Dissecting 2023 Amazon Shopping Landscape: A Deep Dive into Consumer Behavior and Market Trends

Introduction

In this project, I analyzed extensive Amazon data to uncover valuable insights into consumer behavior and market trends. Using SQL, Python, and platforms like Databricks, I focused on exploring market dynamics, pricing fluctuations, and customer behavior across various product categories. Handling this massive dataset presented challenges, particularly in integrating multiple data sources and cleaning raw data to ensure usability.

My main goal was to build an efficient ETL (Extract, Transform, Load) process using SQL and Python, enabling me to effectively process and analyze the data. Through this analysis, I identified key patterns, such as seasonal pricing trends and bestselling products in specific categories. These insights are critical for businesses looking to make data-driven decisions, allowing them to adjust strategies, meet customer demands, and maintain a competitive edge. Ultimately, I transformed raw Amazon data into actionable insights that drive business success.

About the dataset:

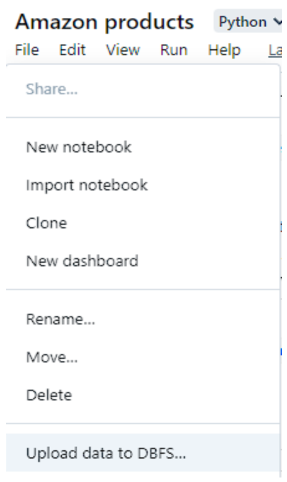
Amazon, a major player in the U.S. online retail market, offers a rich dataset containing over 12 million products, providing a valuable opportunity to extract meaningful insights. This analysis seeks to uncover top-selling products, assess the effectiveness of SEO titles in boosting sales, determine ideal price ranges within specific categories, and reveal other important trends. The focus is on identifying popular product categories, analyzing their sales performance, and highlighting highly-rated products based on customer reviews. Additionally, the project aims to create a product title generator by analyzing patterns in successful sales titles. This dataset is key to discovering profitable niches, understanding online shoppers' spending habits, and refining database management and optimization skills.

Columns in the data source and their data types:

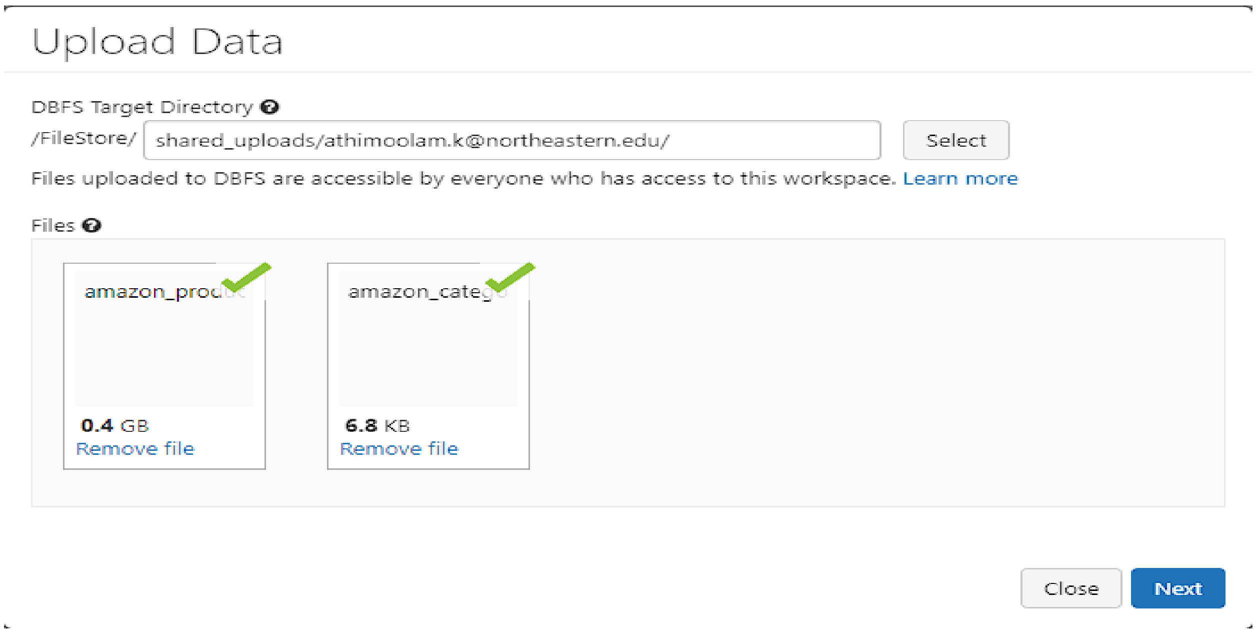
- asin: Product ID from Amazon. (type:str)
- title: Title of the product. (type:str)
- imgUrl: Url of the product image. (type:str)
- productURL: Url of the product. (type:str)
- stars: Product rating. If 0, no ratings were found. (type:float)
- reviews: Number of reviews. If 0, no reviews were found. (type:int)
- price: Buy now price of the product. If 0, price was unavailable. (type:float, currency: USD)
- listPrice: Original price of the product before discount. If 0, no list price was found AKA, no discounts. (type:float, currency: USD)
- category_id: Use the amazon_categories.csv to find the actual category name. (type:int)
- isBestSeller: Whether the product had the Amazon BestSeller status or not. (type:bool)

Uploading the dataset to DBFS:

- Amazon Products Dataset 2023 was obtained from the Kaggle which contained 2 files namely amazon_produt and amazon_categories.
- To upload the file into the system we select the File option from the top left corner of Data Bricks and then choose Upload data to DBFS option.



- Next Upload Data popup will come and there we can upload the files from the file system by selecting next and it will move to the Data Bricks notebook


Upload Data

DBFS Target Directory
/FileStore/ shared_uploads/athimoolam.k@northeastern.edu/

Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

Files

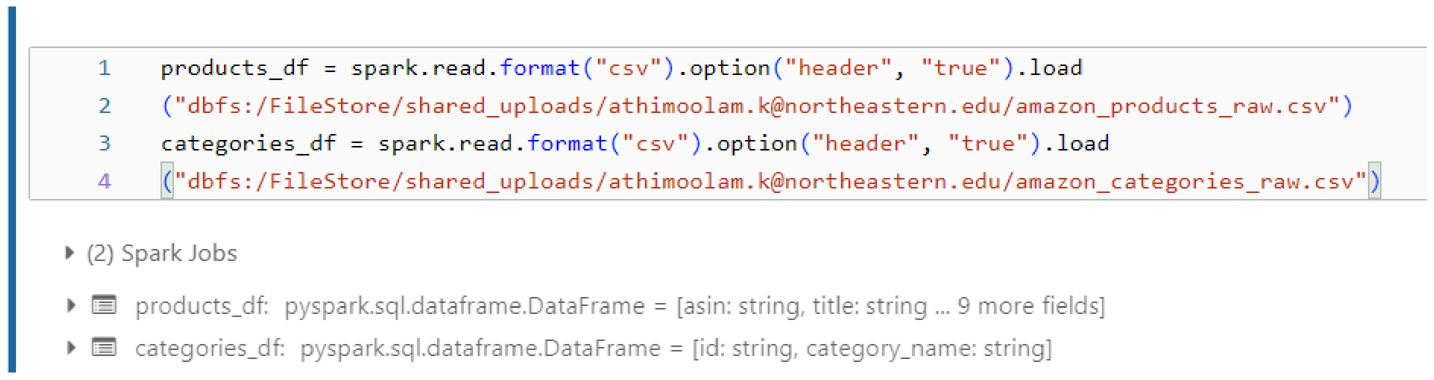
amazon_produ
0.4 GB

amazon_catego
6.8 KB

Code work with Data Bricks:

- Once the files are uploaded to work with the data or to create a table we need to first load the CSV files into a data frame.
- Here we created 2 data frames namely products_df and categories_df and loaded both files respectively.

```
products_df = spark.read.format("csv").option("header", "true").load
("dbfs:/FileStore/shared_uploads/athimoolam.k@northeastern.edu/amazon_products_
raw.csv")
categories_df = spark.read.format("csv").option("header", "true").load
("dbfs:/FileStore/shared_uploads/athimoolam.k@northeastern.edu/amazon_categorie
s_raw.csv")
```

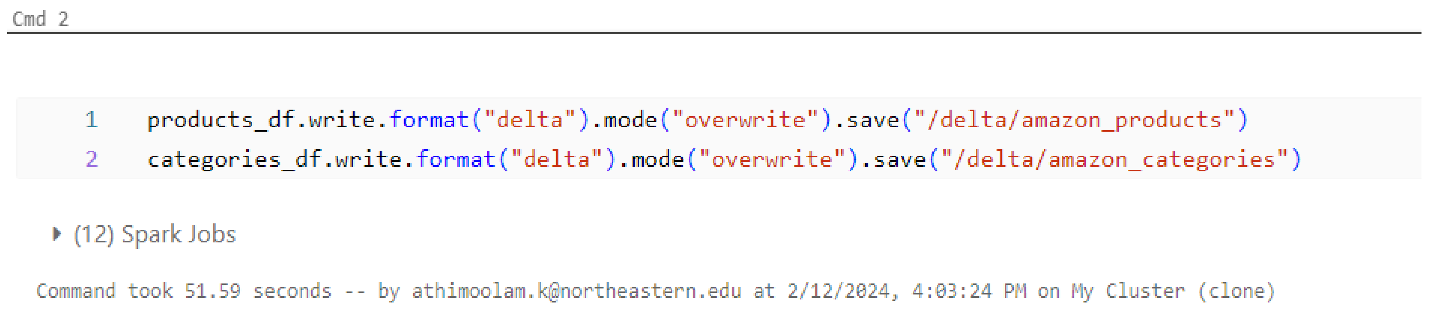


```
products_df = spark.read.format("csv").option("header", "true").load
("dbfs:/FileStore/shared_uploads/athimoolam.k@northeastern.edu/amazon_products_raw.csv")
categories_df = spark.read.format("csv").option("header", "true").load
("dbfs:/FileStore/shared_uploads/athimoolam.k@northeastern.edu/amazon_categories_raw.csv")
```

```
▸ (2) Spark Jobs
▸ products_df:  pyspark.sql.dataframe.DataFrame = [asin: string, title: string ... 9 more fields]
▸ categories_df:  pyspark.sql.dataframe.DataFrame = [id: string, category_name: string]
```

- Here we write the 2 data frames into Delta Lake format, with an overwrite mode, meaning that if any data exists in those locations, it will be replaced by the new DataFrames.
- The .save() method is used to specify the location where the DataFrames will be saved.

```
products_df.write.format("delta").mode("overwrite").save("/delta/amazon_products")

categories_df.write.format("delta").mode("overwrite").save("/delta/amazon_categories")
```



Cmd 2

```
products_df.write.format("delta").mode("overwrite").save("/delta/amazon_products")
categories_df.write.format("delta").mode("overwrite").save("/delta/amazon_categories")
```

```
▸ (12) Spark Jobs
Command took 51.59 seconds -- by athimoolam.k@northeastern.edu at 2/12/2024, 4:03:24 PM on My Cluster (clone)
```

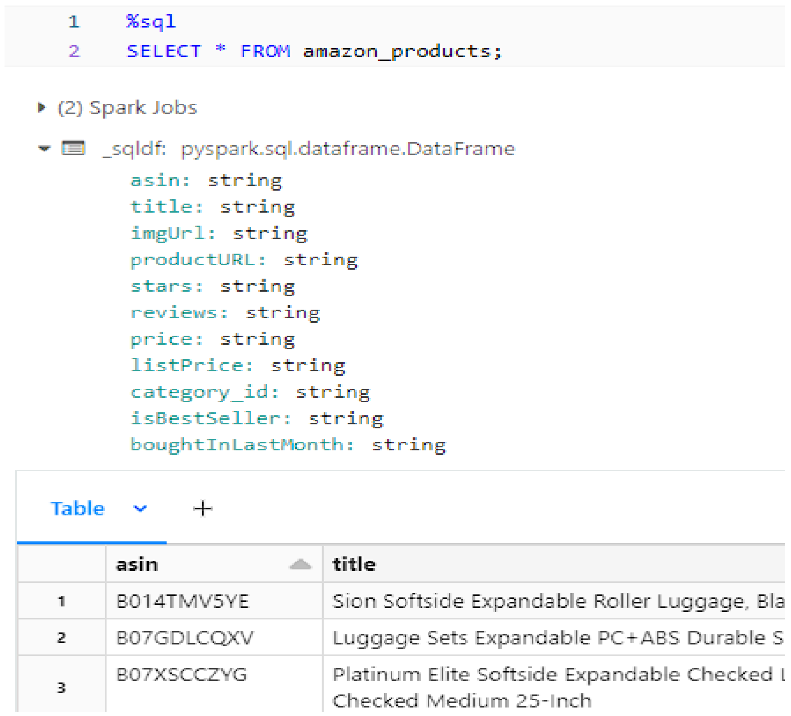
- Now we are creating Delta tables in Databricks using SQL commands.
- The CREATE TABLE statement with the USING DELTA syntax is used to create a Delta table, and the LOCATION parameter specifies the location where the Delta files are stored.

```
%sql

CREATE TABLE amazon_products USING DELTA LOCATION '/delta/amazon_products/';

CREATE TABLE amazon_categories USING DELTA LOCATION '/delta/amazon_categories/';
```

```
%sql
CREATE TABLE amazon_products USING DELTA LOCATION '/delta/amazon_products/';
CREATE TABLE amazon_categories USING DELTA LOCATION '/delta/amazon_categories/';

```

- Once the table is loaded we need to make sure that all data is available hence we run the select statement on both the tables.
- We do the select query for both amazon_products and amazon_categories to see if the data is loaded correctly or not.
- Here we can see that all the data has been loaded correctly and the data type for each column is a string.
- As all the data type is string we need to make sure that they are converted in to proper data types before we make a final table.

```
%sql

SELECT * FROM amazon_products;
```



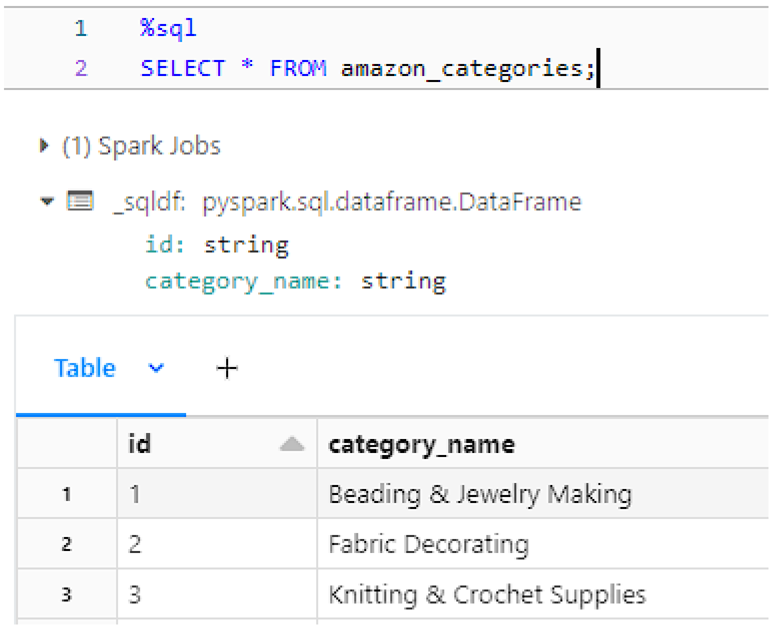
```
%sql
SELECT * FROM amazon_products;
```

```
▸ (2) Spark Jobs
▾ _sqldf: pyspark.sql.dataframe.DataFrame
    asin: string
    title: string
    imgUrl: string
    productURL: string
    stars: string
    reviews: string
    price: string
    listPrice: string
    category_id: string
    isBestSeller: string
    boughtInLastMonth: string
```

Table +

	asin	title
1	B014TMV5YE	Sion Softside Expandable Roller Luggage, Bla
2	B07GDLCQXV	Luggage Sets Expandable PC+ABS Durable S
3	B07XSCCZYG	Platinum Elite Softside Expandable Checked L Checked Medium 25-Inch

```
%sql

SELECT * FROM amazon_categories;
```



```
%sql
SELECT * FROM amazon_categories;
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame

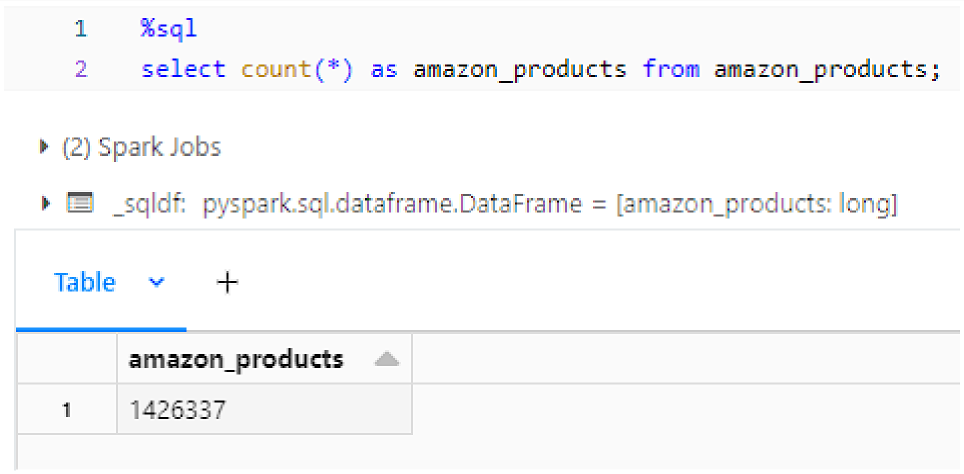
```
id: string
category_name: string
```

Table

	id	category_name
1	1	Beading & Jewelry Making
2	2	Fabric Decorating
3	3	Knitting & Crochet Supplies

- To check the size or the number of rows present in the data we run a count query on the products table and categories table.
- In our case, amazon_products has 1426337 number of records, and amazon_categories has 248 records.

```
%sql

select count(*) as amazon_products from amazon_products;
```



```
%sql
select count(*) as amazon_products from amazon_products;
```

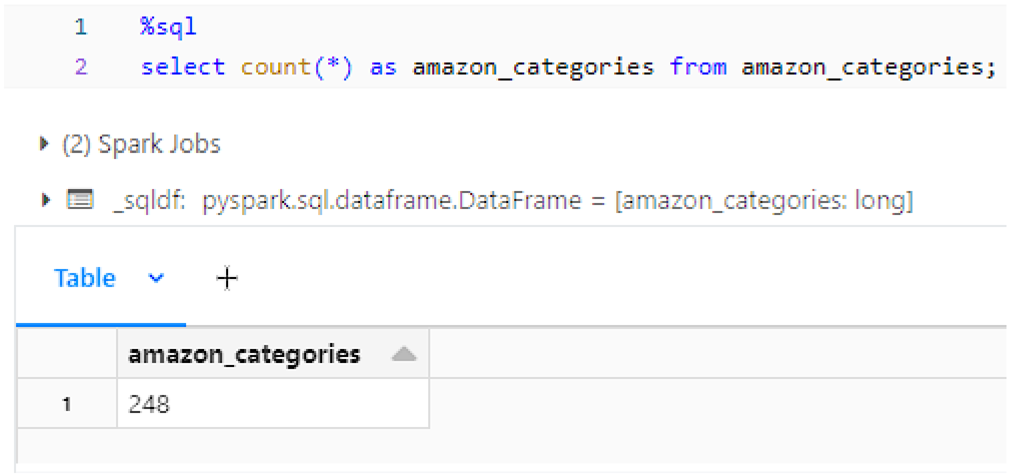
(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [amazon_products: long]

Table

	amazon_products
1	1426337

```
%sql

select count(*) as amazon_categories from amazon_categories;
```



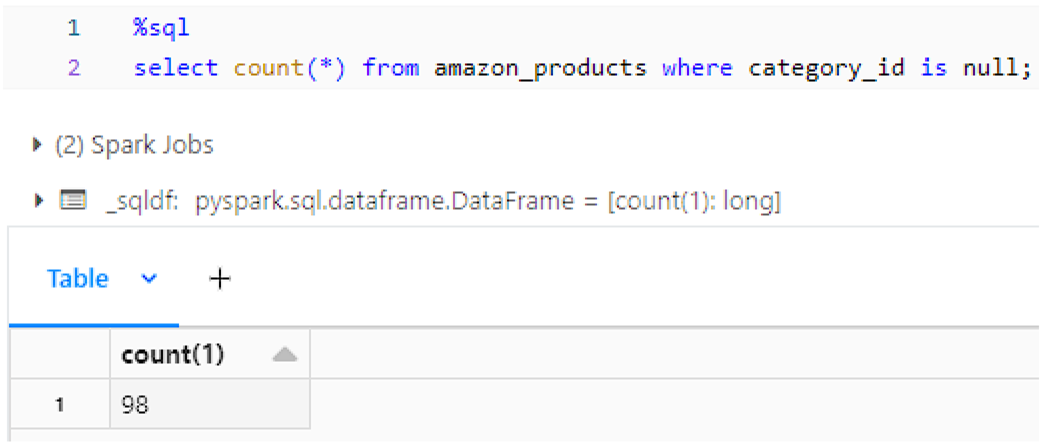
```
%sql
select count(*) as amazon_categories from amazon_categories;
```

```
(2) Spark Jobs
_sqldf:  pyspark.sql.dataframe.DataFrame = [amazon_categories: long]
```

	amazon_categories
1	248

- When we inspect the data we find that the id column in the amazon_cateogires table and category_id in the amazon_products table are the keys for joining the table.
- In Amazon categories id is the primary key and amazon_products table category_id is the foreign key for joining these 2 tables.
- Since these are the joining keys we need to check for junk values mainly for null values before joining.
- As we run the query to check the null values we find that there are 98 null values for the column category_id hence these need to be cleaned up before joining both tables.

```
%sql

select count(*) from amazon_products where category_id is null;
```



```
%sql
select count(*) from amazon_products where category_id is null;
```

(2) Spark Jobs

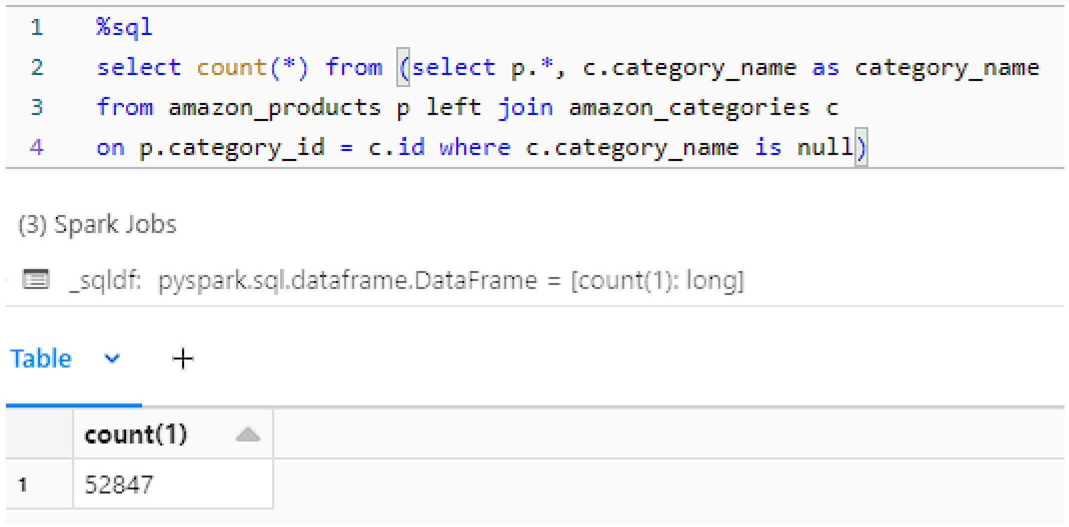
_sqldf: pyspark.sql.dataframe.DataFrame = [count(1): long]

	count(1)
1	98

- Also, we need to check in the final table where we join the data from both tables will have any issues among the columns.
- The subquery performs a left join between the amazon_products table and the amazon_categories table on the category_id column from amazon_products and the id column from amazon_categories. It selects all columns from amazon_products and the category_name column from amazon_categories.
- The join condition p.category_id = c.id links the amazon_products table to the amazon_categories table based on their respective id and category_id columns.
- The `left join` ensures that all records from the amazon_products table are included in the result set, even if there's no matching record in the amazon_categories table.
- The where clause c.category_name is null filters the result set to include only records where the category_name from the amazon_categories table is null, meaning there was no matching category found for the category_id in the amazon_products table.
- Finally, the outer query wraps this subquery and applies a count(*) aggregation function, counting the number of records in the result set.
- This count represents the number of records in the amazon_products table where there is no corresponding category name found in the amazon_categories table.
- Here we find out that there are several null values in the category_name column as well where the count stand at 52847.

```
%sql
```

```
select count(*) from (select p.*, c.category_name as category_name from
amazon_products p left join amazon_categories c on p.category_id = c.id where
c.category_name is null)
```



```
%sql
select count(*) from (select p.*, c.category_name as category_name
from amazon_products p left join amazon_categories c
on p.category_id = c.id where c.category_name is null)
```

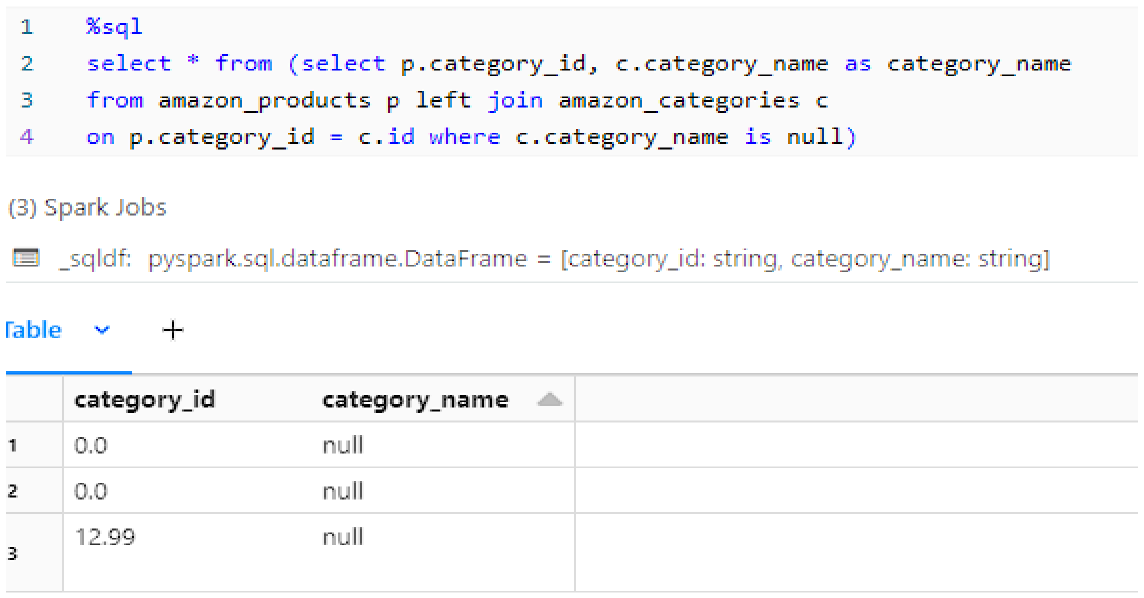
(3) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [count(1): long]

	count(1)
1	52847

- In the next query we are trying to see the data before deciding on the removal of the null valued data.
- Here instead of the count(*) function we use the select function to display the data present in the table.
- As we can see the data here is not proper and they are junk values. Hence it is important that we either remove the junk values or fill them with relevant values.
- Since the junk values were not relevant it is better that we removed the data from the table to make sure it is clean.
- This removal can be done with final table loading.

```
%sql

select * from (select p.category_id, c.category_name as category_name from
amazon_products p left join amazon_categories c on p.category_id = c.id where
c.category_name is null)
```



```
%sql
select * from (select p.category_id, c.category_name as category_name
from amazon_products p left join amazon_categories c
on p.category_id = c.id where c.category_name is null)
```

(3) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [category_id: string, category_name: string]

	category_id	category_name
1	0.0	null
2	0.0	null
3	12.99	null

- Once we understand the null values and junk values we also need to make sure that we only use the required columns.
- Here 2 columns namely imgUrl and productURL are not useful for our analysis as they contain the URL for the image and procut respectively.
- Hence while creating the final table with the below mentioned query we remove these columns to make the table as per the requirement.

In the next SQL query, it creates or replaces a table named amazon_data. It populates this table by selecting specific columns from the amazon_products table and joining it with the amazon_categories table.

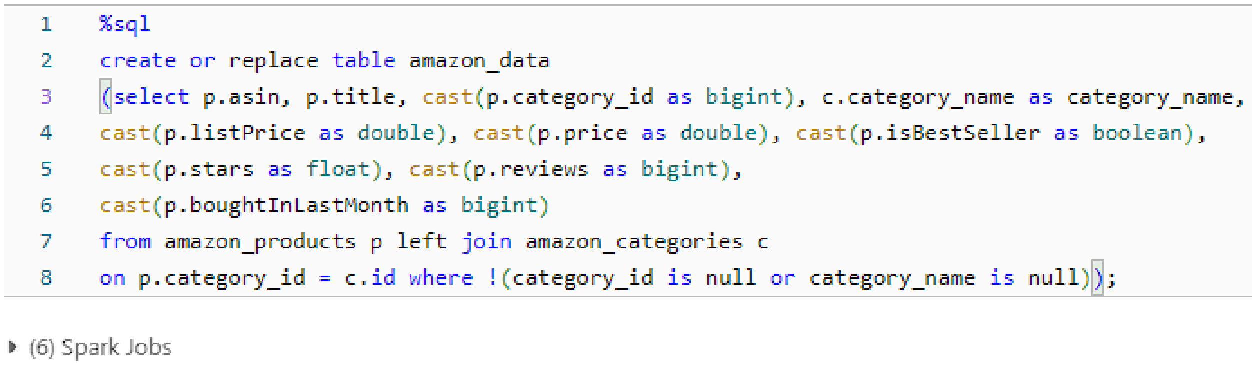
Here's a breakdown of what it does:

1. It selects columns asin, title, category_id, listPrice, price, isBestSeller, stars, reviews, and boughtInLastMonth from the amazon_products table (p alias).
2. It selects the category_name column from the amazon_categories table (c alias).
3. It performs a left join between amazon_products and amazon_categories based on the category_id column.
4. It filters out rows where either category_id or category_name is null.
5. It casts certain columns to specific data types:

6. category_id is cast to bigint.
7. listPrice and price are cast to double.
8. isBestSeller is cast to a boolean.
9. stars is cast to a float.
10. reviews and boughtInLastMonth are cast to bigint.

The resulting table amazon_data will contain data from amazon_products along with the corresponding category name from amazon_categories, ensuring that only rows with valid category information are included and with the specified data types.

```
%sql

create or replace table amazon_data(select p.asin, p.title, cast(p.category_id
as bigint), c.category_name as category_name,cast(p.listPrice as double),
cast(p.price as double), cast(p.isBestSeller as boolean),cast(p.stars as
float), cast(p.reviews as bigint),cast(p.boughtInLastMonth as bigint)from
amazon_products p left join amazon_categories c on p.category_id = c.id where
!(category_id is null or category_name is null));
```

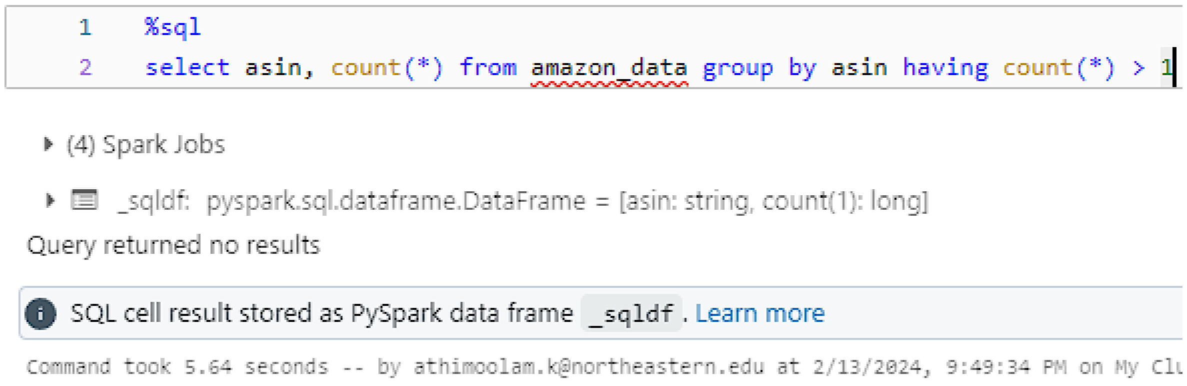


```
%sql
create or replace table amazon_data
(select p.asin, p.title, cast(p.category_id as bigint), c.category_name as category_name,
cast(p.listPrice as double), cast(p.price as double), cast(p.isBestSeller as boolean),
cast(p.stars as float), cast(p.reviews as bigint),
cast(p.boughtInLastMonth as bigint)
from amazon_products p left join amazon_categories c
on p.category_id = c.id where !(category_id is null or category_name is null));
```

- Once the final table is created, we did a sanity check to identify if there are any duplicates with respect to the primary column 'asin'.
- The screenshot below shows that there were no duplicate entries found in the data.

```
%sql
```

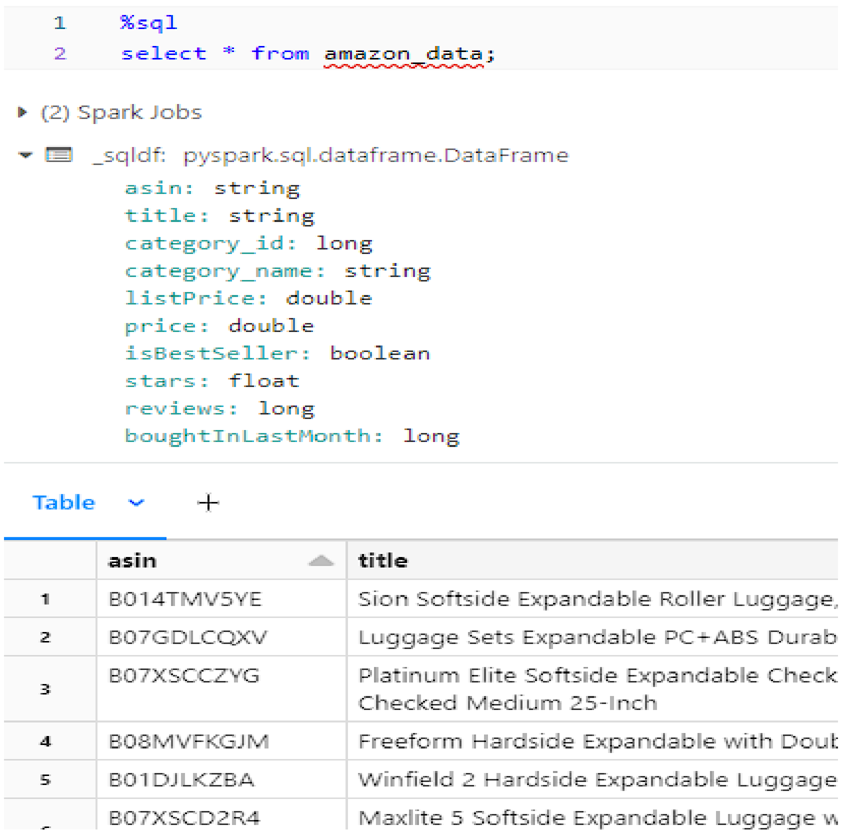
```
select asin, count(*) from amazon_data group by asin having count(*) > 1
```



```
%sql
select asin, count(*) from amazon_data group by asin having count(*) > 1
```

```
(4) Spark Jobs
_sqldf:  pyspark.sql.dataframe.DataFrame = [asin: string, count(1): long]
Query returned no results
SQL cell result stored as PySpark data frame _sqldf . Learn more
Command took 5.64 seconds -- by athimoolam.k@northeastern.edu at 2/13/2024, 9:49:34 PM on My Clu
```

- Now once all the joining and data cleanup is done we have the final table with us names as amazon_data.
- To check if the data is loaded properly we run the below query.
- As we can see that the data is loaded properly into the table hence we have the final table called amaon_data with us for our analysis purposes.



```
%sql
select * from amazon_data;
```

```
(2) Spark Jobs
_sqldf:  pyspark.sql.dataframe.DataFrame
    asin:  string
    title:  string
    category_id:  long
    category_name:  string
    listPrice:  double
    price:  double
    isBestSeller:  boolean
    stars:  float
    reviews:  long
    boughtInLastMonth:  long
```

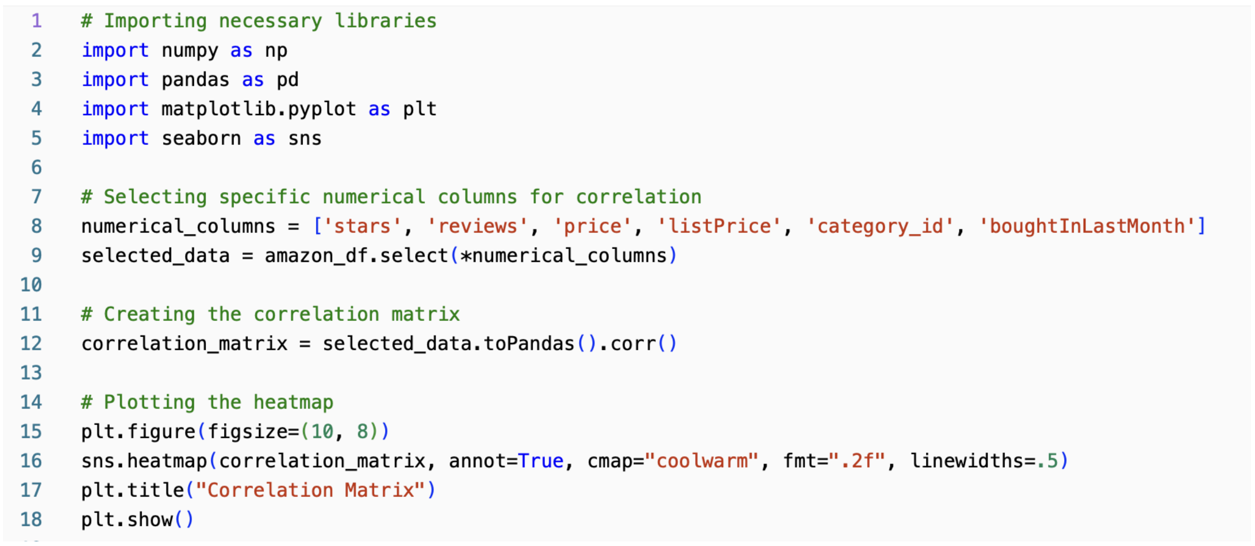
Table

	asin	title
1	B014TMV5YE	Sion Softside Expandable Roller Luggage,
2	B07GDLCQXV	Luggage Sets Expandable PC+ABS Durab
3	B07XSCCZYG	Platinum Elite Softside Expandable Check Checked Medium 25-Inch
4	B08MVFKGJM	Freeform Hardside Expandable with Douk
5	B01DJLKZBA	Winfield 2 Hardside Expandable Luggage
6	B07XSCD2R4	Maxlite 5 Softside Expandable Luggage w

Analysis

In this in-depth analysis, I explore key aspects of market dynamics, customer preferences, and demand patterns using three essential visualizations: a correlation matrix, bar graph, and scatter plot. Each visualization offers distinct insights into various elements of the business environment, allowing for a clearer understanding of market trends, pricing strategies, and consumer behavior.

Correlation Matrix: (Market Trends and Pricing)



```python
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Selecting specific numerical columns for correlation
numerical_columns = ['stars', 'reviews', 'price', 'listPrice', 'category_id', 'boughtInLastMonth']
selected_data = amazon_df.select(*numerical_columns)

# Creating the correlation matrix
correlation_matrix = selected_data.toPandas().corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=.5)
plt.title("Correlation Matrix")
plt.show()
```

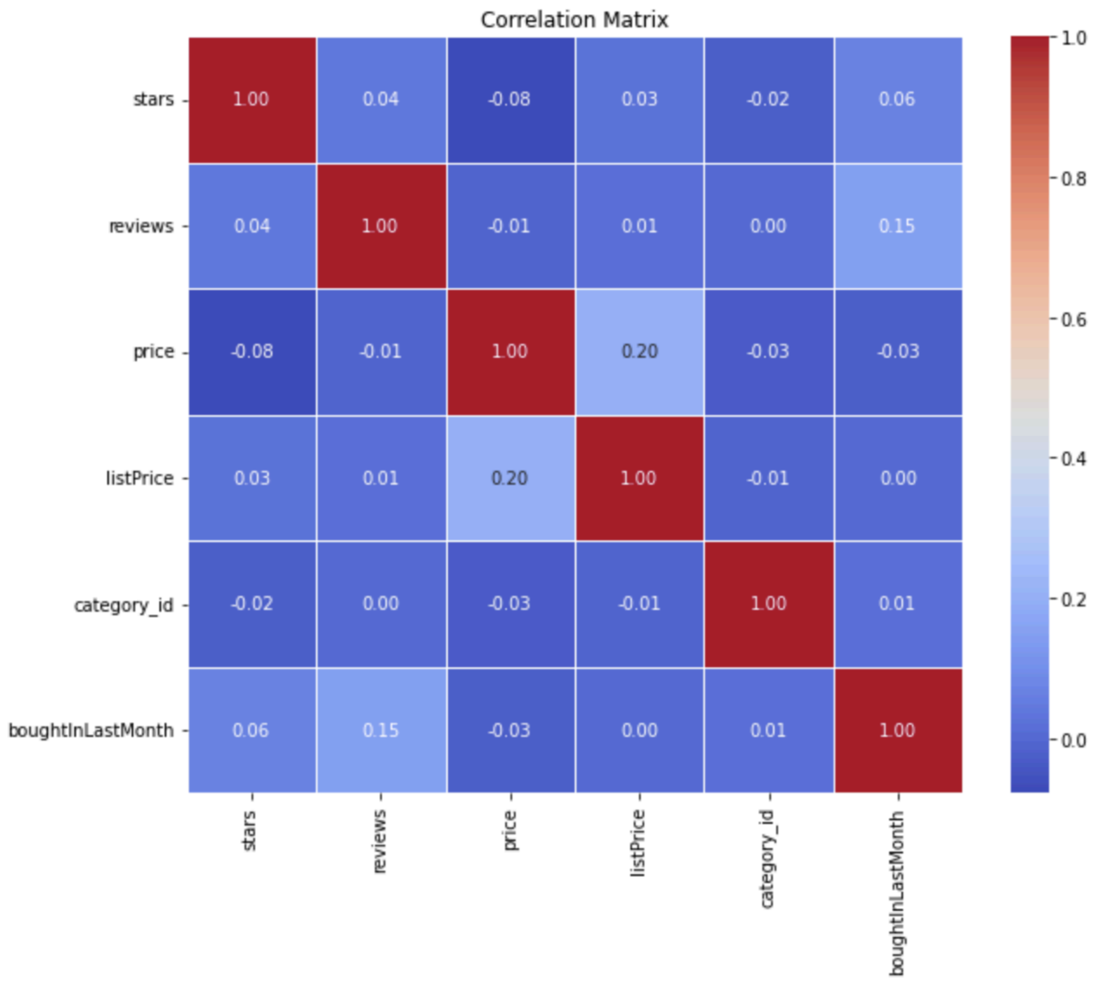
In this code snippet, we perform a correlation analysis on a dataset containing Amazon product information. We begin by importing essential libraries like NumPy, Pandas, Matplotlib, and Seaborn for data manipulation, analysis, and visualization.

Next, we define the key numerical columns from the dataset that will be used for the correlation analysis: 'stars', 'reviews', 'price', 'listPrice', 'category_id', and 'boughtInLastMonth'. These columns are then extracted from the Amazon dataset, which is stored as a DataFrame called 'amazon_df'.

Once the relevant columns are selected, we generate a correlation matrix using Pandas. This matrix calculates the correlation coefficients between pairs of numerical variables, revealing the strength and direction of their linear relationships.

After creating the correlation matrix, we visualize it using a heatmap with Seaborn and Matplotlib. The heatmap uses color to indicate correlation strength—warmer tones for strong positive correlations, cooler tones for strong negative correlations, and neutral colors for weak or no correlations. To enhance clarity, we annotate the heatmap with the actual correlation values.

Finally, we display the heatmap with a title for context, allowing us to easily spot any notable correlations among the selected variables. This visualization helps us uncover relationships within the dataset that could provide valuable insights.



	stars	reviews	price	listPrice	category_id	boughtInLastMonth
stars	1.00	0.04	-0.08	0.03	-0.02	0.06
reviews	0.04	1.00	-0.01	0.01	0.00	0.15
price	-0.08	-0.01	1.00	0.20	-0.03	-0.03
listPrice	0.03	0.01	0.20	1.00	-0.01	0.00
category_id	-0.02	0.00	-0.03	-0.01	1.00	0.01
boughtInLastMonth	0.06	0.15	-0.03	0.00	0.01	1.00

The correlation matrix helps us explore the relationships between various numeric variables, such as pricing, ratings, purchases, and reviews. By analyzing these correlations, we can gain valuable insights into how pricing strategies align with market trends and their impact on overall business performance.

For example, the correlation coefficient of 0.20 between list price and price indicates a positive but relatively weak relationship. This suggests that while list price and actual price tend to increase or decrease together, the connection is not particularly strong, implying that other factors likely influence pricing strategies beyond the list price alone.

In contrast, the correlation coefficient of 0.15 between boughtInLastMonth and reviews suggests a slightly stronger positive correlation. This means that products purchased more frequently in the past month tend to have more reviews. One possible explanation for this is that frequently bought products attract more customer attention, leading to an increased likelihood of receiving reviews. Additionally, products with a higher number of reviews may appear more trustworthy or popular, further boosting their sales.

Bar Graph: (Customer Preferences)

```
# Calculate the total count of purchases for each category
count_purchase_by_category = amazon_df.groupBy('category_name').agg({'boughtInLastMonth': 'count'}).toPandas()

# Select the top ten categories
top_ten_categories = count_purchase_by_category.sort_values(by='count(boughtInLastMonth)', ascending=False).head(10)

# Create a horizontal bar graph for the top ten categories
plt.figure(figsize=(12, 8))
ax = sns.barplot(x='count(boughtInLastMonth)', y='category_name', data=top_ten_categories, color='purple')
plt.xlabel('Total Purchase Count')
plt.ylabel('Category')
plt.title('Top Ten Categories by Total Purchase Count')

plt.show()
```

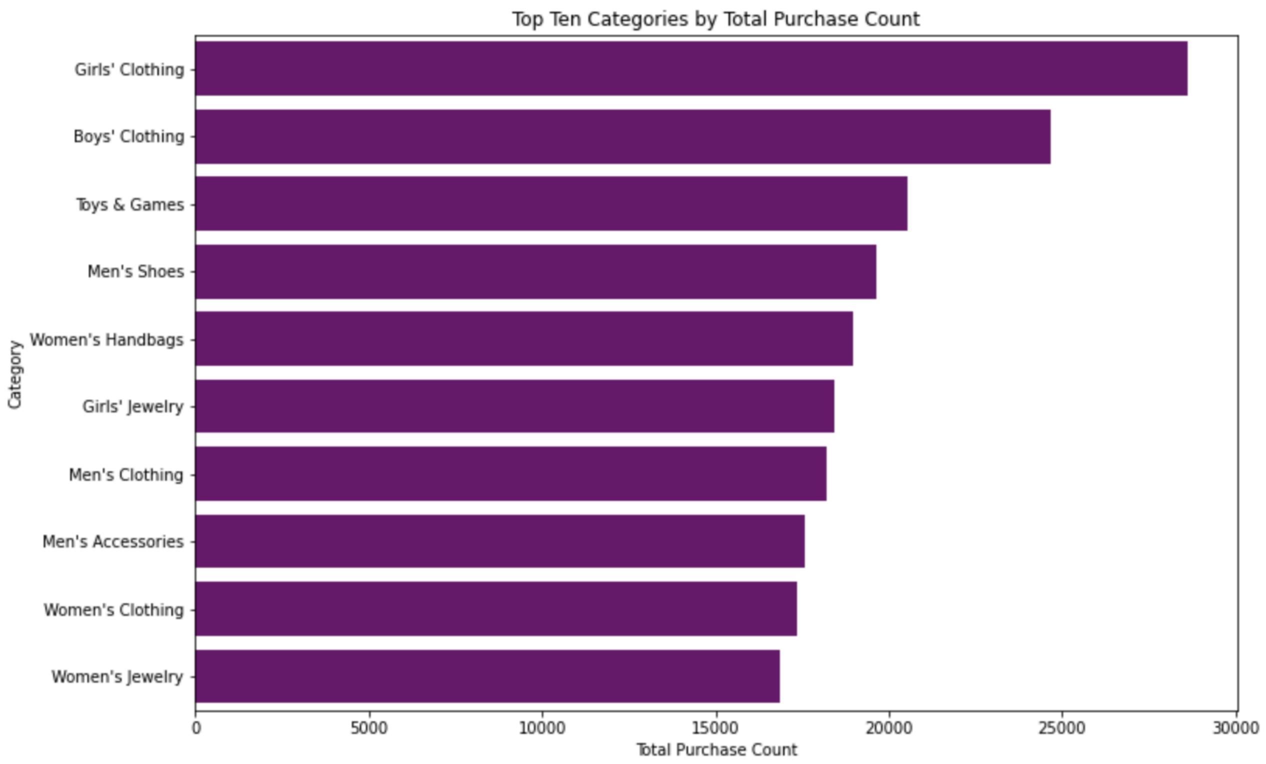
In this code, we're analyzing the total number of purchases for each product category within the Amazon dataset. We begin by grouping the dataset by the 'category_name' column and calculating the number of purchases ('boughtInLastMonth') for each category. This aggregation is done using PySpark's groupBy and agg functions, resulting in a Pandas DataFrame that displays the total purchase count for each category.

Next, we identify the top ten categories based on their purchase totals by sorting the DataFrame in descending order and selecting the first ten rows.

We then visualize these top ten categories by creating a horizontal bar graph. Using Seaborn's barplot function, the x-axis represents the total purchase count ('count(boughtInLastMonth)'), while the y-axis displays the category names. The bars are colored purple to enhance visual appeal.

We further customize the plot by labeling the x-axis as 'Total Purchase Count', the y-axis as 'Category', and adding a title: 'Top Ten Categories by Total Purchase Count'.

Finally, the bar graph is displayed using Matplotlib, illustrating the popularity of each category based on purchase data. This visualization provides key insights into customer preferences and market trends by highlighting the most in-demand product categories on Amazon.



The bar graph offers a visual overview of customer preferences across various product categories. By examining the total purchase counts for each category, we can pinpoint key market segments, gain insight into consumer buying behaviors, and refine marketing strategies to better align with customer needs.

The graph, which displays the top ten categories by total purchase count on Amazon, reveals several noteworthy trends. Children's products dominate the list, with categories such as Girls' Clothing, Boys' Clothing, and Toys & Games leading the way. This highlights a strong preference for these items among Amazon shoppers, likely due to the frequent need to update children's wardrobes and provide entertainment.

Fashion for both men and women also ranks highly, with categories like Men's Shoes, Women's Handbags, Men's Clothing, Women's Clothing, and Women's Jewelry appearing prominently. This underscores the broad shopping habits of Amazon customers, showcasing a demand for both practical and fashionable apparel and accessories.

Interestingly, the graph shows a distinct gap in purchase counts between children's and adult categories, suggesting different shopping behaviors and priorities for those buying for themselves versus for children. Additionally, the similar purchase counts in categories like Men's Shoes, Women's Handbags, and Girls' Jewelry suggest comparable levels of popularity, signaling potential opportunities for deeper market analysis and strategic targeting.

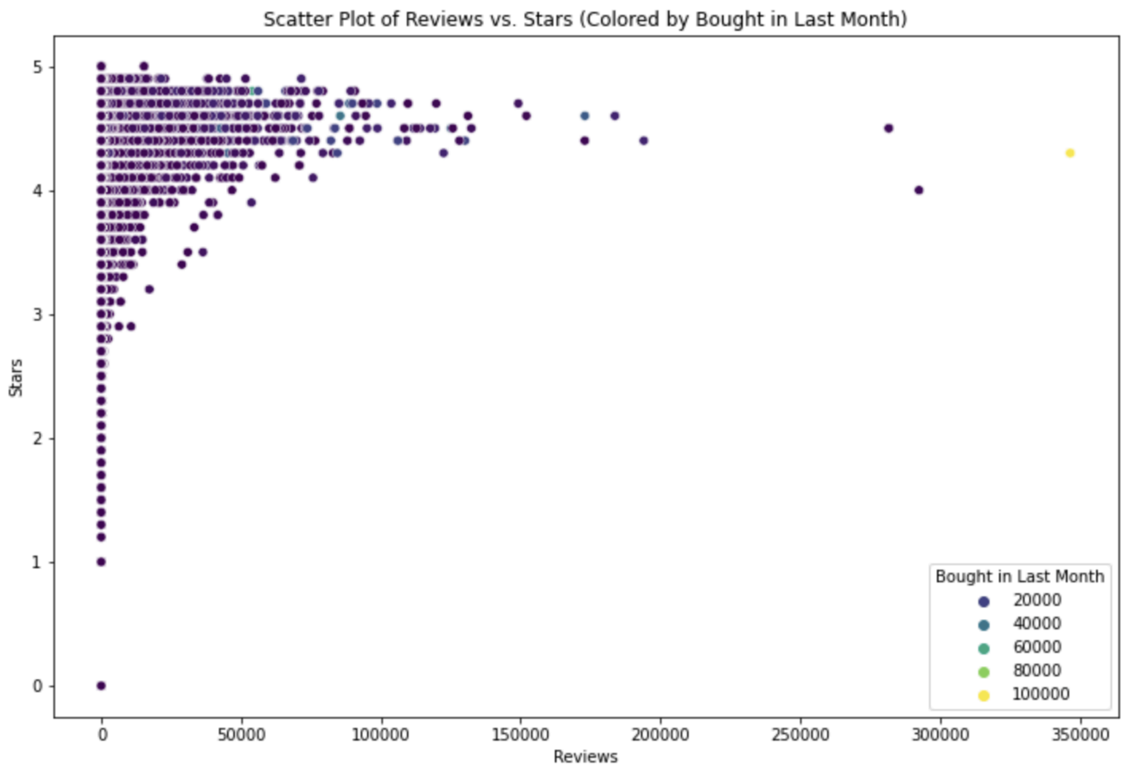
Scatter Plot: (Demand Analysis)

```python
# Convert Spark DataFrame to Pandas DataFrame for plotting
amazon_pd = amazon_df.toPandas()

# Filter out rows where 'boughtInLastMonth' is zero
amazon_pd_filtered = amazon_pd[amazon_pd['boughtInLastMonth'] != 0]

# Scatter plot of reviews vs. stars colored by 'boughtInLastMonth'
plt.figure(figsize=(12, 8))
sns.scatterplot(x='reviews', y='stars', data=amazon_pd_filtered, hue='boughtInLastMonth', palette='viridis')
plt.title('Scatter Plot of Reviews vs. Stars (Colored by Bought in Last Month)')
plt.xlabel('Reviews')
plt.ylabel('Stars')
plt.legend(title='Bought in Last Month')
plt.show()
```

In this code snippet, we're creating a scatter plot to explore the relationship between product reviews and star ratings, with a focus on whether the product was purchased in the last month. We start by converting the Spark DataFrame, 'amazon_df', into a Pandas DataFrame called 'amazon_pd' for easier data manipulation and visualization with Pandas and Matplotlib. Next, we filter out rows where the 'boughtInLastMonth' column is zero, which means we only include products that have been recently purchased. This helps us concentrate on items with current purchase activity.

Then, we use Seaborn's scatterplot function to create the scatter plot. The x-axis represents the 'reviews' column, while the y-axis corresponds to the 'stars' column. Each point on the plot represents a product, positioned based on its review count and star rating. We also color the points based on the 'boughtInLastMonth' column, distinguishing between products purchased recently and those that were not.

Further customizations include setting the title ('Scatter Plot of Reviews vs. Stars (Colored by Bought in Last Month)'), labeling the x-axis ('Reviews') and y-axis ('Stars'), and adding a legend to clarify the color coding for recent purchases. Finally, we display the scatter plot using Matplotlib, enabling us to visually analyze any patterns or trends related to reviews, star ratings, and recent purchase activity. This visualization helps in understanding customer sentiment, the influence of recent purchases on reviews, and identifying any potential outliers in the data.



The scatter plot provides a detailed view of demand dynamics by showcasing the relationship between product reviews, star ratings, and recent purchases. By analyzing the clustering of data points and identifiable patterns, we can gain insights into customer satisfaction, the effect of recent purchases on reviews, and identify potential outliers that may require further examination.

Upon reviewing the scatter plot, it's clear that a strong relationship exists between product reviews and star ratings. The concentration of data points in the upper right quadrant indicates a generally positive sentiment, with many products receiving high star ratings along with a significant number of reviews. This suggests that customers are largely satisfied with these products and feel encouraged to share their feedback.

The differentiation based on recent purchases adds an intriguing dimension to the analysis. The points are colored blue or orange, with blue likely indicating recently purchased products. While there is a slight cluster of blue points in the upper right quadrant, implying that products bought recently may have somewhat higher average reviews and counts, there is still considerable overlap between the blue and orange points.

Pros and Cons of ETL Process:

Pros:

★ **Data Quality and Consistency:** ETL allows for thorough cleaning and transformation of data before it enters the target system, ensuring accuracy and consistency. This enhances data-driven decision making.

★ **Flexibility and Control:** ETL offers greater control over data manipulation, allowing for complex transformations and tailoring to specific needs. This is beneficial for data warehousing environments.

★ **Performance and Efficiency:** By transforming data before loading, ETL can optimize storage and improve query performance in the target system.

★ **Security and Compliance:** ETL allows for implementing data security measures and ensuring compliance with data regulations.

★ **Proven Technology:** ETL is a well-established technology with many mature tools and experienced professionals available.

Cons:

➢ **Development Time and Cost:** Designing and implementing ETL pipelines can be complex and time-consuming, requiring skilled professionals and potentially expensive tools.

➢ **Inflexibility and Latency:** Changes to data sources or transformations require modifying the entire pipeline, leading to delays and reduced adaptability. This can be a challenge for agile environments.

➢ **Limited Scalability:** Traditional ETL processes can struggle with large, high-volume data sets, impacting performance and scalability.

➢ **Operational Complexity:** Managing and maintaining ETL pipelines can be intricate, requiring constant monitoring and error correction.

Difficulties and issues:

☐ **Null Values:** To be able to join the 2 tables we needed the column category_id to be not null. But when we checked the product data we could find there were some null values. hence before joining we had to clean these null values so that there were no junk values in the final table.

☐ **Data cleanup:** We had to create a temporary table to analyze the category_name data column and make sure that it is not null. Once we created the temporary table we found the null values and rechecked the catgory_id column to see if it had any data but there were some junk values. Hence as the category_name and category_id both did not make sense we had to decide to drop this data.

Pros, Cons, & Hurdles Of Analysis

Pros:

- ❖ Comprehensive Analysis: By employing multiple visualization techniques such as correlation matrices, bar graphs, and scatter plots, we gain a holistic understanding of various aspects of the business landscape including market trends, customer preferences, and demand dynamics.
- ❖ Insights Generation: Each visualization provides unique insights that complement and reinforce each other, enabling us to uncover hidden patterns, relationships, and trends within the dataset.
- ❖ Effective Communication: Visualizations offer an intuitive and visually appealing way to communicate complex findings and insights to stakeholders, facilitating better decision-making and strategy formulation.
- ❖ Code Reusability: The code snippets provided are modular and reusable, allowing for easy adaptation and extension to analyze similar datasets or explore different aspects of the business.

Cons:

- ➔ Data Limitations: The analysis is contingent upon the quality and completeness of the dataset. Incomplete or inaccurate data can lead to skewed insights and erroneous conclusions.
- ➔ Interpretation Challenges: While visualizations aid in data exploration and interpretation, they can sometimes be subject to misinterpretation or ambiguity, especially if proper context is not provided or if underlying assumptions are not adequately addressed.
- ➔ Scalability Issues: For larger datasets, processing and analyzing data using certain techniques such as correlation matrices may become computationally intensive and time-consuming, potentially limiting scalability.

Challenges:

- ➢ Code Migration Issues: Transitioning from Python Jupyter Notebook to Databricks posed challenges due to differences in the underlying environment and data processing capabilities. Attempting to replicate the existing code in Databricks resulted in numerous errors and inconsistencies, primarily stemming from discrepancies in data cleaning methodologies between the two platforms. Additionally, integrating existing code and workflows into the Databricks platform proved to be more complex than anticipated, requiring significant adjustments and troubleshooting efforts.
- ➢ Data Cleaning Discrepancies: The data cleaning processes implemented in Python Jupyter Notebook are not the same as what we implemented in Databricks, leading to unexpected outcomes and errors during code execution. Variances in data formats, handling of missing values, and preprocessing techniques between the two environments contributed to the difficulties in achieving consistent results. Data preprocessing, including cleaning and preparing the dataset for analysis, was particularly time-consuming and labor-intensive.
- ➢ Integration Complexity: Aligning data processing steps, dependencies, and libraries across different platforms added layers of complexity to the migration process. The intricacies of integrating existing code and workflows into the Databricks platform required a steep learning curve and adaptation period for the team. Acquiring familiarity with Databricks-specific functionalities, syntax, and best practices presented a significant hurdle in achieving seamless integration and execution.
- ➢ Interpretation Ambiguity: Interpreting the results of the analysis, especially in the context of real-world business implications, may involve subjective judgment and uncertainty. Addressing interpretation ambiguities required an iterative problem-solving approach, with experimentation, debugging, and collaboration essential in refining the code and data processing pipelines to align with the requirements and capabilities of the Databricks platform.

Key Strategies for Effective Analysis

★ Leveraging Python Libraries: Making use of widely-used Python libraries like Pandas, NumPy, Matplotlib, and Seaborn streamlines the process of handling data, performing various analyses, and creating insightful visualizations. These libraries offer powerful tools and functions that expedite data manipulation and enhance visualization capabilities.

★ Thorough Documentation: Providing comprehensive documentation alongside the analysis code enhances transparency and reproducibility. Detailed explanations and comments offer insights into the analysis process, making it easier for team members to understand, replicate, and build upon previous work. Clear documentation also aids in troubleshooting and maintaining the analysis pipeline over time.

★ Modular Code Design: Breaking down the analysis into modular code snippets promotes code reusability, simplifies maintenance, and facilitates collaboration among team members. Modular design allows different components of the analysis to be developed, tested, and modified independently, resulting in more manageable and scalable codebases.

★ Best Practices in Visualization: Adhering to best practices in visualization design ensures that visualizations are effective in conveying insights. Proper labeling, color coding, and annotation help to clarify complex information and highlight key findings. Following visualization best practices enhances the interpretability and impact of the visual representations of data analysis results.

★ Iterative Approach to Analysis: Adopting an iterative approach to analysis involves continuously refining and improving the analysis based on feedback and new insights. By incorporating feedback from stakeholders and revisiting analysis assumptions, teams can enhance the accuracy and relevance of their findings. Iterative analysis allows for the exploration of alternative approaches and the adaptation of the analysis to evolving requirements, leading to more robust and actionable insights.

Conclusion

In this project, I undertook a comprehensive exploration of Amazon's extensive dataset to extract meaningful insights that could guide strategic decision-making. Utilizing a mix of SQL, Python, and visualization tools within the Databricks platform, I navigated the complexities of data integration, cleaning, and analysis to reach actionable conclusions.

My analysis journey began with the establishment of an efficient ETL process, which set the foundation for further investigations. By carefully examining market trends, pricing dynamics, and customer preferences, I uncovered valuable patterns and relationships within the data.

Key findings from my analysis illuminated various aspects of Amazon's business landscape. I identified correlations between pricing strategies and customer reviews, as well as insights into popular product categories and demand dynamics. Each visualization provided unique perspectives that contributed to a comprehensive understanding of the dataset.

Although I faced challenges like data preprocessing complexities and interpretation ambiguities, I successfully navigated these hurdles, ensuring the integrity and reliability of my findings. By adhering to best practices and taking an iterative approach, I refined my analysis to ensure that my conclusions were robust and actionable.

In conclusion, this project highlights the significance of data-driven decision-making in today's competitive market. By leveraging the power of data analytics, businesses can uncover valuable insights that foster innovation, optimize operations, and ultimately achieve strategic success. My findings demonstrate the transformative potential of data analytics in unlocking new opportunities and driving sustainable growth.

References

- Shah, H. (2023, March 13). *The Pros and cons of ETL and ELT for data warehousing*. Medium. https://hardiks.medium.com/the-pros-and-cons-of-etl-and-elt-for-data-warehousing-1b22e565614d
- Trotino, G. (2023, September 21). *What are ETL tools? advantages, disadvantages and Innovations revealed*. Delivering Data Products in a Data Fabric & Data Mesh. https://www.k2view.com/blog/what-are-etl-tools
- *Amazon Products Dataset 2023 (1.4M Products). (2024, February 14). Kaggle. https://www.kaggle.com/datasets/asaniczka/amazon-products-dataset-2023-1-4m-products*
- *D. (2021, August 31). Learn to Use Databricks for Data Science. YouTube. https://www.youtube.com/watch?v=mvezlGqmpYc*