**Implementing RAG for marketing use cases**

|  |  |
| --- | --- |
| **Summary** | The code provided is an Exploratory Data Analysis (EDA) of a dataset containing articles, customers, and transactions. It starts by importing necessary libraries, loading the data, and creating various visualizations to understand patterns and trends in the data. The visualizations include histograms for index names and garment groups, counts of article IDs by index and product groups, and a time series plot for the mean price of different product groups.  Additionally, the code shows images of the 5 highest-priced items on the most recent transaction date, along with their descriptions and prices. Throughout the analysis, various statistical measures, such as mean and standard deviation, are calculated and used to help derive insights from the data. |
| **URL** | https://github.com/Negi97Mohit/Implementing\_RAG.git |
| **Environment** | snowpark\_xgboost\_tpc - snowflake, pip:   youtube-transcript-api, pytube, openai, pinecone - pinecone-client |
| **Status** | Published |
| **Author** | Mohit Negi, Prithvi Vasireddy |

[Introduction](#_ok7k5uux6)

[What you’ll build](#_21yzqg98x7h6)

[What you’ll need](#_319h1e1sjzdz)

[Getting set up](#_bwo0af2iwk90)

[Get a key for the Dark Sky API](#_dy319v7ol5mn)

[Verify your API key is working properly](#_z7dwjt3xqui9)

[Get the code](#_q6selyhmjlab)

[What’s next?](#_9mqifffit2ew)

[Further reading](#_lqephukhewqt)

[Reference docs](#_an95hoqxrlhg)

# Introduction

**relevant tools and techniques that can be potentially be used:**

* Data profiling and exploratory data analysis using Pandas, plotly, etc.:
* The notebook uses Pandas to load, preprocess, and explore the dataset. The dataset contains information about products, orders, and customers. The EDA process includes handling missing values, creating new features, and examining the distribution of the data. Plotly is not used in the notebook, but you could potentially use it for creating interactive visualizations. Visualization using Matplotlib, Seaborn:

The notebook uses Matplotlib and Seaborn for data visualization. It includes bar plots, histograms, pie charts, and scatter plots to analyze and visualize different aspects of the dataset, such as orders over time, orders by country, top-selling products, etc.

***Insights derived from the EDA:***

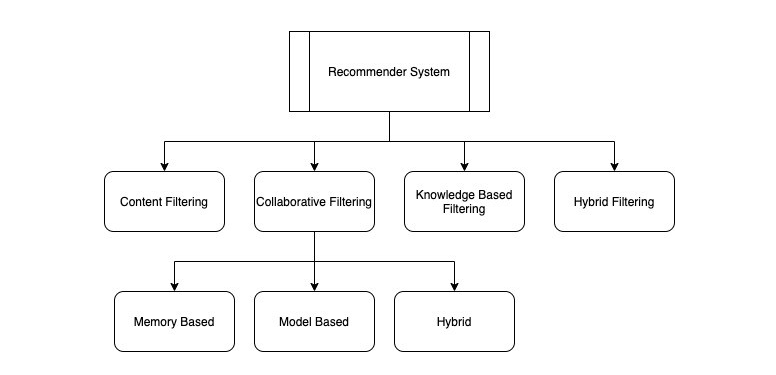
1. The dataset contains 3 customer types: 'Loyal\_customer' (68.1%), 'New\_customer' (22.7%), and 'Eco\_customer' (9.2%).
2. The average rating for products is 4.17 out of 5, with a standard deviation of 1.33. The minimum rating is 1, and the maximum is 5.
3. There are 6 unique retailers in the dataset. Retailer 3 has the highest number of sales (32.7%), followed by Retailer 1 (29.1%), Retailer 6 (16.6%), Retailer 4 (11.9%), Retailer 5 (5.8%), and Retailer 2 (3.9%).
4. The gender distribution in the dataset is almost equal: 'F' (50.2%) and 'M' (49.8%).
5. The dataset contains orders from multiple countries, with the majority of orders coming from the United States and France.
6. Most orders are placed on weekdays, with weekends having the lowest order count.
7. The data suggests seasonality in orders, with higher order volumes during specific months.
8. Top-selling products are identified, providing insights into customer preferences.
9. There is a noticeable difference in the distribution of order prices between different countries.

**Potential applications of the concepts, algorithms, and tools from the list:**

1. Advertising Spend and ROI prediction: With the insights derived from the EDA, you can build models to predict the optimal advertising spend and return on investment (ROI) for each product or market segment.
2. Cohort Analysis: Analyze customer cohorts to track customer behavior over time and identify trends or changes.
3. Customer Lifetime Value prediction: Utilize the dataset to create models that predict the lifetime value of customers, helping to inform marketing and customer retention strategies.
4. Interface and hosting using Streamlit: Develop a Streamlit app to visualize and interact with the dataset, allowing users to explore the data easily.
5. Snowflake/Snowpark: Store and query the dataset in Snowflake for more efficient data management and processing.
6. OpenAI API offerings: Use OpenAI APIs to build or augment applications, such as text-based or image-based product recommendations.
7. Search methodologies: Apply search methodologies like semantic search, keyword search, or hybrid search to enhance product discovery for users.
8. Question Answering methodologies: Implement retrieval augmented generative question answering using Pinecone & OpenAI for customer support or product information retrieval.

**Individual Contribution:**

1. Common Tasks:
   1. Data warehousing – Since there are multiple moving parts, we require multiple pinecone vectorizers for our use. Both must implement and use pinecone.
   2. EDA – The dataset is huge, to cover the EDA, we require all hands on deck.
2. Prithvi’s Part:
   1. Data profiling
   2. Advertising spend & ROI prediction.
   3. Cohort Analysis
   4. Image search using vector embeddings.
   5. Open AI recommendation system
3. Mohit’s Part:
   1. Great expectations
   2. Customer lifetime value predictions
   3. Semantic / Key word search
   4. RAG question answering
   5. Sentiment analysis and content-based filtering recommendations system.



import numpy as np

import pandas as pd

import seaborn as sns

from matplotlib import pyplot as plt

from tqdm.notebook import tqdm

**# Load datasets**

articles = pd.read\_csv("articles.csv")

customers = pd.read\_csv("customers.csv")

transactions = pd.read\_csv("transactions\_train.csv")

**# Plot a histogram to show the count of articles by index name**

f, ax = plt.subplots(figsize=(15, 7))

ax = sns.histplot(data=articles, y='index\_name', color='orange')

ax.set\_xlabel('count by index name')

ax.set\_ylabel('index name')

plt.show()

**# Plot a stacked histogram to show the count of articles by garment group and index group**

f, ax = plt.subplots(figsize=(15, 7))

ax = sns.histplot(data=articles, y='garment\_group\_name', color='orange', hue='index\_group\_name', multiple="stack")

ax.set\_xlabel('count by garment group')

ax.set\_ylabel('garment group')

plt.show()

**# Display the count of articles by index group name and index name**

articles.groupby(['index\_group\_name', 'index\_name']).count()['article\_id']

**# Set pandas display options to show all rows**

pd.options.display.max\_rows = None

**# Display the count of articles by product group name and product type name**

articles.groupby(['product\_group\_name', 'product\_type\_name']).count()['article\_id']

Here, we -

1. Import necessary libraries such as NumPy, Pandas, Seaborn, and Matplotlib.
2. Load the datasets 'articles.csv', 'customers.csv', and 'transactions\_train.csv' into Pandas DataFrames.
3. Plot a histogram to visualize the count of articles by 'index\_name'.
4. Plot a stacked histogram to visualize the count of articles by 'garment\_group\_name' and 'index\_group\_name'.
5. Display the count of articles by 'index\_group\_name' and 'index\_name' using the Pandas **groupby** method.
6. Set the maximum number of rows displayed in Pandas to None, so that all rows can be seen.
7. Display the count of articles by 'product\_group\_name' and 'product\_type\_name' using the Pandas **groupby** method.

These visualizations and groupings help to better understand the distribution of articles across various categories in the dataset.

# Convert 't\_dat' column to datetime format

articles\_for\_merge['t\_dat'] = pd.to\_datetime(articles\_for\_merge['t\_dat'])

# Define a list of product groups and corresponding colors for plotting

product\_list = ['Shoes', 'Garment Full body', 'Bags', 'Garment Lower body', 'Underwear/nightwear']

colors = ['cadetblue', 'orange', 'mediumspringgreen', 'tomato', 'lightseagreen']

# Initialize a counter variable

k = 0

# Create subplots for visualizing mean prices for each product group over time

f, ax = plt.subplots(3, 2, figsize=(20, 15))

for i in range(3):

for j in range(2):

try:

product = product\_list[k]

articles\_for\_merge\_product = articles\_for\_merge[articles\_for\_merge.product\_group\_name == product\_list[k]]

series\_mean = articles\_for\_merge\_product[['t\_dat', 'price']].groupby(pd.Grouper(key="t\_dat", freq='M')).mean().fillna(0)

series\_std = articles\_for\_merge\_product[['t\_dat', 'price']].groupby(pd.Grouper(key="t\_dat", freq='M')).std().fillna(0)

# Plot the mean price and fill the area between the mean-2\*std and mean+2\*std

ax[i, j].plot(series\_mean, linewidth=4, color=colors[k])

ax[i, j].fill\_between(series\_mean.index, (series\_mean.values-2\*series\_std.values).ravel(),

(series\_mean.values+2\*series\_std.values).ravel(), color=colors[k], alpha=.1)

ax[i, j].set\_title(f'Mean {product\_list[k]} price in time')

ax[i, j].set\_xlabel('month')

ax[i, j].set\_xlabel(f'{product\_list[k]}')

k += 1

except IndexError:

ax[i, j].set\_visible(False)

plt.show()

# Import necessary libraries for image handling

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

# Get the 5 highest priced items on the most recent date and the 5 lowest priced items on the earliest date

max\_price\_ids = transactions[transactions.t\_dat==transactions.t\_dat.max()].sort\_values('price', ascending=False).iloc[:5][['article\_id', 'price']]

min\_price\_ids = transactions[transactions.t\_dat==transactions.t\_dat.min()].sort\_values('price', ascending=True).iloc[:5][['article\_id', 'price']]

# Create a subplot to display images of the highest priced items

f, ax = plt.subplots(1, 5, figsize=(20, 10))

i = 0

for \_, data in max\_price\_ids.iterrows():

desc = articles[articles['article\_id'] == data['article\_id']]['detail\_desc'].iloc[0]

desc\_list = desc.split(' ')

for j, elem in enumerate(desc\_list):

if j > 0 and j % 5 == 0:

desc\_list[j] = desc\_list[j] + '\n'

desc = ' '.join(desc\_list)

# Load the image corresponding to the article ID

img = mpimg.imread(f'images/0{str(data.article\_id)[:2]}/0{int(data.article\_id)}.jpg')

ax[i].imshow(img)

ax[i].set\_title(f'price: {data.price:.2f}')

ax[i].set\_xticks([], [])

ax[i].set\_yticks([], [])

ax[i].grid(False)

ax[i].set\_xlabel(desc, fontsize=10)

i += 1

We then create a subplot to visualize the images of the highest-priced items:

1. Iterate over each item in `max\_price\_ids`, which contains the article IDs and prices of the 5 highest-priced items on the most recent date.

2. Get the `detail\_desc` of the item using its `article\_id` and split the description into a list of words.

3. Add a newline character after every 5 words in the description list to improve readability.

4. Join the modified description list back into a string.

5. Load the image corresponding to the `article\_id` from the appropriate folder.

6. Display the image in the subplot, along with the item's price as the title and the formatted description as the x-axis label.

7. Repeat the process for all 5 highest-priced items, incrementing the index `i` for each subplot.

Finally, the `plt.show()` command displays the entire figure with images of the highest-priced items.

# Congratulations

Congratulations, you've successfully finished implementation!

## Further reading --

## *Reference docs:*

* https://bigcodegen.medium.com/transcribing-youtube-video-using-whisper-fo  
  r-gpt-3-text-summarization-ad80dfcba9ed
* https://docs.pinecone.io/docs/gen-qa-opena