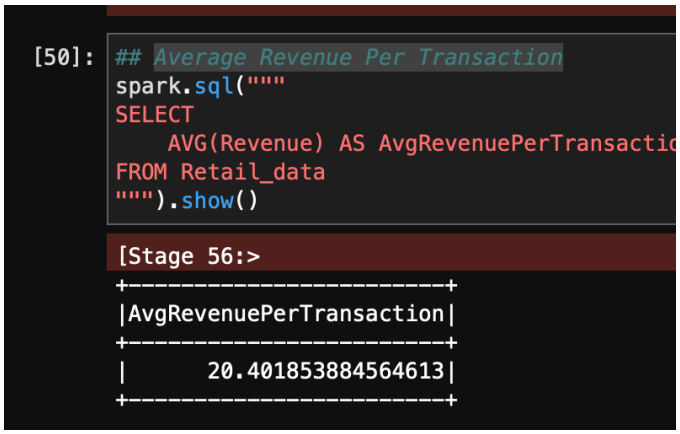
```
[48]: #Top 10 Customers by Revenue
spark.sql("""
SELECT
    CustomerID,
    SUM(Revenue) AS TotalCustomerRevenue
FROM Retail_data
GROUP BY CustomerID
ORDER BY TotalCustomerRevenue DESC
LIMIT 10
""").show()

[Stage 47:>
+----------+--------------------+
|CustomerID|TotalCustomerRevenue|
+----------+--------------------+
|   14646.0|   279489.0199999999|
|   18102.0|  256438.49000000005|
|   17450.0|  187482.17000000013|
|   14911.0|   132572.6199999998|
|   12415.0|  123725.44999999987|
|   14156.0|  113384.13999999985|
|   17511.0|   88125.37999999996|
|   16684.0|   65892.07999999999|
|   13694.0|   62653.10000000003|
|   15311.0|   59419.34000000011|
+----------+--------------------+
```

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

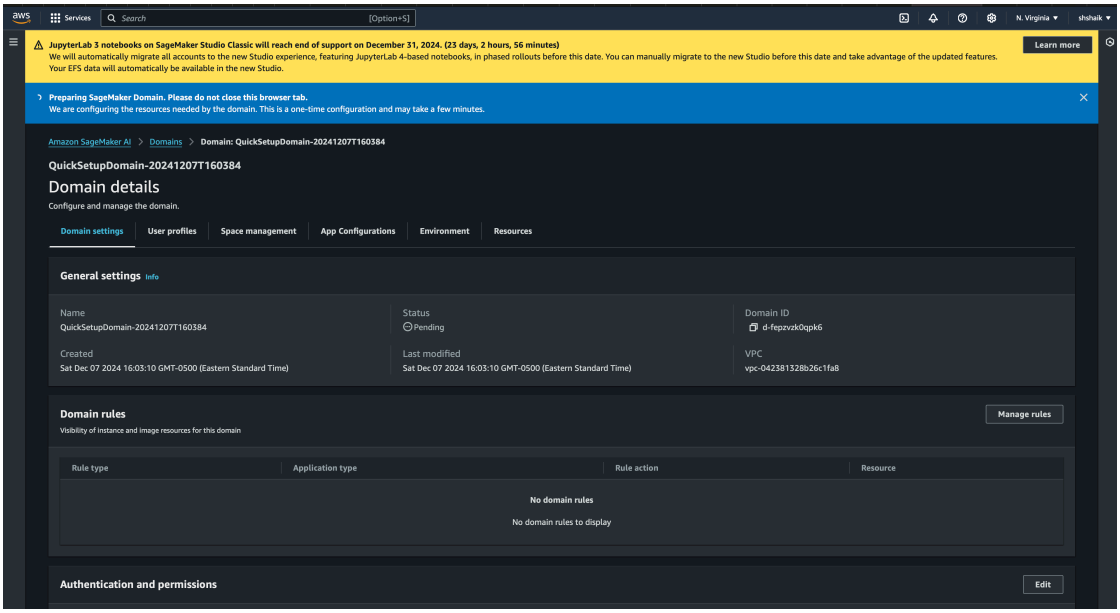


```
[49]: # Count of Unique Transactions
spark.sql("""
SELECT
    COUNT(DISTINCT InvoiceNo) AS UniqueTran
FROM Retail_data
""").show()

[Stage 50:>
+----------------------+
|UniqueTransactionCount|
+----------------------+
|                 22190|
+----------------------+
```

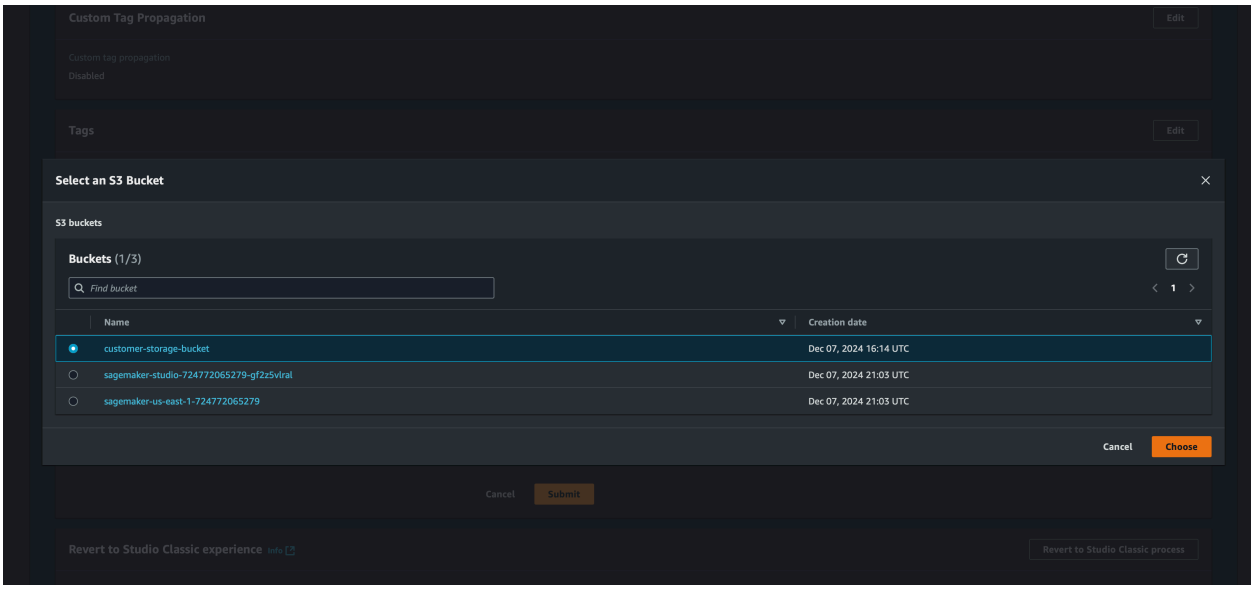
e. Average Revenue Per Transaction: This brings into view how much return is usually obtained per transaction.



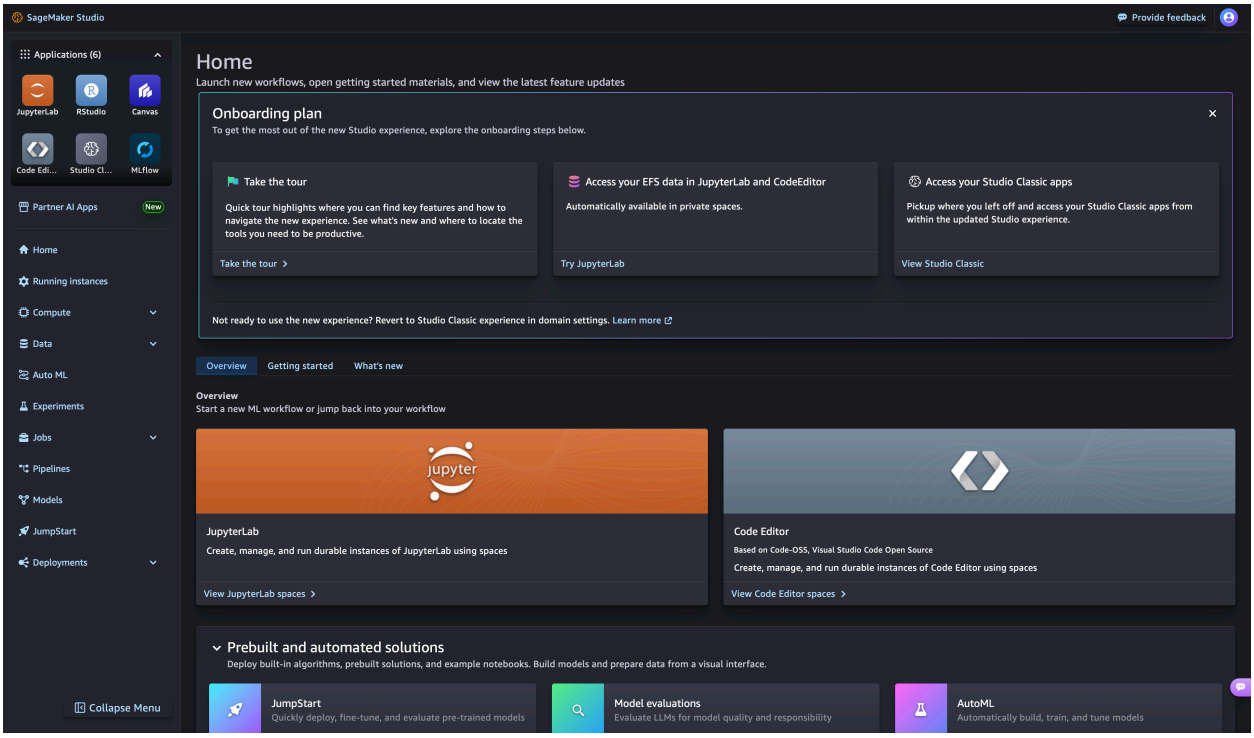
5. Task 5: Machine Learning with AWS SageMaker Autopilot – Created domain for sagemaker



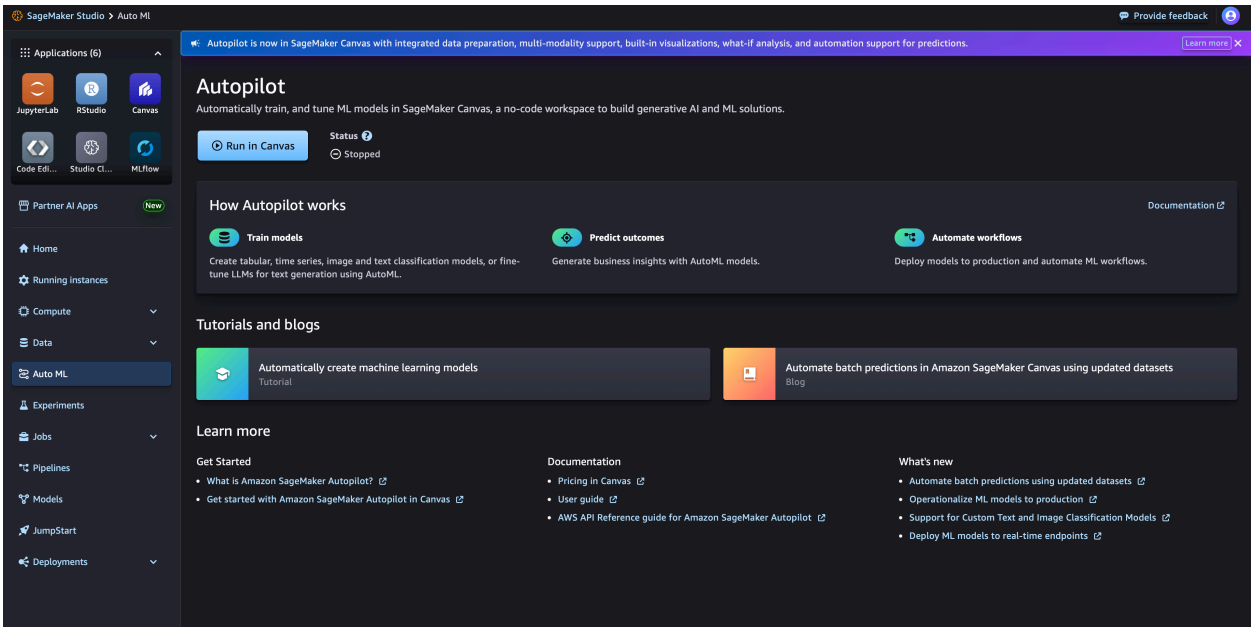
Select the S3 bucket while configuring



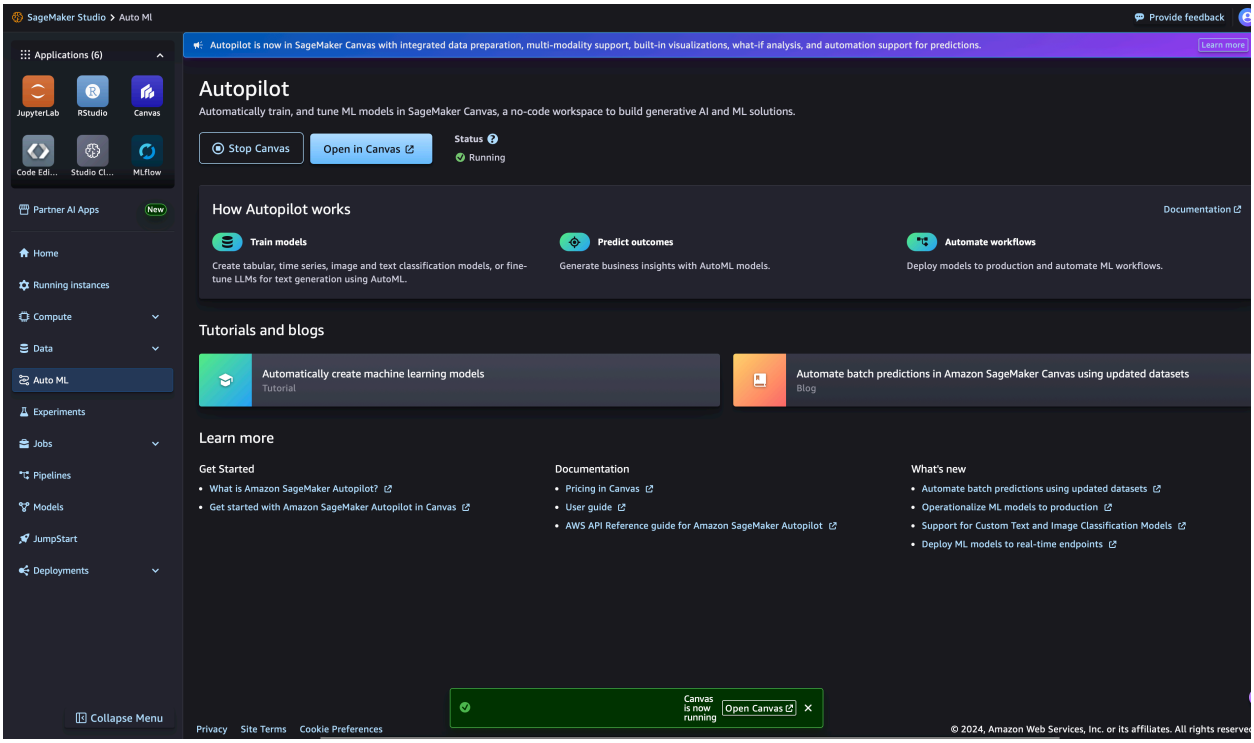
Launch SageMaker-



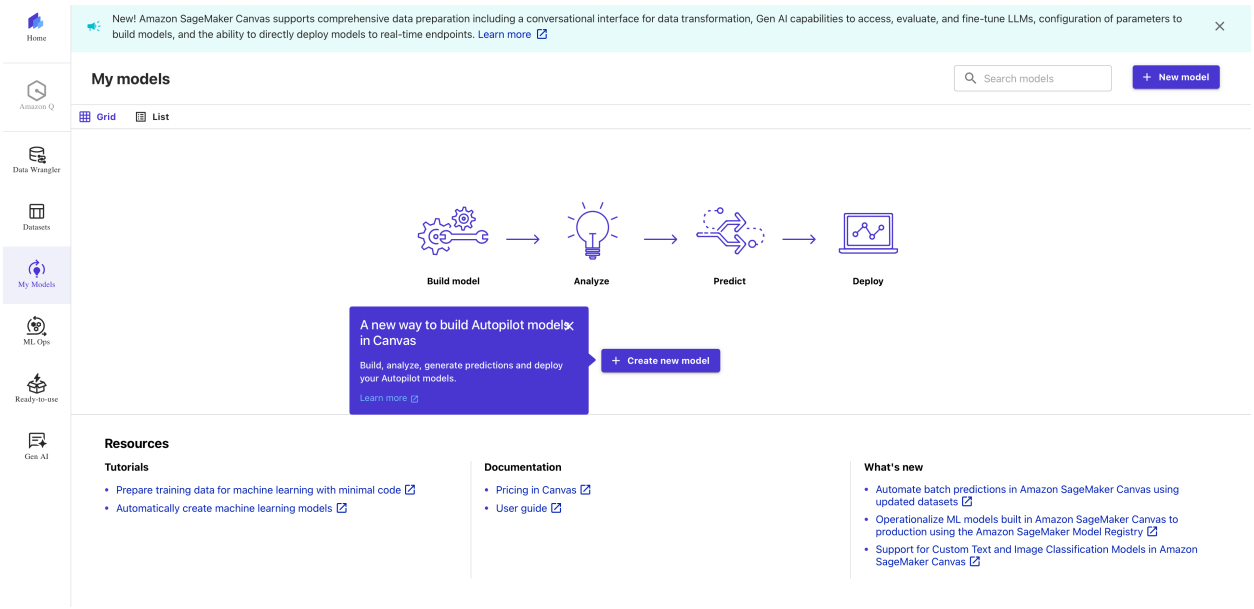
Select the AutoML, from the column in the left and Start the canvas -



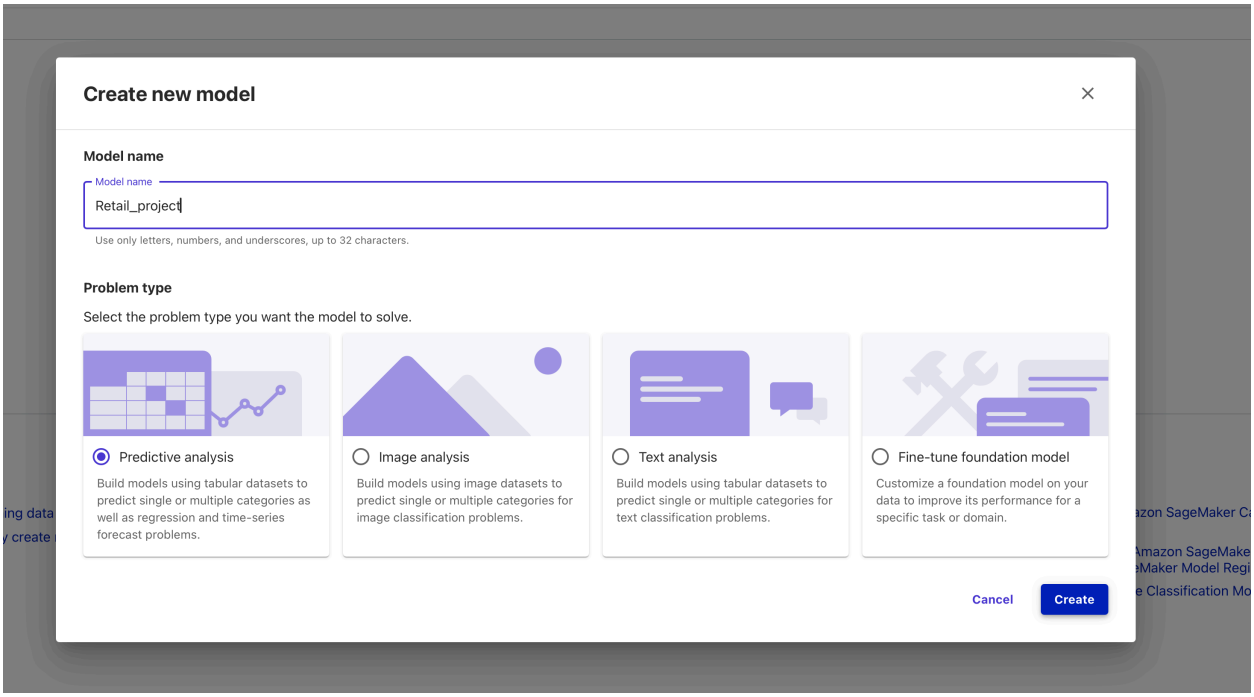
Now, its time to Open in Canvas,



AWS CANVAS LAUNCH –

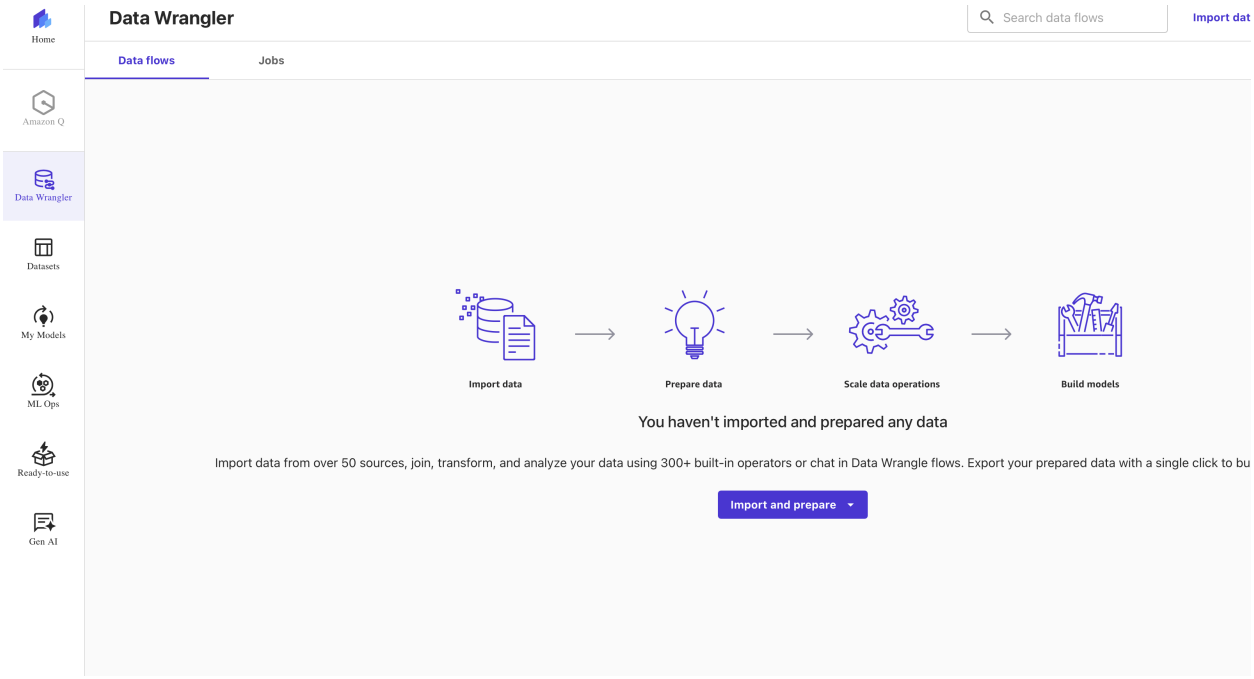


Click, create new model

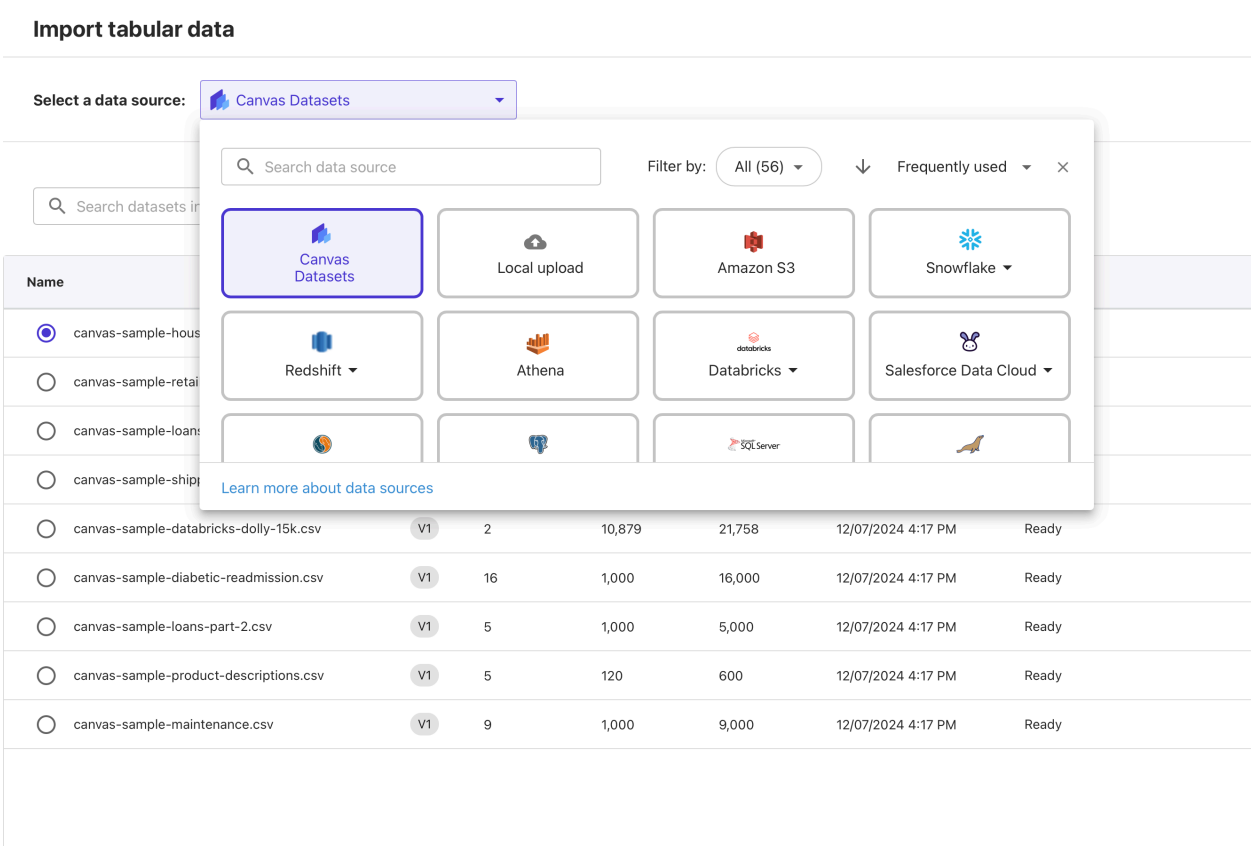


a. Import Processed Data:

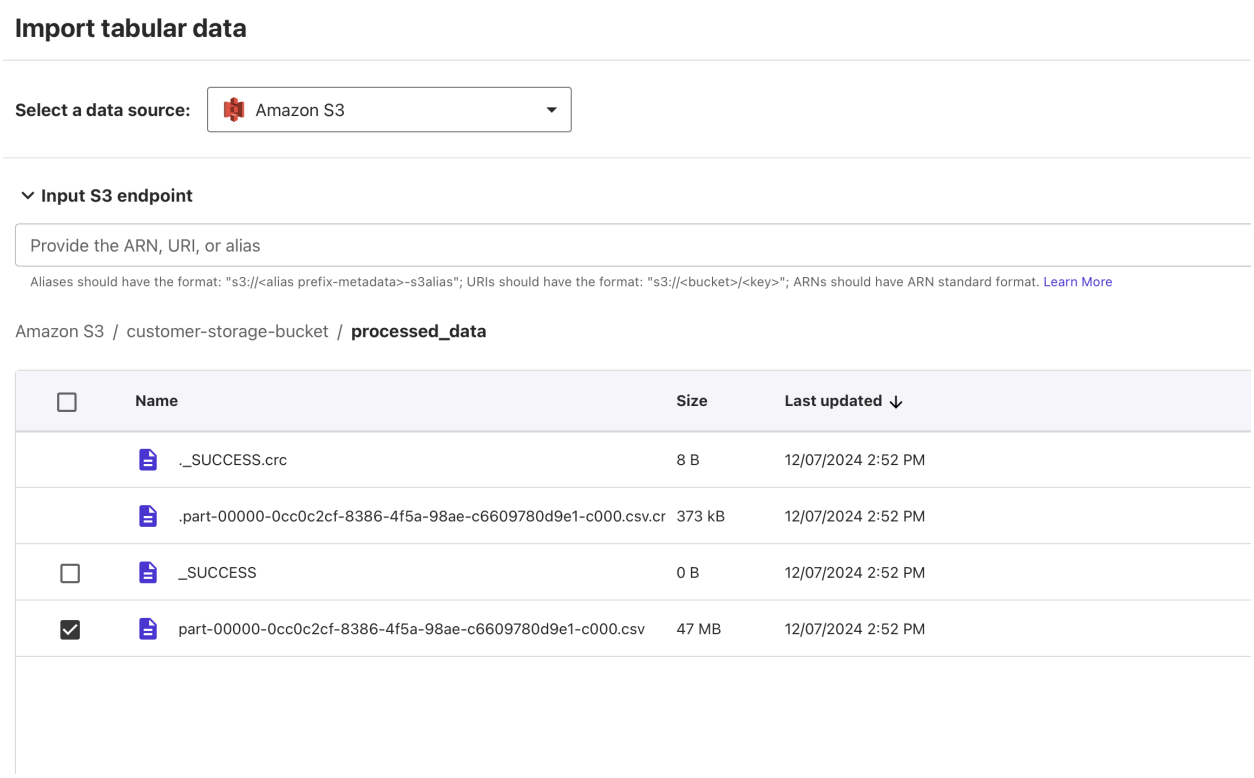
DATAWRANGLER -> IMPORT AND PREPARE


Home
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

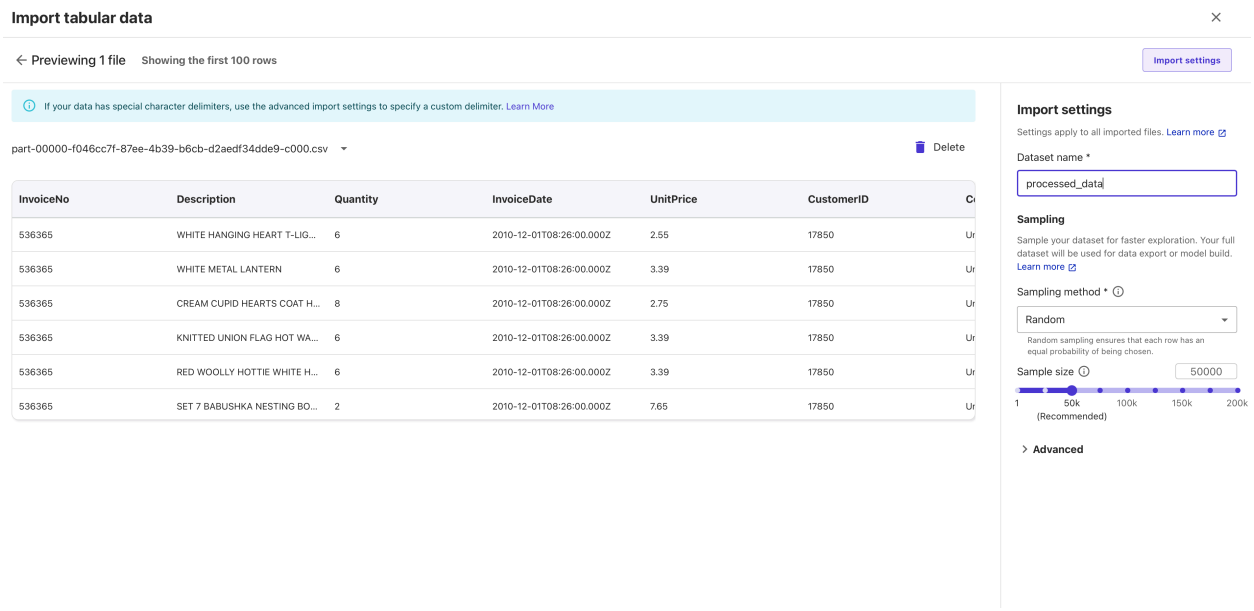
Select s3 –


Import tabular data
Select a data source:
Canvas Datasets
Search data source
Filter by:
All (56)
Frequently used
Canvas Datasets
Local upload
Amazon S3
Snowflake
Redshift
Athena
Databricks
Salesforce Data Cloud
Learn more about data sources
Search datasets in
Name
canvas-sample-hous
canvas-sample-retai
canvas-sample-loans
canvas-sample-shipp
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

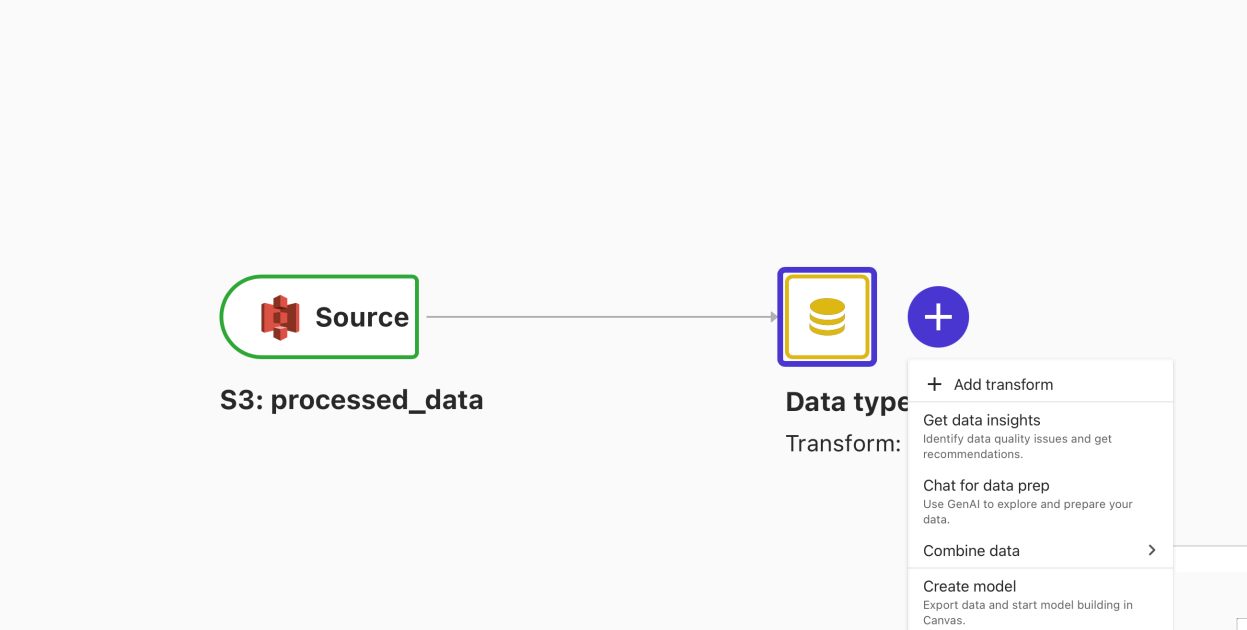
Select the processed file –



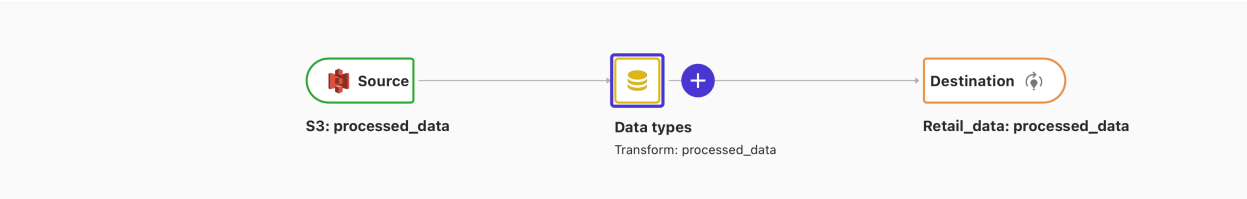
Data Preview -



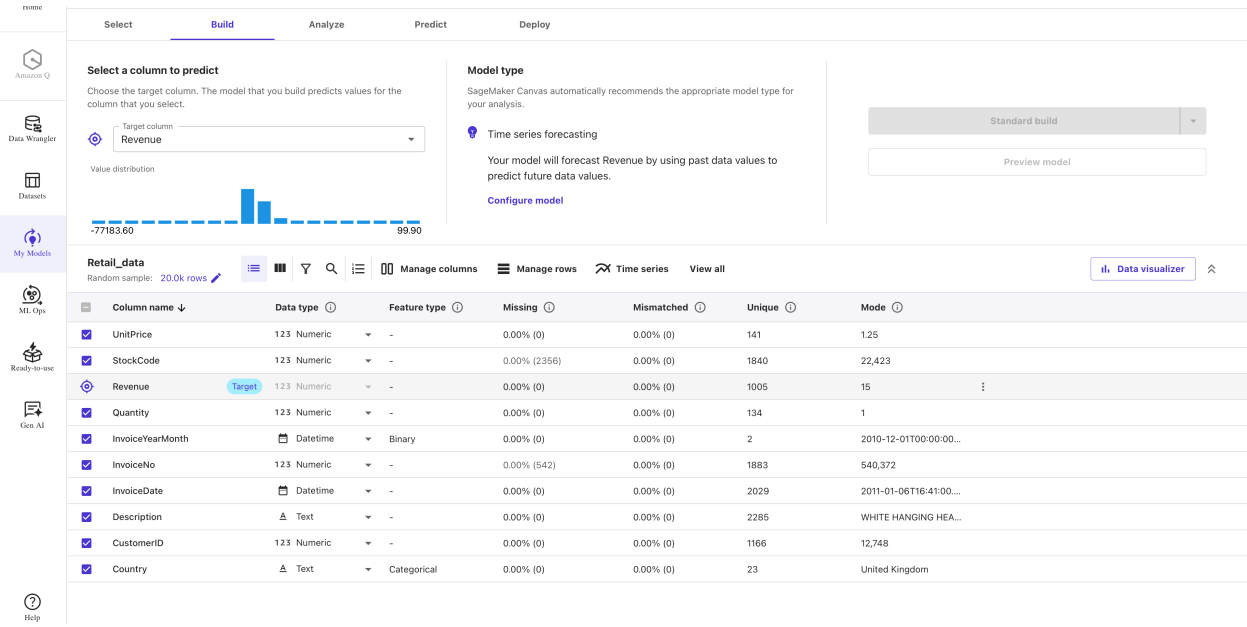
Data Imported Successfully, its time to create model (click on create model as shown below)



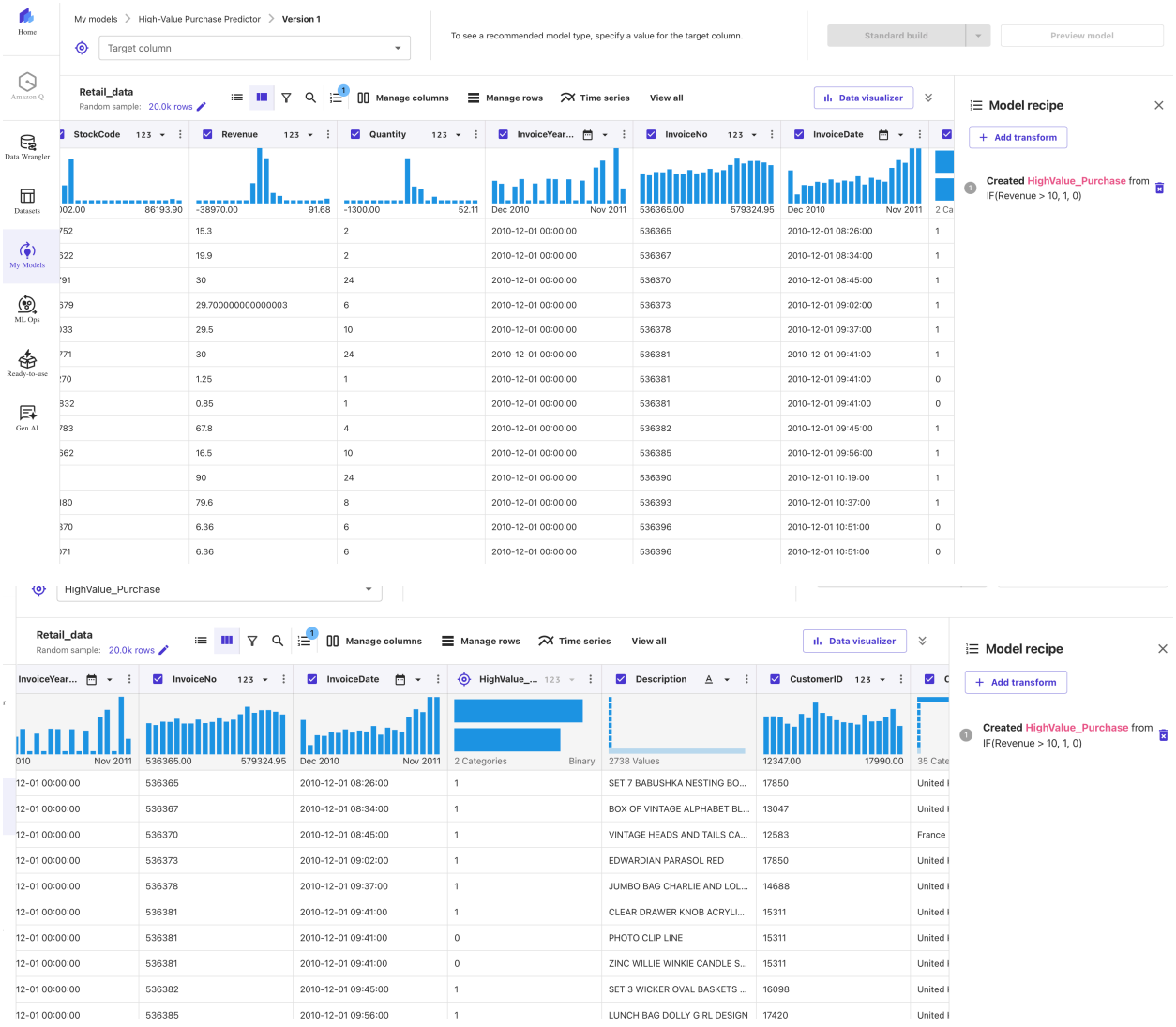
b. Run Autopilot Experiment:



This page opens up when we start creating model -

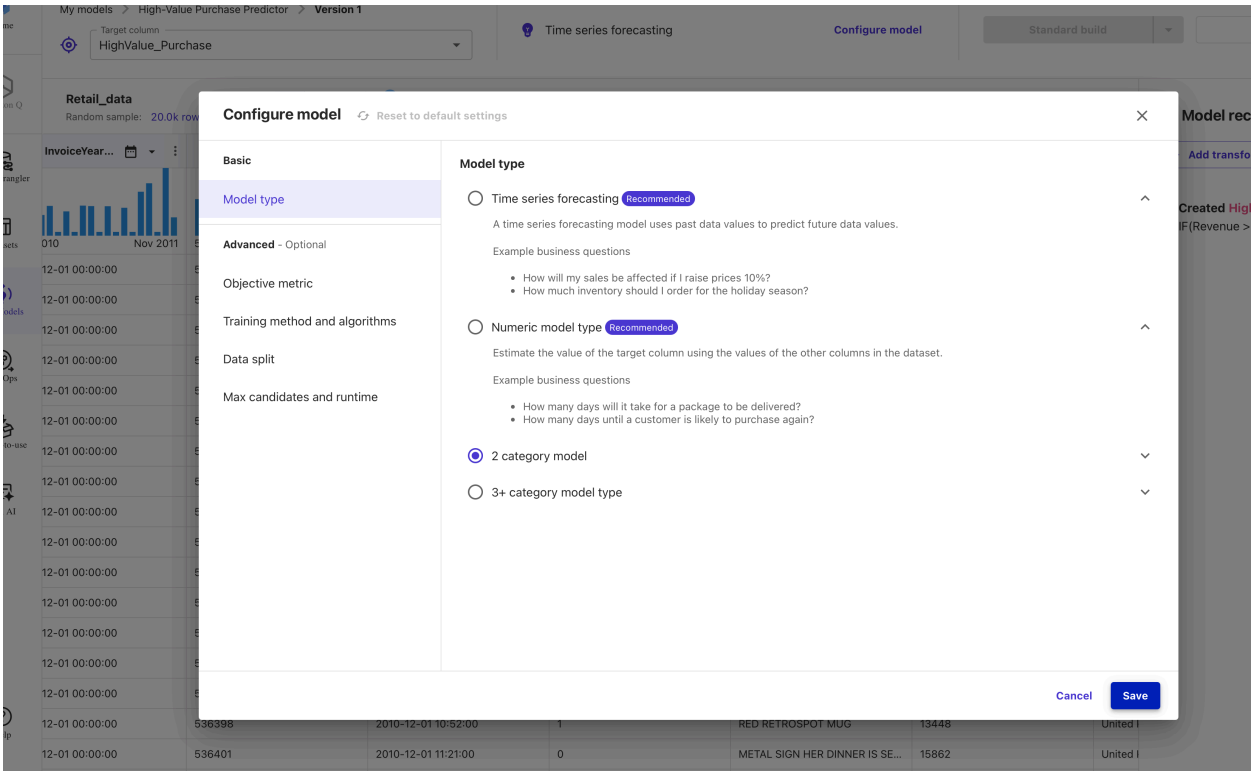


Created a new column called HighValue_Purchase with condition on Revenue.

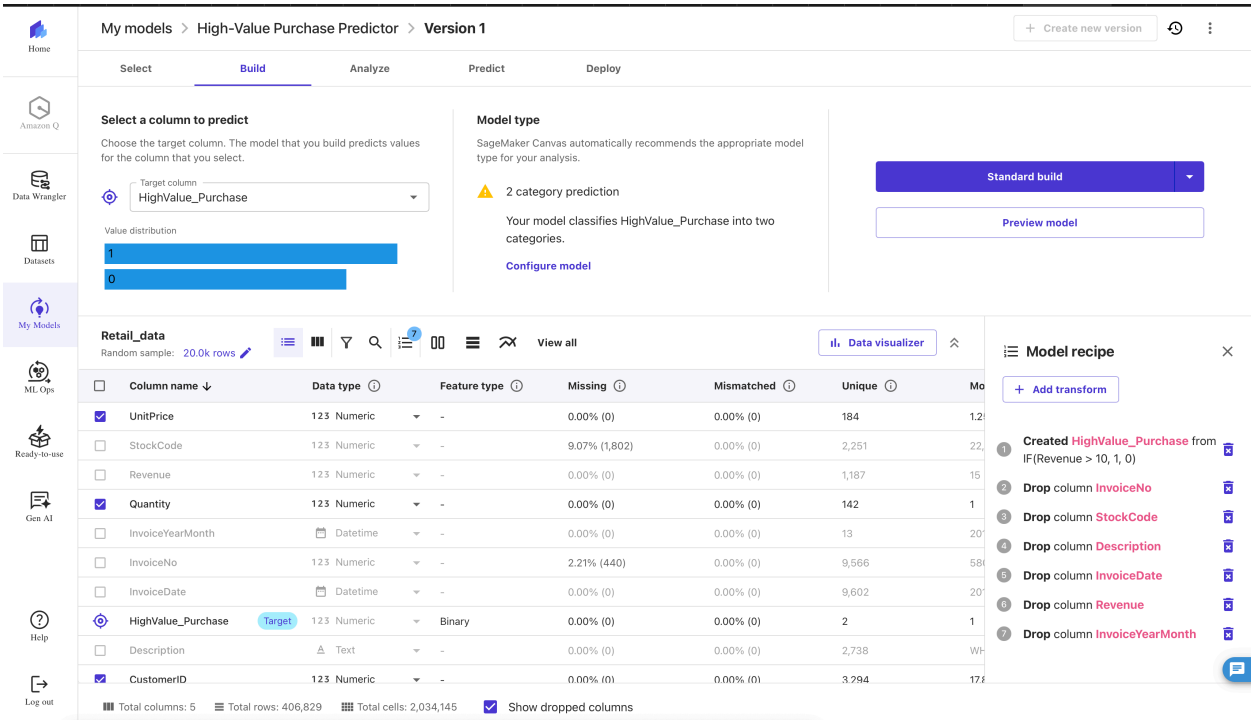


As we are planning to predict HighValue_Purchase, which is a classification problem, i.e 1 or 0. (2 category model)

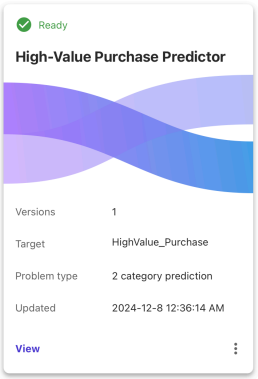
Lets configure ml model accordingly as shown below,



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



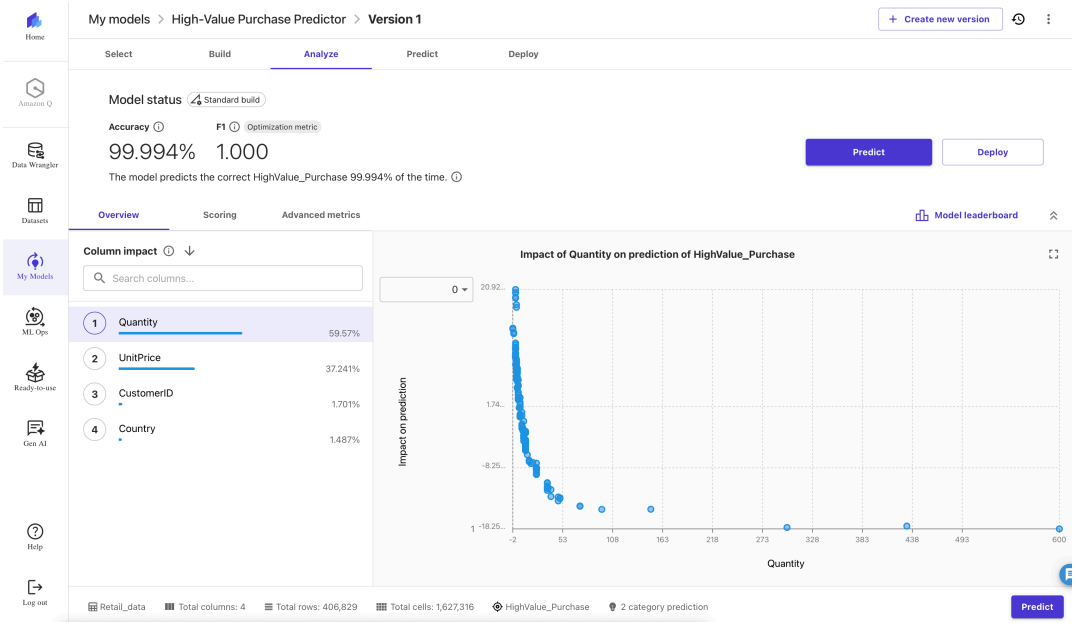
c. Screenshots: when we go through My models section -

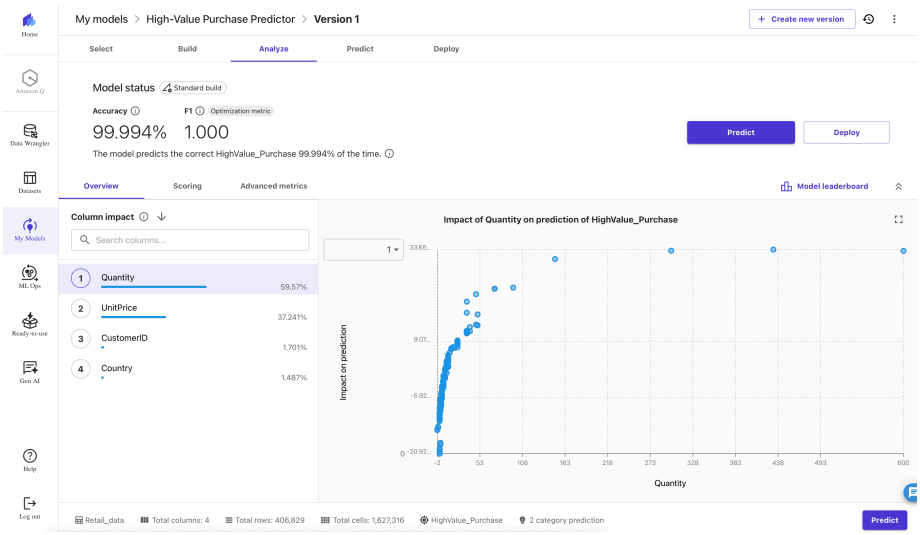


d. Review Results: (After successfully building the model)

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

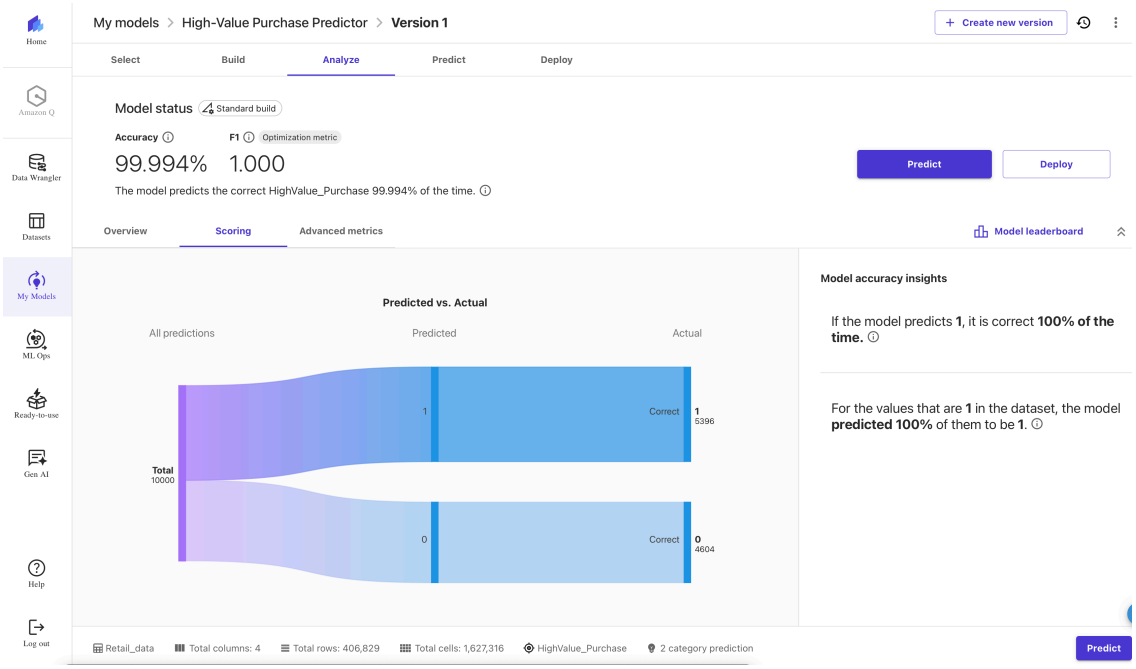
The model's predictive power relies on four main features, ranked by impact:

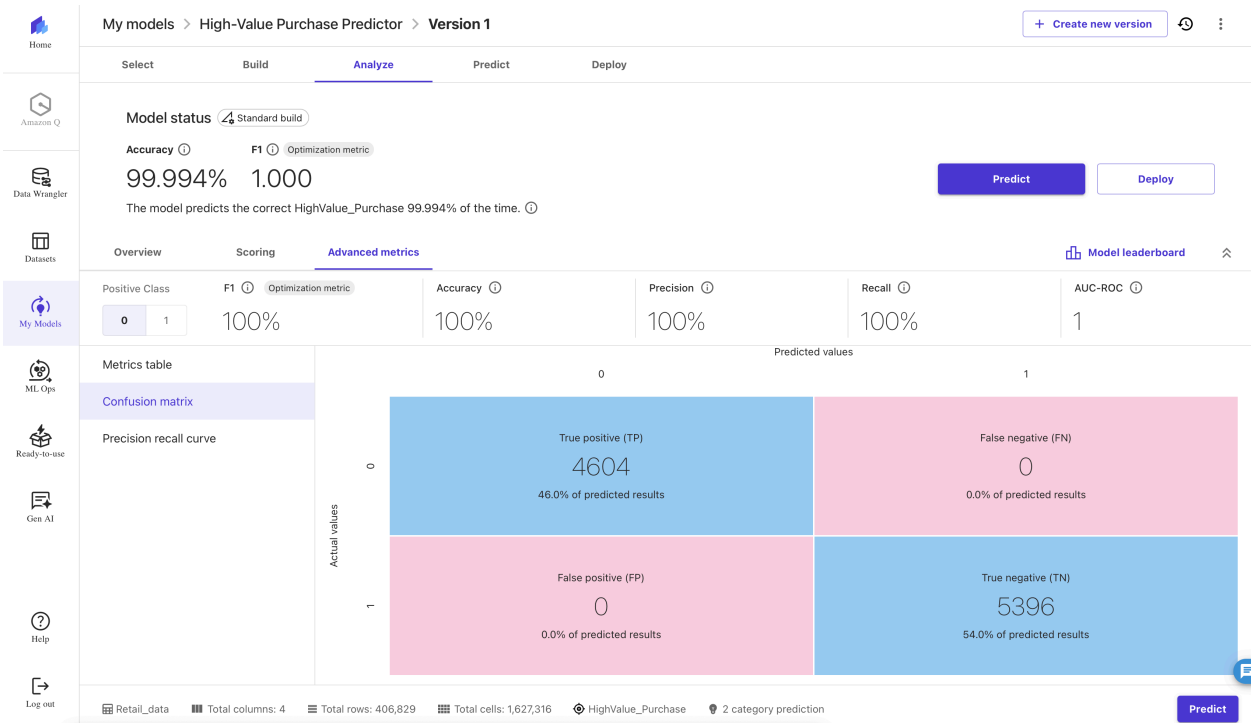
Quantity (59.67% impact)

UnitPrice (37.24% impact)

CustomerID (1.70% impact)

Country (1.457% impact)





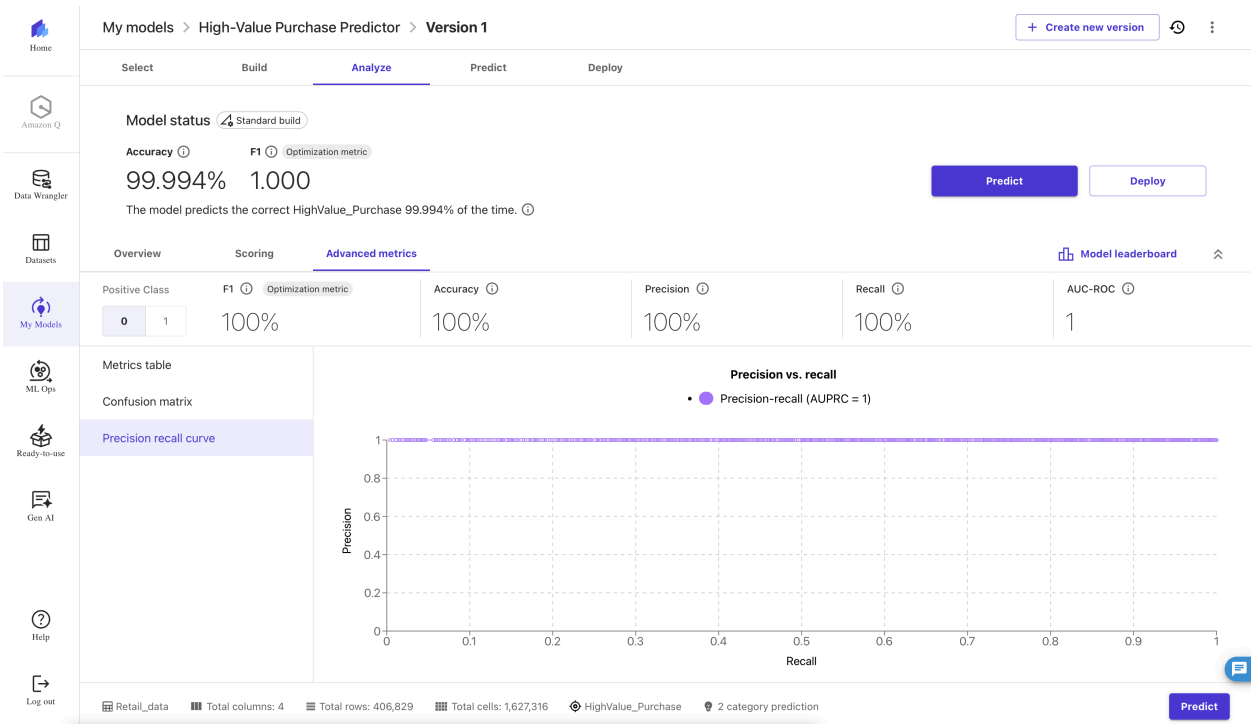
Prediction Analysis –

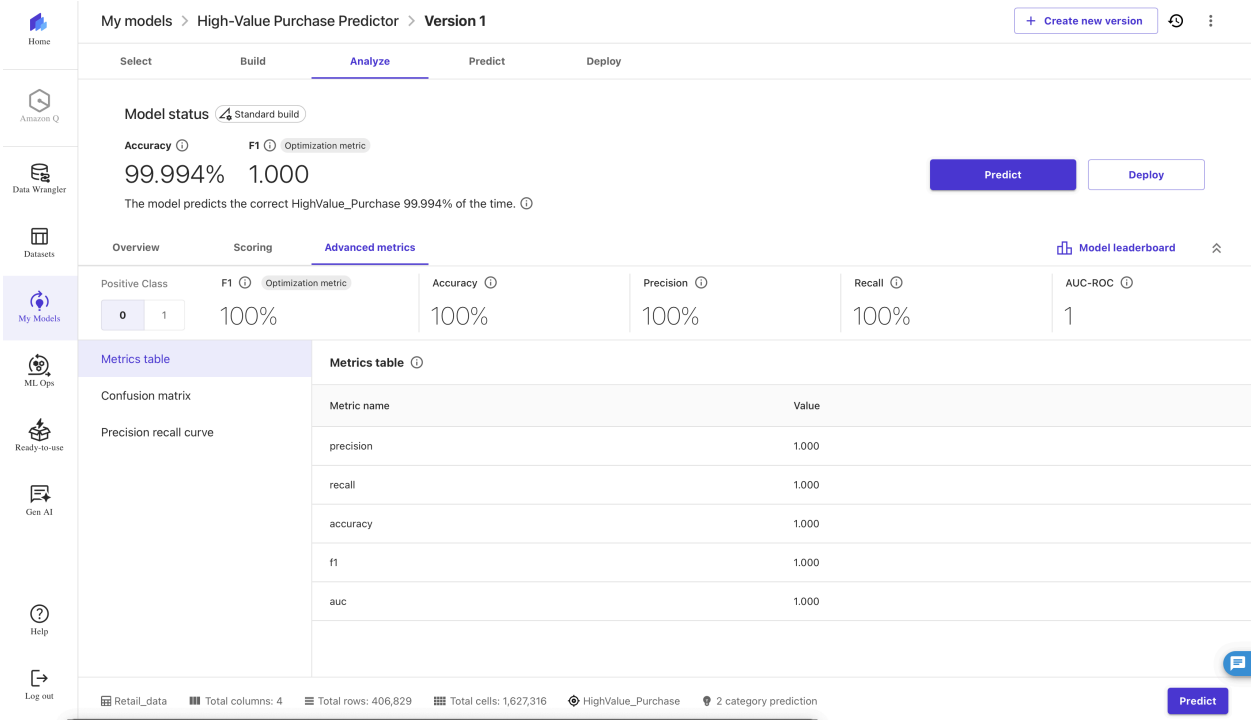
- The model correctly classified 4,604 true positives and 5,396 true negatives.
- Zero false positives and false negatives, indicating perfect classification4
- For all values labeled as 1 in the dataset, the model achieved 100% prediction accuracy.

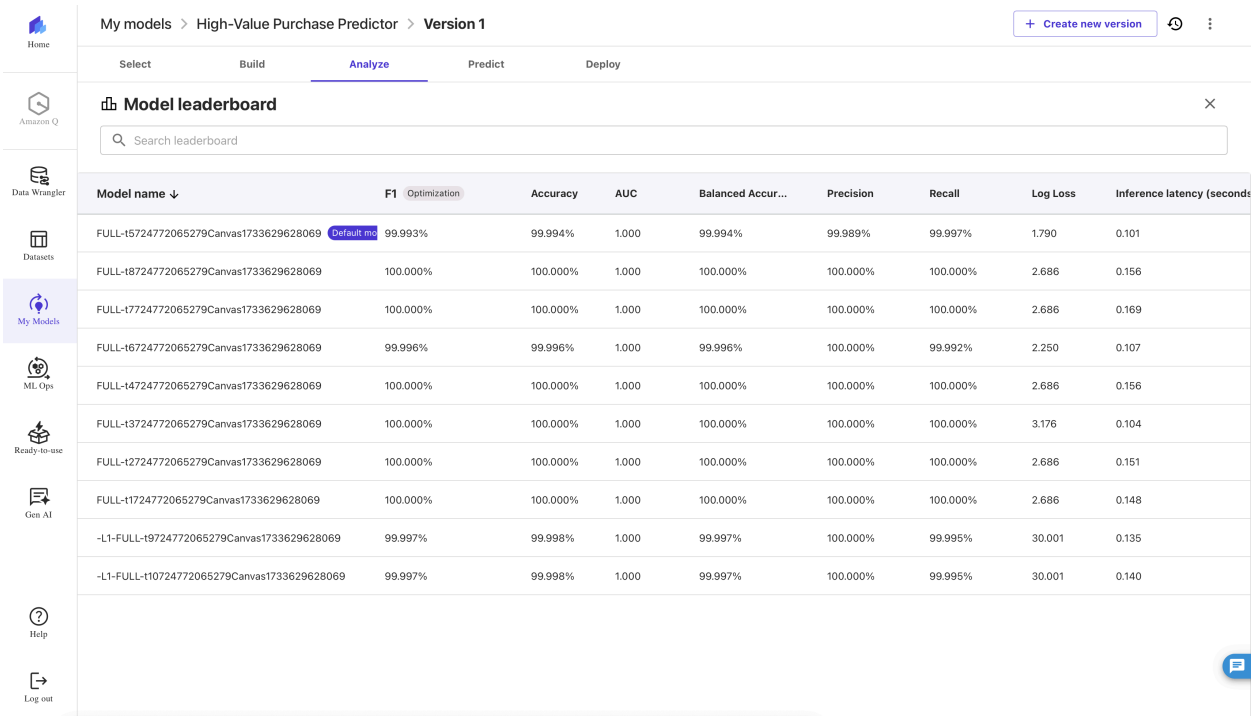
Impact Analysis -

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







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

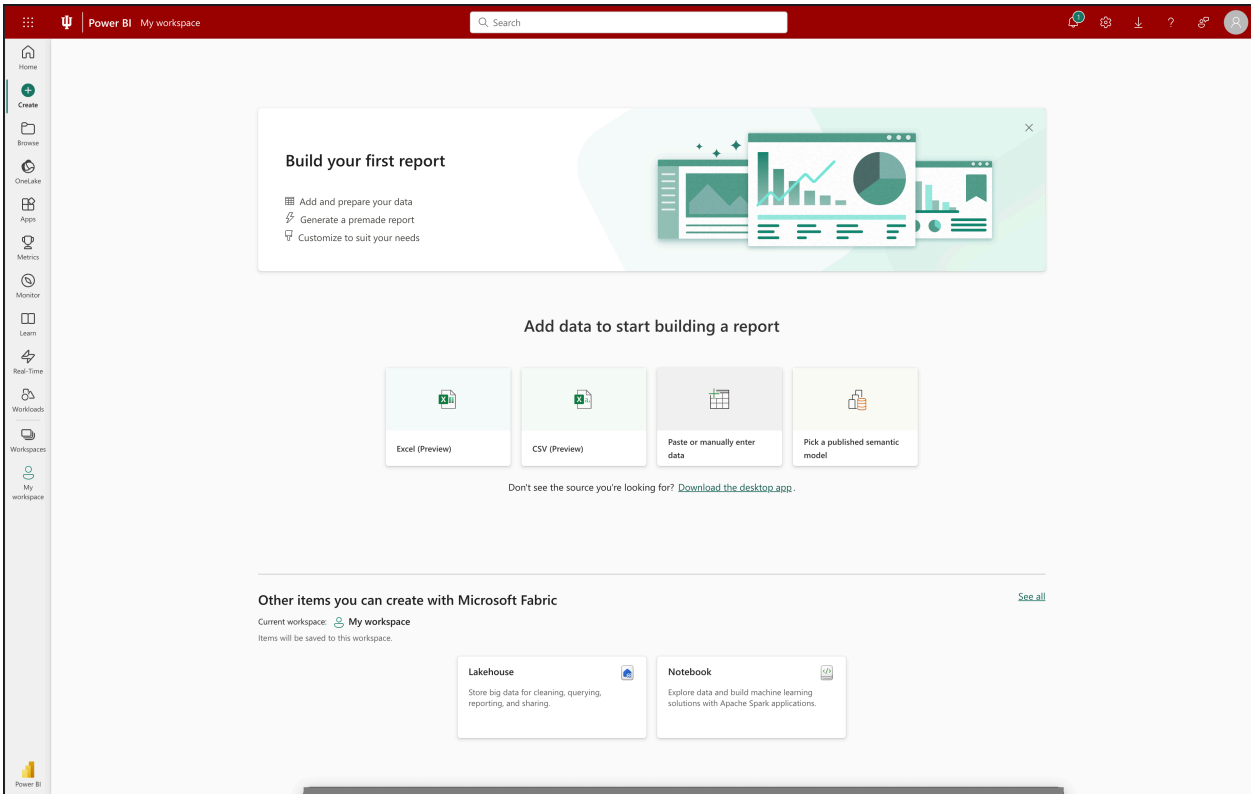
2-category prediction for HighValue_Purchase

4) Visualization –

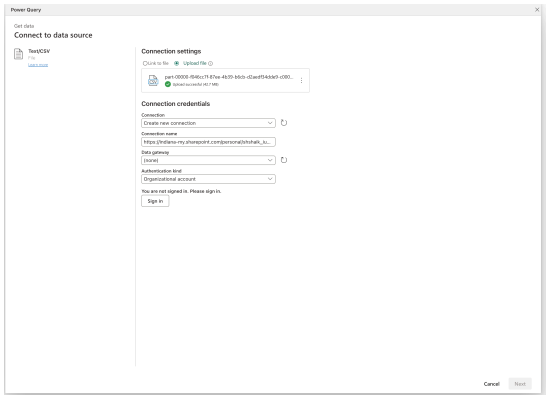
Gathered all files and exported to local – (As PowerBi doesn't support S3)

```
(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-81e8-70aeffc690b1-c000.csv                                    100%  107KB 613.6KB/s   00:00
.part-00000-21780367-3d60-4e5c-81e8-70aeffc690b1-c000.csv.crc                               100%  868    12.5KB/s   00:00
._SUCCESS.crc                                                                               100%    8     0.1KB/s   00:00
(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-a1d3-42da-b0e0-0dd69bc314eb-c000.csv                                    100%   97KB 203.9KB/s   00:00
.part-00000-97f63484-a1d3-42da-b0e0-0dd69bc314eb-c000.csv.crc                               100%  784    11.0KB/s   00:00
._SUCCESS.crc                                                                               100%    8     0.1KB/s   00:00
(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-00000-010364c0-0a71-4c10-8112-3230afae36cc-c000.csv                                    100%  359     5.4KB/s   00:00
.part-00000-010364c0-0a71-4c10-8112-3230afae36cc-c000.csv.crc                               100%   12     0.2KB/s   00:00
._SUCCESS.crc                                                                               100%    8     0.1KB/s   00:00
part-00000-010364c0-0a71-4c10-8112-3230afae36cc-c000-checkpoint.csv                         100%  359     5.1KB/s   00:00
(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                                    100%   43MB   3.6MB/s   00:11
.part-00000-f046cc7f-87ee-4b39-b6cb-d2aedf34dde9-c000.csv.crc                               100%  342KB   1.8MB/s   00:00
._SUCCESS.crc                                                                               100%    8     0.1KB/s   00:00
(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                                    100%  560     8.5KB/s   00:00
.part-00000-9f6650c7-c4c8-4dcc-a483-d4b38f51ab95-c000.csv.crc                               100%   16     0.2KB/s   00:00
._SUCCESS.crc                                                                               100%    8     0.1KB/s   00:00
(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                                    100%  952    12.5KB/s   00:00
.part-00000-e67377ec-eb59-40ab-85d4-47b2c1b61305-c000.csv.crc                               100%   16     0.2KB/s   00:00
._SUCCESS.crc                                                                               100%    8     0.1KB/s   00:00
(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                                                                           100%   46MB   7.5MB/s   00:06
(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                                                                               100%   51KB 352.8KB/s   00:00
(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 %
```

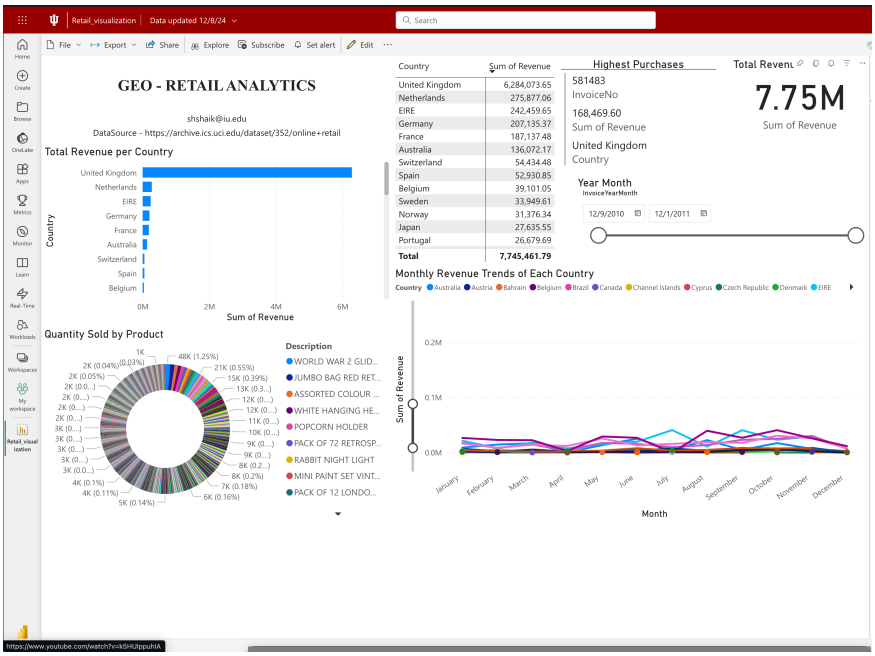
Task: Create Dashboards



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

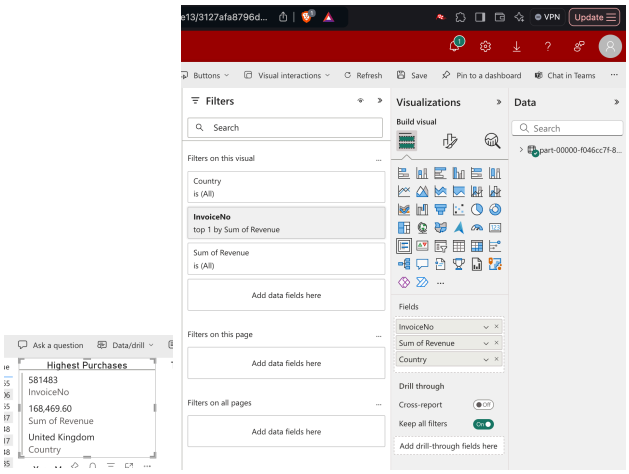


2. Design a dashboard with at least 4 insightful visualizations.



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)

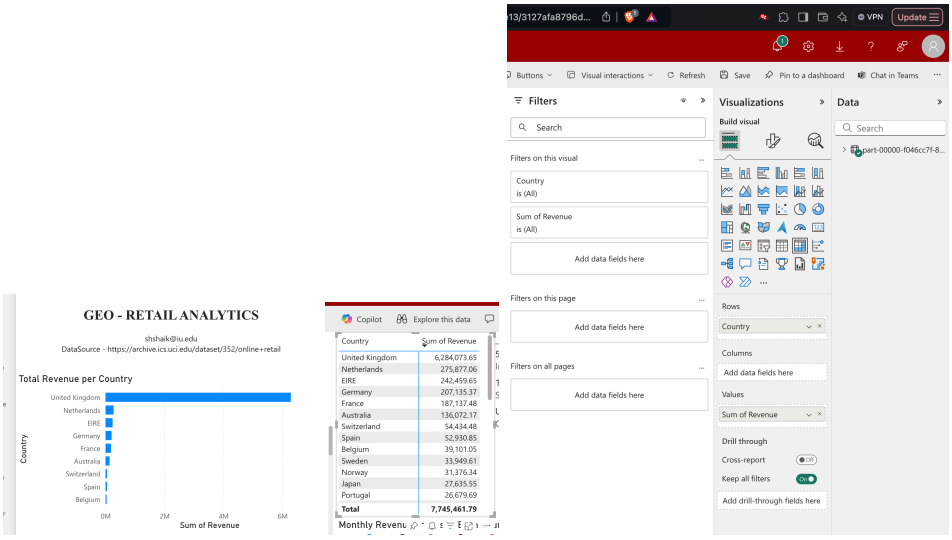
Highest Purchases – (This displays the highest invoice billed ever)



Country and their total revenue –

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

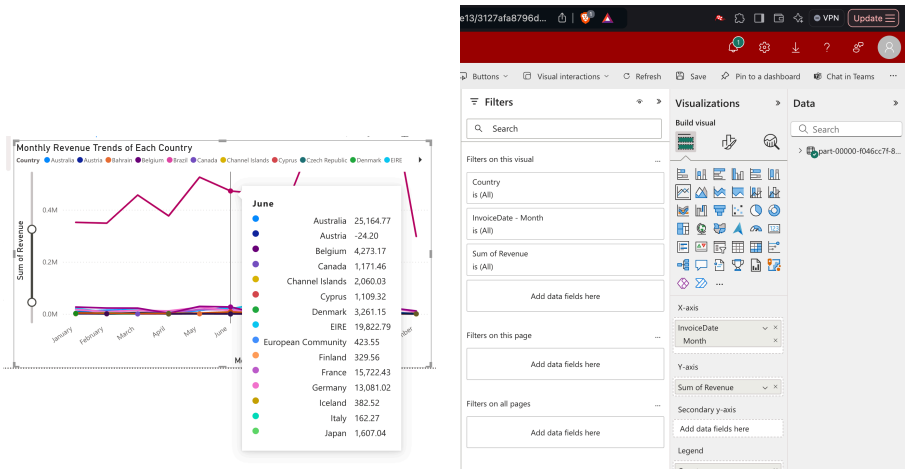
Total revenue of 7.75M across all markets displayed prominently



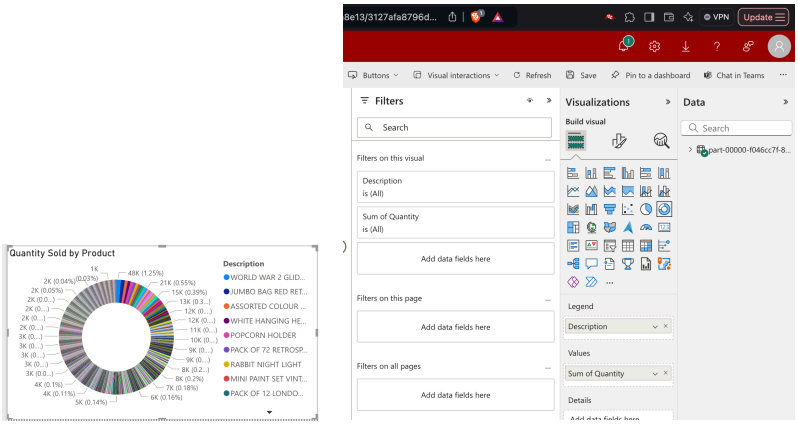
Monthly Revenue trends for each country –

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------