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

2. Dataset Collection

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

1. Import Libraries used in scraping task

```
[1]: import requests # use it for send request to page url that i want scrapping data from
     from bs4 import BeautifulSoup  # use it in web scraping process 'parsing HTML content'
     import pandas as pd
     import time # add delays between requests to avoid blocke
     import random  # use it to add delays between requests "to not blocked from site"
     from fake_useragent import UserAgent  # used to to generate random user agents for each request .. same user agent get 503 can send req only one
     import re  # used for regular expressions -  some data shoud extacted from specif expressions
```

2. 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]: # get the soup object "to parse" from the URL
     def get_soup(url, headers):
         for _ in range(5):  # Retry up to 5 times if the request fails
             try:
                 response = requests.get(url, headers=headers)  # sedn GET HTTP request to the URL
                 response.raise_for_status()  # raise an HTTP Error for bad responses 400 and 500 "error has occurred during the process"
                 return BeautifulSoup(response.content, 'html.parser')  # parse HTML content to use it and extract data
             except requests.RequestException as e:
                 # print(f"Failed to retrieve page {url}, error: {e}")  # error message
                 time.sleep(random.randint(1, 5))  # Wait for a random time between 1 to 5 seconds before retrying
         return None  # Return None if the request fails after 5 retries
```

3. 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}.

```
[3]: def parse_product(product, page_number,category):
         def safe_find(element, search_dict, text=False):
             try:
                 found = element.find(**search_dict)  # search for the element using provided criteria "clas , attribute"
                 return found.text.strip() if (found and text) else found  # text if text=True, else return element itself
             except AttributeError:
                 return None  # if element not found

         # extract product name ( product to search in ,  criteria to search based on , test =True)
         name = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-size-medium a-color-base a-text-normal'}}, text=True)
         if not name:
             name = safe_find(product, {'name': 'div', 'attrs': {'data-cy': 'title-recipe'}}, text=True)

         # extract product price
         price = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-offscreen'}}, text=True)
         if not price:
             price = safe_find(product, {'name': 'div', 'attrs': {'class': 'a-row a-size-base a-color-secondary'}}, text=True)

         # extract src of product image
         image_element = safe_find(product, {'name': 'img', 'attrs': {'class': 's-image'}})
         image = image_element['src'] if image_element else None

         # extract product rating
         rating_text = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-icon-alt'}}, text=True)
         rating = rating_text.split()[0] if rating_text else None
```

```python
    # extract rating count
    try:
        rating_count_element = product.find("div", {"class": "s-csa-instrumentation-wrapper"}).find("span", {"aria-label": True})
        rating_count_text = rating_count_element.text.strip() if rating_count_element else None
        rating_count = re.sub(r'[^0-9]', '', rating_count_text) if rating_count_text else None
    except AttributeError:
        rating_count = None

    # extract delivery information
    try:
        delivery_element = product.find("div", {"data-cy": "delivery-recipe"})
        delivery = delivery_element.find("span", {"aria-label": True}).text.strip() if delivery_element else None
    except AttributeError:
        delivery = safe_find(product, {'name': 'span', 'attrs': {'aria-label': True}}, text=True)

    # is the product is a "Best Seller"
    best_seller_element = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-badge-text'}}, text=True)
    is_best_seller = 1 if best_seller_element and "Best Seller" in best_seller_element else 0

    # check if the product is an "Overall Pick" ..  Products highlighted as 'Overall Pick' are: Rated 4+ stars , Purchased often ,Returned infrequently
    overall_pick_element = safe_find(product, {'name': 'span', 'attrs': {'class': 'a-badge-text', 'data-a-badge-color': 'sx-cloud'}}, text=True)
    is_overall_pick = 1 if overall_pick_element and "Overall Pick" in overall_pick_element else 0

    # Return product details
    return {
        'page': page_number,
        'name': name,
```


Helper Function: `safe_find`

```python
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`.

4. 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.

```python
[4]: #scrape each page "url"
def scrape_page(url, headers, page_number, category):
    soup = get_soup(url, headers)  # html to parse it and extract data
    if not soup:
        return []

    # extract product divs that contain the required data-component-type attribute
    product_divs = soup.find_all('div', {"data-component-type": "s-search-result"})
    # Print the product divs to inspect their structure
    #for index, product_div in enumerate(product_divs):
    #    print(f"Product div {index}:\n", product_div.prettify(), "\n **********\n") #get what this piece of code does

    # parse each product in the product divs
    products = [parse_product(product, page_number, category) for product in product_divs]

    return [product for product in products if product]  # filter None values
```

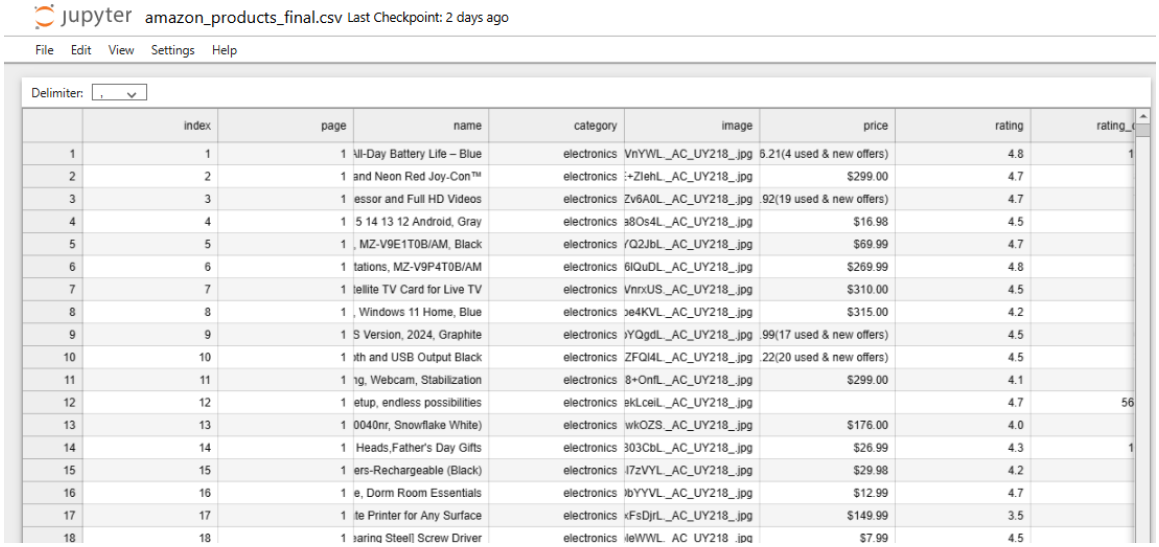
5. Start Scraping Process: for 9 different category in amazon site I loop in 10 pages to extract products in it.. define headers used in request , dynamic URL “dynamic with category and page_number” and use function defined in fourth step to scrape each page and store all_products in DataFrame and convert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

```
[5]: ua = UserAgent()  #  UserAgent for random headers - one for each req

headers = {
    "accept-language": "en-US,en;q=0.9",  # accept-Language header
    "accept-encoding": "gzip, deflate, br",  #accept-encoding header
    "User-Agent": ua.random,  # random User-Agent for each request
    "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7",
    "Connection": "close"  # close connection after each request to avoid block
}

categories = ['electronics', 'toys', 'mens', 'womens', 'foods', 'clothes', 'printers', 'flowers', 'accessories']
all_products = []

# for each category
for category in categories:
    page_number = 1  # reset page number for each category

    # 10 pages for each category
    while page_number < 10:  # cant do this to be 404 becouse if page number not found display last avaliable so no condition to stop loop and huge data
        headers['User-Agent'] = ua.random  # new random User-Agent for each requ
        url = f'https://www.amazon.com/s?k={category}&page={page_number}&_encoding=UTF8&content-id=amzn1.sym.ce070039-db53-47a0-8017-250744e811c9&pd_rd_
        products = scrape_page(url, headers, page_number, category)  # get product data

        if not products:
            break

        all_products.extend(products)  # add the products to the list
        page_number += 1  # next page "iteration"

        # delay to avoid block by site
        time.sleep(random.randint(1, 5))


# convert all products to a DataFrame
df = pd.DataFrame(all_products)
df.index += 1  # index from 1
df.to_csv('amazon_products_final.csv', index_label='index')  # Save the DataFrame to a csv file
```

6. Finally Ensure by reading csv file info()

```
[6]: data = pd.read_csv('amazon_products_final.csv')
data.info() #info about scrapped data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2015 entries, 0 to 2014
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   index            2015 non-null   int64
 1   page             2015 non-null   int64
 2   name             2015 non-null   object
 3   category         2015 non-null   object
 4   image            2015 non-null   object
 5   price            1950 non-null   object
 6   rating           1849 non-null   float64
 7   rating_count     1782 non-null   float64
 8   delivery         1782 non-null   object
 9   is_best_seller   2015 non-null   int64
 10  is_overall_pick  2015 non-null   int64
dtypes: float64(2), int64(4), object(5)
memory usage: 173.3+ KB
```



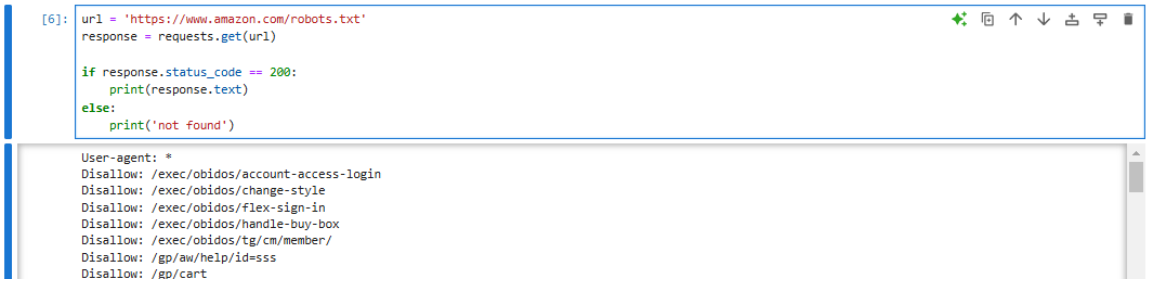
jupyter amazon_products_final.csv Last Checkpoint: 2 days ago

File Edit View Settings Help

Delimiter: ,

	index	page	name	category	image	price	rating	rating_
1	1	1	ll-Day Battery Life – Blue	electronics	VnYWL._AC_UY218_.jpg	6.21(4 used & new offers)	4.8	1
2	2	1	and Neon Red Joy-Con™	electronics	:+ZlehL._AC_UY218_.jpg	$299.00	4.7	
3	3	1	essor and Full HD Videos	electronics	Zv6A0L._AC_UY218_.jpg	.92(19 used & new offers)	4.7	
4	4	1	5 14 13 12 Android, Gray	electronics	a8Os4L._AC_UY218_.jpg	$16.98	4.5	
5	5	1	, MZ-V9E1T0B/AM, Black	electronics	/Q2JbL._AC_UY218_.jpg	$69.99	4.7	
6	6	1	tations, MZ-V9P4T0B/AM	electronics	6lQuDL._AC_UY218_.jpg	$269.99	4.8	
7	7	1	ellite TV Card for Live TV	electronics	VnrxUS._AC_UY218_.jpg	$310.00	4.5	
8	8	1	, Windows 11 Home, Blue	electronics	e4KVL._AC_UY218_.jpg	$315.00	4.2	
9	9	1	S Version, 2024, Graphite	electronics	YQgdL._AC_UY218_.jpg	.99(17 used & new offers)	4.5	
10	10	1	th and USB Output Black	electronics	ZFQl4L._AC_UY218_.jpg	.22(20 used & new offers)	4.5	
11	11	1	g, Webcam, Stabilization	electronics	8+OnfL._AC_UY218_.jpg	$299.00	4.1	
12	12	1	etup, endless possibilities	electronics	ekLceiL._AC_UY218_.jpg		4.7	56
13	13	1	0040nr, Snowflake White)	electronics	wkOZS._AC_UY218_.jpg	$176.00	4.0	
14	14	1	Heads,Father's Day Gifts	electronics	303CbL._AC_UY218_.jpg	$26.99	4.3	1
15	15	1	ers-Rechargeable (Black)	electronics	I7zVYL._AC_UY218_.jpg	$29.98	4.2	
16	16	1	e, Dorm Room Essentials	electronics	bYYVL._AC_UY218_.jpg	$12.99	4.7	
17	17	1	te Printer for Any Surface	electronics	xFsDjrL._AC_UY218_.jpg	$149.99	3.5	
18	18	1	aring Steell Screw Driver	electronics	leWWL._AC_UY218_.jpg	$7.99	4.5	

Ensure to follow the website's robots.txt policy



```
[6]: url = 'https://www.amazon.com/robots.txt'
response = requests.get(url)

if response.status_code == 200:
    print(response.text)
else:
    print('not found')
```

```
User-agent: *
Disallow: /exec/obidos/account-access-login
Disallow: /exec/obidos/change-style
Disallow: /exec/obidos/flex-sign-in
Disallow: /exec/obidos/handle-buy-box
Disallow: /exec/obidos/tg/cm/member/
Disallow: /gp/aw/help/id=sss
Disallow: /gp/cart
```

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]: # Import libraries used in data preparation
import pandas as pd
```

2. Load dataset and Read it

```
[2]: # Load the heart dataset .. in same folder
dataset = pd.read_csv("amazon_products_final.csv") #Load dataset into a pandas DataFrame.
dataset # ensure read it successfully
```

[2]:

	index	page	name	category	image	price	rating	rating_count	delivery	is_best_seller	is_overall_pick
0	1	1	Apple iPad (10th Generation): with A14 Bionic ...	electronics	https://m.media-amazon.com/images/I/61uA2UVnYW...	No featured offers available$276.21(4 used & n...	4.8	19683.0	NaN	0	1
1	2	1	Nintendo Switch™ with Neon Blue and Neon Red J...	electronics	https://m.media-amazon.com/images/I/71wpE+ZIeh...	$299.00	4.7	8012.0	$45.66 delivery	0	0
2	3	1	Canon EOS Rebel T7 DSLR Camera with 18-55mm Le...	electronics	https://m.media-amazon.com/images/I/71Is-Zv6A0...	No featured offers available$453.92(19 used & ...	4.7	7318.0	NaN	1	0
3	4	1	LISEN Retractable Car Charger [69W USB C Car C...	electronics	https://m.media-amazon.com/images/I/71R6ka8Os4...	$16.98	4.5	1154.0	FREE delivery Thu, Dec 12 to Palestinian Terri...	1	0

3. Dataset Size

```
[3]: # get the dataset Size
size = dataset.shape
size
```

```
[3]: (2015, 11)
```

4. Dataset dimensions

```
[4]: # get the dataset dimension
dimension = dataset.ndim
dimension
```

```
[4]: 2
```

5. Get dataset columns

```
[5]: # get columns' titles
titles = dataset.columns
titles
```

```
[5]: Index(['index', 'page', 'name', 'category', 'image', 'price', 'rating',
       'rating_count', 'delivery', 'is_best_seller', 'is_overall_pick'],
      dtype='object')
```

6. Head of dataset ‘”first rows”

```
[6]: # Step 1 : Data Exploration
# Display the first 5 rows of the dataset using head method
dataset.head()
```

[6]:

	index	page	name	category	image	price	rating	rating_count	delivery	is_best_seller	is_overall_pick
0	1	1	Apple iPad (10th Generation): with A14 Bionic ...	electronics	https://m.media-amazon.com/images/I/61uA2UVnYW...	No featured offers available$276.21(4 used & n...	4.8	19683.0	NaN	0	1
1	2	1	Nintendo Switch™ with Neon Blue and Neon Red J...	electronics	https://m.media-amazon.com/images/I/71wpE+Zleh...	$299.00	4.7	8012.0	$45.66 delivery	0	0
2	3	1	Canon EOS Rebel T7 DSLR Camera with 18-55mm Le...	electronics	https://m.media-amazon.com/images/I/71ls-Zv6A0...	No featured offers available$453.92(19 used & ...	4.7	7318.0	NaN	1	0
3	4	1	LISEN Retractable Car Charger [69W USB C Car C...	electronics	https://m.media-amazon.com/images/I/71R6ka8Os4...	$16.98	4.5	1154.0	FREE delivery Thu, Dec 12 to Palestinian Terri...	1	0
4	5	1	SAMSUNG 990 EVO SSD 1TB, PCIe Gen 4x4, Gen 5x2...	electronics	https://m.media-amazon.com/images/I/71FHPYQ2Jb...	$69.99	4.7	1383.0	FREE delivery Thu, Dec 12 to Palestinian Terri...	1	0

Activ

7. Explore the Structure using info()

```
[7]: # Display information about the dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2015 entries, 0 to 2014
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   index            2015 non-null   int64
 1   page             2015 non-null   int64
 2   name             2015 non-null   object
 3   category         2015 non-null   object
 4   image            2015 non-null   object
 5   price            1950 non-null   object
 6   rating           1849 non-null   float64
 7   rating_count     1782 non-null   float64
 8   delivery         1782 non-null   object
 9   is_best_seller   2015 non-null   int64
 10  is_overall_pick  2015 non-null   int64
dtypes: float64(2), int64(4), object(5)
memory usage: 173.3+ KB
```

8. Display descriptive statistic using describe()

```
[8]: # Display descriptive statistics of the dataset "for numeric columns"
dataset.describe()
```

[8]:

	index	page	rating	rating_count	is_best_seller	is_overall_pick
count	2015.000000	2015.000000	1849.000000	1782.000000	2015.00000	2015.00000
mean	1008.000000	4.166749	4.420227	9411.675645	0.08933	0.00397
std	581.824716	2.255741	0.394074	33841.556774	0.28529	0.06290
min	1.000000	1.000000	1.000000	1.000000	0.00000	0.00000
25%	504.500000	2.000000	4.300000	52.000000	0.00000	0.00000
50%	1008.000000	4.000000	4.500000	648.000000	0.00000	0.00000
75%	1511.500000	6.000000	4.600000	5042.750000	0.00000	0.00000
max	2015.000000	9.000000	5.000000	603391.000000	1.00000	1.00000

```
[9]: # Display summary statistics for all columns
dataset.describe(include='all')
```

[9]:

	index	page	name	category	image	price	rating	rating_count	delivery	is_best_seller	is_ove
count	2015.000000	2015.000000	2015	2015	2015	1950	1849.000000	1782.000000	1782	2015.00000	20
unique	NaN	NaN	1580	8	1652	643	NaN	NaN	67	NaN	
top	NaN	NaN	SponsoredSponsored You're seeing this ad based...	flowers	https://m.media-amazon.com/images/I/61s+9FIpRk...	$19.99	NaN	NaN	Delivery Thu, Dec 12 to Palestinian Territories	NaN	
freq	NaN	NaN	7	380	8	53	NaN	NaN	601	NaN	
mean	1008.000000	4.166749	NaN	NaN	NaN	NaN	4.420227	9411.675645	NaN	0.08933	
std	581.824716	2.255741	NaN	NaN	NaN	NaN	0.394074	33841.556774	NaN	0.28529	
min	1.000000	1.000000	NaN	NaN	NaN	NaN	1.000000	1.000000	NaN	0.00000	
25%	504.500000	2.000000	NaN	NaN	NaN	NaN	4.300000	52.000000	NaN	0.00000	
50%	1008.000000	4.000000	NaN	NaN	NaN	NaN	4.500000	648.000000	NaN	0.00000	
75%	1511.500000	6.000000	NaN	NaN	NaN	NaN	4.600000	5042.750000	NaN	0.00000	
max	2015.000000	9.000000	NaN	NaN	NaN	NaN	5.000000	603391.000000	NaN	1.00000	

9. Data Types

```
[10]: dataset.dtypes
```

[10]:

```
index              int64
page               int64
name              object
category          object
image             object
price             object
rating           float64
rating_count     float64
delivery          object
is_best_seller     int64
is_overall_pick    int64
dtype: object
```

2. Data Cleaning

1. Extract exact price value “preprocessing”

[12]:
```
import numpy as np
import re

# Define a function to extract the numeric price from the string
def extract_price(value):
    if isinstance(value, str):
        match = re.search(r'(\d+\.\d{2})', value)
        return float(match.group(1)) if match else np.nan
    else:
        return np.nan

dataset['price'] = dataset['price'].apply(extract_price)

# processed price column
dataset['price']
```

[12]:
```
0       276.21
1       299.00
2       453.92
3        16.98
4        69.99
         ...
2010     31.99
2011     16.78
2012     87.19
2013     24.99
2014     40.99
Name: price, Length: 2015, dtype: float64
```


2. Missing/Incomplete Values

[11]:
```
#count missing values in each attribute
dataset.isnull().sum() #or pd.DataFrame({'missing': dataset.isnull().sum()})
```

[11]:
```
index              0
page               0
name               0
category           0
image              0
price             65
rating           166
rating_count     233
delivery         233
is_best_seller     0
is_overall_pick    0
dtype: int64
```

Handel Missing Values

[13]:
```
# Handle missing values
dataset['price'] = dataset['price'].fillna(dataset['price'].mode()[0])# most frequent value
dataset['rating'] = dataset['rating'].fillna(dataset['rating'].median()) #median value
dataset['rating_count'] = dataset['rating_count'].fillna(0) # with 0
dataset['delivery'] = dataset['delivery'].fillna('Unknown') # with 'Unknown'
```

[14]:
```
dataset.isnull().sum()
```

[14]:
```
index            0
page             0
name             0
category         0
image            0
price            0
rating           0
rating_count     0
delivery         0
is_best_seller   0
is_overall_pick  0
dtype: int64
```

3. Duplicated Values

```
[16]: dataset.duplicated() # duplicates in datset
```

```
[16]: 0       False
      1       False
      2       False
      3       False
      4       False
              ...
      2010    False
      2011    False
      2012    False
      2013    False
      2014    False
      Length: 2015, dtype: bool
```

```
[18]: dataset.drop(columns=['page'], inplace=True) #usless coulmn
      dataset.describe()
```

[18]:

	index	price	rating	rating_count	is_best_seller	is_overall_pick
count	2015.000000	2015.000000	2015.000000	2015.000000	2015.00000	2015.00000
mean	1008.000000	40.922501	4.426799	8323.377667	0.08933	0.00397
std	581.824716	67.808313	0.378121	31965.911534	0.28529	0.06290
min	1.000000	0.000000	1.000000	0.000000	0.00000	0.00000
25%	504.500000	14.990000	4.300000	20.000000	0.00000	0.00000
50%	1008.000000	22.390000	4.500000	373.000000	0.00000	0.00000
75%	1511.500000	35.990000	4.600000	3915.500000	0.00000	0.00000
max	2015.000000	999.990000	5.000000	603391.000000	1.00000	1.00000

No Duplicated Values

4. Noisy Values

In Image below we notice no error, noisy or outlier

```
[19]: dataset.describe(include='all')
```

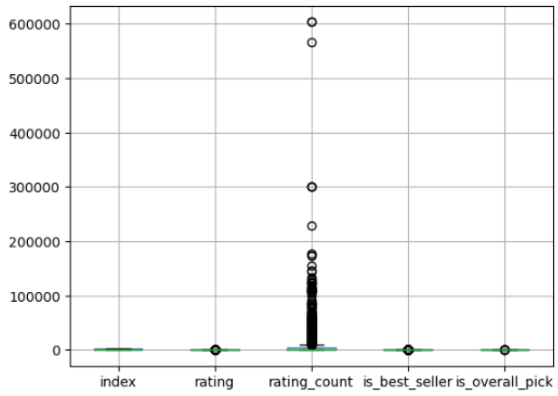
[19]:

	index	name	category	image	price	rating	rating_count	delivery	is_best_seller	is_overall_pick
count	2015.000000	2015	2015	2015	2015.000000	2015.000000	2015.000000	2015	2015.00000	2015.00000
unique	NaN	1580	8	1652	NaN	NaN	NaN	68	NaN	NaN
top	NaN	SponsoredSponsored You're seeing this ad based...	flowers	https://m.media-amazon.com/images/I/61s+9FIpRk...	NaN	NaN	NaN	Delivery Thu, Dec 12 to Palestinian Territories	NaN	NaN
freq	NaN	7	380	8	NaN	NaN	NaN	601	NaN	NaN
mean	1008.000000	NaN	NaN	NaN	40.922501	4.426799	8323.377667	NaN	0.08933	0.00397
std	581.824716	NaN	NaN	NaN	67.808313	0.378121	31965.911534	NaN	0.28529	0.06290
min	1.000000	NaN	NaN	NaN	0.000000	1.000000	0.000000	NaN	0.00000	0.00000
25%	504.500000	NaN	NaN	NaN	14.990000	4.300000	20.000000	NaN	0.00000	0.00000
50%	1008.000000	NaN	NaN	NaN	22.390000	4.500000	373.000000	NaN	0.00000	0.00000
75%	1511.500000	NaN	NaN	NaN	35.990000	4.600000	3915.500000	NaN	0.00000	0.00000
max	2015.000000	NaN	NaN	NaN	999.990000	5.000000	603391.000000	NaN	1.00000	1.00000

use boxplot to show the outlier

```
[18]: dataset.boxplot()  # use boxplot to show the outlier
```

```
[18]: <Axes: >
```

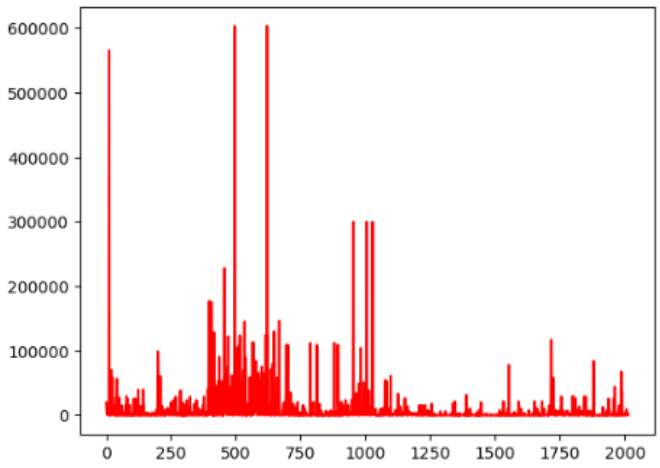


3. Data Transformation

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

```
[20]: dataset['rating_count'].plot.line(color = 'red')
```

```
[20]: <Axes: >
```

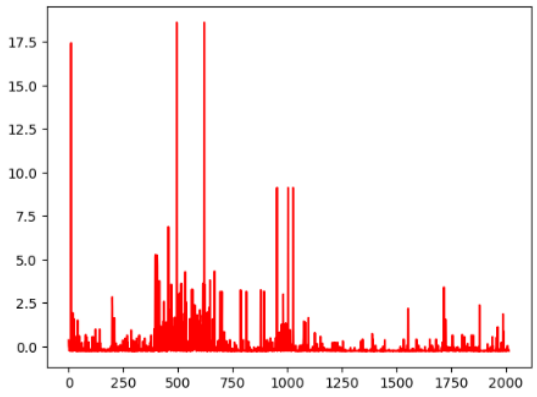



```
[21]: from sklearn.preprocessing import StandardScaler

# Normalize rating_count column
scaler = StandardScaler() # initializes the scaler.
dataset['rating_count'] = scaler.fit_transform(dataset[['rating_count']]) #The fit_transform method is applied to the rating_count column.
#fit: calculates the mean and standard deviation of rating_count
#transform:  applies the transformation to standardize the data

dataset['rating_count']
```

```
[21]: 0       0.355455
      1      -0.009743
      2      -0.031459
      3      -0.224338
      4      -0.217172
                ...
      2010   -0.199180
      2011   -0.151492
      2012   -0.231034
      2013   -0.260135
      2014   -0.260448
      Name: rating_count, Length: 2015, dtype: float64
```

```
[22]: dataset['rating_count'].plot.line(color = 'red')
```

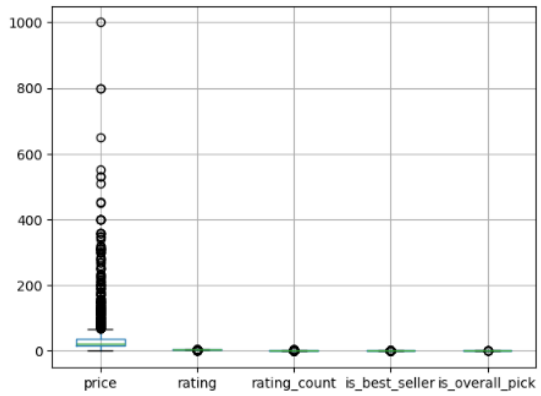
```
[22]: <Axes: >
```



After normalization to rating_count, I need to drop max values to make dataset more suitable "consider is outlier"

```
[23]: # indices of highest 20 rows rating_count values
indices = dataset['rating_count'].nlargest(20).index

# drop it rows from the dataset
dataset = dataset.drop(indices)

columns = ['price','rating', 'rating_count', 'is_best_seller', 'is_overall_pick']
dataset.boxplot(columns)
```

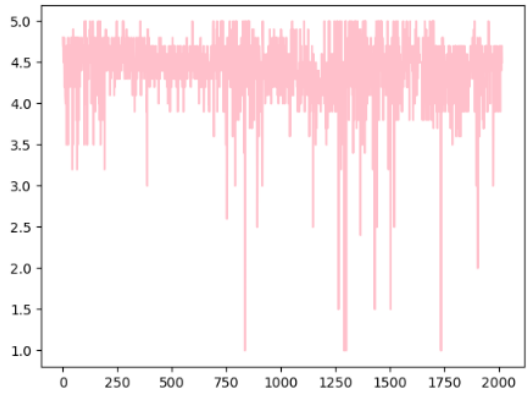
```
[23]: <Axes: >
```



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

```
[24]: dataset['rating'].plot.line(color = 'pink')
```

```
[24]: <Axes: >
```




```
[24]: # Normalize price column
scaler = StandardScaler() # initializes the scaler.
dataset['rating'] = scaler.fit_transform(dataset[['rating']])

dataset['rating']
```

```
[24]: 0       0.987575
1       0.724037
2       0.724037
3       0.196960
4       0.724037
          ...
2010   -0.066578
2011    0.724037
2012    0.460499
2013    0.724037
2014    0.196960
Name: rating, Length: 1995, dtype: float64
```

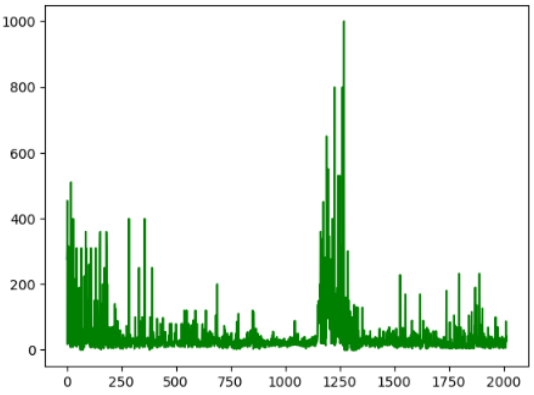
[25]:
```
dataset['rating'].plot.line(color = 'pink')
```

[25]: <Axes: >

[26]:
```
# indices of smallest 20 rows rating values
indices = dataset['rating'].nsmallest(20).index

# drop it rows from the dataset
dataset = dataset.drop(indices)
```

[27]:
```
dataset['price'].plot.line(color = 'green')
```

[27]: <Axes: >



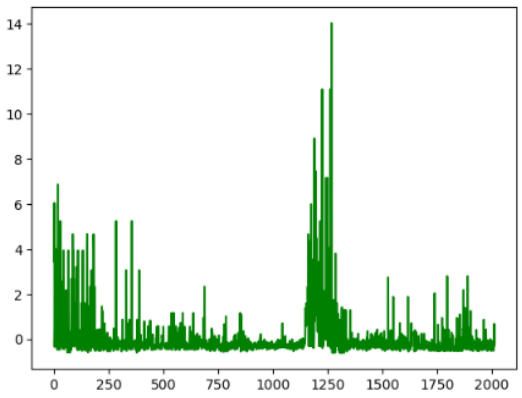
Normalize price

[28]:
```
# Normalize price column
scaler = StandardScaler() # initializes the scaler.
dataset['price'] = scaler.fit_transform(dataset[['price']])

dataset['price']
```

[28]:
```
0       3.443591
1       3.777381
2       6.046388
3      -0.353173
4       0.423229
          ...
2010   -0.133331
2011   -0.356102
2012    0.675145
2013   -0.235856
2014   -0.001515
Name: price, Length: 1975, dtype: float64
```

```
[29]: dataset['price'].plot.line(color = 'green')
```

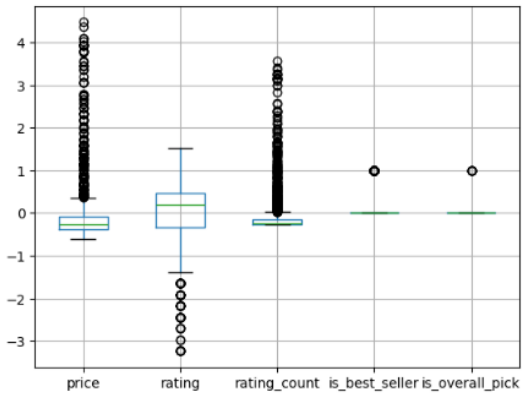
```
[29]: <Axes: >
```




```
[30]: # indices of highest 20 rows price values
indices = dataset['price'].nlargest(20).index

# drop it rows from the dataset
dataset = dataset.drop(indices)
```

```
[31]: columns = ['price','rating', 'rating_count', 'is_best_seller', 'is_overall_pick']
dataset.boxplot(columns)
```

```
[31]: <Axes: >
```



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

4. Feature Engineering

1. ***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.

```
[33]: # price and rating columns should be numeric
      # price_per_rating feature
      dataset['price_per_rating'] = dataset['price'] / dataset['rating']
```

2. ***Create a New Feature: rating_density "in compute customer engagement"***

Indicates how densely a product is rated, calculated as rating_count / price.

```
[34]: # rating_density feature
      dataset['rating_density'] = dataset['rating_count'] / dataset['price']
```

```
[35]: dataset
```

[35]:

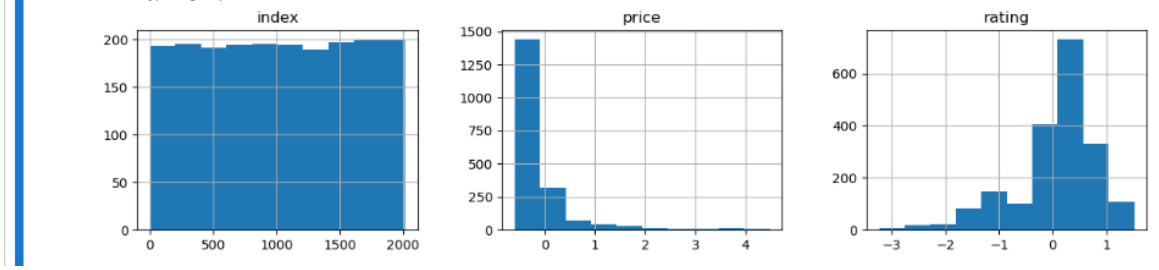
name	category	image	price	rating	rating_count	delivery	is_best_seller	is_overall_pick	price_per_rating	rating_density
le iPad (10th eration): with A14 Bionic ...	electronics	https://m.media-amazon.com/images/I/61uA2UVnYW...	3.443591	0.987575	0.355455	Unknown	0	1	3.486915	0.103222
Nintendo Switch™ with eon Blue and Neon Red J...	electronics	https://m.media-amazon.com/images/I/71wpE+Zleh...	3.777381	0.724037	-0.009743	$45.66 delivery	0	0	5.217110	-0.002579
l Retractable harger [69W USB C Car C...	electronics	https://m.media-amazon.com/images/I/71R6ka8Os4...	-0.353173	0.196960	-0.224338	FREE delivery Thu, Dec 12 to Palestinian Terri...	1	0	-1.793116	0.635207

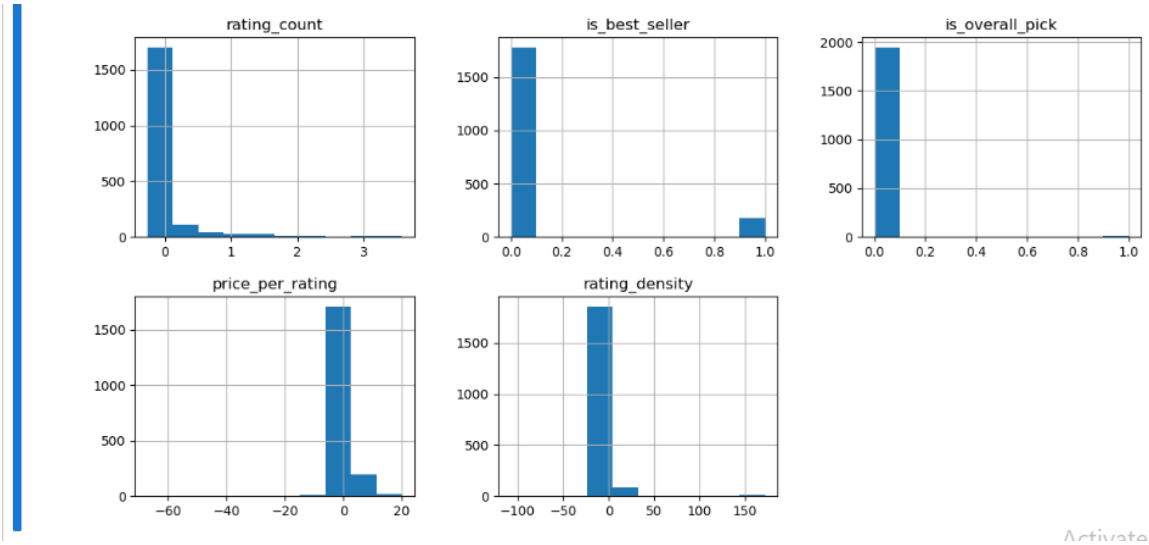
Activate

Data Exploration after all **Data Preparation**

```
[74]: # Generate histograms for each column
      dataset.hist(figsize=(14, 10))
```

```
[74]: array([[<Axes: title={'center': 'index'}>,
              <Axes: title={'center': 'price'}>,
              <Axes: title={'center': 'rating'}>],
             [<Axes: title={'center': 'rating_count'}>,
              <Axes: title={'center': 'is_best_seller'}>,
              <Axes: title={'center': 'is_overall_pick'}>],
             [<Axes: title={'center': 'price_per_rating'}>,
              <Axes: title={'center': 'rating_density'}>, <Axes: >]],
            dtype=object)
```




rating_count
is_best_seller
is_overall_pick
price_per_rating
rating_density