Big Data Applications
Mini Project

Alvan Dmello **IU ID: 2001251305**

Introduction:

The goal of this project is to analyze and derive actionable insights from the Online Retail dataset, which contains transaction data from a UK-based, non-store online retailer. The dataset includes sales transactions between December 1, 2010, and September 9, 2011, from a company that primarily sells unique all-occasion gifts. This analysis aims to provide valuable insights into the company's sales performance, customer behavior, and product demand patterns during this period. The dataset consists of the following columns:

- **Invoice number**: A unique identifier for each purchase.
- **Invoice date**: The date the invoice was generated.
- **Stock code**: The identifier of the products purchased.
- **Description**: A brief description of the products.
- **Quantity**: The quantity of items purchased.
- **Unit price**: The price of a single unit of the product.
- **Customer ID**: The unique identifier for the customer.
- **Country**: The country of the customer.

Methodology:

The Project workflow is as follows:

1. Dataset Selection

As mentioned in the introduction, it is an Online Retail dataset, which contains transaction data from a UK-based, non-store online retailer. The dataset was obtained from Kaggle. The dataset contains various columns which allow for cleaning and also columns such as quantity, invoice date and unit price, which allow for transformation and aggregation.

As the free tier had a limited processing capacity, the dataset has been sampled down to fewer rows which help with faster processing. Here’s a snippet of the data.

	A	B	C	D	E	F	G	H	I
1	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerI	Country	
2	543451	22167	OVAL WALL MIRROR DIAMANTE	1	02-08-2011 12:13	19.96		United Kingdom	
3	577522	22944	CHRISTMAS METAL POSTCARD WITH BELLS	6	11/20/2011 13:23	0.39	15988	United Kingdom	
4	580367	22284	HEN HOUSE DECORATION	1	12-02-2011 16:39	3.29		United Kingdom	
5	576245	23569	TRADTIONAL ALPHABET STAMP SET	4	11/14/2011 13:40	4.95	12553	France	
6	578293	22086	PAPER CHAIN KIT 50'S CHRISTMAS	12	11/23/2011 14:36	2.95	15640	United Kingdom	
7	573248	23247	BISCUIT TIN 50'S CHRISTMAS	2	10/28/2011 12:09	2.89	14498	United Kingdom	
8	C569985	22617	BAKING SET SPACEBOY DESIGN	-3	10-06-2011 19:51	4.95	15365	United Kingdom	
9	C557971	37449	CERAMIC CAKE STAND + HANGING CAKES	-1	6/24/2011 10:15	9.95	18118	United Kingdom	
10	549586	21213	PACK OF 72 SKULL CAKE CASES	4	04-11-2011 10:00	2.08		United Kingdom	
11	566301	21165	BEWARE OF THE CAT METAL SIGN	1	09-11-2011 16:06	1.69	16474	United Kingdom	
12	573277	23419	HOME SWEET HOME BOTTLE	1	10/28/2011 13:18	2.08	14606	United Kingdom	
13	559518	22893	MINI CAKE STAND T-LIGHT HOLDER	6	07-08-2011 16:11	0.83		United Kingdom	
14	551414	85099B	JUMBO BAG RED RETROSPOT	10	4/28/2011 13:35	2.08	15622	United Kingdom	
15	549130	20676	RED RETROSPOT BOWL	6	04-06-2011 15:02	1.25	14701	United Kingdom	
16	567461	84030e	ENGLISH ROSE HOT WATER BOTTLE	1	9/20/2011 12:31	8.29		United Kingdom	
17	572909	23360	SET 8 CANDLES VINTAGE DOILY	4	10/26/2011 15:48	1.95	15821	United Kingdom	
18	546762	22431	WATERING CAN BLUE ELEPHANT	1	3/16/2011 14:12	1.95	17961	United Kingdom	
19	567673	21155	RED RETROSPOT PEG BAG	1	9/21/2011 15:43	4.96		United Kingdom	
20	571667	22197	POPCORN HOLDER	4	10/18/2011 13:04	0.85	14554	United Kingdom	
21	557153	23241	TREASURE TIN GYMKHANA DESIGN	2	6/17/2011 11:07	2.08	13735	United Kingdom	
22	553228	21231	SWEETHEART CERAMIC TRINKET BOX	12	5/16/2011 10:48	1.25	16496	United Kingdom	
23	578347	23370	SET 36 COLOURING PENCILS DOILY	4	11/24/2011 9:26	2.46		United Kingdom	
24	549744	21669	BLUE STRIPE CERAMIC DRAWER KNOB	36	04-12-2011 10:30	1.25	15240	United Kingdom	
25	567668	21326	AGED GLASS SILVER T-LIGHT HOLDER	1	9/21/2011 15:29	1.63		United Kingdom	
26	563712	23256	CHILDRENS CUTLERY SPACEBOY	4	8/18/2011 15:44	4.15	12680	France	
27	580879	84519B	CARROT CHARLIE+LOLA COASTER SET	1	12-06-2011 12:18	1.25	17346	United Kingdom	

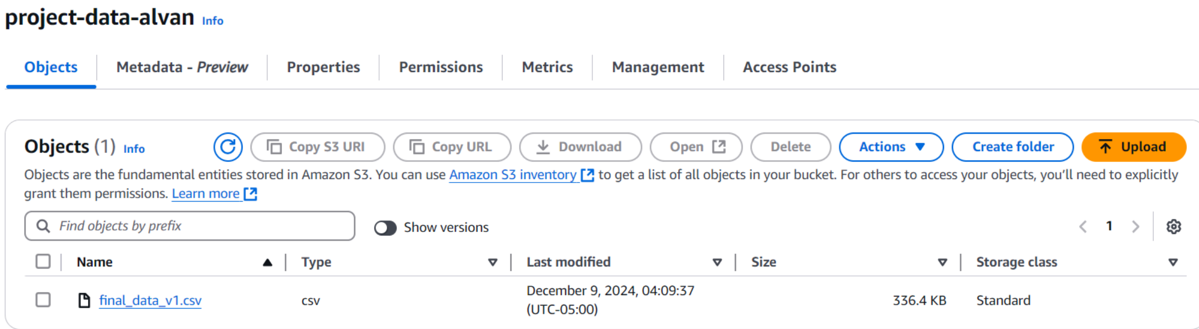
2. Environment Setup

1. AWS S3 for data storage

a. Step 1: Create an S3 bucket to store both raw and processed data

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

After creating the S3 bucket ‘project-data-alvan’, the raw dataset was uploaded into the bucket.

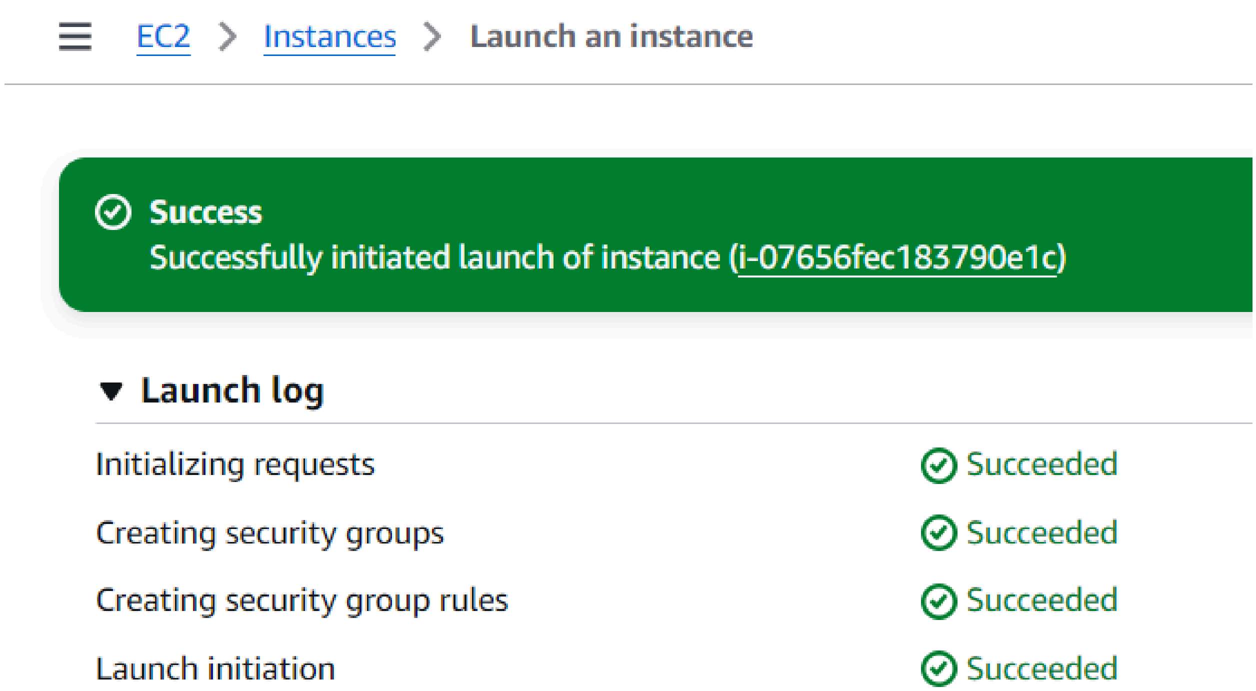


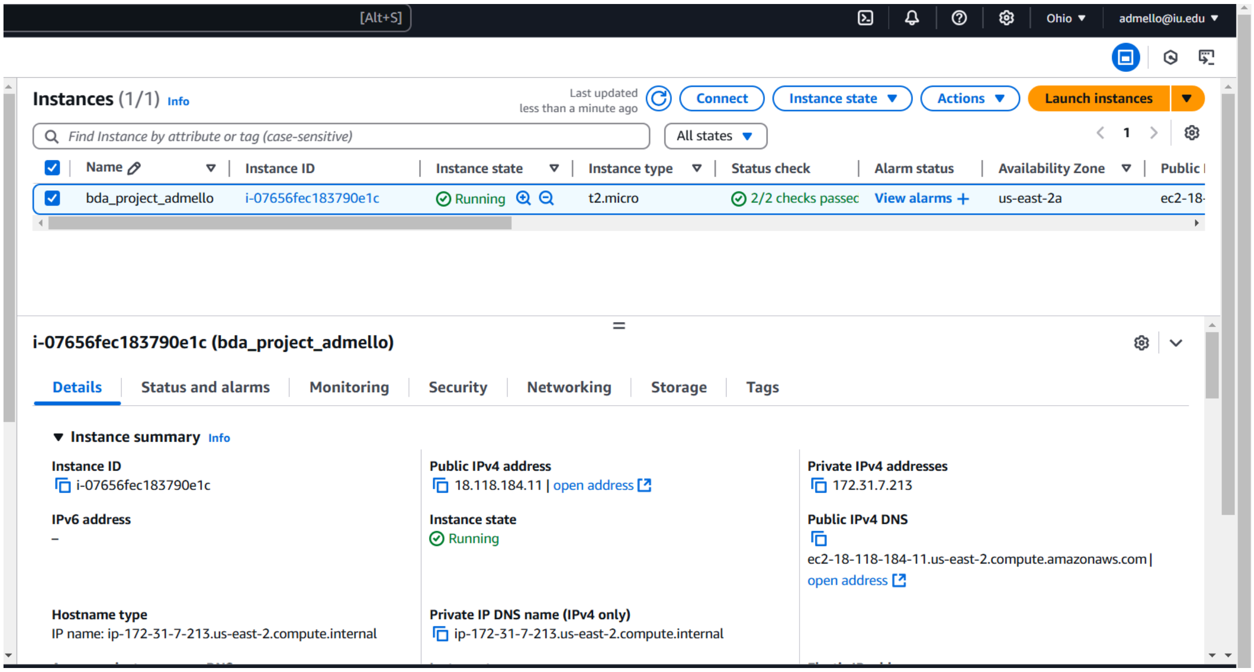
2. Linux Environment with PySpark

a. Step 1: Set up a Linux-based environment, either locally or using an AWS EC2 instance.

Recommendation: Use an AWS EC2 instance for better scalability and AWS integration.

Created an instance ‘bda_project_admello’ on EC2





b. Step 2: Install PySpark for distributed data processing.

Before installing PySpark, there were multiple dependencies such as Java and Python

```
ubuntu@ip-172-31-7-213:~$ sudo apt update && sudo apt upgrade -y
Hit:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:5 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:6 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [498 kB]
Get:8 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:9 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:10 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:11 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:12 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:13 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:14 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [675 kB]
Get:15 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [158 kB]
Get:16 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [132 kB]
Get:17 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [722 kB]
Get:18 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [215 kB]
Get:19 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [309 kB]
Get:20 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [19.9 kB]
Get:21 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [498 kB]
Get:22 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted Translation-en [95.7 kB]
Get:23 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
Get:24 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [16.0 kB]
Get:25 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse Translation-en [3844 B]
```

Installing Python:

```
ubuntu@ip-172-31-7-213:~$ sudo apt install -y python3 python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.12.3-0ubuntu2).
python3 set to manually installed.
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot
  fontconfig-config fonts-dejavu-core fonts-dejavu-mono g++ g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu
  gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libaom3 libasan8 libatomic1 libbinutils
  libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libde265-0 libdeflate0 libdpkg-perl libexpat1-dev libfakeroot
  libfile-fcntllock-perl libfontconfig1 libgcc-13-dev libgd3 libgomp1 libgprofng0 libheif-plugin-aomdec libheif-plugin-aomenc libheif-plugin-libde265 libheif1
  libhwasan0 libisl23 libitm1 libjbig0 libjpeg-turbo8 libjpeg8 libjs-jquery libjs-sphinxdoc libjs-underscore liblerc4 liblsan0 libmpc3 libpython3-dev
  libpython3.12-dev libquadmath0 libsframe1 libsharpyuv0 libstdc++-13-dev libtiff6 libtsan2 libubsan1 libwebp7 libxpm4 linux-libc-dev lto-disabled-list make
  manpages-dev python3-dev python3-wheel python3.12-dev rpcsvc-proto zlib1g-dev
Suggested packages:
  binutils-doc gprofng-gui bzip2-doc cpp-doc gcc-13-locales cpp-13-doc debian-keyring g++-multilib g++-13-multilib gcc-13-doc gcc-multilib autoconf automake libtool
  flex bison gdb gcc-doc gcc-13-multilib gdb-x86-64-linux-gnu apache2 | lighttpd | httpd glibc-doc bzr libgd-tools libheif-plugin-x265 libheif-plugin-ffmpegdec
  libheif-plugin-jpegdec libheif-plugin-jpegenc libheif-plugin-j2kdec libheif-plugin-j2kenc libheif-plugin-rav1e libheif-plugin-svtenc libstdc++-13-doc make-doc
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot
  fontconfig-config fonts-dejavu-core fonts-dejavu-mono g++ g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu
  gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libaom3 libasan8 libatomic1 libbinutils
  libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libde265-0 libdeflate0 libdpkg-perl libexpat1-dev libfakeroot
  libfile-fcntllock-perl libfontconfig1 libgcc-13-dev libgd3 libgomp1 libgprofng0 libheif-plugin-aomdec libheif-plugin-aomenc libheif-plugin-libde265 libheif1
  libhwasan0 libisl23 libitm1 libjbig0 libjpeg-turbo8 libjpeg8 libjs-jquery libjs-sphinxdoc libjs-underscore liblerc4 liblsan0 libmpc3 libpython3-dev
  libpython3.12-dev libquadmath0 libsframe1 libsharpyuv0 libstdc++-13-dev libtiff6 libtsan2 libubsan1 libwebp7 libxpm4 linux-libc-dev lto-disabled-list make
```

```
ubuntu@ip-172-31-7-213:~$ python3 --version
pip3 --version
Python 3.12.3
pip 24.0 from /usr/lib/python3/dist-packages/pip (python 3.12)
ubuntu@ip-172-31-7-213:~$
```

Installing Java:

```
ubuntu@ip-172-31-7-213:~$ sudo apt install -y openjdk-8-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  adwaita-icon-theme alsa-topology-conf alsa-ucm-conf at-spi2-common at-spi2-core ca-certificates-java dconf-gsettings-backend dconf-service fontconfig
  fonts-dejavu-extra gsettings-desktop-schemas gtk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common libasound2-data libasound2t64 libasyncns0
  libatk-bridge2.0-0t64 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0t64 libatspi2.0-0t64 libavahi-client3 libavahi-common-data libavahi-common3
  libcairo-gobject2 libcairo2 libcups2t64 libdatrie1 libdconf1 libdrm-amdgpu1 libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libflac12t64 libgail-common libgail18t64
  libgdk-pixbuf-2.0-0 libgdk-pixbuf2.0-bin libgdk-pixbuf2.0-common libgif7 libgl1 libgl1-amber-dri libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0 libglx0
  libgraphite2-3 libgtk2.0-0t64 libgtk2.0-bin libgtk2.0-common libharfbuzz0b libice-dev libice6 liblcms2-2 libllvm17t64 libmp3lame0 libmpg123-0t64 libogg0 libopus0
  libpango-1.0-0 libpangocairo-1.0-0 libpangoft2-1.0-0 libpciaccess0 libpcsclite1 libpixman-1-0 libpthread-stubs0-dev libpulse0 librsvg2-2 librsvg2-common libsm-dev
  libsm6 libsndfile1 libthai-data libthai0 libvorbis0a libvorbisenc2 libvulkan1 libwayland-client0 libx11-dev libx11-xcb1 libxau-dev libxaw7 libxcb-dri2-0
  libxcb-dri3-0 libxcb-glx0 libxcb-present0 libxcb-randr0 libxcb-render0 libxcb-shape0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0 libxcb1-dev libxcomposite1 libxcursor1
  libxdamage1 libxdmcp-dev libxfixes3 libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxrandr2 libxrender1 libxshmfence1 libxt-dev libxt6t64 libxtst6 libxv1
  libxxf86dga1 libxxf86vm1 mesa-vulkan-drivers openjdk-8-jdk-headless openjdk-8-jre openjdk-8-jre-headless session-migration ubuntu-mono x11-common x11-utils
  x11proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  default-jre alsa-utils libasound2-plugins cups-common gvfs libice-doc liblcms2-utils opus-tools pcscd pulseaudio librsvg2-bin libsm-doc libx11-doc libxcb-doc
  libxt-doc openjdk-8-demo openjdk-8-source visualvm libnss-mdns fonts-nanum fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei fonts-wqy-zenhei
  fonts-indic mesa-utils
Recommended packages:
  luit
The following NEW packages will be installed:
  adwaita-icon-theme alsa-topology-conf alsa-ucm-conf at-spi2-common at-spi2-core ca-certificates-java dconf-gsettings-backend dconf-service fontconfig
```

```
ubuntu@ip-172-31-7-213:~$ java -version
openjdk version "1.8.0_432"
OpenJDK Runtime Environment (build 1.8.0_432-8u432-ga~us1-0ubuntu2~24.04-ga)
OpenJDK 64-Bit Server VM (build 25.432-bga, mixed mode)
ubuntu@ip-172-31-7-213:~$
```

I also created a virtual environment to install Pyspark

```
ubuntu@ip-172-31-7-213:~$ sudo apt-get install python3-venv -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  python3-pip-whl python3-setuptools-whl python3.12-venv
The following NEW packages will be installed:
  python3-pip-whl python3-setuptools-whl python3-venv python3.12-venv
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 2425 kB of archives.
After this operation, 2777 kB of additional disk space will be used.
Get:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3-pip-whl all 24.0+dfsg-1ubuntu1.1 [1703 kB]
Get:2 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3-setuptools-whl all 68.1.2-2ubuntu1.1 [716 kB]
Get:3 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3.12-venv amd64 3.12.3-1ubuntu0.3 [5678 B]
Get:4 http://us-east-2.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3-venv amd64 3.12.3-0ubuntu2 [1034 B]
Fetched 2425 kB in 0s (35.2 MB/s)
Selecting previously unselected package python3-pip-whl.
(Reading database ... 124408 files and directories currently installed.)
Preparing to unpack .../python3-pip-whl_24.0+dfsg-1ubuntu1.1_all.deb ...
Unpacking python3-pip-whl (24.0+dfsg-1ubuntu1.1) ...
Selecting previously unselected package python3-setuptools-whl.
Preparing to unpack .../python3-setuptools-whl_68.1.2-2ubuntu1.1_all.deb ...
Unpacking python3-setuptools-whl (68.1.2-2ubuntu1.1) ...
Selecting previously unselected package python3.12-venv.
Preparing to unpack .../python3.12-venv_3.12.3-1ubuntu0.3_amd64.deb ...
```

Installing PySpark within Virtual Environment:

```
ubuntu@ip-172-31-7-213:~$ python3 -m venv venv
ubuntu@ip-172-31-7-213:~$ source venv/bin/activate
(venv) ubuntu@ip-172-31-7-213:~$ sudo apt-get install openjdk-11-jdk -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openjdk-11-jdk is already the newest version (11.0.25+9-1ubuntu1~24.04).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
(venv) ubuntu@ip-172-31-7-213:~$ java -version
openjdk version "11.0.25" 2024-10-15
OpenJDK Runtime Environment (build 11.0.25+9-post-Ubuntu-1ubuntu124.04)
OpenJDK 64-Bit Server VM (build 11.0.25+9-post-Ubuntu-1ubuntu124.04, mixed mode, sharing)
(venv) ubuntu@ip-172-31-7-213:~$ pip install pyspark
Collecting pyspark
  Downloading pyspark-3.5.3.tar.gz (317.3 MB)
     ---------------------------------------- 317.3/317.3 MB 1.9 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl.metadata (1.5 kB)
Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
    ---------------------------------------- 200.5/200.5 kB 30.9 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (pyproject.toml) ... done
  Created wheel for pyspark: filename=pyspark-3.5.3-py2.py3-none-any.whl size=317840629 sha256=d80e2687e660404a0cf6b24edef9ef66b532cacc01308c6ad861fd86d46f2d11
```

Running PySpark

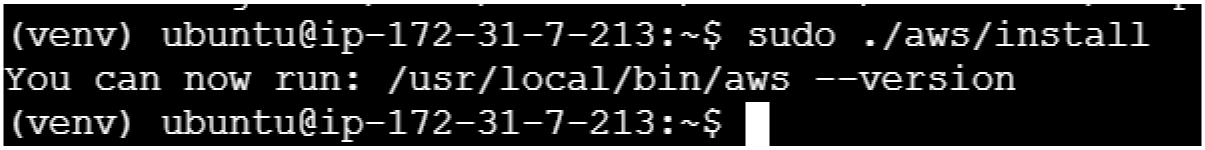
```
(venv) ubuntu@ip-172-31-7-213:~$ pyspark
Python 3.12.3 (main, Nov  6 2024, 18:32:19) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/08 20:31:14 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.3
      /_/

Using Python version 3.12.3 (main, Nov  6 2024 18:32:19)
Spark context Web UI available at http://ip-172-31-7-213.us-east-2.compute.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1733689876230).
SparkSession available as 'spark'.
>>>
```

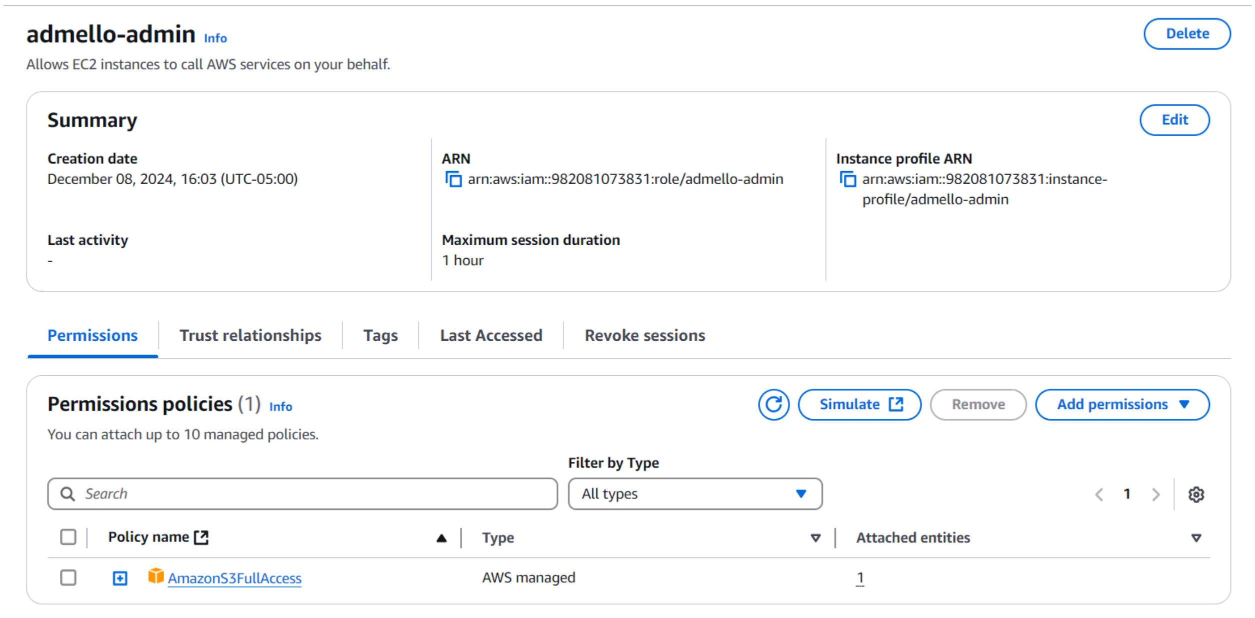
c. Step 3: Configure AWS CLI to interact with S3 buckets.

Performed the steps below to establish connection between EC2 instance and S3 buckets.

```
(venv) ubuntu@ip-172-31-7-213:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 64.2M  100 64.2M    0     0  58.7M      0  0:00:01  0:00:01 --:--:-- 58.7M
(venv) ubuntu@ip-172-31-7-213:~$
```



Created role in IAM:



Granted EC2 instance access to role and checking if the instance can access S3

Bucket name showing up confirms that the instance now has access to S3

3. Data Pipeline Tasks

Task 1: Data Ingestion from S3

● **Steps:**

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

2. Confirm successful ingestion by inspecting the dataset.

For this task, I configured and used a Jupyter notebook to perform data preprocessing

I used the following command to access the Jupyter notebook:

```
upload: ./SQL.py to s3://project-data-alvan/code_files/SQL.py
(venv) ubuntu@ip-172-31-7-213:~/code_files$ jupyter notebook --no-browser --ip=0.0.0.0 --port=8888
[I 2024-12-15 16:58:49.235 ServerApp] jupyter_lsp | extension was successfully linked.
```

After this command, the Jupyter notebook could be accessed in a different tab/browser using the instance's Public IP

Next, I utilized PySpark's built-in support to load dataset directly:

Data Ingestion

```
[1]: from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Data Preprocessing Online Store UK") \
    .config("spark.jars.packages", "org.apache.hadoop:hadoop-aws:3.3.1") \
    .config("spark.sql.legacy.timeParserPolicy", "LEGACY") \
    .getOrCreate()

hadoop_conf = spark._jsc.hadoopConfiguration()
hadoop_conf.set("fs.s3a.endpoint", "s3.amazonaws.com")

s3_path = f"s3a://project-data-alvan/final_data_v1.csv"

try:
    df = spark.read.csv(s3_path, header=True, inferSchema=True)
    print("Dataset loaded successfully!")
except Exception as e:
    print(f"Error loading dataset: {e}")
    spark.stop()
    exit()

print("Schema of the dataset:")
df.printSchema()
```

```
:: loading settings :: url = jar:file:/home/ubuntu/venv/lib/python3.12/site-packages/pyspark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysetti
ngs.xml
Ivy Default Cache set to: /home/ubuntu/.ivy2/cache
The jars for the packages stored in: /home/ubuntu/.ivy2/jars
org.apache.hadoop#hadoop-aws added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-f1b3bc02-247f-428e-8471-e8e689cb0f47;1.0
        confs: [default]
        found org.apache.hadoop#hadoop-aws;3.3.1 in central
        found com.amazonaws#aws-java-sdk-bundle;1.11.901 in central
        found org.wildfly.openssl#wildfly-openssl;1.0.7.Final in central
:: resolution report :: resolve 597ms :: artifacts dl 25ms
        :: modules in use:
        com.amazonaws#aws-java-sdk-bundle;1.11.901 from central in [default]
        org.apache.hadoop#hadoop-aws;3.3.1 from central in [default]
        org.wildfly.openssl#wildfly-openssl;1.0.7.Final from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   3   |   0   |   0   |   0   ||   3   |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-f1b3bc02-247f-428e-8471-e8e689cb0f47
        confs: [default]
        0 artifacts copied, 3 already retrieved (0kB/16ms)
24/12/14 21:27:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/12/14 21:27:54 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hadoop-metrics2.properties

Dataset loaded successfully!
Schema of the dataset:
root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: string (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: double (nullable = true)
 |-- Country: string (nullable = true)
```

Displayed the first five rows to confirm successful ingestion

```
[3]: # Displaying the first 5 rows of the data
df.show(5)
```

```
+---------+---------+--------------------+--------+----------------+---------+----------+--------------+
|InvoiceNo|StockCode|         Description|Quantity|     InvoiceDate|UnitPrice|CustomerID|       Country|
+---------+---------+--------------------+--------+----------------+---------+----------+--------------+
|   543451|    22167| OVAL WALL MIRROR...|       1|  2/8/2011 12:13|    19.96|      NULL|United Kingdom|
|   577522|    22944|CHRISTMAS METAL P...|       6|11/20/2011 13:23|     0.39|   15988.0|United Kingdom|
|   580367|    22284|HEN HOUSE DECORATION|       1| 12/2/2011 16:39|     3.29|      NULL|United Kingdom|
|   576245|    23569|TRADTIONAL ALPHAB...|       4|11/14/2011 13:40|     4.95|   12553.0|        France|
|   578293|    22086|PAPER CHAIN KIT 5...|      12|11/23/2011 14:36|     2.95|   15640.0|United Kingdom|
+---------+---------+--------------------+--------+----------------+---------+----------+--------------+
only showing top 5 rows
```

Task 2: Data Processing with PySpark

Data Cleaning

For cleaning, I checked if there were any null values in the dataset.

```
[5]: from pyspark.sql import functions as F

# Count null values in each column
null_counts = df.select([F.sum(F.col(c).isNull().cast("int")).alias(c) for c in df.columns])
null_counts.show()
```

```
+---------+---------+-----------+--------+-----------+---------+----------+-------+
|InvoiceNo|StockCode|Description|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---------+---------+-----------+--------+-----------+---------+----------+-------+
|        0|        0|          8|       0|          0|        0|       989|      0|
+---------+---------+-----------+--------+-----------+---------+----------+-------+
```

Since there were a few rows with null values, I decided the best strategy would be to drop these rows since they belonged to the customer ID column.

```
[2]: # Drop rows with any null values in any column
df = df.dropna()
```

Data Transformation

I created 3 new columns from the existing columns. From the invoice date column, I created the columns ‘Month’ and ‘Year’. From the columns, Unit Price and Quantity, I created the column ‘Revenue’

[5]:
```python
# Extract Month and Year
df = df.withColumn('Month', month(col('InvoiceDate')))
df = df.withColumn('Year', year(col('InvoiceDate')))
```

[9]:
```python
# Creating column revenue
df = df.withColumn("Revenue", round(col("Quantity") * col("UnitPrice"),2))
```

The final preprocessed data looks like this with 3 additional columns:

[10]:
```python
df.show(5)
```

```
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+----+--------+-------+
|InvoiceNo|StockCode|         Description|Quantity|        InvoiceDate|UnitPrice|CustomerID|       Country|Year|   Month|Revenue|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+----+--------+-------+
|   577522|    22944|CHRISTMAS METAL P...|       6|2011-11-20 13:23:00|     0.39|   15988.0|United Kingdom|2011|November|   2.34|
|   576245|    23569|TRADTIONAL ALPHAB...|       4|2011-11-14 13:40:00|     4.95|   12553.0|        France|2011|November|   19.8|
|   578293|    22086|PAPER CHAIN KIT 5...|      12|2011-11-23 14:36:00|     2.95|   15640.0|United Kingdom|2011|November|   35.4|
|   573248|    23247|BISCUIT TIN 50'S ...|       2|2011-10-28 12:09:00|     2.89|   14498.0|United Kingdom|2011| October|   5.78|
|  C569985|    22617|BAKING SET SPACEB...|      -3|2011-10-06 19:51:00|     4.95|   15365.0|United Kingdom|2011| October| -14.85|
+---------+---------+--------------------+--------+-------------------+---------+----------+--------------+----+--------+-------+
only showing top 5 rows
```

Data Aggregation

The following metrics have been created using aggregate functions

1. Total Revenue by Country

This metric calculates the total revenue generated by sales in each country. By summing up the revenue column for each country and sorting the results in descending order, it highlights the regions that contribute the most to the overall sales performance.

1. Total Revenue by Country

[10]:
```python
from pyspark.sql.functions import sum

# Calculate total revenue by Country
total_revenue_by_region = df.groupBy("Country") \
    .agg(sum("Revenue").alias("TotalRevenue")) \
    .orderBy(col("TotalRevenue").desc())

total_revenue_by_region.show()
```

```
[Stage 3:>                                                          (0 + 1) / 1]
+--------------+-----------------+
|       Country|     TotalRevenue|
+--------------+-----------------+
| United Kingdom| 51720.96000000006|
|          EIRE|2108.0899999999997|
|       Germany|1674.2599999999998|
|        France|1625.3500000000006|
|   Netherlands|1615.0799999999997|
|     Australia|          1277.53|
|   Switzerland|360.97999999999996|
|         Spain|286.65999999999997|
|Channel Islands|           205.75|
|       Belgium|           173.62|
|         Italy|138.29000000000002|
|        Norway|           136.89|
|      Portugal|           129.38|
|       Denmark|            123.0|
|        Sweden|            92.59|
|       Finland|            83.58|
|        Cyprus|             53.6|
|        Poland|             51.4|
|         Japan|48.650000000000006|
| Czech Republic|             45.9|
+--------------+-----------------+
only showing top 20 rows
```

2. **Revenue Growth by Year**

This metric aggregates the total revenue for each year in the dataset. By grouping transactions by year and summing up the revenue, it provides a clear view of the business's revenue growth or decline over time.

2. Revenue Growth Over the years

[11]:
```python
revenue_growth = df.groupBy("Year") \
    .agg(sum("Revenue").alias("TotalRevenue")) \
    .orderBy("Year")

revenue_growth.show()
```

```
+----+-----------------+
|Year|     TotalRevenue|
+----+-----------------+
|2010|3955.940000000002|
|2011|55840.30000000006|
+----+-----------------+
```

3. **Average Monthly Revenue by Region**

This metric calculates the average revenue per month for each country. It first computes the monthly revenue by summing up the revenue for each country, year, and month. Then, it calculates the average across all months for each country, offering insight into the regions with consistent revenue generation.

3. Average Revenue by Month

```
[12]: from pyspark.sql.functions import avg

# Calculate average monthly revenue by region
avg_monthly_revenue_by_region = df.groupBy("Country", "Year", "Month") \
    .agg(sum("Revenue").alias("MonthlyRevenue")) \
    .groupBy("Country") \
    .agg(avg("MonthlyRevenue").alias("AvgMonthlyRevenue")) \
    .orderBy(col("AvgMonthlyRevenue").desc())

avg_monthly_revenue_by_region.show()
```

```
+--------------+-----------------+
|       Country| AvgMonthlyRevenue|
+--------------+-----------------+
| United Kingdom|3978.5353846153853|
|    Netherlands|230.72571428571428|
|          EIRE| 162.1607692307692|
|     Australia|159.69125000000003|
|       Germany|128.78923076923078|
|        France|125.02692307692307|
|    Switzerland|51.568571428571424|
| Czech Republic|             45.9|
|Channel Islands|            41.15|
|       Denmark|             41.0|
|     Lithuania|             35.4|
|           USA|             32.4|
|      Portugal|           32.345|
|       Belgium|28.936666666666667|
|         Spain|28.666000000000004|
|         Italy|27.658000000000005|
|        Norway|27.377999999999997|
|        Canada|            26.52|
|        Poland|             25.7|
|        Sweden|          23.1475|
+--------------+-----------------+
only showing top 20 rows
```

4. Average Transaction Value by Customer

This metric computes the average value of a transaction for each customer. By dividing the total revenue generated by a customer by the number of invoices they have, it helps identify customers who make high-value purchases on average, which can be crucial for customer segmentation and targeted marketing.

4. Average transaction value per customer

```
[13]: from pyspark.sql.functions import count
avg_transaction_value = df.groupBy("CustomerID") \
    .agg((sum("Revenue") / count("InvoiceNo")).alias("AvgTransactionValue")) \
    .orderBy(col("AvgTransactionValue").desc())

avg_transaction_value.show()
```

```
+----------+-------------------+
|CustomerID|AvgTransactionValue|
+----------+-------------------+
|   16041.0|              675.0|
|   14145.0|              594.0|
|   18102.0|             562.66|
|   12931.0|              507.5|
|   17389.0|             414.48|
|   15769.0|              358.0|
|   14031.0|             343.76|
|   13629.0|              330.0|
|   12939.0|             325.44|
|   17396.0|              246.1|
|   14154.0|              195.0|
|   16684.0| 186.42399999999998|
|   18092.0|              180.0|
|   18064.0|              179.0|
|   14607.0|              179.0|
|   13093.0|              175.2|
|   13798.0|              172.5|
|   16553.0|              169.5|
|   16029.0|              163.2|
|   14680.0|            154.375|
+----------+-------------------+
only showing top 20 rows
```

5. Top 10 Products by Quantity Sold

This metric identifies the top 10 most popular products based on the quantity sold. By summing up the quantities sold for each product and sorting in descending order, it provides a view of the best-performing products, helping in inventory management and product strategy.

5. Total Quantity Sold per product

[14]:
```
# Calculate total quantity sold per product
top_products = df.groupBy("Description") \
    .agg(sum("Quantity").alias("TotalQuantitySold")) \
    .orderBy(col("TotalQuantitySold").desc()) \
    .limit(10)

top_products.show()
```

```
+--------------------+-----------------+
|         Description|TotalQuantitySold|
+--------------------+-----------------+
|SET OF 60 I LOVE ...|              537|
|JUMBO BAG RED RET...|              448|
|LUNCH BAG ALPHABE...|              392|
|WOODLAND CHARLOTT...|              310|
|DOORMAT KEEP CALM...|              306|
|DOORMAT UNION JAC...|              304|
|MEDIUM CERAMIC TO...|              300|
|PACK OF 72 RETROS...|              277|
|PLASTERS IN TIN S...|              263|
|JAZZ HEARTS ADDRE...|              248|
+--------------------+-----------------+
```

Task 3: Store Processed Data Back to S3

Steps:

1. Export data in CSV or Parquet format.
2. Upload the processed data to a designated S3 location for easy access

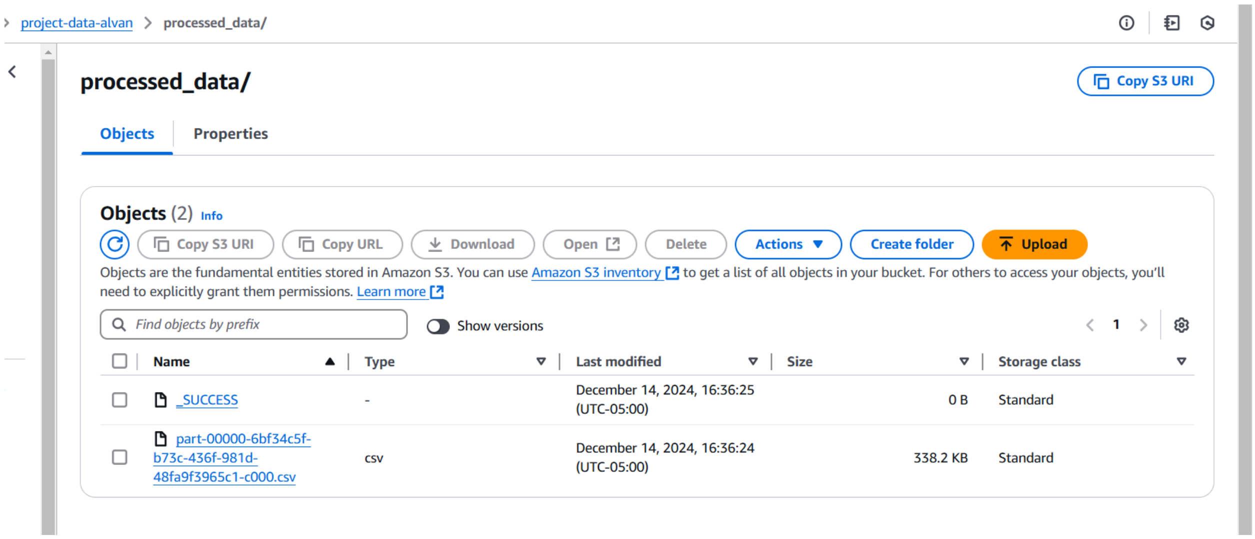
The data was directly pushed into the S3 buckets. First I pushed the preprocessed data and then I pushed each of the individual metrics obtained using aggregate functions.

Store Data Back to S3

Storing preprocessed data in S3

•[21]:
```
df.write.csv("s3a://project-data-alvan/processed_data",header=True)
```

```
24/12/14 21:36:21 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
24/12/14 21:36:22 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
```



Storing aggregated data in S3

```
[16]: top_products.write.csv("s3a://project-data-alvan/aggregated_data/top_products",header=True)
```

```
24/12/14 21:30:50 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
24/12/14 21:30:51 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
```

```
[17]: avg_transaction_value.write.csv("s3a://project-data-alvan/aggregated_data/avg_transaction_value",header=True)
```

```
24/12/14 21:31:30 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
24/12/14 21:31:30 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
```

```
[18]: avg_monthly_revenue_by_region.write.csv("s3a://project-data-alvan/aggregated_data/avg_monthly_revenue_by_region",header=True)
```

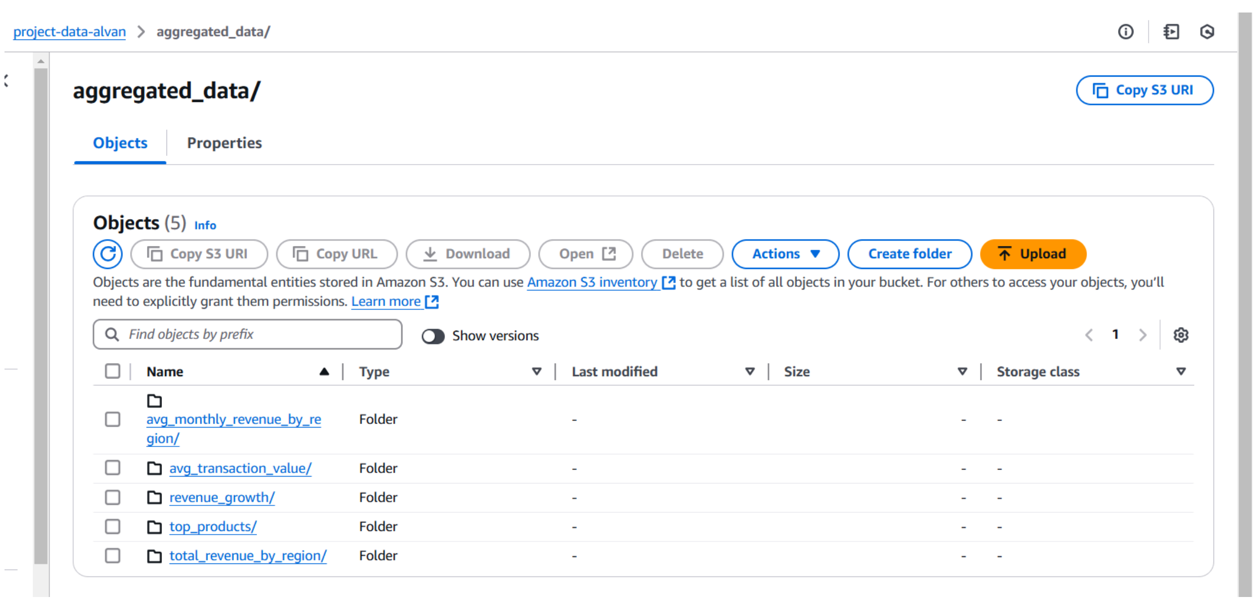
```
24/12/14 21:32:05 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
24/12/14 21:32:05 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
```

```
[19]: revenue_growth.write.csv("s3a://project-data-alvan/aggregated_data/revenue_growth",header=True)
```

```
24/12/14 21:32:54 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
24/12/14 21:33:00 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
```

```
[20]: total_revenue_by_region.write.csv("s3a://project-data-alvan/aggregated_data/total_revenue_by_region",header=True)
```

```
24/12/14 21:33:42 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
24/12/14 21:33:55 WARN AbstractS3ACommitterFactory: Using standard FileOutputCommitter to commit work. This is slow and potentially unsafe.
```



Task 4: Data Analysis Using Spark SQL

The queries and their insights are as follows

#1. Total Revenue by Country

This query calculates the total revenue generated in each country by summing up the Revenue column for each country. The results are sorted in descending order to highlight the top-performing regions. We can see that the UK generates a very large revenue compared to all the other countries.

Query:

```
print("Total Revenue by Country:")
query = """
SELECT
    Country,
    SUM(Revenue) AS TotalRevenue
FROM
    sales_data
GROUP BY
    Country
ORDER BY
    TotalRevenue DESC
"""
result = spark.sql(query)
result.show()
```

```
----------------------------------------------------------------
Total Revenue by Country:
+---------------+-----------------+
|        Country|     TotalRevenue|
+---------------+-----------------+
| United Kingdom| 51720.96000000006|
|           EIRE|2108.0899999999997|
|        Germany|1674.2599999999998|
|         France|1625.3500000000006|
|    Netherlands|1615.0799999999997|
|      Australia|          1277.53|
|    Switzerland|360.97999999999996|
|          Spain|286.65999999999997|
|Channel Islands|           205.75|
|        Belgium|           173.62|
|          Italy|138.29000000000002|
|         Norway|           136.89|
|       Portugal|           129.38|
|        Denmark|            123.0|
|         Sweden|            92.59|
|        Finland|            83.58|
|         Cyprus|             53.6|
|         Poland|             51.4|
|          Japan|48.650000000000006|
| Czech Republic|             45.9|
+---------------+-----------------+
only showing top 20 rows
```

#2. Top 5 Customers by Revenue

This query identifies the top 5 customers based on the total revenue they have generated. It groups data by CustomerID, sums the revenue for each customer, and sorts the results in descending order. We can see that the customer with ID 18102 purchases comparatively more than the remaining customers and generates a lot of revenue for the store.

Query:

print("Top 5 Customers by Revenue")

query = """

SELECT

```
    CustomerID,
    SUM(Revenue) AS TotalRevenue
FROM
    sales_data
GROUP BY
    CustomerID
ORDER BY
    TotalRevenue DESC
LIMIT 5
"""
result = spark.sql(query)
result.show()
```

```
------------------------------------------------------------------
Top 5 Customers by Revenue
+----------+------------------+
|CustomerID|      TotalRevenue|
+----------+------------------+
|   18102.0|           2250.64|
|   14031.0|            1718.8|
|   14646.0|1594.7999999999997|
|   14911.0|1246.0900000000001|
|   17389.0|           1243.44|
+----------+------------------+
```

#3. Monthly Revenue Trends

This query analyzes revenue changes over time by calculating the total revenue for each year and month. It includes MonthOrder to ensure months are correctly sorted chronologically. Here we can see that October and November have a very high revenue, which can be indicative of seasonal activity.

Query:

```
print("Monthly Revenue Trends")
query = """
SELECT
    Year,
    Month,
    SUM(Revenue) AS MonthlyRevenue
FROM
    sales_data
GROUP BY
    Year, Month, MonthOrder
ORDER BY
    Year, MonthOrder
"""
result = spark.sql(query)
result.show()
```

```
-----------------------------------------------------------------------
Monthly Revenue Trends
+----+---------+-----------------+
|Year|    Month|   MonthlyRevenue|
+----+---------+-----------------+
|2010| December| 3955.940000000002|
|2011|  January|3177.4899999999993|
|2011| February|2827.4900000000002|
|2011|    March|3130.5600000000004|
|2011|    April|-113.3000000000002|
|2011|      May| 4805.099999999997|
|2011|     June|            6074.35|
|2011|     July| 6149.689999999997|
|2011|   August| 4683.939999999998|
|2011|September|            5531.25|
|2011|  October| 8192.519999999993|
|2011| November| 8303.300000000001|
|2011| December| 3077.909999999999|
+----+---------+-----------------+
```

#4. Average Transaction Value by Month

This query calculates the average value of a transaction for each month by dividing the total revenue by the number of unique invoices (InvoiceNo). The months are sorted using MonthOrder to ensure proper order. This is almost the same over the months, indicating that people prefer buying products that have a price in that particular range.

Query:

```
print("Average Transaction Value by Month")
query = """
SELECT
    Month,
    SUM(Revenue) / COUNT(DISTINCT InvoiceNo) AS AvgTransactionValue
FROM
```

```
    sales_data
GROUP BY
    Month, MonthOrder
ORDER BY
    MonthOrder
"""
result = spark.sql(query)
result.show()
```

```
----------------------------------------------------------------------
Average Transaction Value by Month
+---------+-------------------+
|    Month|AvgTransactionValue|
+---------+-------------------+
|  January| 26.260247933884294|
| February| 20.944370370370358|
|    March| 17.888914285714282|
|    April|-0.8648854961832031|
|      May| 22.143317972350232|
|     June|  30.37175000000001|
|     July|  33.97618784530386|
|   August| 27.232209302325586|
|September| 20.872641509433958|
|  October|  24.97719512195121|
| November| 20.603722084367234|
| December| 23.290894039735072|
+---------+-------------------+
```

#5. Top 5 Products by Quantity Sold

This query finds the 5 most sold products by summing up the quantities sold (Quantity) for each product. The results are sorted in descending order based on total quantity sold. The product with stock code 23309 sells the most

Query:

```
print("Top 5 Products by Quantity Sold")
query = """
SELECT
    StockCode,
    Description,
    SUM(Quantity) AS TotalQuantitySold
FROM
    sales_data
GROUP BY
    StockCode, Description
ORDER BY
    TotalQuantitySold DESC
LIMIT 5
"""
result = spark.sql(query)
result.show()
```

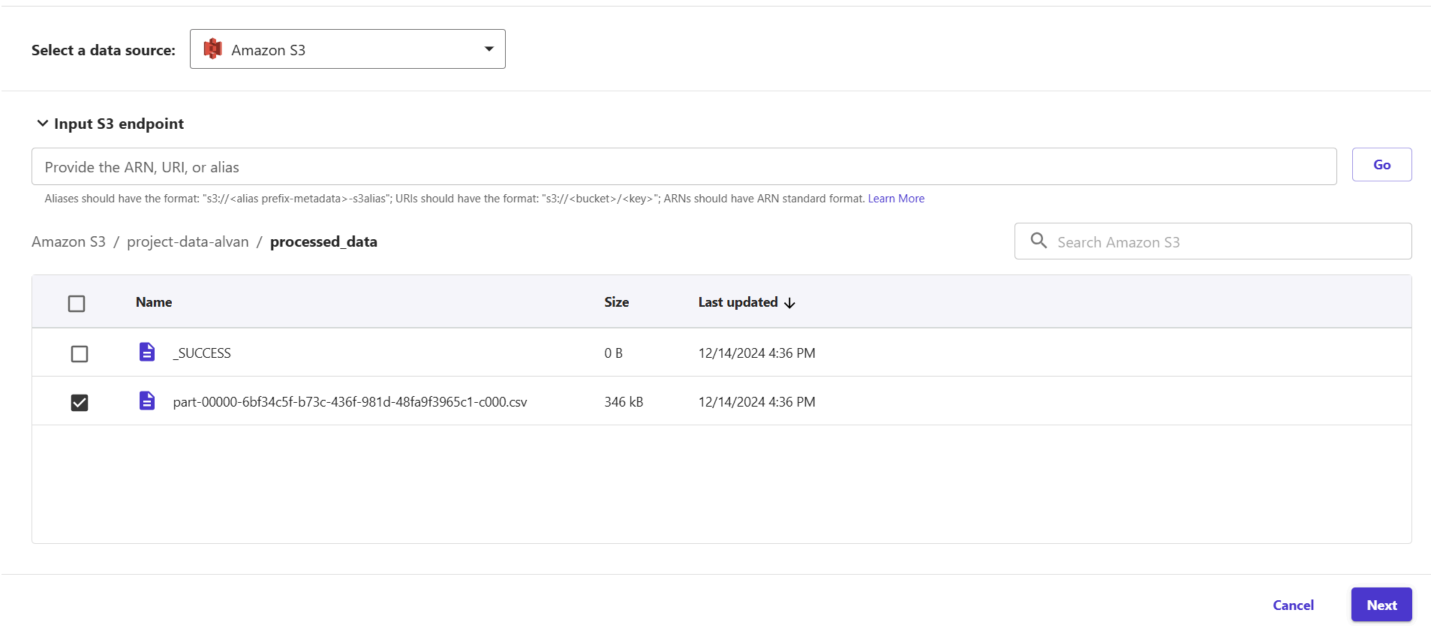
```
------------------------------------------------------------------
Top 5 Products by Quantity Sold
+---------+--------------------+-----------------+
|StockCode|         Description|TotalQuantitySold|
+---------+--------------------+-----------------+
|    23309|SET OF 60 I LOVE ...|              537|
|   85099B|JUMBO BAG RED RET...|              448|
|    23207|LUNCH BAG ALPHABE...|              392|
|    20719|WOODLAND CHARLOTT...|              310|
|    23284|DOORMAT KEEP CALM...|              306|
+---------+--------------------+-----------------+
```

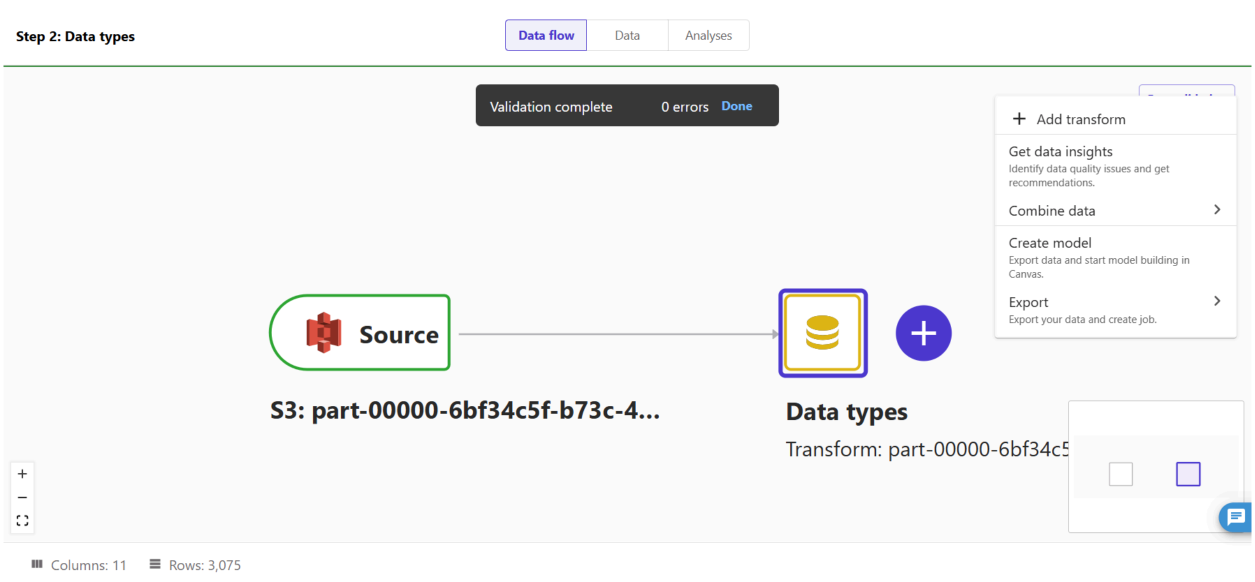
Task 5: Machine Learning with AWS SageMaker Autopilot

Steps:

1. Import Processed Data: Load the processed dataset from S3 into SageMaker Autopilot.

Data has been imported into Sagemaker from S3 buckets

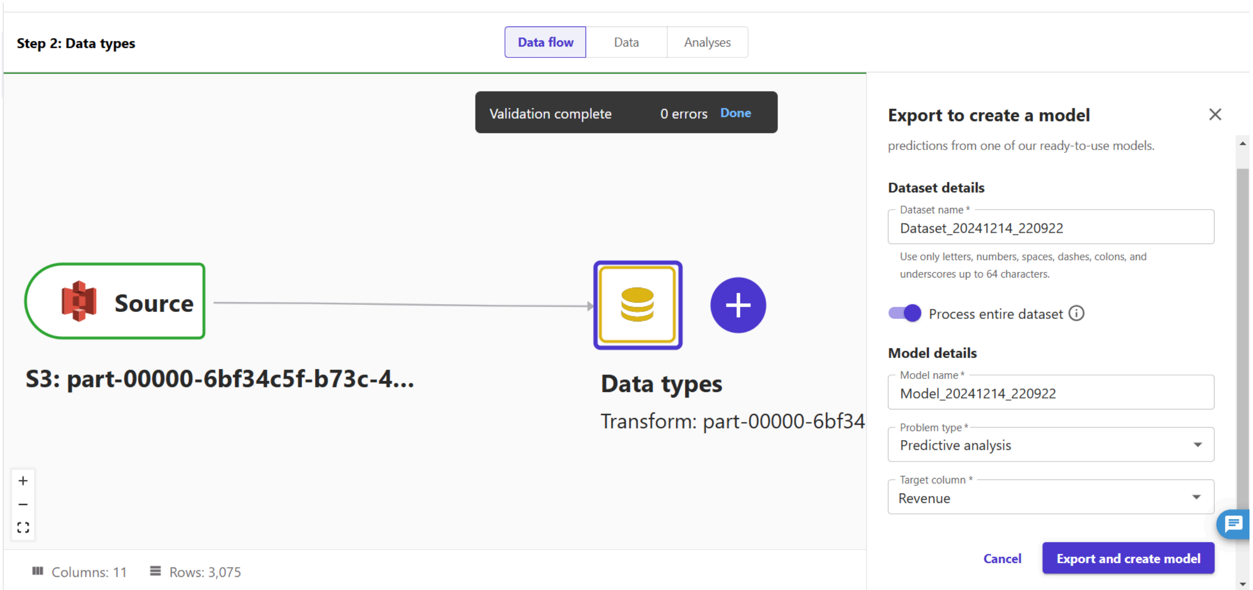


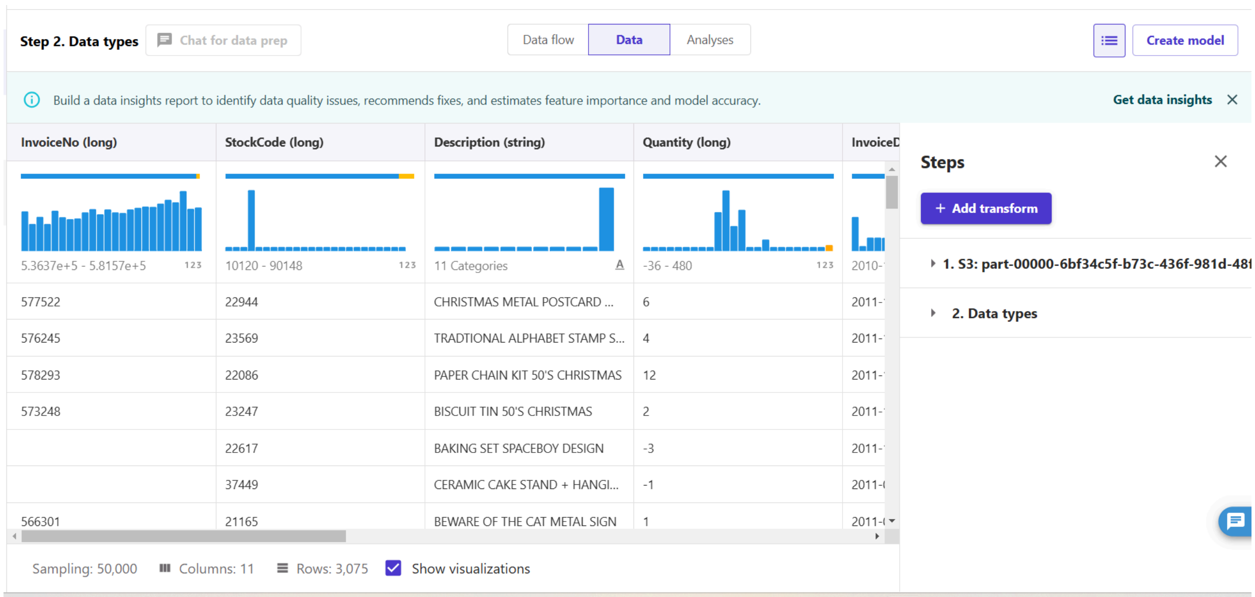


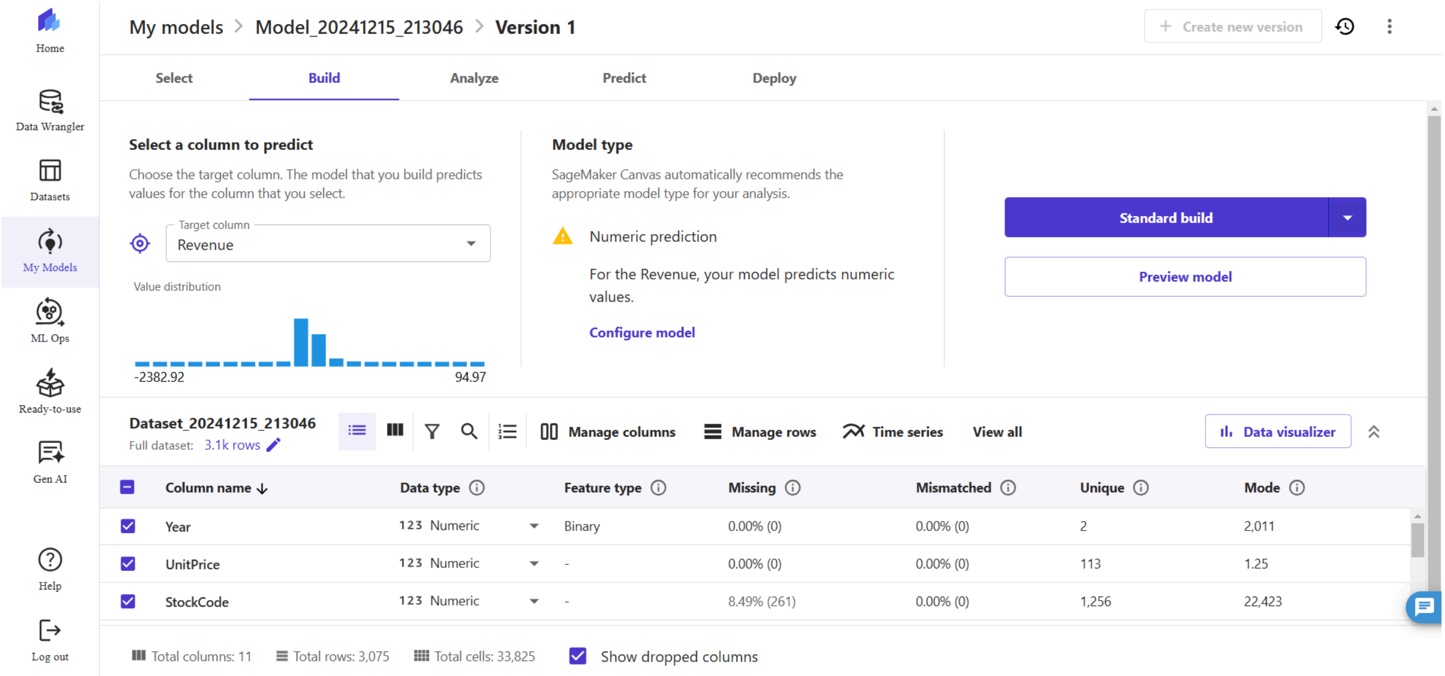
2. Run Autopilot Experiment:

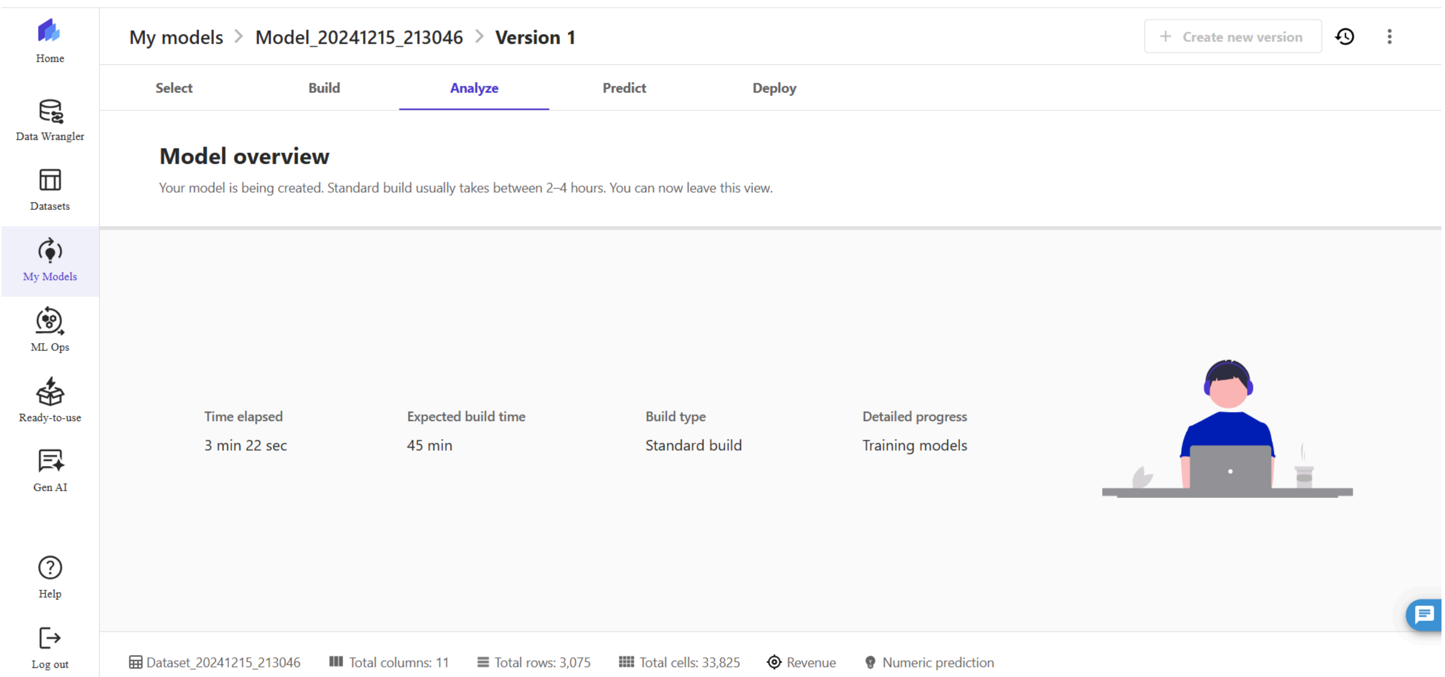
I ran the autopilot experiment on target column Revenue

3. Consider taking screenshots of your working and query results.



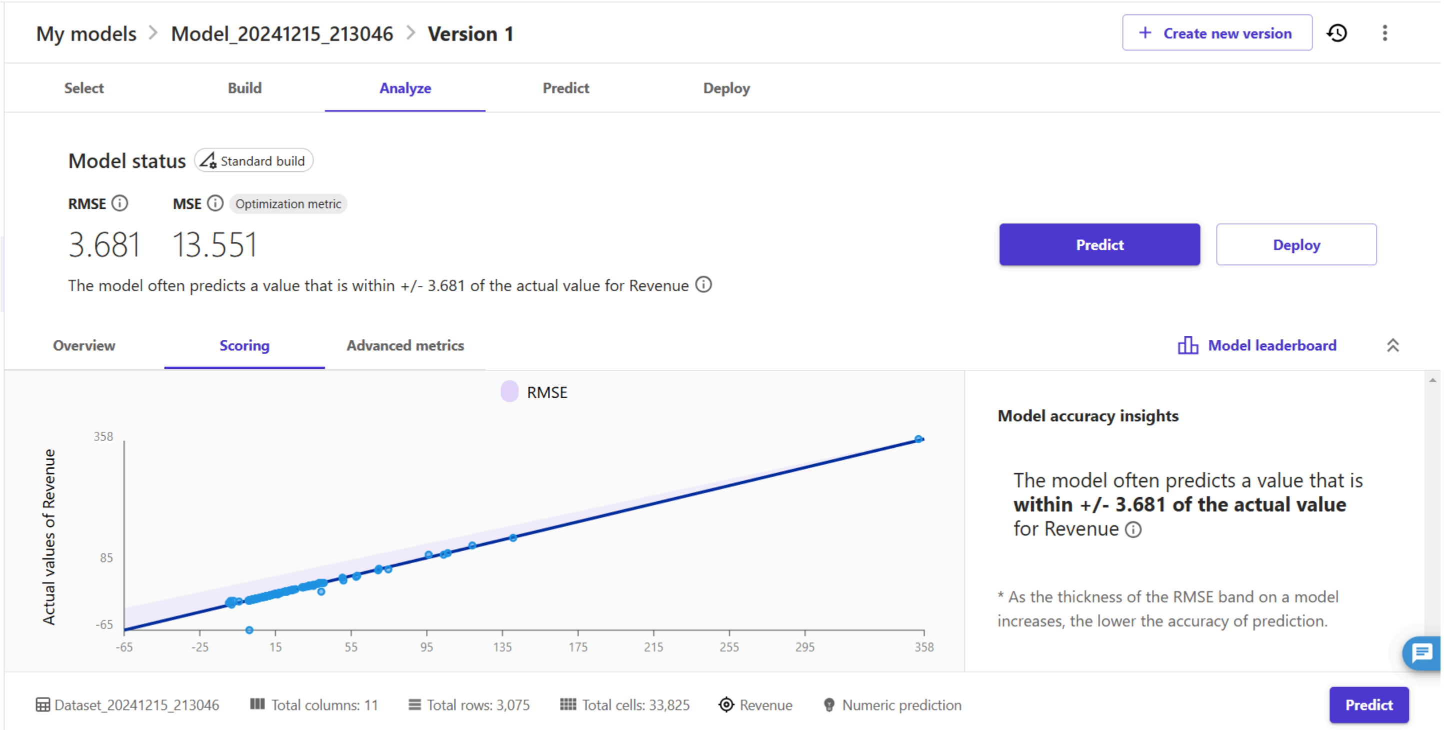


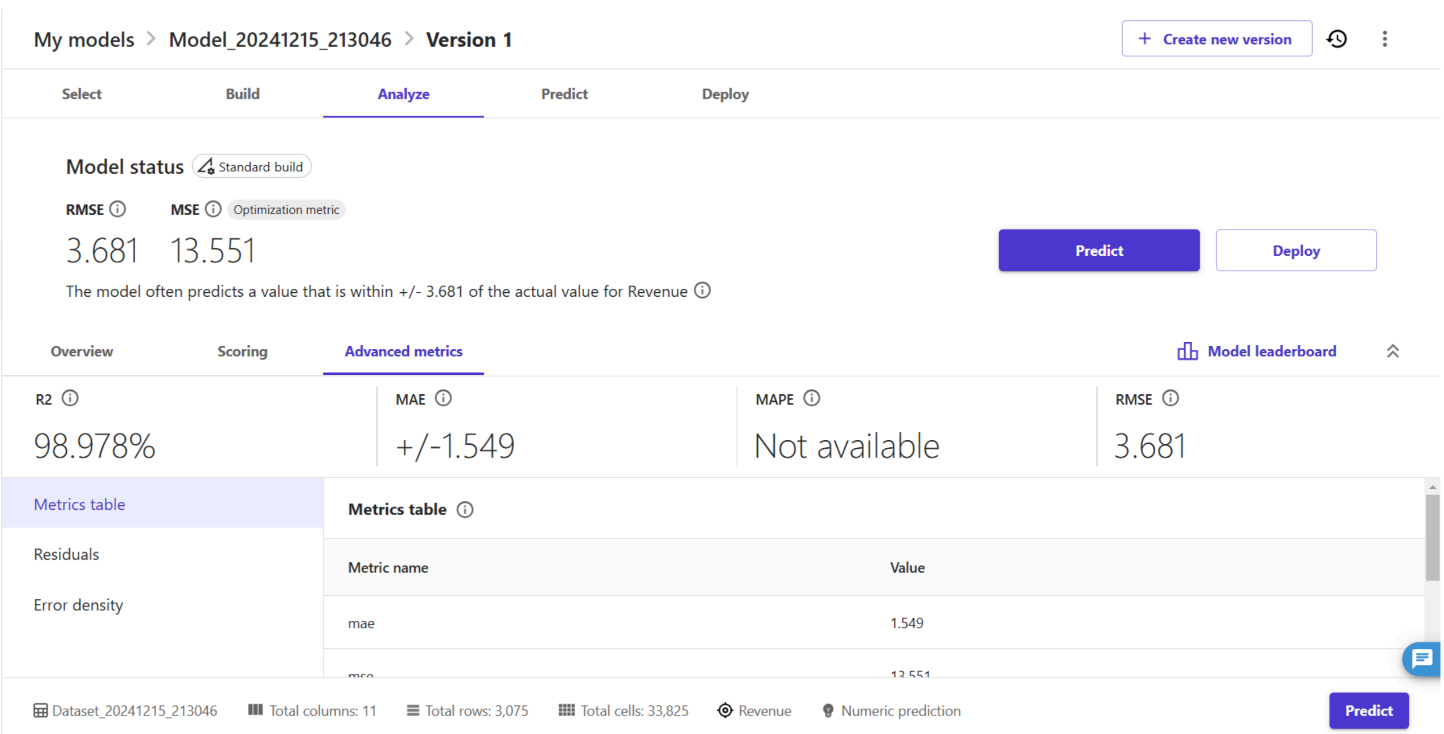

Home
Data Wrangler
Datasets
My Models
ML Ops
Ready-to-use
Gen AI
Help
Log out
My models > Model_20241215_213046 > Version 1
Create new version
Select
Build
Analyze
Predict
Deploy
Select a column to predict
Choose the target column. The model that you build predicts values for the column that you select.
Target column
Revenue
Value distribution
-2382.92
94.97
Model type
SageMaker Canvas automatically recommends the appropriate model type for your analysis.
Numeric prediction
For the Revenue, your model predicts numeric values.
Configure model
Standard build
Preview model
Dataset_20241215_213046
Full dataset: 3.1k rows
Manage columns
Manage rows
Time series
View all
Data visualizer
Column name
Data type
Feature type
Missing
Mismatched
Unique
Mode
Year
123 Numeric
Binary
0.00% (0)
0.00% (0)
2
2,011
UnitPrice
123 Numeric
-
0.00% (0)
0.00% (0)
113
1.25
StockCode
123 Numeric
-
8.49% (261)
0.00% (0)
1,256
22,423
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Show dropped columns

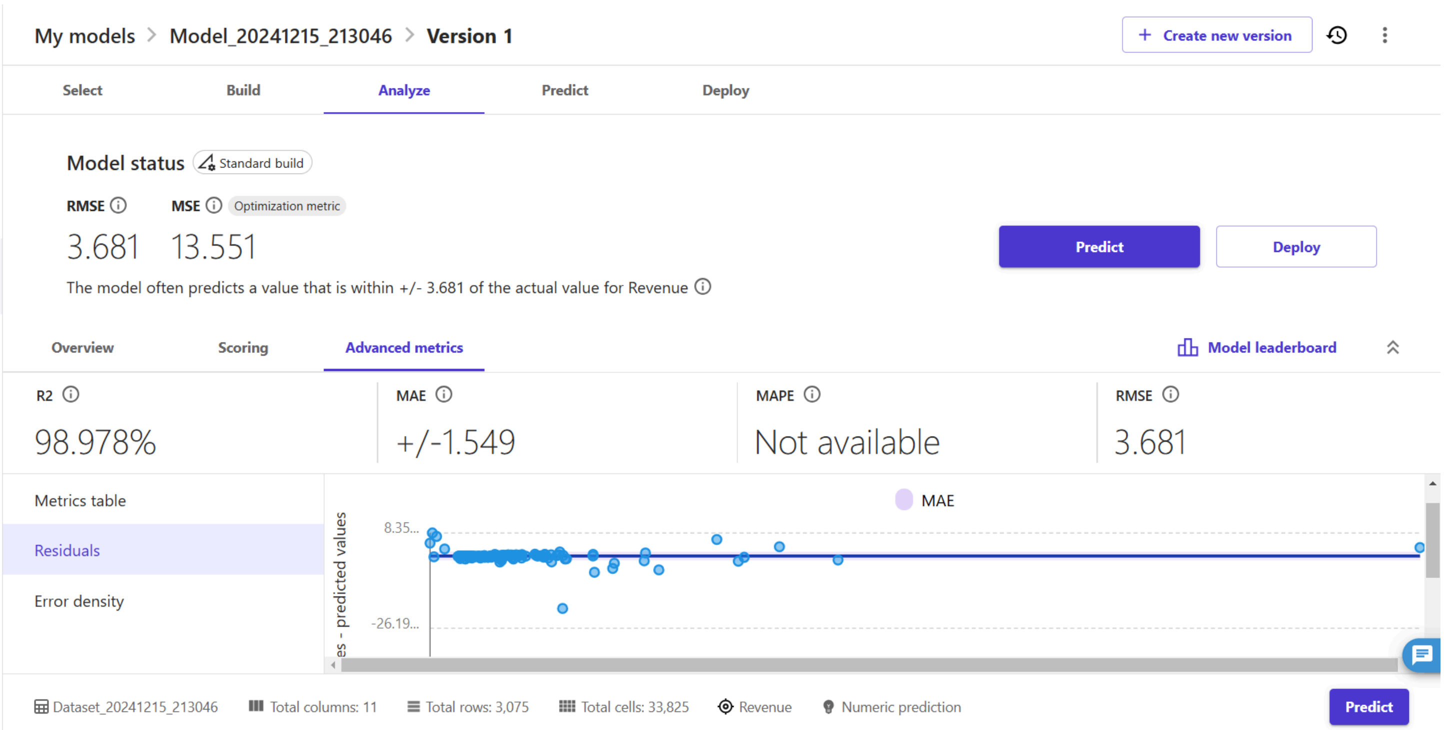

Home
Data Wrangler
Datasets
My Models
ML Ops
Ready-to-use
Gen AI
Help
Log out
My models > Model_20241215_213046 > Version 1
Create new version
Select
Build
Analyze
Predict
Deploy
Model overview
Your model is being created. Standard build usually takes between 2–4 hours. You can now leave this view.
Time elapsed
3 min 22 sec
Expected build time
45 min
Build type
Standard build
Detailed progress
Training models
Dataset_20241215_213046
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Revenue
Numeric prediction

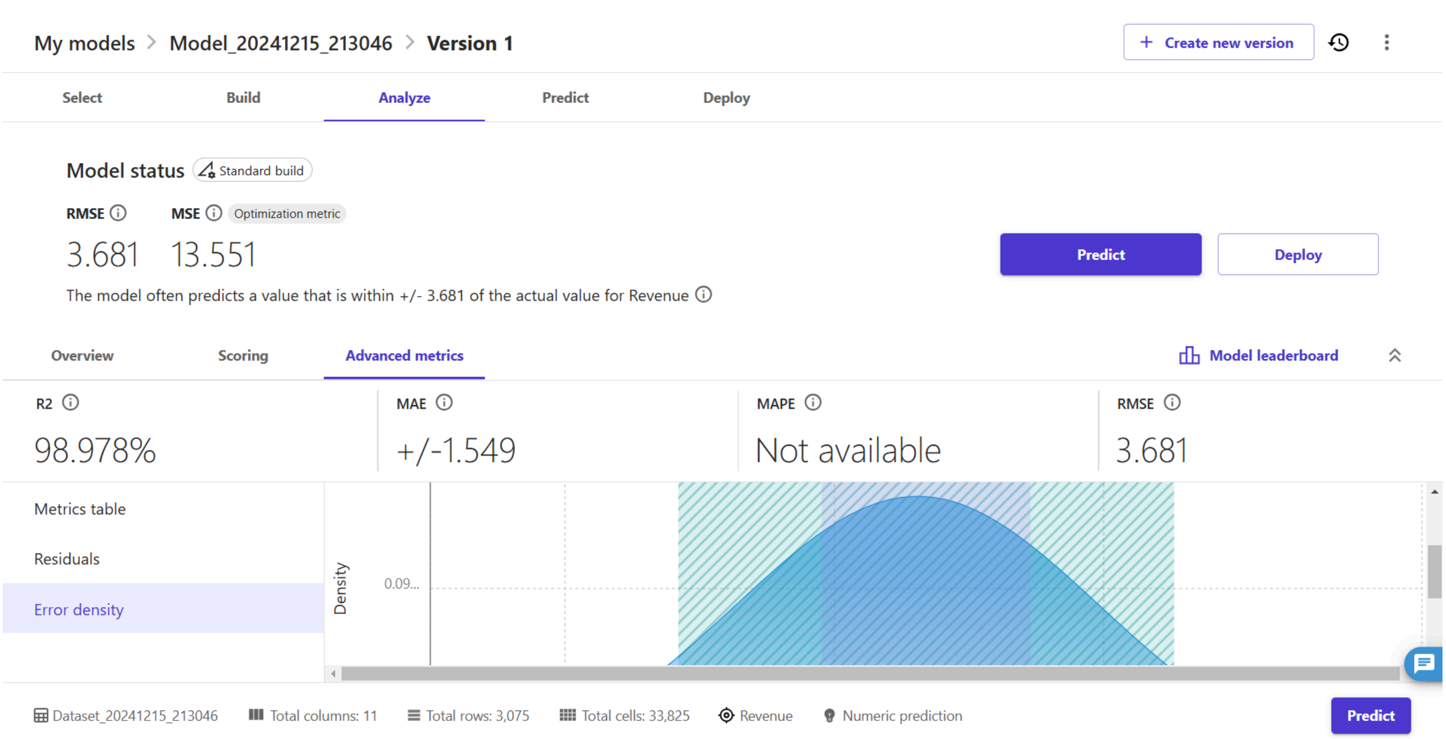

My models > Model_20241215_213046 > Version 1
+ Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Standard build
RMSE
MSE
Optimization metric
3.681
13.551
The model often predicts a value that is within +/- 3.681 of the actual value for Revenue
Predict
Deploy
Overview
Scoring
Advanced metrics
Model leaderboard
Column impact
Search columns...
1 InvoiceNo 31.208%
2 Quantity 20.68%
3 Description 15.548%
Impact of InvoiceNo on prediction of Revenue
Impact on prediction
60.92...
44.21...
29.21...
Dataset_20241215_213046
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Revenue
Numeric prediction
Predict

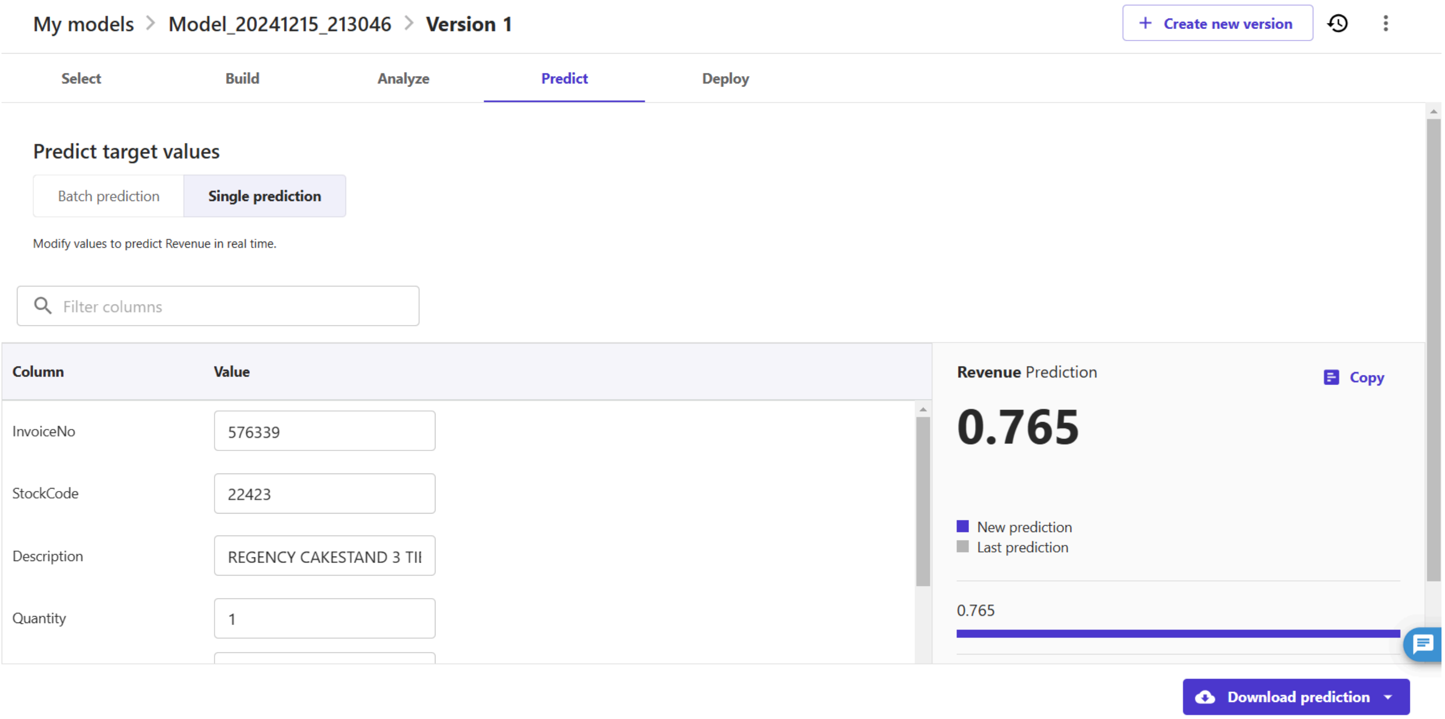

My models > Model_20241215_213046 > Version 1
+ Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Standard build
RMSE
MSE
Optimization metric
3.681
13.551
The model often predicts a value that is within +/- 3.681 of the actual value for Revenue
Predict
Deploy
Overview
Scoring
Advanced metrics
Model leaderboard
Column impact
Search columns...
1 InvoiceNo 31.208%
2 Quantity 20.68%
3 Description 15.548%
Impact of Quantity on prediction of Revenue
Impact on prediction
215.94...
80.65...
10.65...
Dataset_20241215_213046
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Revenue
Numeric prediction
Predict


My models > Model_20241215_213046 > Version 1
Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Standard build
RMSE
MSE
Optimization metric
3.681
13.551
The model often predicts a value that is within +/- 3.681 of the actual value for Revenue
Predict
Deploy
Overview
Scoring
Advanced metrics
Model leaderboard
RMSE
Actual values of Revenue
358
85
-65
-65
-25
15
55
95
135
175
215
255
295
358
Model accuracy insights
The model often predicts a value that is within +/- 3.681 of the actual value for Revenue
* As the thickness of the RMSE band on a model increases, the lower the accuracy of prediction.
Dataset_20241215_213046
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Revenue
Numeric prediction
Predict

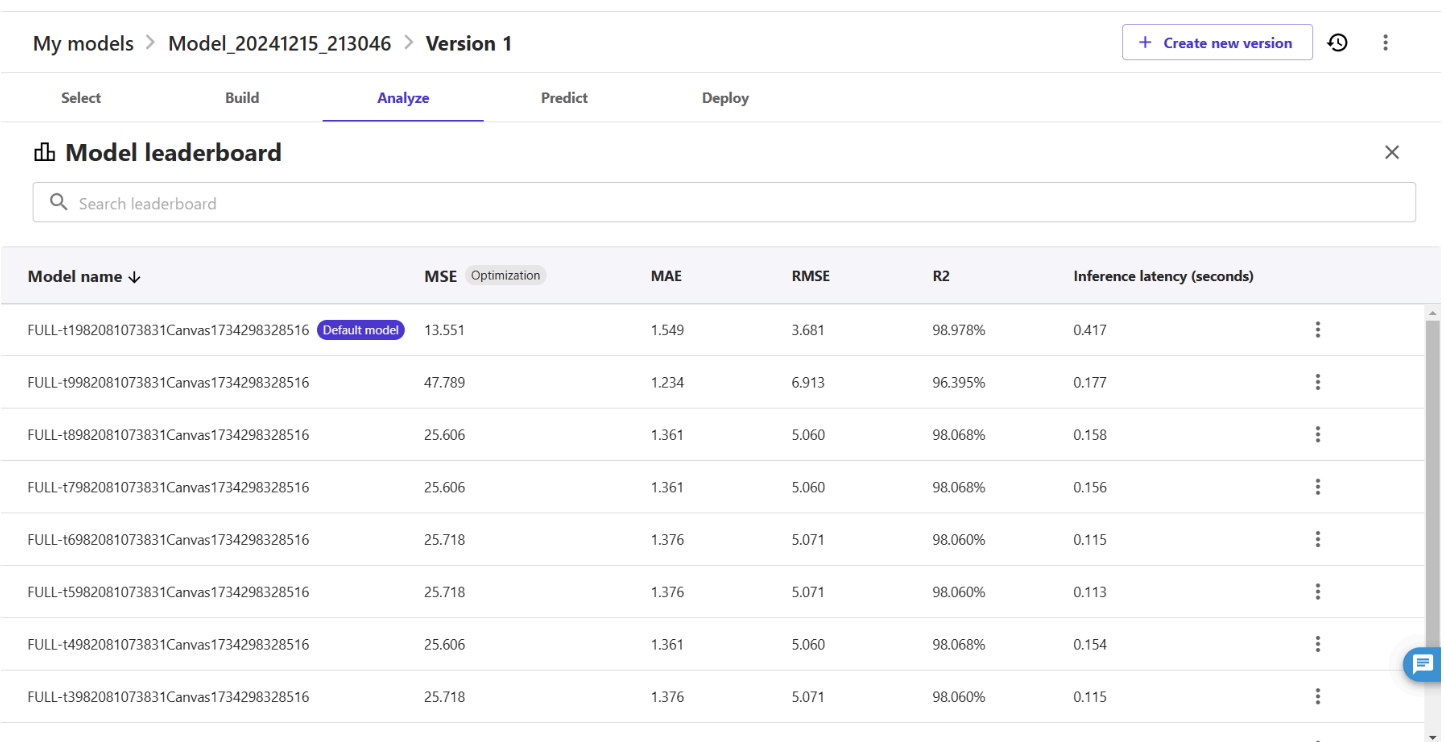

My models > Model_20241215_213046 > Version 1
Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Standard build
RMSE
MSE
Optimization metric
3.681
13.551
The model often predicts a value that is within +/- 3.681 of the actual value for Revenue
Predict
Deploy
Overview
Scoring
Advanced metrics
Model leaderboard
R2
98.978%
MAE
+/-1.549
MAPE
Not available
RMSE
3.681
Metrics table
Residuals
Error density
Metrics table
Metric name
Value
mae
1.549
Dataset_20241215_213046
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Revenue
Numeric prediction
Predict


My models > Model_20241215_213046 > Version 1
Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Standard build
RMSE
MSE
Optimization metric
3.681
13.551
Predict
Deploy
The model often predicts a value that is within +/- 3.681 of the actual value for Revenue
Overview
Scoring
Advanced metrics
Model leaderboard
R2
98.978%
MAE
+/-1.549
MAPE
Not available
RMSE
3.681
Metrics table
Residuals
Error density
MAE
8.35...
-26.19...
es - predicted values
Dataset_20241215_213046
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Revenue
Numeric prediction
Predict


My models > Model_20241215_213046 > Version 1
Create new version
Select
Build
Analyze
Predict
Deploy
Model status
Standard build
RMSE
MSE
Optimization metric
3.681
13.551
Predict
Deploy
The model often predicts a value that is within +/- 3.681 of the actual value for Revenue
Overview
Scoring
Advanced metrics
Model leaderboard
R2
98.978%
MAE
+/-1.549
MAPE
Not available
RMSE
3.681
Metrics table
Residuals
Error density
Density
0.09...
Dataset_20241215_213046
Total columns: 11
Total rows: 3,075
Total cells: 33,825
Revenue
Numeric prediction
Predict



Model Leaderboard:



Model name ↓	MSE Optimization	MAE	RMSE	R2	Inference latency (seconds)
FULL-t1982081073831Canvas1734298328516 Default model	13.551	1.549	3.681	98.978%	0.417
FULL-t9982081073831Canvas1734298328516	47.789	1.234	6.913	96.395%	0.177
FULL-t8982081073831Canvas1734298328516	25.606	1.361	5.060	98.068%	0.158
FULL-t7982081073831Canvas1734298328516	25.606	1.361	5.060	98.068%	0.156
FULL-t6982081073831Canvas1734298328516	25.718	1.376	5.071	98.060%	0.115
FULL-t5982081073831Canvas1734298328516	25.718	1.376	5.071	98.060%	0.113
FULL-t4982081073831Canvas1734298328516	25.606	1.361	5.060	98.068%	0.154
FULL-t3982081073831Canvas1734298328516	25.718	1.376	5.071	98.060%	0.115

4. Review Results: Analyze the model leaderboard and performance metrics.

Performance Metrics Review:

MAE (Mean Absolute Error):

The MAE is 13.551, which signifies the average absolute difference between the predicted and actual revenue values. Given the average revenue of 17.39, this MAE represents approximately 78% of the average revenue. This is a relatively low error, indicating that the model's predictions are fairly close to the actual values and that it performs with good precision in predicting revenue.

RMSE (Root Mean Squared Error):

The RMSE is 3.681, which indicates how much larger errors are impacting the model's predictions. Since RMSE penalizes larger errors more heavily, this relatively low value suggests that the model does not have large outliers skewing the predictions. The fact that the RMSE is significantly lower than the MAE further confirms the model's consistency and that errors are evenly distributed across predictions without excessively large deviations.

R^2 Score:

The R^2 score of 98.978% is a strong indication that the model explains nearly 99% of the variance in the revenue data. This high R^2 score shows that the model does an excellent job of capturing the underlying patterns and trends in the dataset, providing a highly accurate fit for predicting revenue.

Insights:

The performance metrics suggest that the model is performing exceptionally well. The low MAE demonstrates that the model's revenue predictions are quite accurate, while the low RMSE reflects that the model is stable and free from major prediction outliers. With an R^2 score of nearly 99%, the model explains nearly all of the variance in revenue, making it a highly effective tool for predicting future revenue. These results highlight the model's potential for practical, real-world applications in business analytics and forecasting.

Model Leaderboard:

The model at the top of the leaderboard has the lowest MAE of 13.551, suggesting it has the most accurate predictions. The subsequent models show slightly higher MAE values, which indicates a slight drop in prediction accuracy as you move down the leaderboard.

The top model in the leaderboard has an RMSE of 3.681, which is relatively low, indicating that large errors are not significantly skewing predictions. The models lower down on the leaderboard show slightly higher RMSE values, but they are still relatively low, reflecting a generally consistent performance in terms of error distribution.

The top model on the leaderboard has an R^2 score of 98.978%, which means that it successfully explains nearly 99% of the variance in the data, suggesting a highly accurate and well-fitting model. The other models show R^2 values in the range of 96% to 98%, which still reflects a strong ability to explain the data but with slightly less predictive power compared to the top model.

A new metric inference latency is present here. This metric indicates the amount of time the model takes to make a prediction. A lower latency is generally preferred for real-time applications. The top model has an inference latency of 0.417 seconds, which is reasonable, but models further down the leaderboard show lower latencies, with the fastest model taking 0.113 seconds to make predictions.

Insights:

The top model has the best performance with the lowest MAE, RMSE, and the highest R^2 score. It is the most accurate model in terms of prediction, while still having a reasonable inference latency.

Overall, the leaderboard showcases that while inference speed can be improved, the primary focus should remain on optimizing for accuracy, as evidenced by the strong performance of the top-ranked model.

Address ethical issues like bias in training data and privacy concerns.

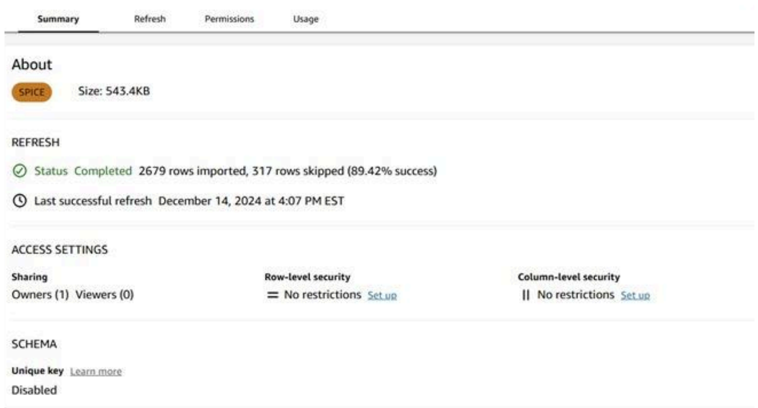
In this dataset, geographical bias may arise due to the over-representation of UK transactions, potentially leading to reduced accuracy for other regions. Similarly, product or customer segment bias could favor more frequently occurring categories or customer types, leaving under-represented groups underserved. Temporal bias is another concern, as the dataset spans a specific period (2010–2011), and the trends it reflects may not align with current or future consumer behavior. To mitigate these biases, balanced sampling techniques should be employed to ensure fair representation of all groups.

The risk of customer identification or revealing private business behaviors increases if data is not adequately anonymized. For instance, predictions based on revenue or quantity might inadvertently expose individual spending habits or operational details of wholesalers. To address these risks, data should be anonymized by removing or masking unique identifiers and presenting results at an aggregated level, such as by country or product category. Strict adherence to data protection standards, such as GDPR, ensures compliance with privacy regulations by minimizing data usage, obtaining explicit consent, and defining clear data handling policies. Secure model deployment practices, including robust access controls, are essential to safeguarding the model and its associated datasets against unauthorized access or misuse.

Stakeholders should also have a clear understanding of the model's purpose, the data used for training, and how predictions are generated

4. Visualization

Task 1: Connect QuickSight to the processed data in S3



Task 2: Design a dashboard with at least 4 insightful visualizations

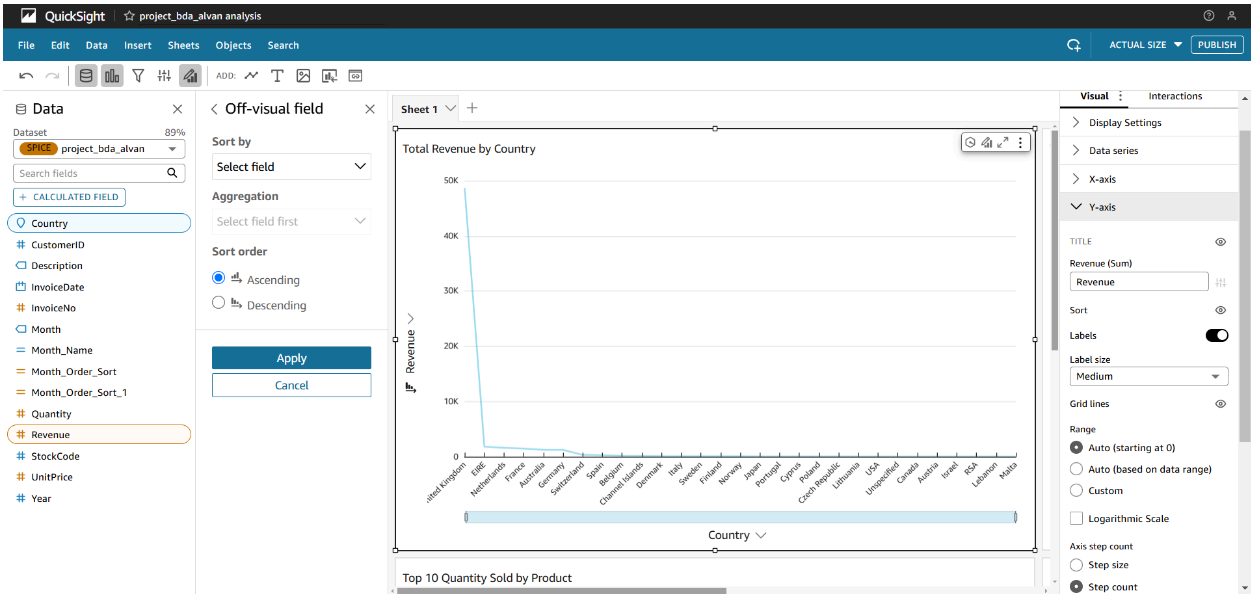
1. Total Revenue by Country

Purpose of the Visualization- This chart highlights the total revenue contributed by each country, helping to identify key markets for the store's operations.

Insights Derived from the Data:

a. The UK generates the highest revenue, which aligns with expectations since the store is based in the UK.
b. Neighboring countries like France, Germany, and Switzerland also contribute significantly to the revenue.
c. Australia stands out as an outlier among the top contributors, suggesting the store offers products that resonate well with Australian customers.

Filters or Parameters Applied - No specific filters were applied. The data represents all countries contributing to the revenue.



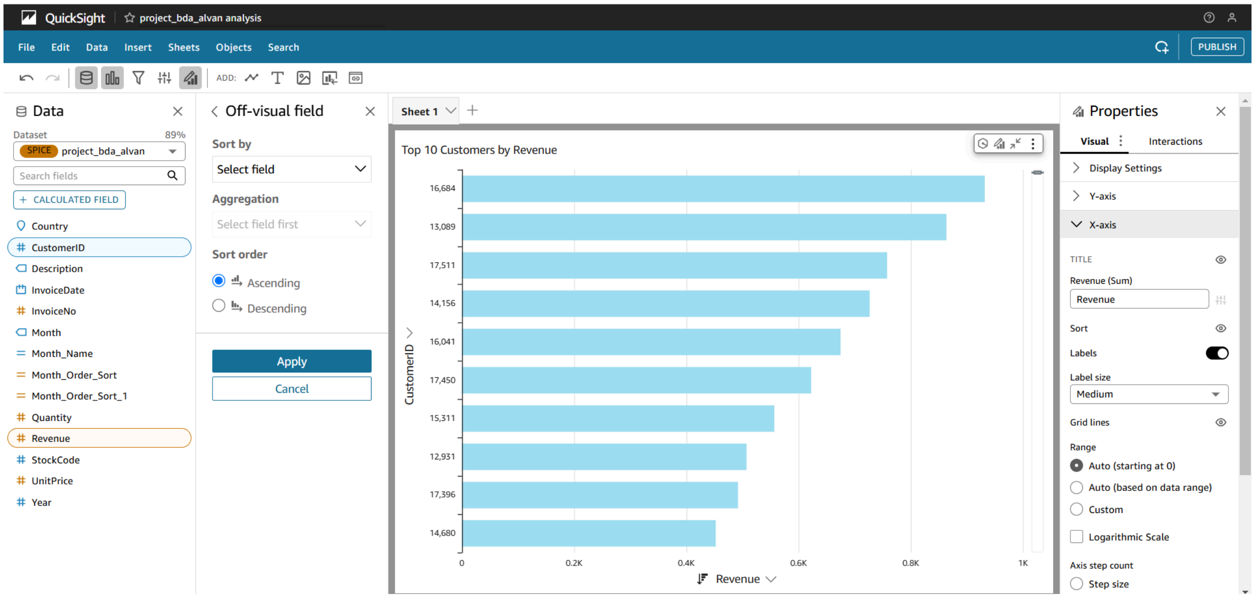
2. Top 10 Customers by Revenue

Purpose of the Visualization- This chart identifies the top 10 customers based on the revenue they generated, helping to focus on high-value customers.

Insights Derived from the Data:

a. The highest revenue was generated by Customer ID 16,684, followed by 13,089, 17,511, 14,156, and 16,041.
b. The revenue range for these top customers is between 600 and 1,000.
c. These insights can guide personalized marketing campaigns and customer loyalty initiatives.

Filters or Parameters Applied - The chart is limited to the top 10 customers by total revenue for better focus and clarity.



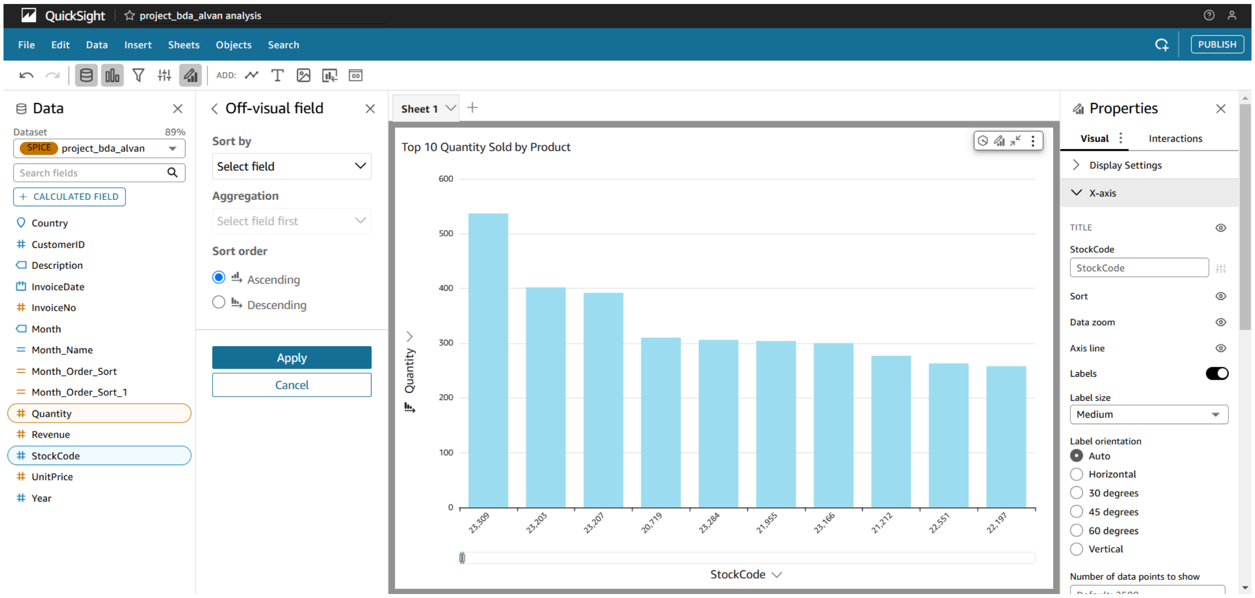
3. Top 10 Quantities Sold by Product

Purpose of the Visualization- This chart showcases the products with the highest quantities sold, helping to identify the store’s most popular items.

Insights Derived from the Data:

a. The product associated with StockCode 23,309 has the highest quantity sold, followed by 23,203, 23,207, 20,719, and 23,284.
b. The quantities for these products range between 300 and 550.
c. These products likely drive significant sales volume, emphasizing the need to maintain adequate stock levels and consider promotional strategies for these items.

Filters or Parameters Applied- The chart focuses on the top 10 products by quantity sold for actionable insights.



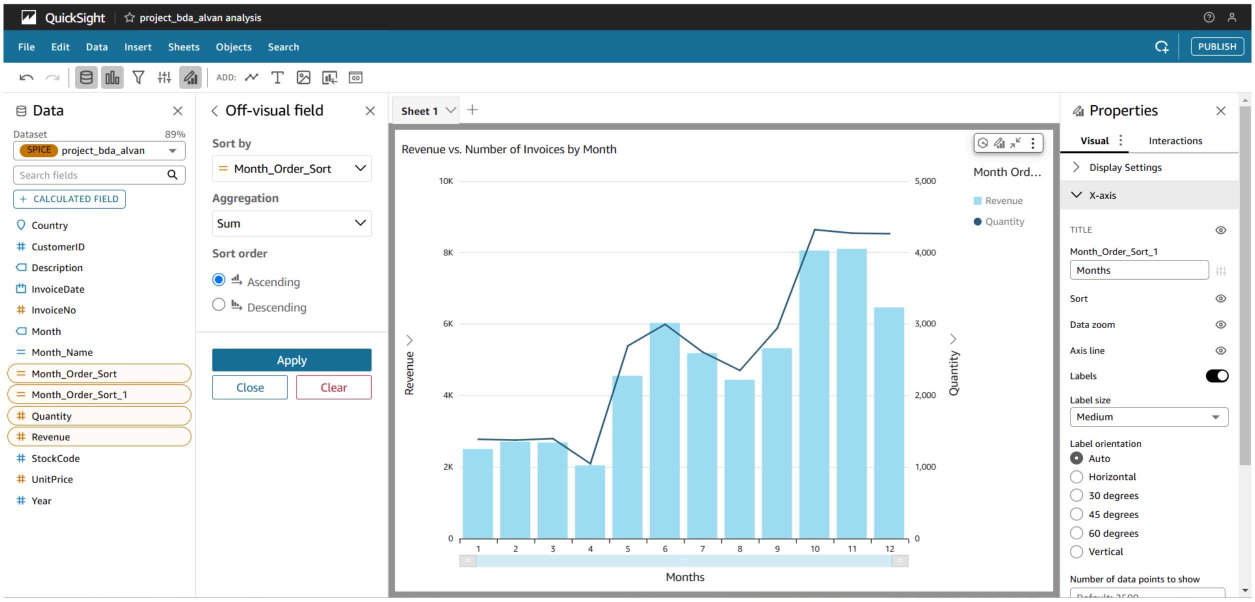
4. Revenue vs Number of Invoices by Month

Purpose of the Visualization- This dual-axis clustered bar combo chart compares revenue with the number of invoices over the months, revealing seasonal trends and purchase behaviors.

Insights Derived from the Data:

a. Revenue shows a progressive increase over the months, peaking in October and November.
b. Despite a decline in revenue in December, the number of invoices remains high, indicating a high volume of smaller-value orders during the holiday season.
c. These trends can inform inventory planning and marketing strategies during peak seasons.

Filters or Parameters Applied A calculated field was used to assign numeric values to months (e.g., January = 1, February = 2), ensuring correct chronological sorting. The chart covers all transactions for the available months in the dataset.



Results:

The analysis and modeling of the online retail dataset yielded significant insights. Using QuickSight, visualizations revealed that the United Kingdom contributed the highest revenue, followed by neighboring countries like France and Germany, with Australia emerging as an outlier. Key customers and top-selling products were identified, while trends showed peak revenues in October and November, likely influenced by seasonal demand.

The predictive model for revenue, built using Amazon SageMaker, showed promising results. The model achieved an R^2 of 92.19%, indicating that it explains a substantial portion of the variance in the revenue data. The RMSE of 10.18 suggests a reasonable prediction error, while the MAE of 103.57 shows the model's average deviation from the actual revenue values. Despite being a strong model, these metrics indicate room for improvement in accuracy.

Conclusion:

Utilizing AWS services such as Amazon SageMaker and QuickSight created a powerful and adaptable framework for analyzing the online retail dataset and constructing predictive models. These services streamlined the entire process, offering seamless data preparation, meaningful visualizations, and efficient model building, all while maintaining a high standard of scalability and security.

AWS delivered an integrated ecosystem that supported every stage of the workflow, from data cleaning to advanced analytics and model deployment. Its versatility, robust performance, and strong security measures made it a standout platform for deriving actionable insights and implementing scalable solutions tailored to the online retail dataset.

Link to Video Demonstration:
https://drive.google.com/file/d/1OJIKzEjkIGaIKkK5mxsyjjgNNXnJ2PWl/view?usp=sharing

References:

https://chatgpt.com/
https://docs.aws.amazon.com/ec2/?nc2=h_ql_doc_ec2
https://docs.aws.amazon.com/s3/?icmpid=docs_homepage_featuredsvcs
https://docs.aws.amazon.com/sagemaker/?icmpid=docs_homepage_ml
https://docs.aws.amazon.com/iam/?icmpid=docs_homepage_security