



# **Final Project**

## **Market Basket Analysis for Online Retail Industry**

**Group Epsilon:**

**Yu Qiu**

(qiu.yu1@northeastern.edu)

**Chengjie Zhang**

(zhang.chengj@northeastern.edu)

**Fatima Nurmakhamadova**

(nurmakhamadova.f@northeastern.edu)

Class ALY6110. 80442: Big Data and Data Management

June 27, 2022

## Summary

In this report, we created a Manual with instructions of doing data analysis, especially the Market Basket Analysis by using Apriori Algorithm for the retail industry. The purpose of this report is to help analysts who have no experience using the big data management tool – Databricks to understand how to use it to tackle the real big data.

We provided the steps from registering the Databrick Community Edition, creating a cluster and a notebook, to writing commands in Notebook and export the final Notebook. Also, we provided instructions on how to collaborate with team members by using Community Edition.

Moreover, we have used a dataset of a UK-based online retail company to do the analysis and Market Basket analysis. We also provided the insights and findings from our analysis.

At the end of the report, we analyzed the pros and cons of using Databricks to manage big data and the challenges we met through the whole process. The programming languages used in our analysis involve Python and SQL.

## Dataset Source and Introduction

The dataset we use is from [Kaggle](#) (Cagirici, n.d.). According to Cagirici, (n.d.), the dataset is from a UK-based online sales store which contains the transaction data from January 12, 2009, to September 12, 2011.

Since the dataset is huge, all the recordings originally are provided in two sheets of the Excel document. After merging the 2 sheets, there are 1,067,371 rows/observations and 8 columns/variables in the dataset.

**Table 1:***Information about Variables*

No.	Variables	Meaning	Data Type
1	<b>Invoice</b>	The invoice number made by a customer; those starts with “C” are the canceled transactions.	Categorical
2	<b>Stock Code</b>	Unique product code for each product	Categorical
3	<b>Description</b>	Product name	Categorical
4	<b>Quantity</b>	Number of products sold in each transaction	Numeric: discrete
5	<b>Invoice Date</b>	The date of invoices happened	Date & Time
6	<b>Price</b>	The price of the product	Numeric: continuous
7	<b>Customer ID</b>	Unique customer ID for each customer; customers who do not register will have not customer ID	Categorical
8	<b>Country</b>	The customers’ location	Categorical

## A real-world problem

A real-word problem we aim to solve in this analysis is how to create a better product bundle sales strategy for online retail companies.

Market basket analysis is an association-rule data mining that could be used to analyze big transaction data from retailers to understand which products frequently appear together in the same order (Pradhan et al., 2022). This analysis could help retailers to better understand their customers and then arrange accurate product placements, create attractive product bundle strategies, and increase customer retention (Loshin & Reifer, 2013).

## Why it is important

Market basket analysis is a classical topic to understand customer purchase patterns. According to Kadlaskar (2021), Market Basket Analysis could be used to increase customer engagement,

increase sales, improve customer experience, optimize marketing campaigns, etc., therefore this is important to retail industries. The specific problem we want to solve is utilizing the Marketing Basket Analysis to improve the online retailers' product bundle sales strategy so that it could help the online retailers to increase revenue.

## Methodology

The methodology that will be used to do the Market Basket Analysis is the Associated Rule Learning. The Association Rule mainly includes 2 parts: one is to find frequent item pairs in one order by pre-setting a threshold on the minimum frequency, and the second is to calculate the confidence of these pairs which is the probability of one product B given that product A is in the cart, like  $P(B|A)$  (Kadlaskar, 2021).

There are several different algorithms that could be used to conduct Market Basket Analysis. One of the most popular algorithms is Apriori Algorithm which could help to find the frequent product combinations and association rules (Kadlaskar, 2021).

While since the transaction data of online retailers is huge, big data management tools are also needed to store and handle the data. I would propose to use Databricks which have APIs with multiple programming languages as the tool to process the big transaction data and it contains Spark. The advantage of using Apache Spark rather than Hadoop MapReduce is because Spark allows real-time data analytics (Garg, 2022). The RDDs in Spark could realize high-speed data processing which could be 100 times faster than Hadoop (Garg, 2022).

## Content

### Get Started with Databricks

In this report, we will use Databricks to complete the report. **Databricks provides a unified, open platform for all your data.** The idea behind the **Databricks** is to provide a separate space for

processing data, independent of the hosting environment and Hadoop cluster management, and the entire process is done in the cloud. The product has several core concepts: Notebooks provide a way to interact with data and build graphs, and when users understand how to display data, they can start building dashboards to monitor certain types of data. Finally, users can schedule the runtime of Apache Spark through the platform's task launcher.

## Step 1 Create an account of DataBricks community

So first we need to log on to the website and register a Databricks account:

<https://databricks.com/try-databricks>

**Try Databricks for free**

An open and unified data analytics platform for data engineering, data science, machine learning, and analytics. From the original creators of Apache Spark™, Delta lake, MLflow, and Koalas.

**aws** **Microsoft Azure** **Google Cloud**

**Databricks trial:**

- Collaborative environment for data teams to build solutions together.
- Interactive notebooks to use Apache Spark™, SQL, Python, Scala, Delta Lake, MLflow, TensorFlow, Keras, Scikit-learn and more.
- Available as a 14-day full trial in your own cloud, or as a lightweight trial hosted by Databricks.

**Used by:**

**Please tell us about yourself**

**First Name: \***  
Chengle

**Last Name: \***  
Zhang

**Company \***  
Northeastern University

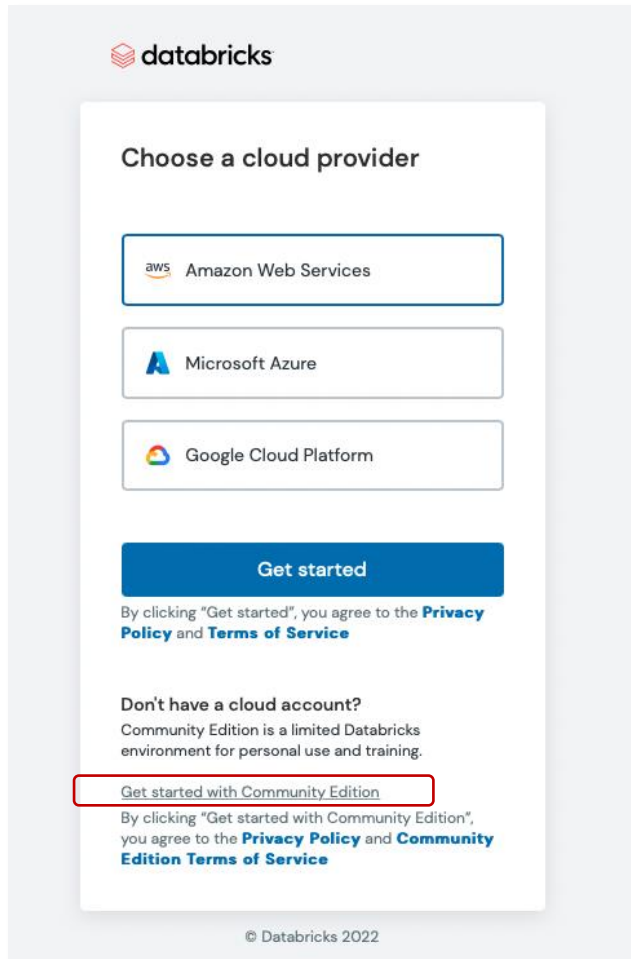
**Company Email \***

**Title \***  
Zhang

**Phone Number**

**Country: \***  
United States

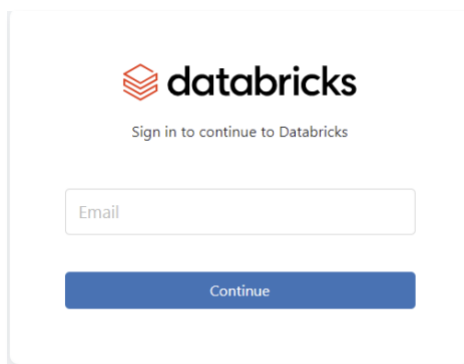
And then we need to choose a cloud provider: such as AWS (Amazon Web Services), Microsoft Azure or Google Cloud Platform. In this project, we used the Community Edition which is free to use.



The image shows the Databricks registration page. At the top is the Databricks logo. Below it is the heading "Choose a cloud provider". There are three buttons for "Amazon Web Services", "Microsoft Azure", and "Google Cloud Platform". Below these is a blue "Get started" button. Underneath the button is a line of text: "By clicking 'Get started', you agree to the [Privacy Policy](#) and [Terms of Service](#)". Below this is a section titled "Don't have a cloud account?" which explains that the Community Edition is for personal use and training. At the bottom of this section is a link "Get started with Community Edition" which is highlighted with a red rectangle. Below the link is another line of text: "By clicking 'Get started with Community Edition', you agree to the [Privacy Policy](#) and [Community Edition Terms of Service](#)". At the very bottom of the page is the copyright notice "© Databricks 2022".

During registration, we could choose the Community Edition at the bottom of the page.

After creating the account, we can now login to the portal of Databricks.



The image shows the Databricks login page. At the top is the Databricks logo. Below it is the text "Sign in to continue to Databricks". There is an input field for "Email". Below the input field is a blue "Continue" button.

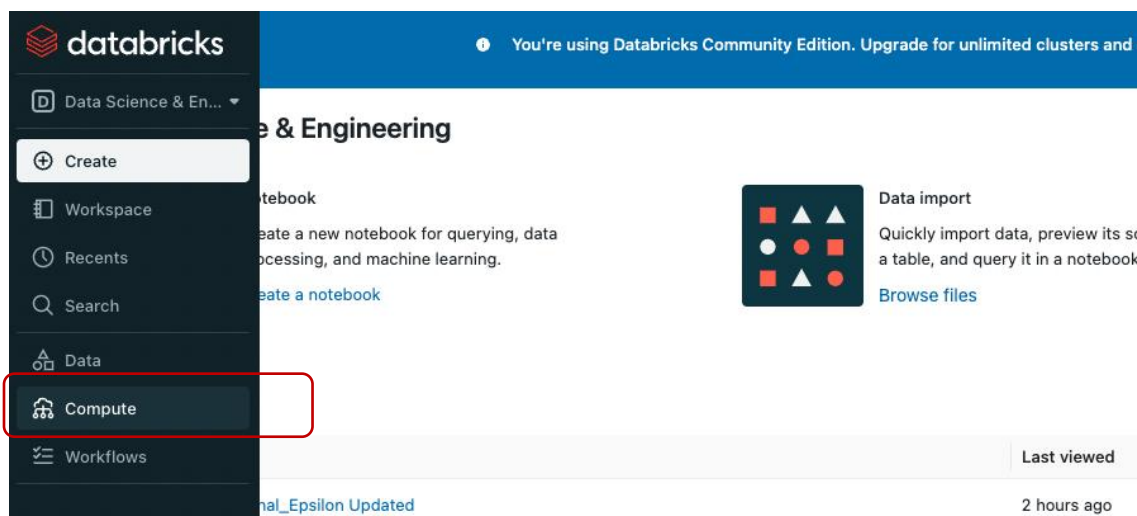
## Step 2 Create a cluster in DataBricks

The user interface of DataBricks Community Edition is like below. In this step we need to create a cluster.

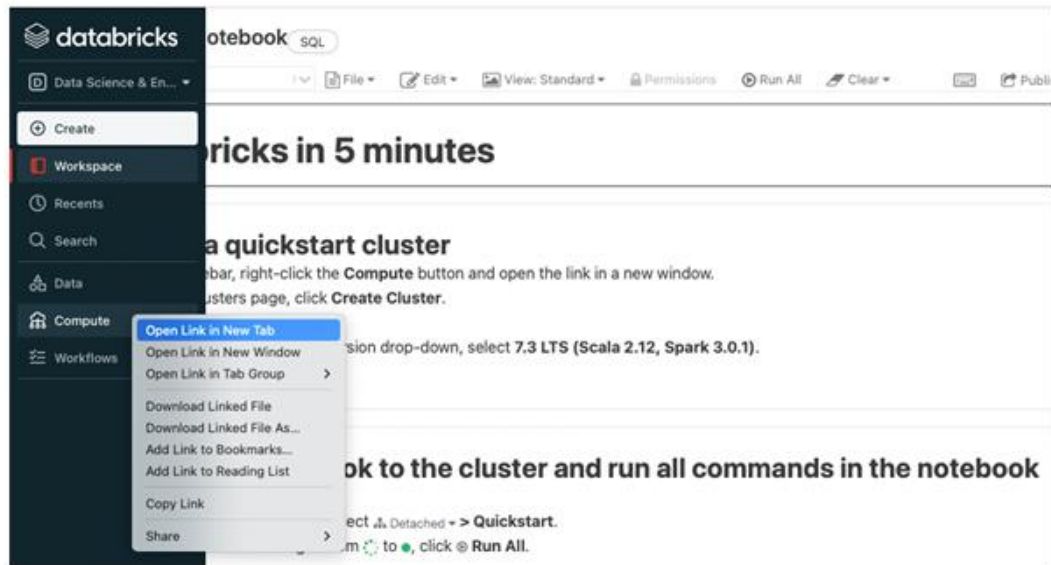
The cluster in Databricks is for running data engineering, data science and data analytics workload which contains computation resources and configurations ("Clusters - Azure Databricks", 2022). Therefore, we could do ETL, streaming analysis, ad-hoc analysis and machine learning on the cluster.



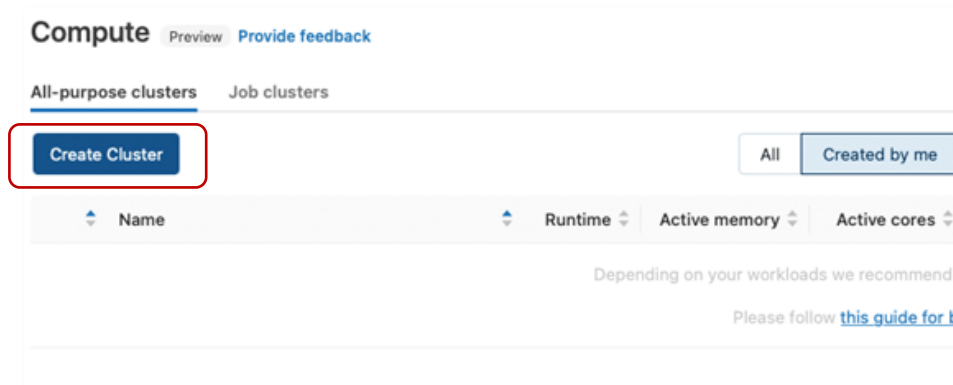
Go to the left-hand side menu bar, we could find there is a Compute button.



We could click on “Compute” in the left-side menu bar, open it in a new tap.



Click the “Create Cluster” button at the top.



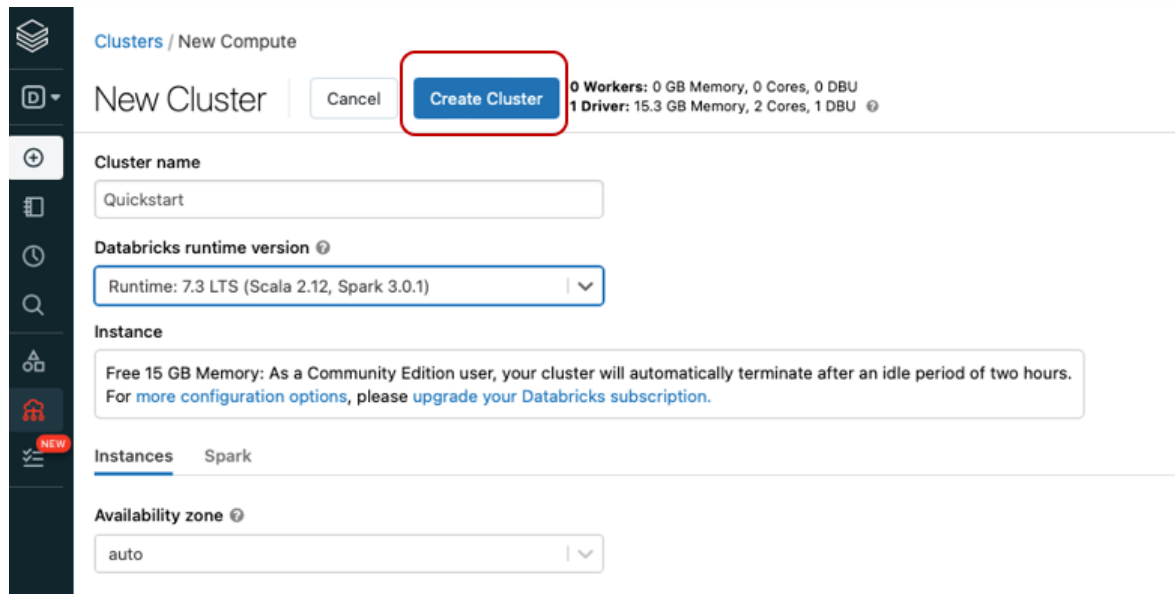
To create a cluster, we need to name the cluster and choose the Databricks runtime version. Databricks runtime runs on clusters and includes both Spark and components which could improve the performance and security of big data analysis ("What is Databricks Runtime? - Databricks", n.d.).

Therefore, different versions of Databricks Runtime include different versions of Apache Spark and Scala. According to the Tutorial, we chose this section, and then by clicking “Create Cluster”,



the new cluster has been created. At the very first, we chose 7.3 LTS (includes Apache Spark 3.0.1, Scala 2.12).

7.3 LTS (includes Apache Spark 3.0.1, Scala 2.12)



Clusters / New Compute

New Cluster

0 Workers: 0 GB Memory, 0 Cores, 0 DBU  
1 Driver: 15.3 GB Memory, 2 Cores, 1 DBU

Cluster name  
Quickstart

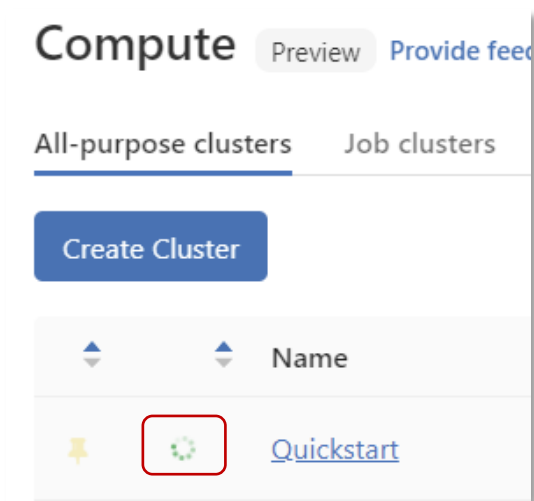
Databricks runtime version  
Runtime: 7.3 LTS (Scala 2.12, Spark 3.0.1)

Instance  
Free 15 GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For [more configuration options](#), please [upgrade your Databricks subscription](#).

Instances Spark

Availability zone  
auto

When the cluster is created, we need to wait for the green tick beside to be shown which means the cluster is fully started and ready to use. Usually it takes 3- 5 minutes, for higher version of runtime, it might take longer time.



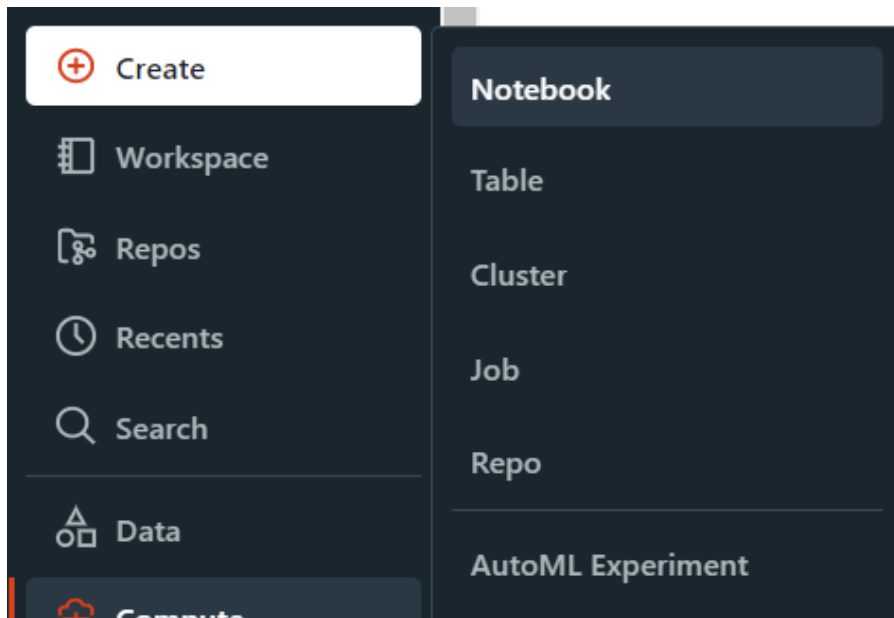
Compute

All-purpose clusters Job clusters

	Name
	<a href="#">Quickstart</a>

### Step 3 Create a Notebook in DataBricks

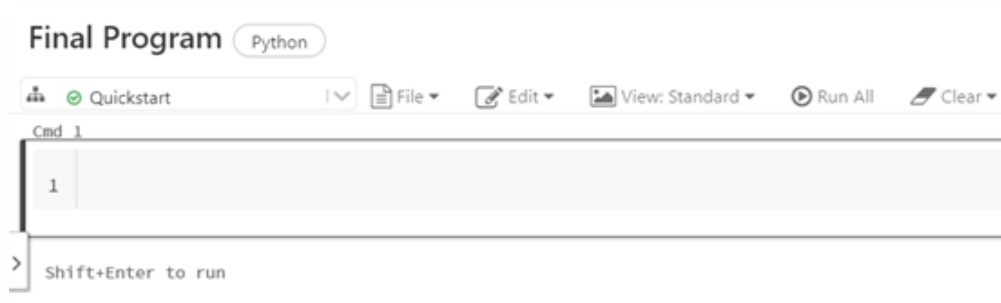
So now we can create Notebook. Go to the homepage and on the left-side menu bar we can find “Create”, and we need to use create function to create a notebook in Databricks.



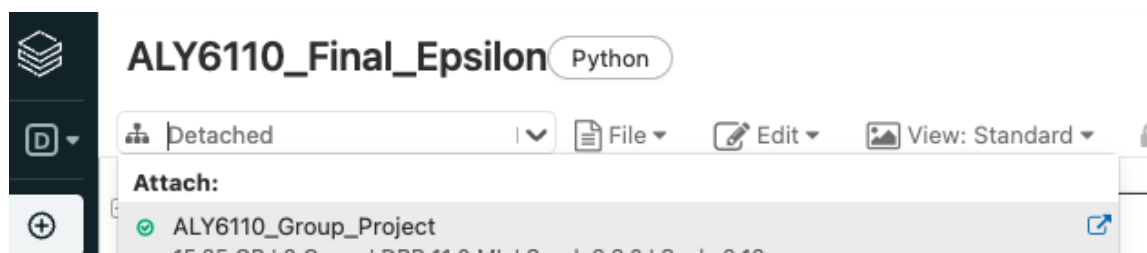
When creating a notebook, we need to name this notebook and select the Default Language. There are four default languages we can select: Python, R, SQL, and Scala. In this program, we select Python as the default language as we will use Python mainly for our analysis.

A screenshot of the 'Create Notebook' dialog box in Databricks. The dialog has a title bar with 'Create Notebook' and a close button. It contains three input fields: 'Name' with the text 'Final Program', 'Default Language' with a dropdown menu showing 'Python', and 'Cluster' with a dropdown menu showing 'Quickstart'. At the bottom right, there are two buttons: 'Cancel' and 'Create'.

After opening the notebook created, we could see that the UI is similar to Jupyter Notebook, we can write code in each command cell. Beside the file name, we can find the Default language is Python.



And below the file title, we need to select a Cluster to run this notebook. Click “Detached” and choose the cluster we just created which is ready to use.



Now, we can start to write code to analyze.

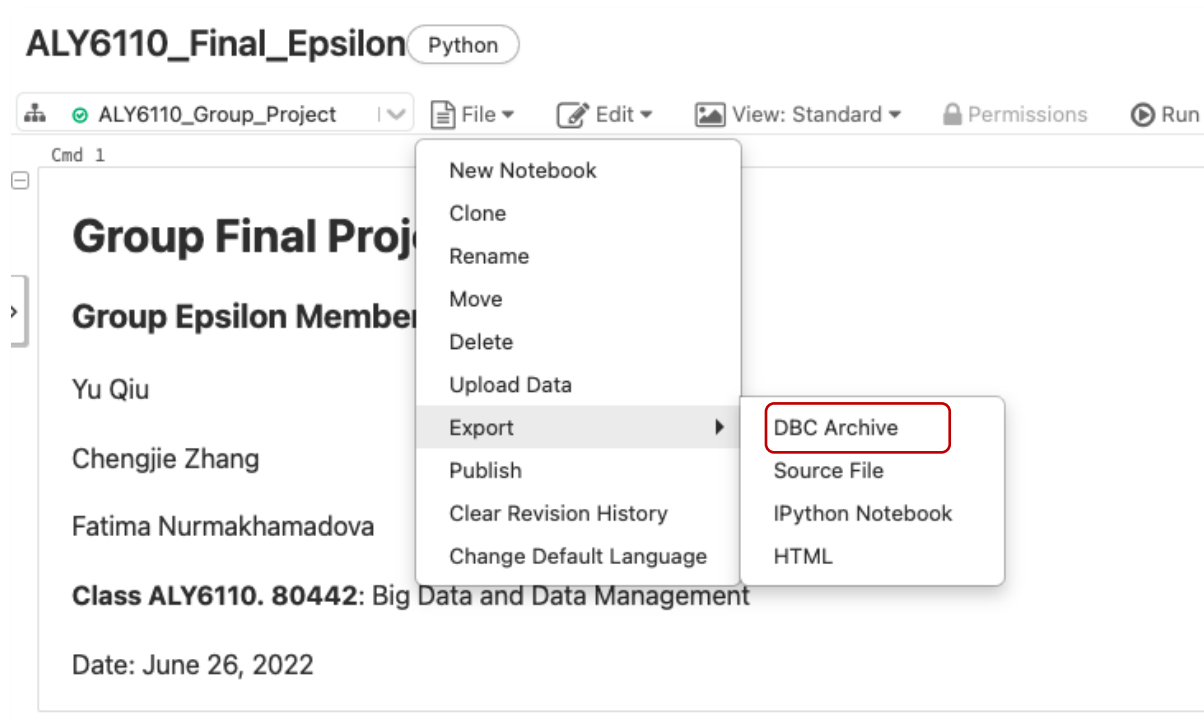
## Collaboration with Community Edition

Databricks Community Edition has no collaborative features, so if we want to integrate the codes of team members, we need to download the Notebook and share with the Notebook to continue writing the following codes. In this way, we could ensure that all the analysis done by different members are based on the same cleaned dataset. Below are the steps how to download Notebook from Databricks and upload it to Databricks.

## Step 1 Download DBC File

1. Go to the top menu of Notebook and click “File”
2. Chose Export and then click DBC Archive

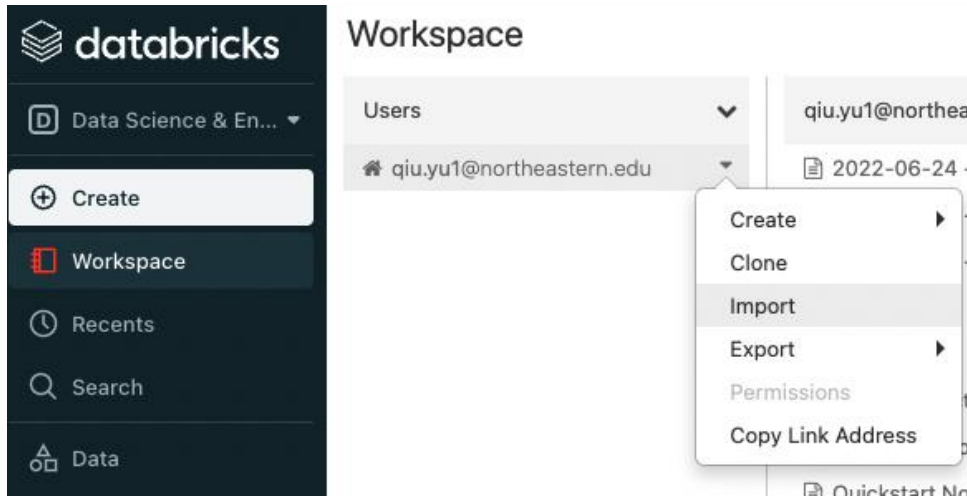
DBC Archive version contains metadata and results of all the commands in the Notebook, therefore it will be easier for teammates to check the previous process.



## Step 2: Share the DBC file with teammates

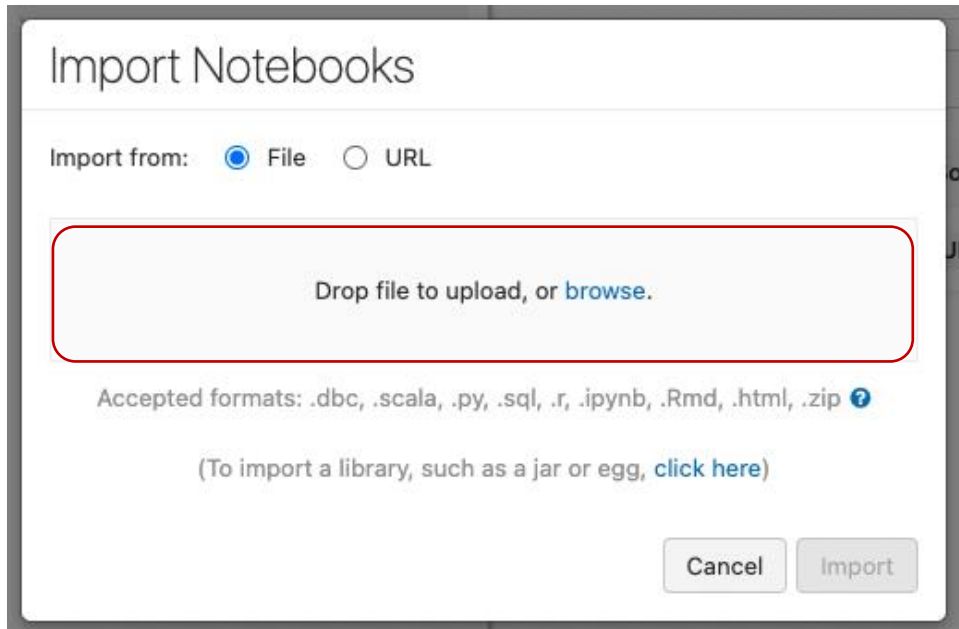
After receiving the shared file, other teammates could use below the steps to upload DBC file and open it in Databricks to continue working on it.

## Step 3: Upload DBC Files



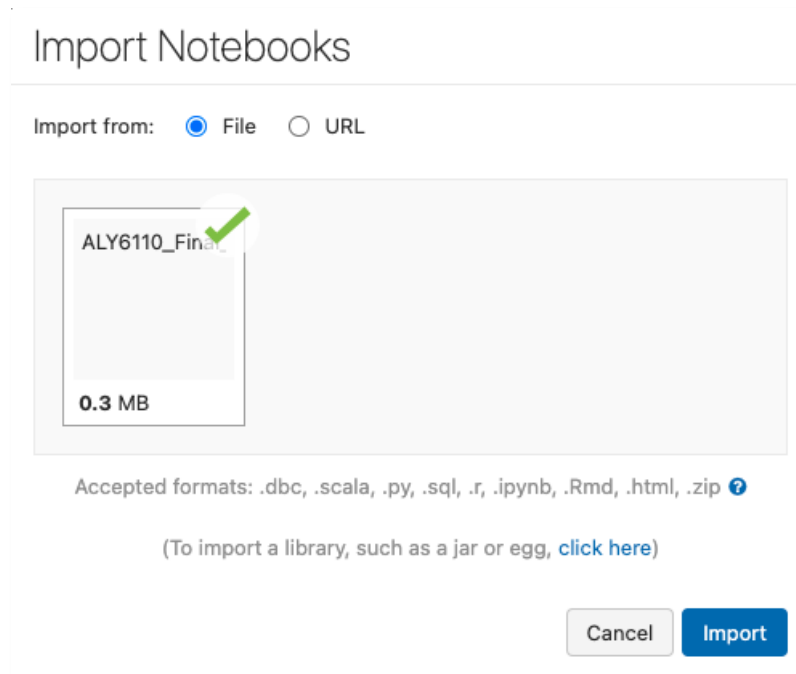
1. Go to Databricks Portal and click “Workspace” in the left menu bar;
2. Then click the arrow beside your user account and choose “Import”;

After clicking “Import”, we could see below the window popping out.



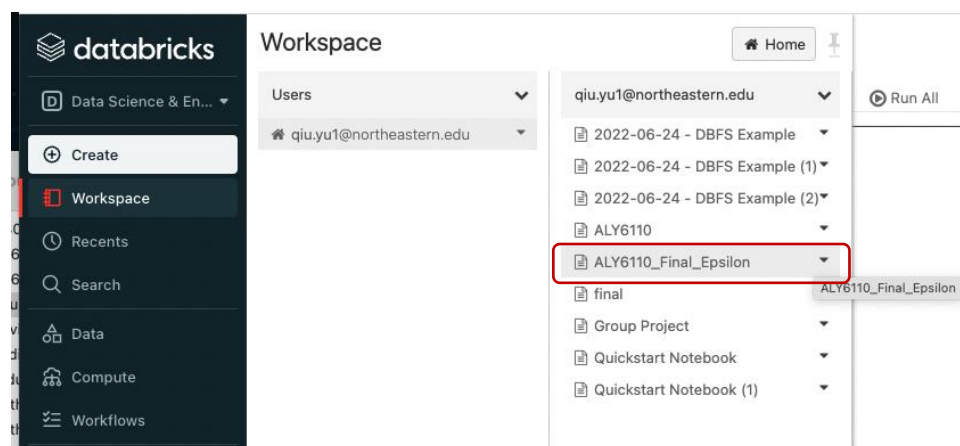
3. Drag the DBC file into the box “Drop file to upload”, or click “browse” to upload;

It might take several minutes to have the whole file uploaded, please be patient. When seeing the green tick shown as below, it is the time to click “Import” button below.



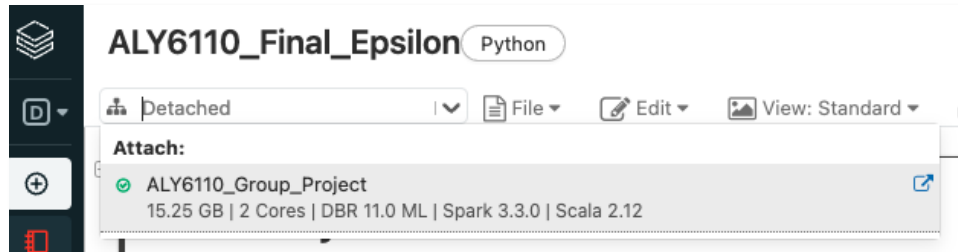
Then we could see the DBC file shown in the list of the user account.

#### 4. Double click the file and the Notebook will be opened in Databricks;



Same as the steps of creating a new notebook, we need to choose a Cluster for running the notebook.

**Step 4: Click Detached in the top menu and choose the cluster which is running (with the green tick ready).**

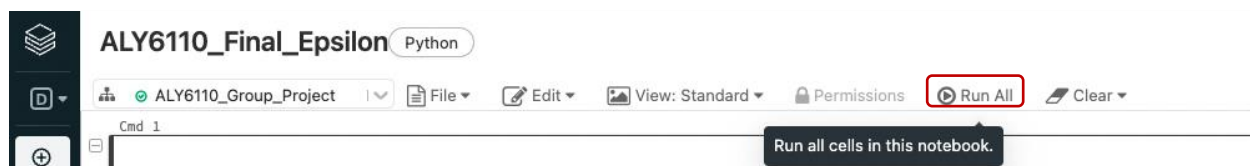


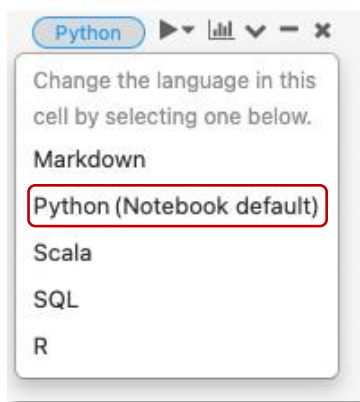
**Step 5: Check the dataset file path in the notebook commands.**

Each team member should upload the dataset file(s) to their own Databricks and check the file path whether the file names are the same with the ones with your uploaded ones. If the file names are the same and the location is the same, then no need to change the file path.

```
# File location and type
#Please upload two attached files to DBFS, particularly in /FileStore/tables/
f1_loc = "/FileStore/tables/online_retail_2009_2010.csv"
f2_loc = "/FileStore/tables/online_retail_2010_2011.csv"
file_type = "csv"
```

**Step 6: Then click “Run All” in the top menu and the exiting command will be ran**



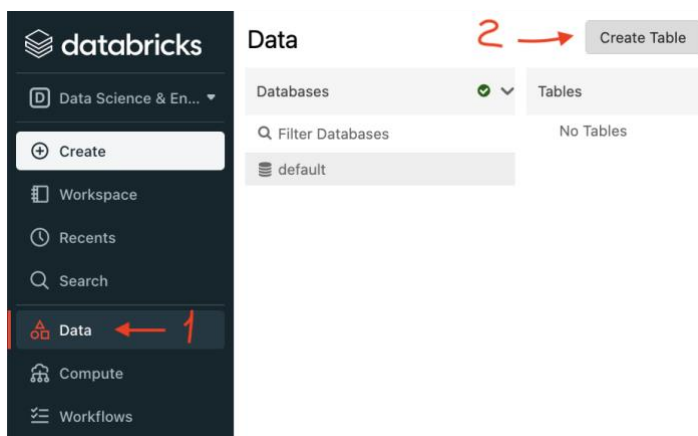
**Step 8: Write your own codes based on the existing codes/commands**

Please note for executing the notebook attached to this report, you need to use Python language unless other is specified. For this you can choose “Python” as the Syntax in top right angle of the cell or write %python command on top of each cell running.

## Data Uploading

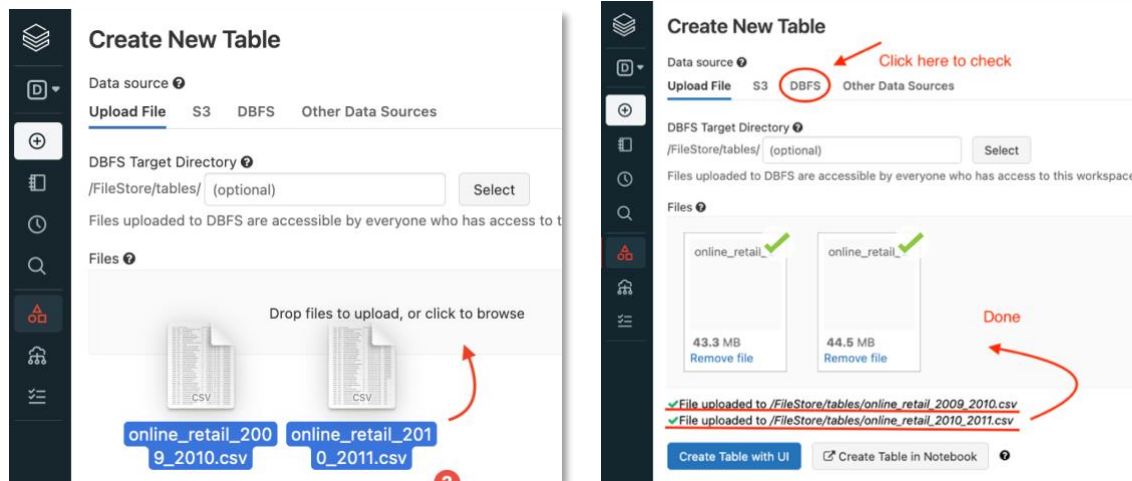
**Step 1: Create tables**

First, we need to upload two CSV files attached to this submission into DBFS, particularly into /FileStore/tables/ directory. For this, we need to click on Data in the left menu, then to Create Table. We will not create a table now, but just upload files to run the notebook we have attached to the submission.





Then, we need to upload two files by dropping them into the 'Drop files' window. It takes some time to be uploaded but once it done, we will see the green tick on each file uploaded. Below that, we can see the file location where it was uploaded. This will be used to read the files into data frames further.



We can double check the file load but going to DBFS tab there on top, select the FileStore, then tables. And here we can see all the uploaded files we have. The Databricks File System is a distributed file system that stores all the files we have that can be accessed by all clusters.



After uploading two CSV files attached to this submission into DBFS, we could go back to the notebook created before.

## Step 2: Read the Datasets

Now we can read the csv files into the Pyspark data frame. For this, we first define file location for each file that we copied while uploading them into DBFS. Here file 1 is Online Retail records for the year 2009 – 2010, and file 2 is for the year 2010-2011.

```
1 # File location and type
2 #Please upload two attached files to DBFS, particularly in /FileStore/tables/
3 f1_loc = "/FileStore/tables/online_retail_2009_2010.csv"
4 f2_loc = "/FileStore/tables/online_retail_2010_2011.csv"
5 file_type = "csv"
```

Then we set options to be considered while reading particularly the CSV files. Here, we have the option `inferSchema` which allows Spark to automatically define column names, data types, and nullability, otherwise, all the columns will have an object data type. We first set it as true, and most of the variables have the correct data types. However, we face some challenges that will be explained in the comments part, thus we kept this option false, and changed the data types manually. Other options we included are to consider the first row as a header, as well as a delimiter as a comma sign.

```
7 # CSV options of file parameters
8 infer_schema = "false" #true
9 first_row_is_header = "true"
10 delimiter = ","
```

Using `spark.read.format()` function, and including the options we defined and data location, we now can read two CSV files into two separate Spark data frames.

```

12 # The applied options are for CSV files. For other file types, these will be ignored.
13 # Import csv file for 2009 - 2010 and read it into Spark data frame
14 df1 = spark.read.format(file_type) \
15     .option("inferSchema", infer_schema) \
16     .option("header", first_row_is_header) \
17     .option("sep", delimiter) \
18     .load(f1_loc)
19
20 # Import csv file for 2010 - 2011 and read it into Pyspark data frame
21 df2 = spark.read.format(file_type) \
22     .option("inferSchema", infer_schema) \
23     .option("header", first_row_is_header) \
24     .option("sep", delimiter) \
25     .load(f2_loc)

```

Here are two Spark data frames we created, the top one is for the retail records of the years 2009-2010, and the below one is for the years 2010-2011. We are provided with an Invoice and date for each order made, stock code, description of the product, quantity ordered price of the products, customer ID, and country. Moreover, we can notice that each order of the same invoice is presented separately so that we know the price, and quantity of every product purchased.

```

27 #Show the datasets
28 display(df1)
29 display(df2)

```

▶ (6) Spark Jobs

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
1	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	12/1/09 7:45	6.95	13085	United Kingdom
2	489434	79323P	PINK CHERRY LIGHTS	12	12/1/09 7:45	6.75	13085	United Kingdom
3	489434	79323W	WHITE CHERRY LIGHTS	12	12/1/09 7:45	6.75	13085	United Kingdom
4	489434	22041	"RECORD FRAME 7"" SINGLE SIZE "	48	12/1/09 7:45	2.1	13085	United Kingdom
5	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	12/1/09 7:45	1.25	13085	United Kingdom
6	489434	22064	PINK DOUGHNUT TRINKET POT	24	12/1/09 7:45	1.65	13085	United Kingdom

Truncated results, showing first 1000 rows.



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
1	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/10 8:26	2.55	17850	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	12/1/10 8:26	3.39	17850	United Kingdom
3	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/10 8:26	2.75	17850	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/10 8:26	3.39	17850	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/10 8:26	3.39	17850	United Kingdom
6	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/10 8:26	7.65	17850	United Kingdom

Truncated results, showing first 1000 rows.



Command took 11.39 seconds -- by nurmakhamadova.f@northeastern.edu at 6/24/2022, 10:53:09 PM on My Cluster

Then we convert the spark data frames into pandas' data frames using `toPandas()`, and combine two datasets together using `concat()` functions. This results in one data frame that contains all the retail records for years 2009-2011 and can be analyzed using python language. Since the

notebook is written in Python, the default cell type is Python. But we can use any other languages, such as SQL, Scala, R, or Markdown by %language syntax.

```
4 # convert Spark data frames to pandas dataframes
5 df1_pandas = df1.toPandas()
6 df2_pandas = df2.toPandas()
7
8 #Join dataset from 2009-2010 and 2010-2011
9 data =pd.concat([df1_pandas, df2_pandas])
```

## Data Cleaning

### Step 1: Check dataset info and unique values of each variable

Our dataset consists of 1,067,371 observations, and 8 features. Looking at the graphs below, we can notice that all the features have object data types, as well as there are some missing values in Description, and Customer ID columns. Moreover, with `nunique()` we can see how many unique values in each column, for example, we can see that there are 43 countries presented in the dataset. As you can notice, we can easily do analysis using the python commands.

```
1 #Get information about the dataset
2 data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1067371 entries, 0 to 541909
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Invoice          1067371 non-null object
1   StockCode       1067371 non-null object
2   Description      1062989 non-null object
3   Quantity        1067371 non-null object
4   InvoiceDate      1067371 non-null object
5   Price           1067371 non-null object
6   Customer ID     824364 non-null object
7   Country         1067371 non-null object
dtypes: object(8)
memory usage: 73.3+ MB
```

```
1 #Show unique values in each column
2 data.nunique()

Out[121]: Invoice          53628
StockCode          5305
Description         5698
Quantity           1057
InvoiceDate        47635
Price              2807
Customer ID        5942
Country            43
dtype: int64
```

## Step 2: Convert variable types to the right types

Thus, first we will start data cleaning with changing the data types. The Invoice Date column has date and time information about each order made. Thus, we converted it to datetime type using `pd.to_datetime()` function. This gives us the date in format YMDHMS (yyy-mm-dd hh:mm:ss). This information can further be used for timeseries analysis.

```
1 #Convert date from object to date
2 data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate']) #.dt.date
3 data.head()
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom
3	489434	22041	"RECORD FRAME 7"" SINGLE SIZE "	48	2009-12-01 07:45:00	2.1	13085	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom

Next, we converted Quantity column to integer type, and Price column to float type using `astype()` function. Now we have correct data types, 1 datetime, 1 float, 1 integer, and 5 objects. We kept Invoice, Stock Code, and Customer ID types as objects as we will not perform any mathematical operations on them.

```
1 #Convert Quantity from object into integer
2 data['Quantity'] = data['Quantity'].astype('int')
3
4 #Convert Price from object into float
5 data['Price'] = data['Price'].astype('float')
6
7 #Check the data types
8 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1067371 entries, 0 to 541909
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Invoice          1067371 non-null object
1   StockCode       1067371 non-null object
2   Description     1062989 non-null object
3   Quantity        1067371 non-null int64
4   InvoiceDate     1067371 non-null datetime64[ns]
5   Price           1067371 non-null float64
6   Customer ID    824364 non-null object
7   Country         1067371 non-null object
dtypes: datetime64[ns](1), float64(1), int64(1), object(5)
memory usage: 73.3+ MB
```

**Step 3: Calculate missing value percentages in each variable**

Further, we checked for missing values percentage using `isnull()` function the count of which is summed and multiplied by 100 to get a percentage. We can see that there are 0.4% of missing values in the Description column, and 22.7% in the Customer ID column.

```
1 # Check percentage of missing values
2 percent_missing = data.isnull().sum() * 100 / len(data)
3 percent_missing
```

```
Out[124]: Invoice      0.000000
StockCode      0.000000
Description     0.410541
Quantity       0.000000
InvoiceDate    0.000000
Price          0.000000
Customer ID    22.766873
Country        0.000000
dtype: float64
```

As there was no data dictionary, we decided to consider those customers without ID as unregistered customers. Thus, we replaced missing values in Customer ID with 'non-members' values using `fillna()` function. Now, we have 243,007 non-members who made an online purchase without registration or logging into the system.

**Step 4: Replace Customer ID missing values with non-member, drop other rows with missing values**

```

1 #Replace na in Customer ID with 'non-members'
2 data['Customer ID'].fillna('non-members',inplace=True)
3
4 #Recheck the column
5 data['Customer ID'].value_counts()

```

```

Out[125]: non-members      243007
17841          13097
14911          11613
12748           7307
14606           6709
...
14613             1
15357             1
14900             1
13841             1
12404             1
Name: Customer ID, Length: 5943, dtype: int64

```

After dealing with the Customer ID column, we dropped the missing values in the Description column using `dropna()` function. This was done because the percentage of missing values is less than 1% thus dropping will not affect the analysis. Moreover, we cannot replace them with other values as probably those orders have not proceeded as they miss the product in it. Thus, by checking `isna().any()` we assure there is no missing value left.

```

1 #Drop rows with missing values
2 data_new=data.dropna()
3
4 #Recheck missing values
5 data_new.isna().any()

```

```

Out[127]: Invoice      False
StockCode      False
Description      False
Quantity        False
InvoiceDate      False
Price           False
Customer ID      False
Country          False
dtype: bool

```

## Step 5: Drop meaningless values

Further, we checked the Price and Quantity columns for negative and 0 values. Since this analysis aims to understand the relationship between items and find the most frequent products purchased together, we should drop the values that refer to unfinished purchases. Thus, we created a subset of the dataset extracting data above 0 for the Price and Quantity and then checked the changes.

```

1 #Check if there are price 0 or below
2 data_new[data_new['Price'] <= 0]
3 #Drop prices equal to or below 0 if any
4 data_clean = data_new[data_new['Price'] > 0]
5
6 #Check if there are price 0 or below
7 data_clean[data_clean['Price'] <= 0]

```

Invoice	StockCode	Description	Quantity	InvoiceDate	Price
---------	-----------	-------------	----------	-------------	-------

```

1 #Check if there are quantities 0 or below
2 data_clean[data_clean['Quantity'] <= 0]
3 #Drop Quantity equal to or below 0 if any
4 data_clean = data_clean[data_clean['Quantity'] > 0]
5
6 #Check if there are Quantity 0 or below
7 data_clean[data_clean['Quantity'] <= 0]

```

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Custc
---------	-----------	-------------	----------	-------------	-------	-------

While checking the outliers in numeric columns, we found some doubtful values in Description column, such as Manual, Postage, DOTCOM POSTAGE, AMAZON FEE, and Adjust bad debt that we believe cannot be a product.



```

1 #Check the largest outlier values in Price
2 data_clean[data_clean['Price'] >= 8000]

```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
74356	496115	M	Manual	1	2010-01-29 11:04:00	8985.60	17949	United Kingdom
135013	502263	M	Manual	1	2010-03-23 15:22:00	10953.50	12918	United Kingdom
135015	502265	M	Manual	1	2010-03-23 15:28:00	10953.50	non-members	United Kingdom
241827	512771	M	Manual	1	2010-06-17 16:53:00	25111.09	non-members	United Kingdom
342147	522796	M	Manual	1	2010-09-16 15:12:00	10468.80	non-members	United Kingdom
358639	524159	M	Manual	1	2010-09-27 16:12:00	10468.80	14063	United Kingdom
372834	525399	M	Manual	1	2010-10-05 11:49:00	10468.80	non-members	United Kingdom
517955	537632	AMAZONFEE	AMAZON FEE	1	2010-12-07 15:08:00	13541.33	non-members	United Kingdom
15017	537632	AMAZONFEE	AMAZON FEE	1	2010-12-07 15:08:00	13541.33	non-members	United Kingdom
173382	551697	POST	POSTAGE	1	2011-05-03 13:46:00	8142.75	16029	United Kingdom
299982	A563185	B	Adjust bad debt	1	2011-08-12 14:50:00	11062.06	non-members	United Kingdom

Thus, we dropped them by indicating the dataset, column, and value to be deleted. Using `inplace = True` allows to make changes and update the same dataset.

```

1 #Drop Manual, Postage, DOTCOM POSTAGE, and AMAZON FEE products in Description column
2 data_clean.drop(data_clean[data_clean['Description'] == 'Manual'].index, inplace = True)
3 data_clean.drop(data_clean[data_clean['Description'] == 'POSTAGE'].index, inplace = True)
4 data_clean.drop(data_clean[data_clean['Description'] == 'DOTCOM POSTAGE'].index, inplace = True)
5 data_clean.drop(data_clean[data_clean['Description'] == 'AMAZON FEE'].index, inplace = True)
6 data_clean.drop(data_clean[data_clean['Description'] == 'Adjust bad debt'].index, inplace = True)
7
8 #Recheck again
9 data_clean[data_clean['Description'] == 'Manual']
10 data_clean[data_clean['Description'] == 'POSTAGE']
11 data_clean[data_clean['Description'] == 'DOTCOM POSTAGE']
12 data_clean[data_clean['Description'] == 'AMAZON FEE']
13 data_clean[data_clean['Description'] == 'Adjust bad debt']

```

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
---------	-----------	-------------	----------	-------------	-------	-------------	---------

So, now our dataset is clean, remaining with 1,033,437 observations, and 8 features with correct data types.

```

1 #Recheck cleaned dataset info
2 data_clean.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1033437 entries, 0 to 541908
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Invoice          1033437 non-null object
1   StockCode       1033437 non-null object
2   Description     1033437 non-null object
3   Quantity        1033437 non-null int64
4   InvoiceDate      1033437 non-null datetime64[ns]
5   Price           1033437 non-null float64
6   Customer ID     1033437 non-null object
7   Country         1033437 non-null object
dtypes: datetime64[ns](1), float64(1), int64(1), object(5)
memory usage: 71.0+ MB

```

## Data Visualization & Analysis

In this part, we firstly did some Exploratory data analysis and visualize the result which we think will provide us insights on the report topic, then we conducted Market basket analysis by using Apriori Algorithm.

### Exploratory Data Analysis

#### *1) Which time period generated most transactions?*


After cleaning the data, we can have a pandas dataframe. In databricks, we can use Python code, Spark code and SQL code, it is very powerful and convenient. So, in this step I will show how to use all three code languages to analyze.

```

1  # Enable Arrow-based columnar data transfers
2  spark.conf.set("spark.sql.execution.arrow.enabled", "true")
3
4
5  # Create a Spark DataFrame from a pandas DataFrame using Arrow
6  data_clean_spark = spark.createDataFrame(data_clean)
7
8  # Convert the Spark DataFrame back to a pandas DataFrame using Arrow
9  data_clean_pandas = data_clean_spark.select("*").toPandas()

```

▶ (1) Spark Jobs

▶  data\_clean\_spark: pyspark.sql.dataframe.DataFrame = [Invoice: string, StockCode: string ... 6 more fields]

We cleaned data in Python code, so if we want to use Spark to analyze, we need to have the same dataframe in Spark environment, so this code can convert pandas dataframe to Spark, and vice versa.

```

1  # Select InvoiceDate and to split this column:
2  df_time = data_clean_pandas['InvoiceDate']
3  df_time = pd.to_datetime(df_time)
4  df_time

```

Out[25]: 0                    2009-12-01 07:45:00

1	2009-12-01 07:45:00
2	2009-12-01 07:45:00
3	2009-12-01 07:45:00
4	2009-12-01 07:45:00
...	
1032637	2011-12-09 12:50:00
1032638	2011-12-09 12:50:00
1032639	2011-12-09 12:50:00
1032640	2011-12-09 12:50:00
1032641	2011-12-09 12:50:00

Name: InvoiceDate, Length: 1032642, dtype: datetime64[ns]

In this part, we need to discover the relationship between the total invoices number and date-time. In our cleaned dataframe, the Datetime is in one column, so now we need to split this column into

the Year column, Month column, Day column, Hour column, and Minute column. In this way, we can analyze the DateTime more easily. So, I got a new dataframe and have one column of DateTime, we can use this dataframe to split DateTime column. So, we define the function of splitting this column into several columns.

```
#def a fuction to splot one row of datetime to Y/M/D/H/M
def get_ymd(date):
    Y,M,D,H,m=[],[],[],[],[]
    for i in range(len(date)):
        oneday=date[i]
        year=oneday.year
        month=oneday.month
        day=oneday.day
        hour=oneday.hour
        minute=oneday.minute

        Y.append(year)
        M.append(month)
        D.append(day)
        H.append(hour)
        m.append(minute)
    date=pd.DataFrame()
    date['year']=Y
    date['month']=M
    date['day']=D
    date['hour']=H
    date['minute']=m
    return date
```

Then, we use this function to split the previous data frame, now we can see we successfully split it into 5 columns. We can find the new date frame has the same rows as before.

```

1 df_time = get_ymd(df_time)
2 df_time

Out[27]:

```

	year	month	day	hour	minute
0	2009	12	1	7	45
1	2009	12	1	7	45
2	2009	12	1	7	45
3	2009	12	1	7	45
4	2009	12	1	7	45
...	...	...	...	...	...
1032637	2011	12	9	12	50
1032638	2011	12	9	12	50
1032639	2011	12	9	12	50
1032640	2011	12	9	12	50
1032641	2011	12	9	12	50

1032642 rows x 5 columns

Then, I use “join” command to join this dataframe to the previously cleaned dataframe, after joining we can see now there are 1032642 rows and 13 columns. The ‘InvoiceDate’ is splitted into “year”, ‘month’, ‘day’, and ‘minute’.

```

1 Fuse join to join the new df_time with the origin data set
2 data_clean_pandas2=data_clean_pandas.join(df_time)
3 data_clean_pandas2

Out[29]:

```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	year	month	day	hour	minute
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	2009	12	1	7	45
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009	12	1	7	45
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009	12	1	7	45
3	489434	22041	"RECORD FRAME 7" SINGLE SIZE "	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	2009	12	1	7	45
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	2009	12	1	7	45
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1032637	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680	France	2011	12	9	12	50
1032638	581587	22699	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680	France	2011	12	9	12	50
1032639	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680	France	2011	12	9	12	50
1032640	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680	France	2011	12	9	12	50
1032641	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680	France	2011	12	9	12	50

1032642 rows x 13 columns

This data frame is a pandas dataframe. Similarly, if we want to analyze by SQL and Spark. We need to change this pandas dataframe to Spark dataframe and SQL table.

```
1 # change into spark
2 data_clean_pyspark2 = spark.createDataFrame(data_clean_pandas2)
```

So, by using the command of “createOrRepalceTempView()” we can change this spark dataframe to a SQL table and names “dfs”.

```
1 #use Spark dataframe to create one table in SQL called "dfs"
2 from pyspark.sql import SparkSession
3 from pyspark import SparkConf
4 spark = SparkSession.builder.config(conf=SparkConf()).getOrCreate()
5 data_clean_pyspark2.createOrReplaceTempView("dfs")
```

Since we create a table in SQL environment. So, we can both use spark.sql and SQL code to run our code, I first use a command of spark.sql() to see whether it works:

```

1 #Here I can use pyspark to write the sql code and get the result
2 spark.sql("select * from dfs").show()

```

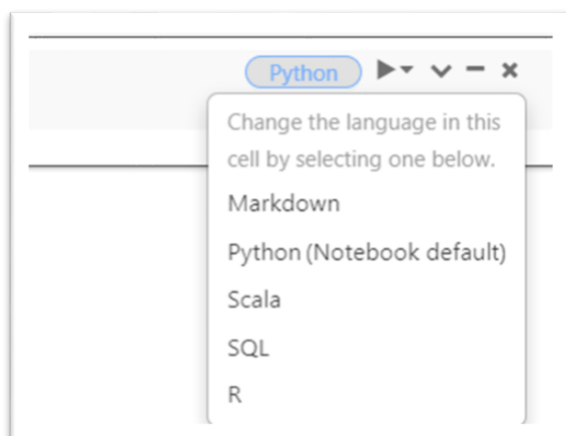
▶ (1) Spark Jobs

[Invoice]	[StockCode]	Description	[Quantity]	InvoiceDate	Price	Customer ID	Country	year	month	day	hour	minute
489434	85048	15CM CHRISTMAS GL...	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	2009	12	1	7	45
489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009	12	1	7	45
489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009	12	1	7	45
489434	22041	*RECORD FRAME 7**...	48	2009-12-01 07:45:00	2.1	13085	United Kingdom	2009	12	1	7	45
489434	21232	STRAWBERRY CERAMI...	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	2009	12	1	7	45
489434	22064	PINK DOUGHNUT TRI...	24	2009-12-01 07:45:00	1.65	13085	United Kingdom	2009	12	1	7	45
489434	21871	SAVE THE PLANET MUG	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	2009	12	1	7	45
489434	21523	FANCY FONT HOME S...	10	2009-12-01 07:45:00	5.95	13085	United Kingdom	2009	12	1	7	45
489435	22350	CAT BOWL	12	2009-12-01 07:46:00	2.55	13085	United Kingdom	2009	12	1	7	46
489435	22349	DOG BOWL , CHASIN...	12	2009-12-01 07:46:00	3.75	13085	United Kingdom	2009	12	1	7	46
489435	22195	HEART MEASURING S...	24	2009-12-01 07:46:00	1.65	13085	United Kingdom	2009	12	1	7	46
489435	22353	LUNCHBOX WITH CUT...	12	2009-12-01 07:46:00	2.55	13085	United Kingdom	2009	12	1	7	46
489436	48173C	DOOR MAT BLACK FL...	10	2009-12-01 09:06:00	5.95	13078	United Kingdom	2009	12	1	9	6
489436	21755	LOVE BUILDING BLO...	18	2009-12-01 09:06:00	5.45	13078	United Kingdom	2009	12	1	9	6
489436	21754	HOME BUILDING BLO...	3	2009-12-01 09:06:00	5.95	13078	United Kingdom	2009	12	1	9	6
489436	84879	ASSORTED COLOUR B...	16	2009-12-01 09:06:00	1.69	13078	United Kingdom	2009	12	1	9	6
489436	22119	PEACE WOODEN BLO...	3	2009-12-01 09:06:00	6.95	13078	United Kingdom	2009	12	1	9	6
489436	22142	CHRISTMAS CRAFT W...	12	2009-12-01 09:06:00	1.45	13078	United Kingdom	2009	12	1	9	6

We can find we write “Select \* from dfs” it shows the result same as we usually seen in SQL.

Then, we need to solve our question about:

So, we know since we want to use SQL to analyze, we need to first change the cell’s language to SQL, because our default language of the Notebook is Python, so we should change this cell to SQL first. Then at the start of this cell a code of %SQL will appear, meaning this cell’s language is SQL now. And then we need to consider writing SQL code of selecting the ‘hour’ and count each hour how many invoices of this online shop.







After writing the code, we get the result table:

```
1 %sql
2 --Similarly I can directly use SQL code to get the result
3 select count(Invoice) as total_invoices, hour from dfs Group by hour order by hour
```

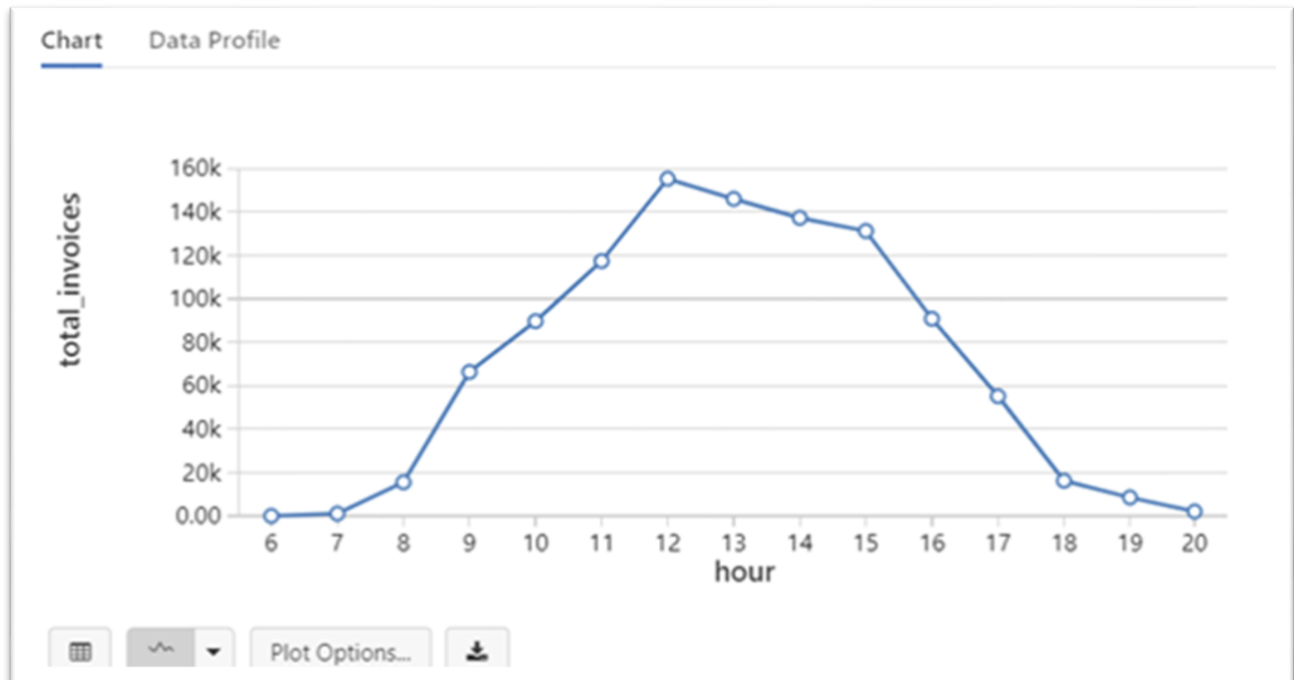
Table		Data Profile	
	total_invoices	hour	
1	1	6	
2	1050	7	
3	15507	8	
4	66329	9	
5	89710	10	
6	117409	11	
7	155309	12	
8	146054	13	
9	137303	14	

Showing all 15 rows.

We can find that when it is 6 a.m. The online shop system may not open, and people slowly start placing orders over time, in lower side of the result, we can find we can change this table to a chart in Databrick:





We can find during 12p.m., 13p.m., 14p.m. have the most invoices in this online shop. People are more likely to buy products at noon and afternoon. From this chart, we can also find this online shop system open only from 8 a.m. to 20 p.m. People are more willing to buy products in the afternoon.

### 1) In a week, when did customers often purchase online?

In the answer to the previous question, we use language based on SQL, so now we need to change back to Python. When we create new cells, we do not need to change the language because the default language of this notebook is Python.

Then, we want to expand from studying the patterns of the day to studying the patterns of the week. So, we need to have another column: “Weekday”, from “Monday” to “Sunday”

To get this column, we need to use the column “Invoice” because it is a datetime column and records detailed time in the dataframe. So, we use Python function to calculate the weekday in calendar. We use `dt.day_name()` function, which will automatically calculate in the calendar and return the day names of the Series or DatetimeIndex with specified locale.

```

1 #use the column of InvoiceDate to get the weekday
2 #from datetime package we can have many functions of dealing with the datetime such as weekday() and other
3 from datetime import datetime
4 data3=data_clean_pandas2
5 df_weekday = data_clean_pandas2['InvoiceDate']
6 df_weekday = pd.to_datetime(df_weekday)
7 data3['weekday']=df_weekday.dt.day_name()
8 data3

```

Now we can get the result, we can find in this new dataframe, there is a new column of weekday:

er ID	Country	year	month	day	hour	minute	weekday
3085	United Kingdom	2009	12	1	7	45	Tuesday
3085	United Kingdom	2009	12	1	7	45	Tuesday
3085	United Kingdom	2009	12	1	7	45	Tuesday
3085	United Kingdom	2009	12	1	7	45	Tuesday
3085	United Kingdom	2009	12	1	7	45	Tuesday
...	...	...	...	...	...	...	...
2680	France	2011	12	9	12	50	Friday
2680	France	2011	12	9	12	50	Friday
2680	France	2011	12	9	12	50	Friday
2680	France	2011	12	9	12	50	Friday
2680	France	2011	12	9	12	50	Friday

Then, we can use 'Weekday' and 'hour' to make a heatmap to find which time of day of the week has more invoices. In this new dataframe called "heatmap".

We need to have a new restructured dataframe, we want to have the y-axis as Weekday, X-axis as each hour in one day. So, we make a new pivot table, and the index is 'weekday', the column is 'hour', and the value is to count total invoice numbers to satisfy the requirement.

```

1 #use pivot_table to get a new dataframe which columns is hour, index is weekday
2 heatmap=pd.pivot_table(data3,index='weekday',columns='hour',values='Invoice',aggfunc='count')
3 heatmap

Out[28]:

```

	hour	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
weekday																
Friday	NaN	178.0	3220.0	12767.0	16355.0	19495.0	20626.0	18474.0	21069.0	15844.0	10626.0	8614.0	626.0	88.0	8.0	
Monday	NaN	154.0	2943.0	15265.0	17297.0	18902.0	25377.0	26288.0	22352.0	22621.0	16839.0	13239.0	1914.0	NaN	NaN	
Saturday	NaN	NaN	NaN	NaN	30.0	4.0	102.0	101.0	56.0	69.0	37.0	NaN	NaN	NaN	NaN	
Sunday	NaN	NaN	NaN	79.0	8889.0	22912.0	28594.0	25683.0	20490.0	21300.0	9248.0	69.0	NaN	NaN	NaN	
Thursday	1.0	287.0	3040.0	13441.0	16936.0	18037.0	24990.0	24951.0	25282.0	20379.0	18582.0	10535.0	9019.0	8114.0	2007.0	
Tuesday	NaN	251.0	2877.0	13178.0	15355.0	18547.0	27672.0	26456.0	23561.0	26500.0	19110.0	10951.0	4487.0	281.0	9.0	
Wednesday	NaN	180.0	3427.0	11599.0	14848.0	19512.0	27948.0	24101.0	24493.0	24533.0	16386.0	11744.0	158.0	15.0	18.0	

Command took 0.51 seconds -- by zcj1138457943@gmail.com at 2022/6/25上午3:46:04 on Quickstart

Since, we have made new pivot table, we can find there is a lot of NaNs in this dataframe, so we need to clean this data, to fill the NaNs as 0 so that we can make a heatmap. And the index sequence doesn't make sense, so we also need to reindex the index column.

```

1 #fillna as 0 and reindex the index to regular consequence
2 heatmap=heatmap.fillna(0)
3 heatmap=heatmap.reindex(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'])
4 heatmap2=heatmap.iloc[:,]
5 heatmap2

Out[29]:

```

	hour	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
weekday																
Monday	0.0	154.0	2943.0	15265.0	17297.0	18902.0	25377.0	26288.0	22352.0	22621.0	16839.0	13239.0	1914.0	0.0	0.0	
Tuesday	0.0	251.0	2877.0	13178.0	15355.0	18547.0	27672.0	26456.0	23561.0	26500.0	19110.0	10951.0	4487.0	281.0	9.0	
Wednesday	0.0	180.0	3427.0	11599.0	14848.0	19512.0	27948.0	24101.0	24493.0	24533.0	16386.0	11744.0	158.0	15.0	18.0	
Thursday	1.0	287.0	3040.0	13441.0	16936.0	18037.0	24990.0	24951.0	25282.0	20379.0	18582.0	10535.0	9019.0	8114.0	2007.0	
Friday	0.0	178.0	3220.0	12767.0	16355.0	19495.0	20626.0	18474.0	21069.0	15844.0	10626.0	8614.0	626.0	88.0	8.0	
Saturday	0.0	0.0	0.0	0.0	30.0	4.0	102.0	101.0	56.0	69.0	37.0	0.0	0.0	0.0	0.0	
Sunday	0.0	0.0	0.0	79.0	8889.0	22912.0	28594.0	25683.0	20490.0	21300.0	9248.0	69.0	0.0	0.0	0.0	

Command took 0.45 seconds -- by zcj1138457943@gmail.com at 2022/6/25上午3:46:04 on Quickstart

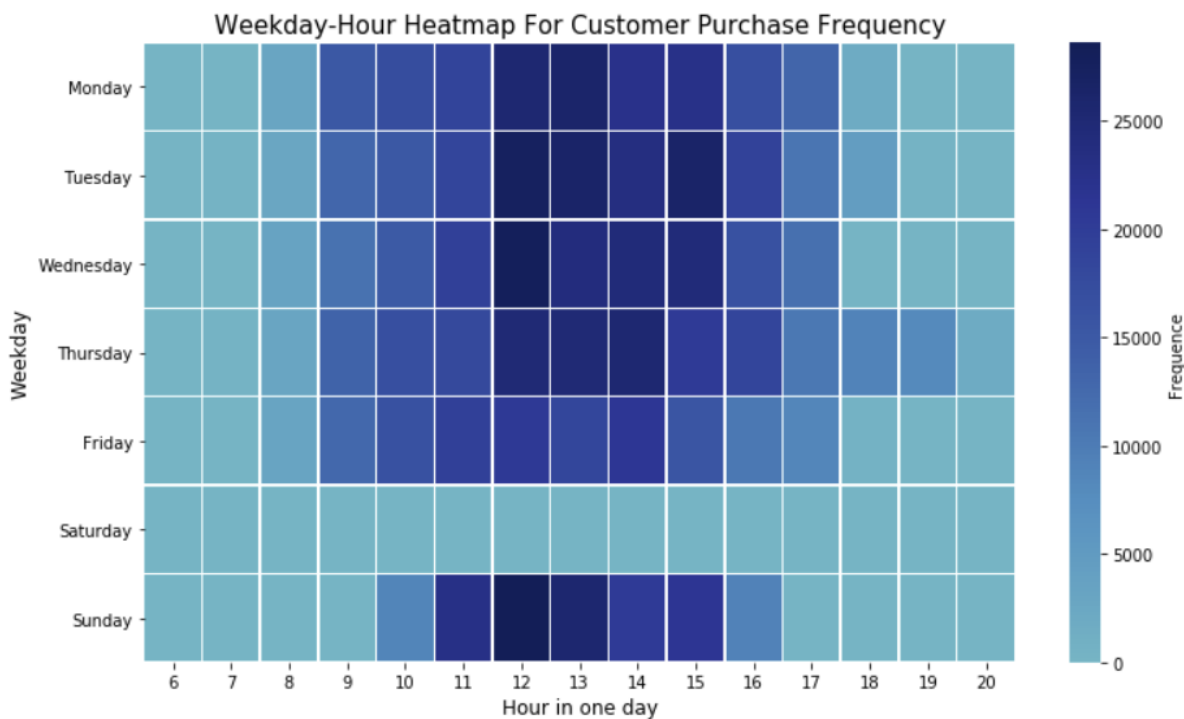
Now, we can use this data set to make a heat map:

```

1 fig = plt.figure(figsize = (12, 7))
2 #use seaborn to draw heatmap
3 import seaborn as sns
4 ax = sns.heatmap(heatmap2,linewidths=.5,center = 0,cmap="YlGnBu",cbar_kws={'label': 'Frequency'})
5 ax.set_xlabel('Hour in one day',fontsize = 12)
6 ax.set_ylabel('Weekday',fontsize = 12)
7 ax.set_title('Weekday-Hour Heatmap For Customer Purchase Frequency',fontsize = 15)
8 ax._legend.set_title('New title')

```

And then get the heatmap:



In this heatmap, we can see the colors are much darker, which means there are more invoices at that time. And this online shop system doesn't work on Saturday and the system only opens from 10 a.m. to 14 p.m. on Sunday. People are more willing to buy products during the noon time. Thursday is the most popular day of the week; we can see the color is much darker than else.

## 2) What are the top 10 products selling the most?

To find products that were sold most, we first grouped products, aka Description, summing up the quantity, and averaging the price. The result was sorted by the quantity of the product in descending order. For better perception, we have limited the product amount to top 10.

```
1 #Calculate the total quantity of products sold and their average prices
2 products_by_quantity = (data_clean.groupby('Description', as_index=False)
3     .agg({'Quantity':'sum', 'Price':'mean'}).sort_values('Quantity', ascending = False)
4     .rename(columns={'Quantity':'Total Quantity', 'Price':'Average Price'}))
5
6 #Extract only Top 10 Product sold the most
7 top_10_sales = products_by_quantity.head(10)
```

Grouping by Description made the column an index, thus we changed it back to a column with the reset\_index() function. Below is the list of the top 10 products that were sold the most.

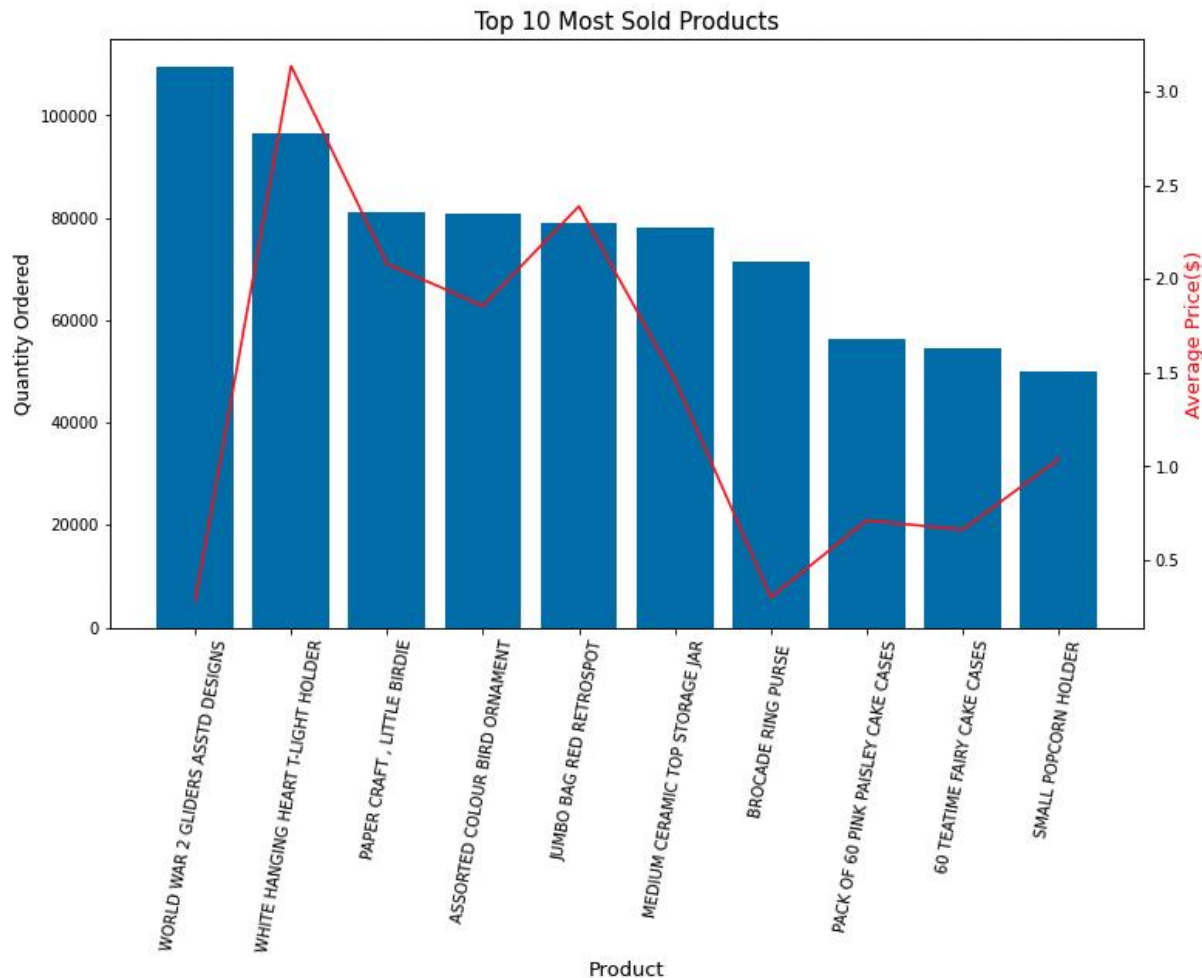
```
6 #Extract only Top 10 Product sold the most
7 top_10_sales = products_by_quantity.head(10)
8 #Convert Index Description to Column
9 top_10_sales.reset_index(inplace=True)
10 #Show the result
11 top_10_sales
```

	index	Description	Total Quantity	Average Price
0	5278	WORLD WAR 2 GLIDERS ASSTD DESIGNS	109418	0.283667
1	5157	WHITE HANGING HEART T-LIGHT HOLDER	96496	3.135235
2	3190	PAPER CRAFT , LITTLE BIRDIE	80995	2.080000
3	342	ASSORTED COLOUR BIRD ORNAMENT	80888	1.857053
4	2469	JUMBO BAG RED RETROSPOT	79047	2.387057
5	2753	MEDIUM CERAMIC TOP STORAGE JAR	78032	1.464498
6	805	BROCADE RING PURSE	71330	0.303032
7	3126	PACK OF 60 PINK PAISLEY CAKE CASES	56489	0.712824
8	200	60 TEATIME FAIRY CAKE CASES	54656	0.663410
9	4610	SMALL POPCORN HOLDER	49909	1.039266

Then we generated a bar plot to visualize the findings. The graph below presents code lines with explanations of how we generated the plot and parameters set used.

```
4 #Plot the top 10 Product sold most with Price
5 fig, ax1 = plt.subplots(figsize = (12, 7))
6
7 ax2 = ax1.twinx()
8 #Bar plot for the Quantity of Products
9 ax1.bar(top_10_sales['Description'], top_10_sales['Total Quantity'])
10 #Line plot for the Price of Products
11 ax2.plot(top_10_sales['Description'], top_10_sales['Average Price'], color = 'r')
12
13 ax1.set_ylabel('Quantity Ordered', fontsize = 12)
14 ax1.set_xlabel('Product', fontsize = 13)
15 plt.grid(False)
16 #Add y label for Line plot of the Price column
17 ax2.set_ylabel('Average Price($)', color = 'r', fontsize = 13)
18 #Add graph title
19 ax1.set_title('Top 10 Most Sold Products', fontsize = 15)
20 #Set x label ticks to rotate for 80 degrees
21 ax1.set_xticklabels(top_10_sales['Description'], rotation = 80, fontsize = 10)
22 plt.show()
```

The graph below demonstrates the top 10 products that were sold the most. The main bar plot and ticks on the left y-axis show product quantity, while the red line of a subplot and ticks on the right y-axis refers to the product price. The top 1st product WORLD WAR 2 GLIDERS ASSTD DESIGNS is the most popular among the customers with the sold quantity above 100,000 items. The rest of the 9 products' quantity is below 100,000 ranging between 90,000 to 50,000 items. Most of these products are small, and not for consuming but decorating purposes. We can notice that the average price for these products ranges from \$0 to \$3 meaning that the products are cheap. The 1st most ordered product has the lowest total price of \$0.28 and high quantity ordered, while the 2nd top product WHITE HANGING HEART T-LIGHT HOLDER has the highest average price of \$3.1. As for the rest 8 most sold products, ASSORTED COLOUR BIRD ORNAMENT, PAPER CRAFT , LITTLE BIRDIE, and JUMBO BAG RED RETROSPOT have an average price above \$1.5 while the rest have below it.



### 3) *What are the top 10 Products Most Often Sold Together?*

Next, we want to find out the top 10 products that most often are sold together. As we remember, each product in the same order is presented in a separate row with the same invoice number. Thus, we first copied the clean original dataset to create a new column that will contain all products of the same order in one row, separating them with coma. For this, we first grouped the data by Invoice and Description and added combined products into the new column Product Bundle. Then, we dropped duplicated data of the same invoices.



```

1 #Joining products with the same Invoice number into the same line
2 #Copy the clean data to add new column
3 data_products = data_clean
4
5 #Group by Invoice and Description assigning all products with same invoice to a new column
6 data_products['Product Bundle'] = data_products.groupby('Invoice')['Description'].transform(lambda x: ','.join(x))
7
8 #Dropping the duplicate values of same invoices
9 data_products = data_products[['Invoice', 'Product Bundle']].drop_duplicates()
10 data_products.head()

```

	Invoice	Product Bundle
0	489434	15CM CHRISTMAS GLASS BALL 20 LIGHTS,PINK CHERR...
8	489435	CAT BOWL ,DOG BOWL , CHASING BALL DESIGN,HEART...
12	489436	DOOR MAT BLACK FLOCK ,LOVE BUILDING BLOCK WORD...
31	489437	CHRISTMAS CRAFT HEART DECORATIONS,CHRISTMAS CR...
54	489438	DINOSAURS WRITING SET ,SET OF MEADOW FLOWER ...

Then we created a for loop to count most frequent product combinations in Product bundles. The result below demonstrates top 10 most common combinations with the 2 product bundles. We can see that the most often products sold together are key fob and key fob with 2,591 transactions. We can notice that key fob product is found in most combinations. Moreover, the WHITE HANGING HEART T-LIGHT HOLDER which was among most sold products is in a product pair with RED HANGING HEART T-LIGHT HOLDER with 830 transactions.

```

1 #Set the counter
2 count = Counter()
3
4 #Create a for loop to count most frequent product combinations in Product bundles
5 for row in data_products['Product Bundle']:
6     row_list = row.split(',') #split each product combination with comma
7     count.update(Counter(combinations(row_list, 2))) #Counting all the 2 product bundles
8
9 #Show top 10 most common combinations
10 for value, count in count.most_common(10):
11     print(count,':', value)

```

```

2591 : ('KEY FOB ', 'KEY FOB ')
2527 : ('FRENCH BLUE METAL DOOR SIGN', 'FRENCH BLUE METAL DOOR SIGN')
1529 : ('KEY FOB ', ' SHED')
1513 : ('KEY FOB ', ' BACK DOOR ')
1135 : ('KEY FOB ', ' FRONT DOOR ')
1025 : ('KEY FOB ', ' GARAGE DESIGN')
980 : ('HOOK', 'MAGIC GARDEN')
927 : ('METAL SIGN', 'CUPCAKE SINGLE HOOK')
880 : ('COFFEE', 'SUGAR')
830 : ('RED HANGING HEART T-LIGHT HOLDER', 'WHITE HANGING HEART T-LIGHT HOLDER')

```



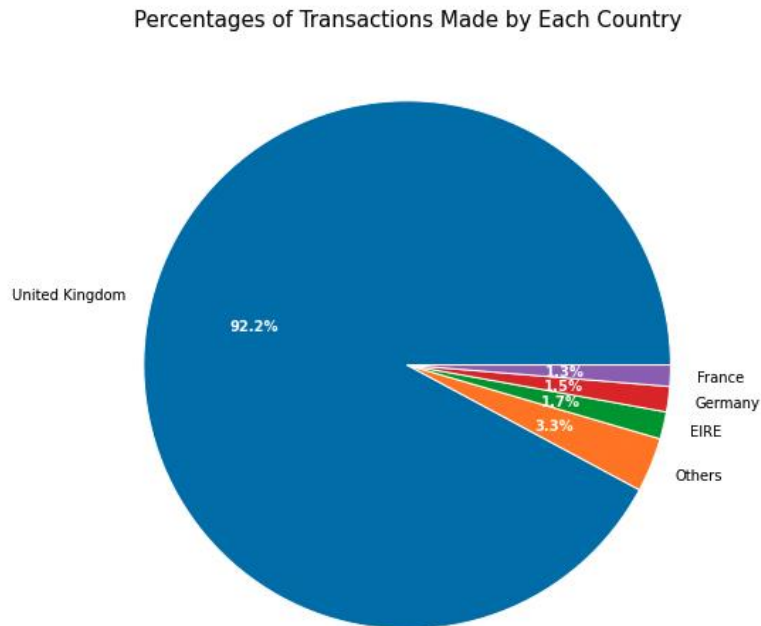
**4) Which country of customers placed most orders from this store?**

**Step 1: Write below the codes in the cell and run**

Each line of codes have explained in below the picture.

```
1 #Percentages of Transactions Made by Each Country
2 country = data_clean.groupby('Country')['Invoice'].count().sort_values(ascending = False)/len(data_clean['Country'])
3
4 #Replace country names which have frequency proportion below 0.005 with "Others"
5 data_clean.loc[data_clean.groupby('Country')['Invoice']
6                 .transform('size')
7                 .div(len(data_clean['Country']))
8                 .lt(0.005),
9                 'Country'] = 'Others'
10
11 #Percentages of Transactions Made by Each Country
12 country = data_clean.groupby('Country')['Invoice'].count().sort_values(ascending = False)/len(data_clean['Country'])
13
14 #define values and labels
15 value = country.values
16 labels = country.index
17
18 #set figure size
19 fig1, ay = plt.subplots(figsize=(12, 7))
20
21 # Capture each of the return elements: add white labels, add white lines with 1.0 width, make text medium size
22 patches, texts, pcts = ay.pie(
23     value, labels=labels, autopct='%.1f%%',
24     wedgeprops={'linewidth': 1.0, 'edgecolor': 'white'},
25     textprops={'size': 'medium'})
26
27 # Style just the percent values
28 plt.setp(pcts, color='white', fontweight='bold')
29 # Set a title for the graph
30 plt.title('Percentages of Transactions Made by Each Country', fontsize = 15)
31 # Adjust the padding between and around subplots
32 plt.tight_layout()
33 #show graph
34 plt.show()
```

**Step 2: Click the triangle beside “Python” and click “Run Cell”.**



### Observations:

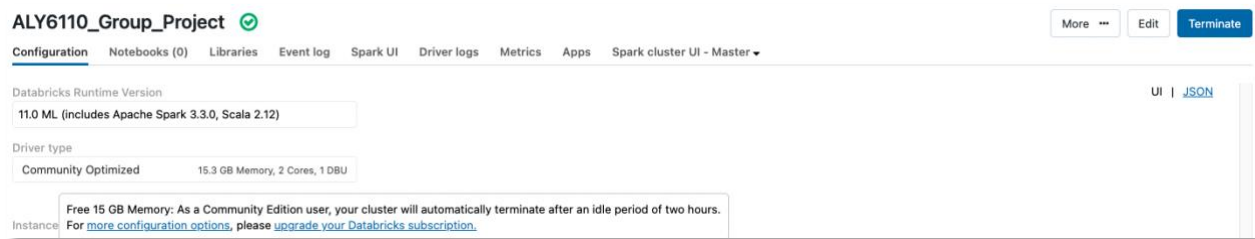
The top 4 countries with most transactions are United Kingdom (UK), EIRE, Germany and France. Transactions from UK occupies the most part of all the transactions which is over 92%. UK based so attract more UK customers

### Market Basket Analysis

In this part, we started to do the Market Basket Analysis using Apriori Algorithm.

#### Step 1: Change the Cluster Runtime Version which contains the ML algorithm needed

Change runtime version of Cluster before installing the library needed and package needed to import in Python.



After changing the version, ensure the notebook is run on the correct cluster.

## Step 2: Install the package 'mlxtend' and import the library

```
1 #Install mlxtend package
2 %pip install mlxtend
```

Python interpreter will be restarted.

Requirement already satisfied: mlxtend in /local\_disk0/.ephemeral\_nfs/envs/pythonEnv-6c710d9f-3d84-49fa-a1b6-dc131c304c30/lib/python3.9/site-packages (0.20.0)

Requirement already satisfied: joblib>=0.13.2 in /databricks/python3/lib/python3.9/site-packages (from mlxtend) (1.0.1)

Requirement already satisfied: numpy>=1.16.2 in /databricks/python3/lib/python3.9/site-packages (from mlxtend) (1.20.3)

Requirement already satisfied: matplotlib>=3.0.0 in /databricks/python3/lib/python3.9/site-packages (from mlxtend) (3.4.3)

Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from mlxtend) (58.0.4)

Requirement already satisfied: scikit-learn>=1.0.2 in /local\_disk0/.ephemeral\_nfs/envs/pythonEnv-6c710d9f-3d84-49fa-a1b6-dc131c304c30/lib/python3.9/site-packages (from mlxtend) (1.1.1)

Requirement already satisfied: pandas>=0.24.2 in /databricks/python3/lib/python3.9/site-packages (from mlxtend) (1.3.4)

Requirement already satisfied: scipy>=1.2.1 in /databricks/python3/lib/python3.9/site-packages (from mlxtend) (1.7.1)

Requirement already satisfied: python-dateutil>=2.7 in /databricks/python3/lib/python3.9/site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)

Requirement already satisfied: kiwisolver>=1.0.1 in /databricks/python3/lib/python3.9/site-packages (from matplotlib>=3.0.0->mlxtend) (1.3.1)

Requirement already satisfied: pyparsing>=2.2.1 in /databricks/python3/lib/python3.9/site-packages (from matplotlib>=3.0.0->mlxtend) (3.0.4)

Requirement already satisfied: cycler>=0.10 in /databricks/python3/lib/python3.9/site-packages (from matplotlib>=3.0.0->mlxtend) (0.10.0)

Requirement already satisfied: pillow>=6.2.0 in /databricks/python3/lib/python3.9/site-packages (from matplotlib>=3.0.0->mlxtend) (8.4.0)

Requirement already satisfied: six in /databricks/python3/lib/python3.9/site-packages (from cycler>=0.10->matplotlib>=3.0.0->mlxtend) (1.16.0)

Requirement already satisfied: pytz>=2017.3 in /databricks/python3/lib/python3.9/site-packages (from pandas>=0.24.2->mlxtend) (2021.3)

Requirement already satisfied: threadpoolctl>=2.0.0 in /databricks/python3/lib/python3.9/site-packages (from scikit-learn>=1.0.2->mlxtend) (2.2.0)

Python interpreter will be restarted.

Command took 0.16 seconds -- by qiu.yul@northeastern.edu at 6/24/2022, 10:03:51 PM on ALY6110\_Group\_Project

Use Python syntax to install the package, we could use pip install command. If it does not work, we could add a % sign before pip. We put this step at the very first of the Notebook, as everytime after installing the package, the notebook will be restarted

```
1 #Import needed libraries
2 from pyspark.sql import SparkSession
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 from mlxtend.frequent_patterns import apriori
8 from mlxtend.frequent_patterns import association_rules
```

Command took 0.90 seconds -- by qiu.yul@northeastern.edu at 6/24/2022, 10:03:51 PM on ALY6110\_Group\_Project

Then import the library needed. For Market Basket Analysis, we choose to use Apriori Algorithm and we also need the association\_rules.

## Step 3: Creating the target subset and restructure the dataset

```
1 #Create a subset basket which only contains UK transactions
2 basketuk = data_clean[data_clean['Country'] == 'United Kingdom']
3 #Restructure the data with Invoice as the Index and fill nan with 0
4 basket = basketuk.groupby(['Invoice', 'Description'])['Quantity'].sum().unstack().fillna(0).applymap(lambda x: 1 if x > 0 else 0)
```

Command took 1.50 minutes -- by qiu.yul@northeastern.edu at 6/26/2022, 4:45:15 PM on ALY6110\_Group\_Project

As customers from different countries might have different purchase behaviors, therefore in this analysis, we mainly focused on the customer from UK.

So we created a subset of UK and sum up the product quantity by invoice and product name (description), display it with descending order.

Next, we restructured the data by using invoice number as the index and put product names as the columns and fill the nan values with 0.

Since the number of each product bought in each order will not influence the MBA analysis, we just consistently replace the values greater than 0 as 1 and others stay as 0.

#### Step 4: Calculate the Support, Confidence and Lift

```
1 #Use Apriori algorithm to calculate the support with the threshold of minimum support 0.03
2 frequent_itemsets = apriori(basket, min_support= 0.03, use_colnames=True)
3
4 #set the rule metric as lift with minimum value of 1
5 rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
```

There are 3 metrics in the Apriori Algorithm: Support, Confidence and Lift. Support means the frequency of the pair of 2 products appear in the same order, confidence which belong to the association rules indicates the percentage of when the rule is true (for example the rule could be that customers buying product A will also likely to buy product B), lift of the rule indicates how strong of the association between 2 products in this rule (). Usually, if lift is higher than 1, the association of products in this rule is strong.

The support in apriori algorithm is a percentage which is the percentage of the 2 products appear in one order against the total number of transactions.

We set the minimum support as 0.03, so the support score which below 0.03 will be filtered out. The reason why we choose 0.03 is because when we choose other values, the command will show error in result.

After filtering out the product pair with low support, the next step to check the association rules. We used the `association_rules` function and set the metric as lift and filtered out those with lift value lower than 1.

Below is the result of the above 2 steps.

```

1 #show the rules result
2 rules

```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(WHITE HANGING HEART T-LIGHT HOLDER)	(RED HANGING HEART T-LIGHT HOLDER)	0.14428	0.04595	0.032234	0.223412	4.862071	0.025604	1.228516
1	(RED HANGING HEART T-LIGHT HOLDER)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.04595	0.14428	0.032234	0.701502	4.862071	0.025604	2.866747

### Observations:

There are only 2 pairs of products which match the criteria we set. The 2 pairs are with 2 same products, while the sequence is different. Comparing antecedent support value and consequent support value, we could see that the antecedent support of buying White Holder first then Red Holder is higher than its consequent support, which means people usually buy White Holder first than Red Holder. While the opposite sequence does not usually happen. The lift value is high which means the 2 products have a strong association.

The 2 confidence values tell us that people buy Red Holder with White Holder together more often than buy Red Holder without White Holder. While people buy white holder without Red Holder more often than buy these 2 together. We will provide recommendations based on these findings.

### Insights & Recommendations

1. Most of the top - most sold items represent small items with short usage time and thus are purchased in large quantities. Thus, we would recommend increasing the profit margins by

selling low-price items and finding the best combinations of low-price items that can be sold together.

2. The top common product combination is the key fob and key fob, with the highest transactions. This product also appears in multiple other combinations with shed, doors, and other items related to doors meaning that most customers purchase these products together. WHITE HANGING HEART T-LIGHT HOLDER which was among the most sold products is in a product pair with RED HANGING HEART T-LIGHT HOLDER. Thus, we can notice that the products that are mostly sold together are those complementing each other. Thus, we would recommend increasing sales by combining these products together so that may be not popular items will get popular for the perfect combination and profitable cost.
3. From this dataset, we found above 92% of the transactions are from UK customers, therefore, to increase the revenue, the store could try the product bundle promotions to non-UK customers and check whether this will help increase the total revenue.
4. Time pattern: provide promotions during non-peak time to release the website pressure and when there is a peak time, we can recommend more relevant products by analyzing the MBA to customers for adding in their shopping carts and expand customers' shopping period.
5. The online UK-store could provide product recommendations when customers are visiting specific product based on the findings of Products Most Often Sold Together.
6. Red Hanging Heart T-Light Holder is usually bought together with White Hanging Heart T-Light Holder, especially when people is going to buy the Red one. Also, we found that White Hanging Heart T-Light Holder were one of the most popular products sold with top number of transactions, therefore we suggest providing a discount on the Red Holders when people put White Hanging Heart T-Light Holder in their carts. This way we could help increase the sales of Red Hanging Heart T-Light Holder and increase the total revenue. This promotion should be mainly provided to UK customers.

## Comments

In this part, we mainly analyzed the pros and cons the Databricks based on our project experience and mainly on the Community edition. Also, we concluded the difficulties and challenges we met during the process.

## Pros

Databricks have APIs to multiple programming languages: R, Scala, Python, SQL. This provide a great flexibility to choose the language we are familiar with. Also, multiply programming languages could be used in the same notebook which provides us the flexibility to choose the syntax based on our familiarity and needs.

Databricks can be connected to the databases we often use in our daily work and study, such as AWS, AZURE, etc. These data usually store a lot of data, resulting in a huge file size. If we don't store these data in the cloud, it will cause inconvenience to our use of the database. Databricks has a good connection method to connect to these storage clouds.

When we select a table from SQL environment, it can be easily transformed into a graph, since there is a function with multiple types of graphs, and we could directly choose the graph type and the result will be shown in chart.

There are many useful packages are available in the environment, so we don't need to install many packages from pip install command

The speed to run the commands is much quicker than Jupiter Notebooks.

## Cons

There is a limited number of clusters could be created in Databricks community edition. The terminated cluster could not be restarted, so every time to reopen the notebook, we need to create a new cluster for the notebook.

When we need to use different ways to analyze one dataframe, we need to establish it as a Spark dataframe, Python dataframe and a SQL table, though they are all the same.

It needs a relatively long time to have the cluster fully started, usually 3- 5 minutes.

## Challenges during the process

Directly upload the original Excel file which contains 2 sheets in it and use Databricks to read and merge the 2 sheets.

When we need to use Spark, Python, and SQL code, we think we are much more familiar with Python, and spend much time on finding how to write code in Spark.

The option inferSchema in Spark read function was able to correctly define most of the columns' data types, except the DateTime type. Moreover, the null values in the Customer ID column were not defined with isnull() function as well as we could not replace these null values with 'non-members' values. While, when we set the inferSchema option as false and changed the data types manually, we were able to identify all null values as well as replace them with the desired values.



```
1 #Use Apriori algorithm to calculate the support with the threshold of minimum support 0.03
2 frequent_itemsets = apriori(basket, min_support= 0.003, use_colnames=True)
3
4 #set the rule metric as lift with minimum value of 1
5 rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
6
7 #show the rules result
8 rules
```

/databricks/python/lib/python3.8/site-packages/ipykernel/ipkernel.py:287: DeprecationWarning: 'should\_run\_async' will not call 'transform\_cell' automatically in the future. Please pass the result to 'transformed\_cell' argument and any exception that happen during the transform in 'preprocessing\_exc\_tuple' in IPython 7.17 and above.

/local\_disk0/.ephemeral\_nfs/envs/pythonEnv-d817e589-012e-468c-af4f-ac102aa0fbb7/lib/python3.8/site-packages/mlxtend/frequent\_patterns/fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types result in worse computational performance and their support might be discontinued in the future. Please use a DataFrame with bool type

MemoryError: Unable to allocate 1.22 TiB for an array with shape (2316628, 2, 36235) and data type int64

Command took 2.46 seconds -- by qiu.yu1@northeastern.edu at 6/25/2022, 11:47:14 PM on ALY6110\_Group\_Project

MBA min support value could only be 0.03. When we set a smaller value for example 0.003, the result shows error “Unable to allocate 1.22 TiB for an array with shape (2316628, 2, 36235) and data type int64”. This made our MBA analysis only show 2 pairs of products.

## Conclusion

In this project, we have illustrated how to use Databricks to tackle big data. We used an online sales store dataset which contains more than 1 billion rows as an example and conducted data mining, specifically association rules with Apriori algorithm and provides recommendations based on our analysis results to the company.



The analysis of the huge dataset using big data tools is indeed quick, and effective. The Databricks platform allows data processing and manipulation in an easy-to-understand environment with familiar tools. Moreover, the multilanguage coding option expands restrictions that we have while analyzing using one tool. We were able to upload and store two files, then combine them into one data frame. Another important fact is that Databricks has a web-based interface thus, there is no need to launch a virtual machine every time being under time pressure.

While the version of Databricks we used is Community Edition, therefore there are several limitations on the features. For companies which have enough budget, we strongly recommend using AWS cloud, and use the full platform.

Future analysis could conduct Market Basket Analysis focusing on other countries customers to see if there is any difference. Also, to find the optimal minimum support for the Market Basket Analysis is important, while we did not find the approach in Python. There are some research which have find out the formular to calculate the best minimum support, and some industries use domain knowledge to choose the threshold, future analysis could try different minimum support value to see which value is the best one for retail industry.

## References

- Cagirici, O. *Online Retail Dataset*. Kaggle.com. Retrieved 17 June 2022, from <https://www.kaggle.com/datasets/ozlemilgun/online-retail-dataset?resource=download>.
- Clusters - Azure Databricks*. Docs.microsoft.com. (2022). Retrieved 18 June 2022, from <https://docs.microsoft.com/en-us/azure/databricks/clusters/>.
- Garg, A. (2022). *Apache Spark Tutorial - Learn Spark & Scala with Hadoop - Intellipaat*. Intellipaat Blog. Retrieved 10 June 2022, from <https://intellipaat.com/blog/tutorial/spark-tutorial/>.
- Kadlaskar, A. (2021). *Market basket Analysis / Guide on Market Basket Analysis*. Analytics Vidhya. Retrieved 11 June 2022, from <https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-market-basket-analysis/>.
- What is Databricks Runtime? - Databricks*. Databricks. Retrieved 18 June 2022, from <https://databricks.com/glossary/what-is-databricks-runtime>.