**2. Web scraping**

**Amazon Product Dataset**

Scraped Dataset

Src: WEB

**Description:**

The Amazon Products Dataset contains information about various products listed on Amazon. It includes features such as product name, category, price, rating, rating count. This data can be used for analysis and modeling related to product performance, customer preferences, and market trends.

**Data Collection and Understanding:**

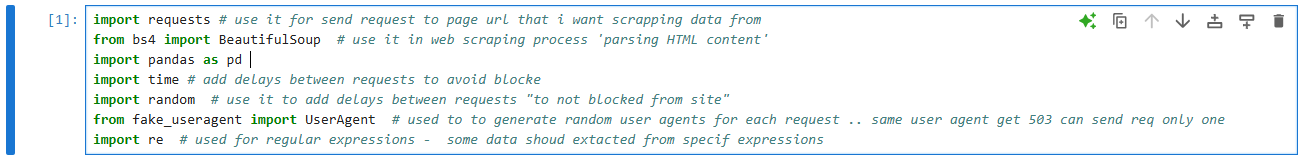
1. **Attributes / Feature Description**

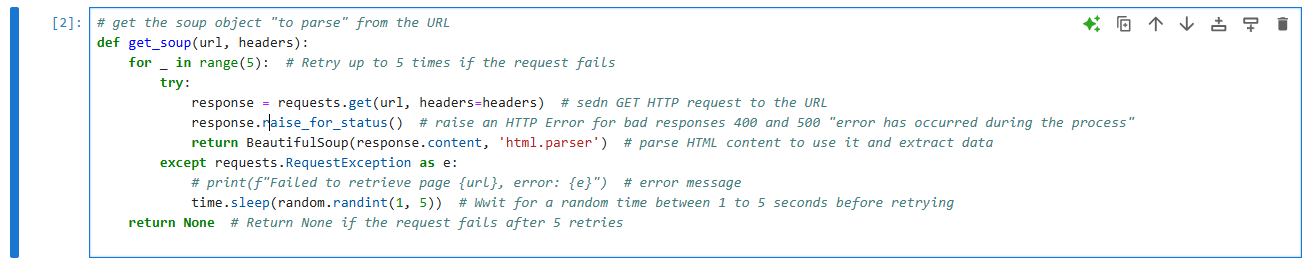
|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | Description | Data Type | Attribute Type |
| index | The index number for each entry. | int64 | Discrete |
| page | The page number where the product was found. | int64 | Discrete |
| name | The name of the product. | object | Nominal |
| category | The category the product belongs to. | object | Nominal |
| image | URL to the product image. | object | Nominal |
| price | The price of the product. | object | Numeric (Ratio-Scaled) |
| rating | The average customer rating for the product (0 to 5). | float64 | Numeric (Interval-Scaled) |
| rating\_count | The number of ratings received by the product. | float64 | Numeric (Ratio-Scaled) |
| delivery | The delivery option or availability. Ex: "Prime", "Free Shipping", "Standard" | object | Nominal |
| is\_best\_seller | If the product is a best seller (1 = yes, 0 = no). | int64 | Binary |
| is\_overall\_pick | If product is an overall pick (1 = yes, 0 = no). | int64 | Binary |

1. **Dataset Collection**

In this Part I collect data using Scraping Techniques “Python”:

1. Import Libraries used in scraping task



1. Get Soup Function: using BeautifulSoup library “is a Python library for pulling data out of HTML and XML files.” This allowing to parse HTML documents and extract the data needed & Request library to send request to specific page to parse it’s HTML content
2. Parse Product: takes a product element from the HTML, the page number, and the product category. It then extracts various attributes related to the product such as name, price, image src, rating and returns these details in a dictionary {key : value}.





### **Helper Function: safe\_find**

def safe\_find(element, search\_dict, text=False):

try:

found = element.find(\*\*search\_dict)

return found.text.strip() if (found and text) else found

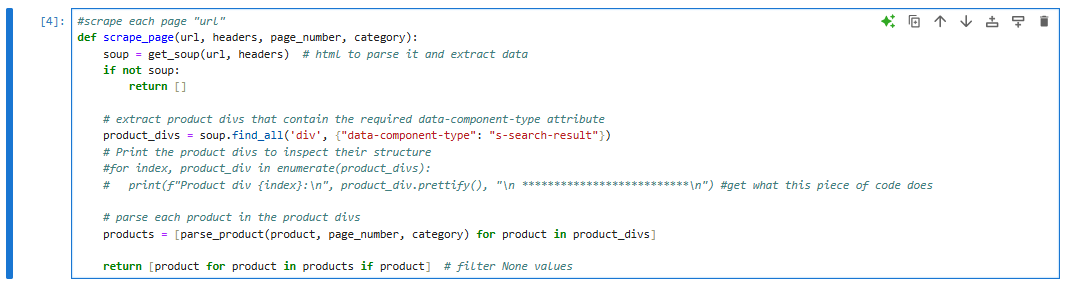
except AttributeError:

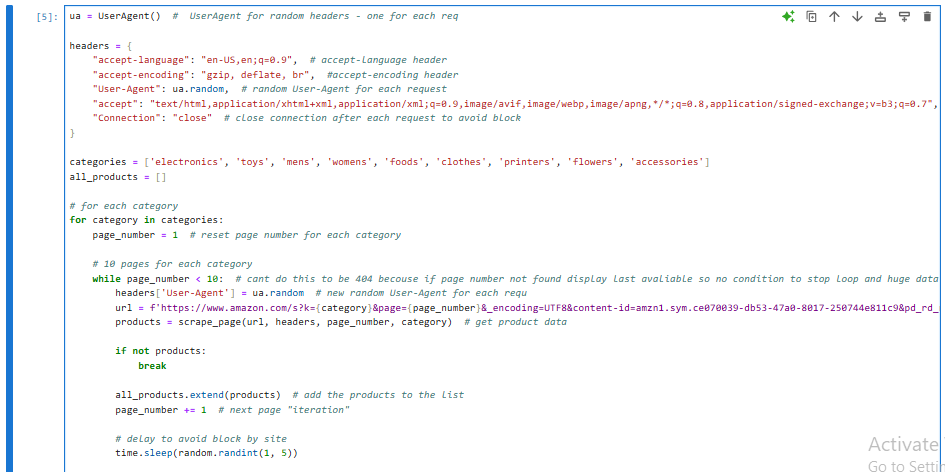
return None

* **element**: The HTML element to search within.
* **search\_dict**: Dictionary specifying the tag and attributes to search for “criteias”.
* **text**: If True, return the text content of the found element; otherwise, return the element itself.

This function ensures that if the element is not found, it handles the AttributeError gracefully and returns None.

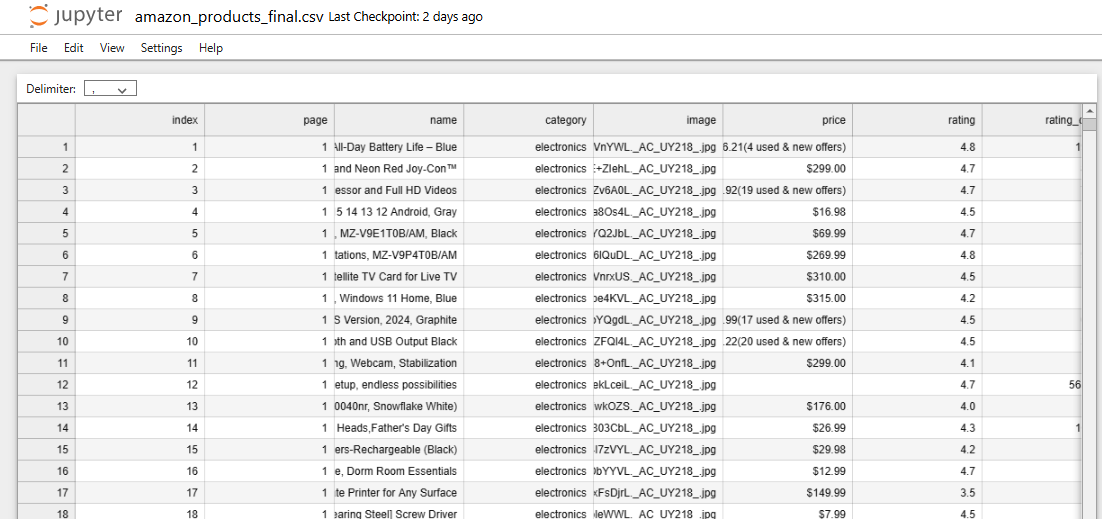
1. Scrape specific page: cut part of soupe object returned from second step “parsed HTML Content” using specific criteria like dive has data-component-type attribute and then pass each extracted result to parse product to extract product details.

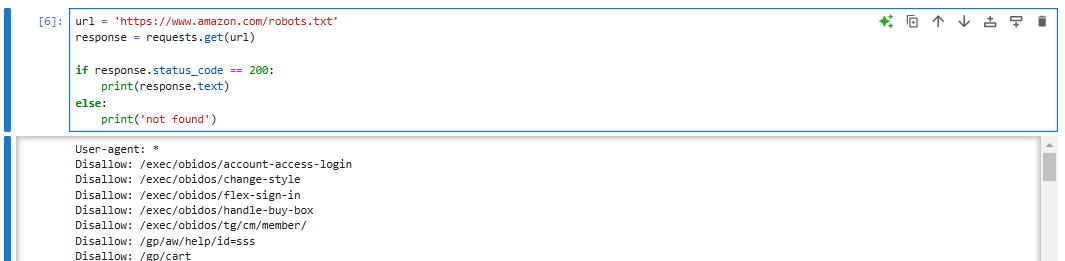


1. Start Scraping Process: for 9 different category in amazon site I loop in 10 pages to extract products in it.. define headers used in request , dynamic URL “dynamic with category and page\_number” and use function defined in fourth step to scrape each page and store all\_products in DataFrame and convert to csv file

1. Finally Ensure by reading csv file info()





Ensure to follow the website's robots.txt policy

In **compliance with the website's `robots.txt` policy**, I ensure that none of the URLs used in my assignemnt are disallowed for scraping. But To prevent being blocked "service unavaliuble" based on many requests, I implemented a delay of 1 to 5 seconds between each request, used different user-agent for each request and close the connection after each request to avoid overloading the server.

**Data Preparation:**

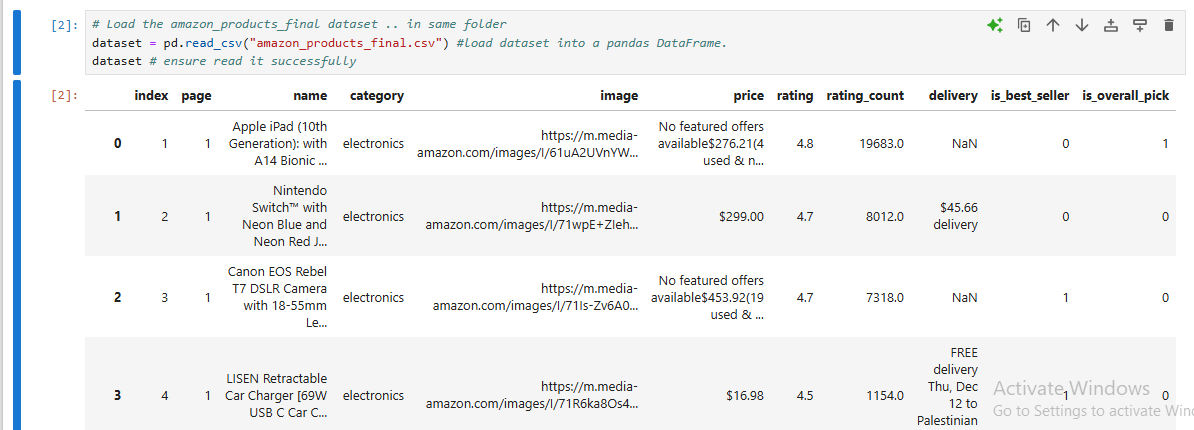
1. **Data Exploration**

**Explore the dataset to understand its structure and content.**

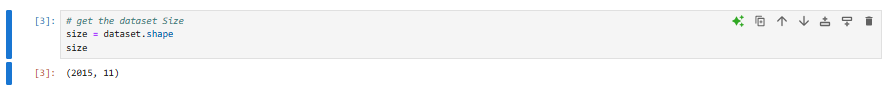
1. **Import Necessary Libraries**



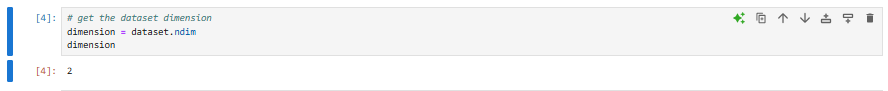
1. **Load dataset and Read it**



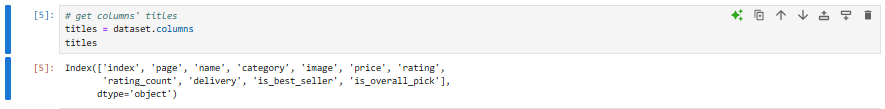
1. **Dataset Size**



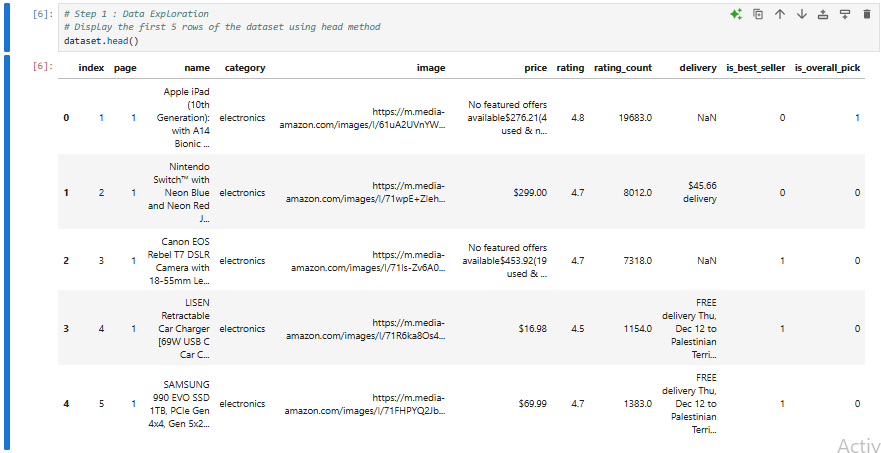
1. **Dataset dimensions**



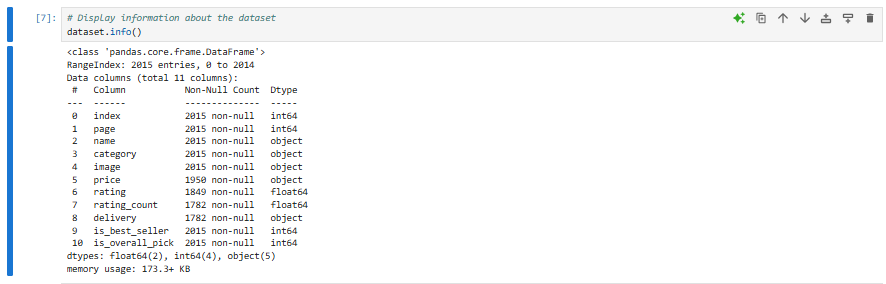
1. **Get dataset columns**



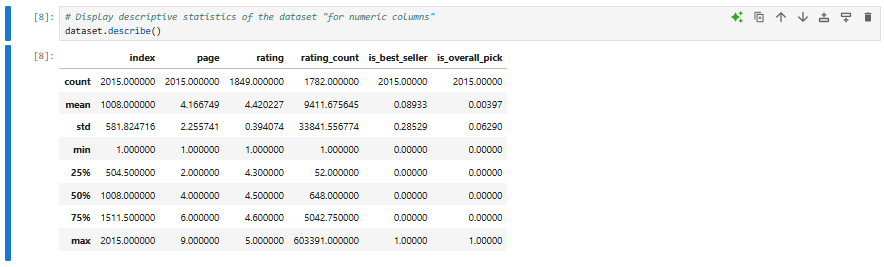
1. **Head of dataset ‘”first rows”**

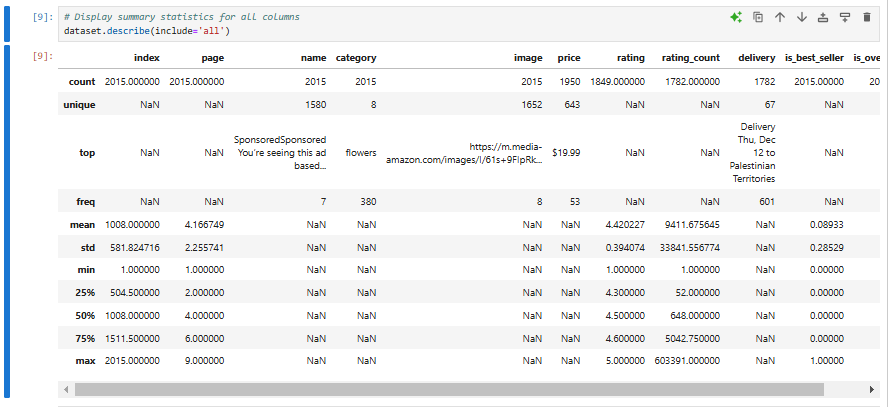


1. **Explore the Structure using info()**

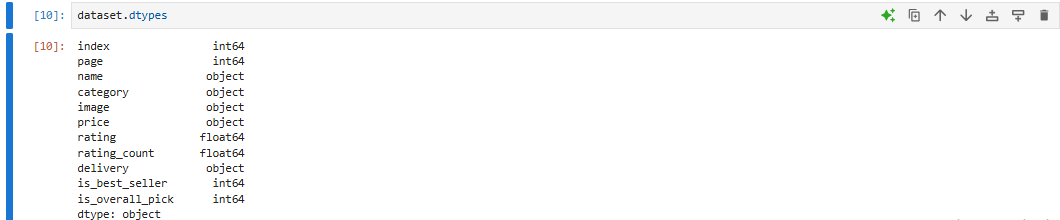


1. **Display descriptive statistic using describe()**





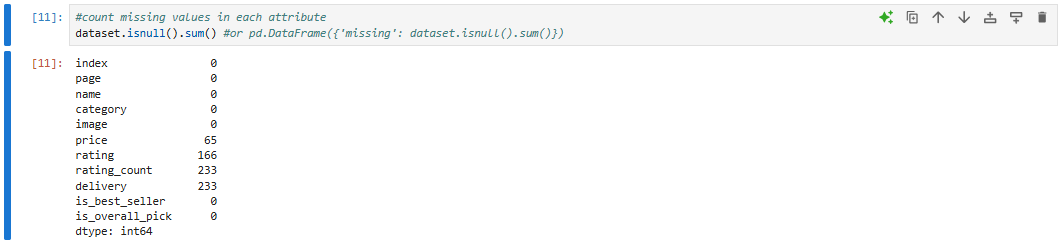
1. **Data Types**



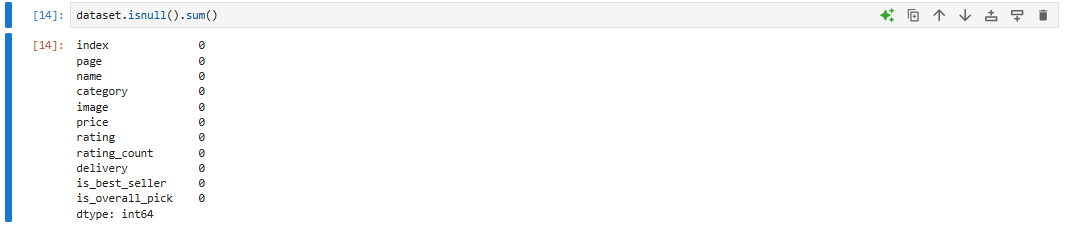
1. **Data Cleaning**
2. **Extract exact price value “preprocessing”**



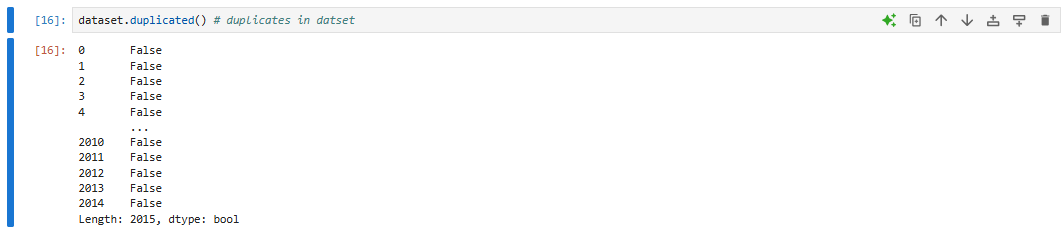
1. **Missing/Incomplete Values**

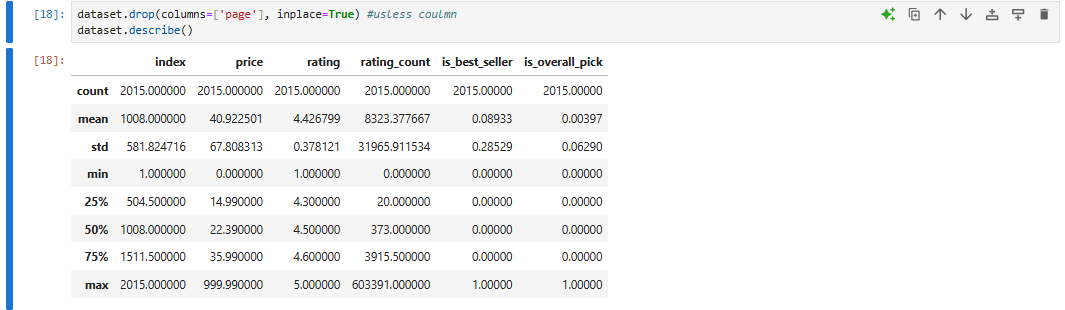


Handel Missing Values



1. **Duplicated Values**

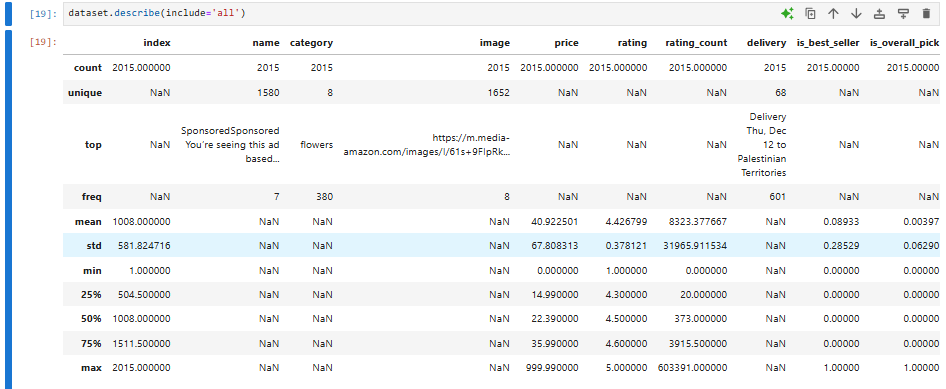


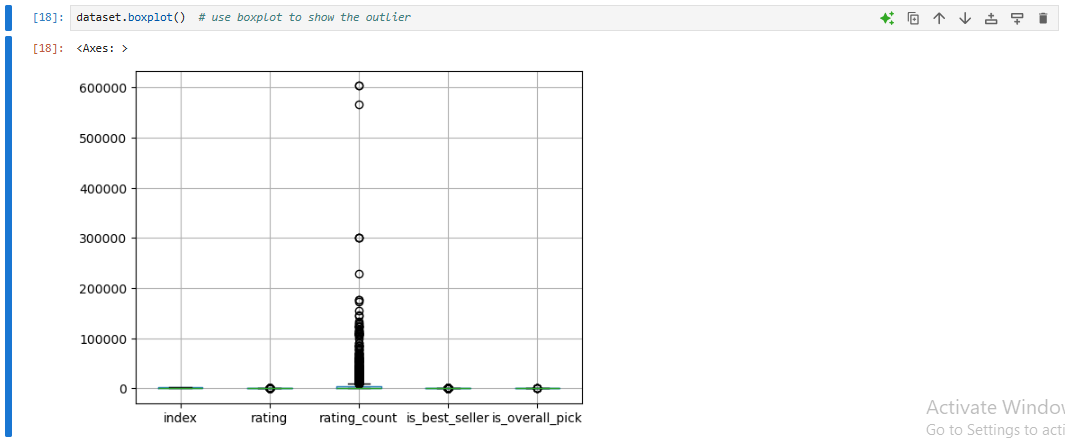


No Duplicated Values

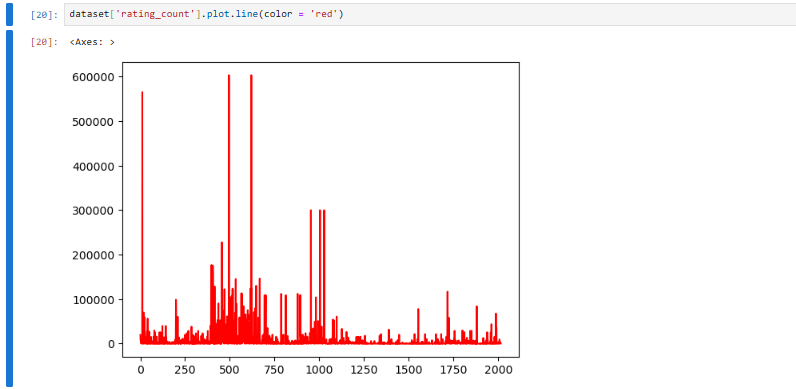
1. **Noisy Values**

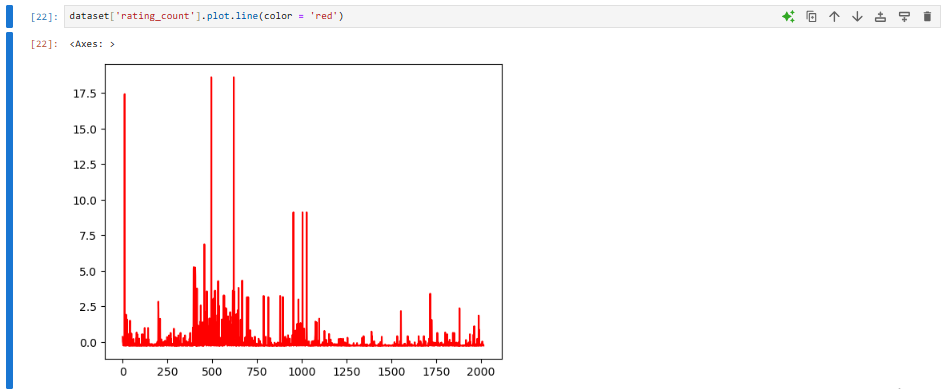
In Image below we notice no error, noisy or outlier



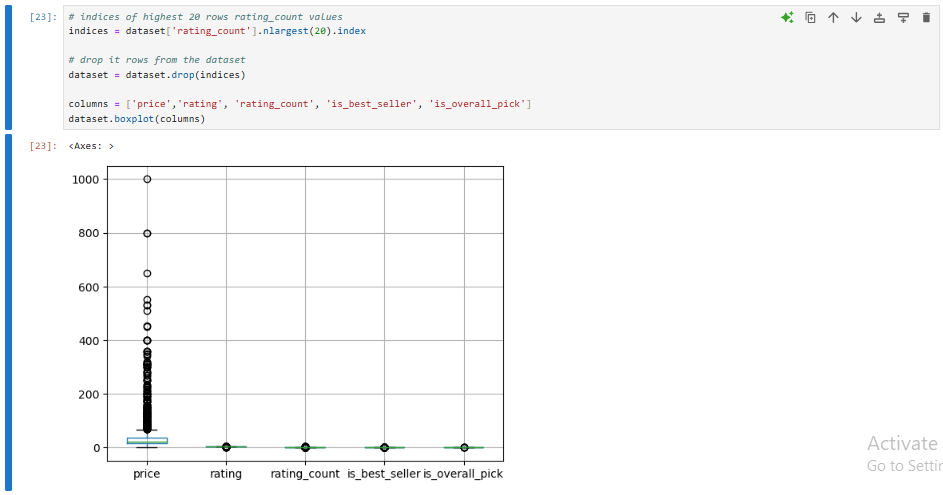
use boxplot to show the outlier

1. **Data Transformation**

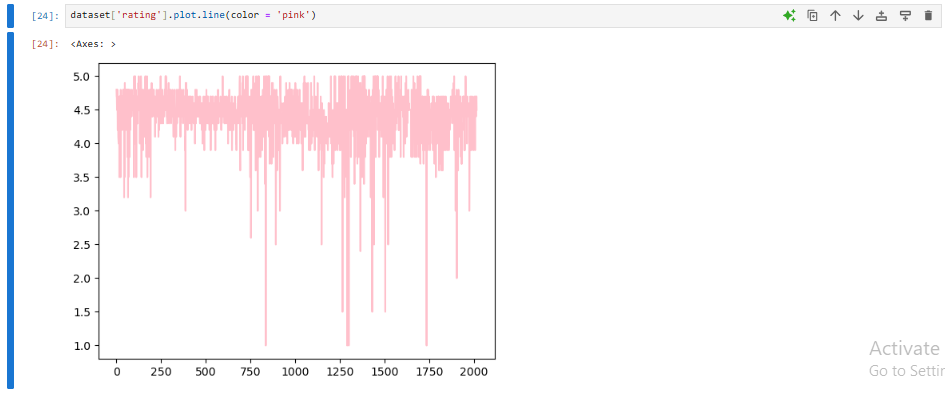
as privious we need do transformation “normalization” in rating account column using StandardScaler() : depend on mean and standard deviation of column



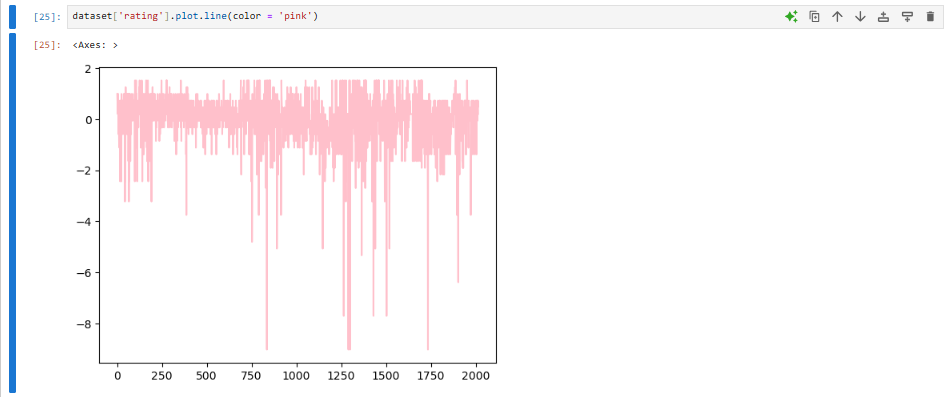
After normalization to rating\_count, I need to drop max values to make dataset more suitable “consider is outlier”

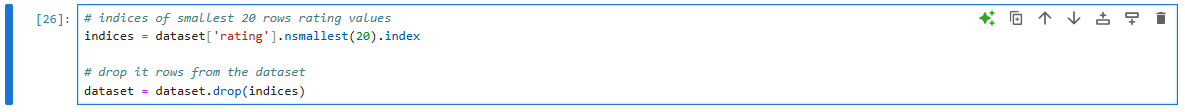


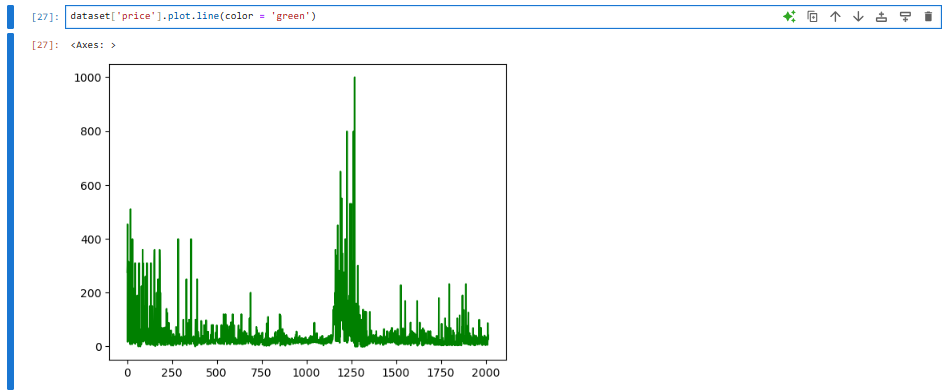
Same normalization and outlier handeling for **rating**, **price** columns



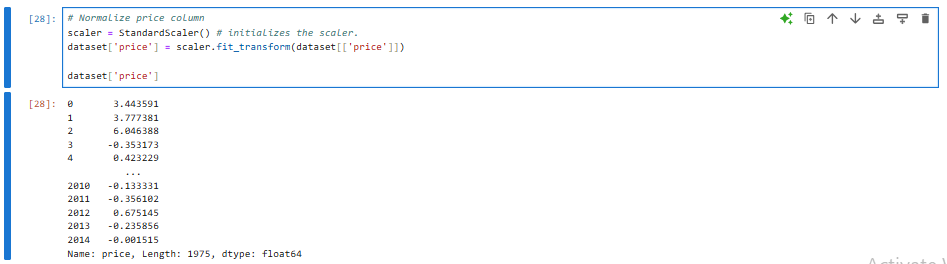


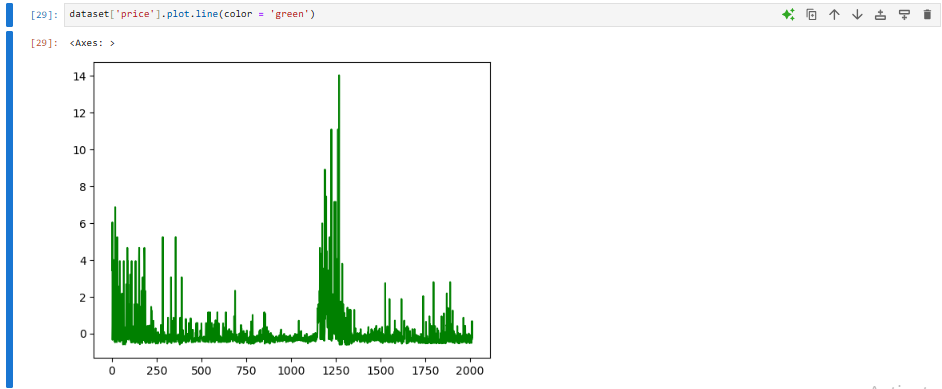


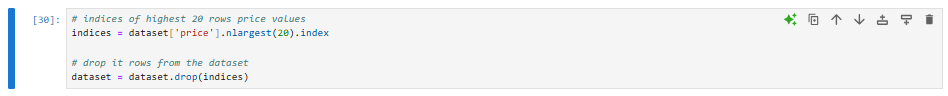


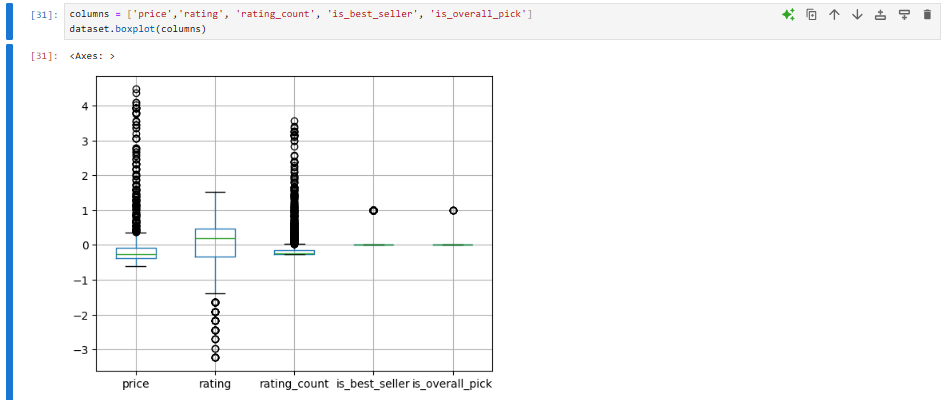


Normalize price







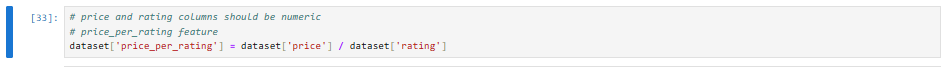


Now data more standardized and more representative so it will enhance model performance and more accurate and efficient analysis

1. **Feature Engineering**

#### **Create a New Feature: price\_per\_rating “cost efficiency of a product based on its rating.”**

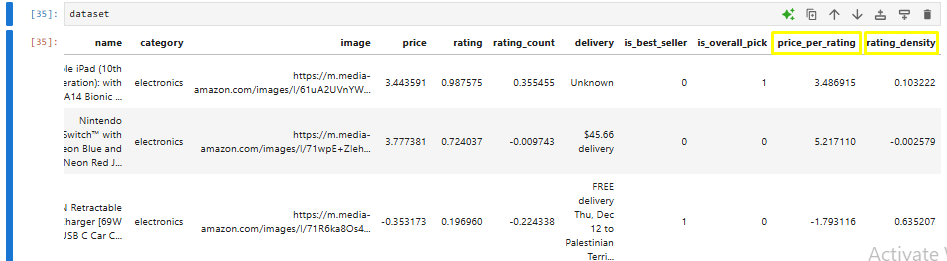
Indicate the value for money, calculated as price / rating.



1. ***Create a New Feature: rating\_density “in compute customer engagement”***

Indicates how densely a product is rated, calculated as rating\_count / price.





Data Exploration after all **Data Preparation**



