Explanation

Step 1: import the necessary module. Here I imported only one module called pandas which is a powerful tool for faster data analysis, data cleaning, and data pre-processing.

Step 2: a quick EDA is performed to understand the pattern of the data.

TASK 1: Products without prices

Step 1: for this task, I only used the columns "price_string" and "product_type". there may be chance of having a value in price_string column for a nan value in product_type column, so deleting all the rows having nan value in product_type column is better (to find out the accurate count of Products with prices in the future).

Step 2: fetch all the rows having zero dollar in price_string column. The products in the product type column will be the products without price.

```
df1[df1.price_string ==  "$0.00"]
```

	price_string	product_type
23	$0.00	QmVkc2hlZXRz
56	$0.00	Q2hlZWsgbWFrZSB1cA
250	$0.00	TGlwIEJhbG0
267	$0.00	RXIlIFNoYWRvdw
397	$0.00	RmFjZSBtYWtlIHVw
...	...	...
13729016	$0.00	RHV2ZXQgY292ZXJz
13729128	$0.00	TGlwIEJhbG0
13729425	$0.00	TGlwc3RpY2svIExpcCBjb2xvcg
13729603	$0.00	RmFjZSBtYWtlIHVw
13729729	$0.00	Q2hlZWsgbWFrZSB1cA

240000 rows × 2 columns

TASK 2: Count of products without prices and with prices in each Product Type, Category, Level 1

Step 1: similar to task 1, the count of products that have no price will be the no.of rows in above table.

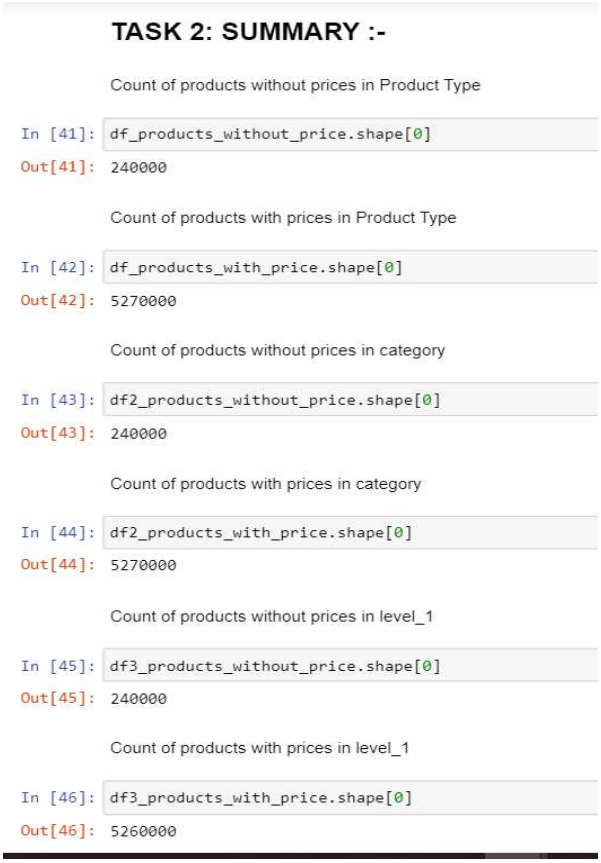
Step 2: to find the count of products that have a valid price, fetch all the rows that don't have zero dollar in price_string column.

```
df_products_with_price = df_cleaned_price_string[df_cleaned_price_string["price_string"] !=  "$0.00"] #selecting only the rows th
```

```
df_products_with_price.head(10)
```

	price_string	product_type
2	$19.95	R3VtbWlcyB2aXRhbWlucyBhbmQgbWluZXJhbHMgZm9yIG...
3	$92.00	U2VydW1z
4	11.50	RWF0aW5nIFV0ZW5zaWxzL0N1dGxlcnk
6	$24.99	TW9wcyBhbmQgYnJvb21z
7	$148.00	V29tZW5zIFBhbnRz
8	$89	V29tZW5zIFBhbnRz
9	$14.95	Um9sbGluZyBQaW4
10	$55.00	Q2FyZHN0b2Nrcw
11	32.99	Rm9vZCBTdG9yYWdl
12	$92.00	U2VydW1z

Step 3: repeat the step 1 and step 2 for category and level_1 column. Final result will be:



TASK 2: SUMMARY :-

Count of products without prices in Product Type

```
In [41]: df_products_without_price.shape[0]
Out[41]: 240000
```

Count of products with prices in Product Type

```
In [42]: df_products_with_price.shape[0]
Out[42]: 5270000
```

Count of products without prices in category

```
In [43]: df2_products_without_price.shape[0]
Out[43]: 240000
```

Count of products with prices in category

```
In [44]: df2_products_with_price.shape[0]
Out[44]: 5270000
```

Count of products without prices in level_1

```
In [45]: df3_products_without_price.shape[0]
Out[45]: 240000
```

Count of products with prices in level_1

```
In [46]: df3_products_with_price.shape[0]
Out[46]: 5260000
```

TASK 3: Correct Product Prices in the correct format (eg: $56) wherever possible and separate them into currency and value columns.

Step 1: for this task, I only used the column “price_string”. Then I cleaned the data by dropping all the NAN values in that column.

Step 2: derived a new column called value from price_string. Removed the dollar sign from all the rows and converted the values into float.

Step 3: added a column called “currency” which have value Dollar.

Step 4: derived “cleaned_price_string” column from value column. edited Product Prices in the correct format (eg: $56) wherever possible.

```
cdf["currency"] = "Dollar"
cdf["value"] = cdf["price_string"].replace("[$,]", "", regex=True).astype(float)
cdf["cleaned_price_string"] = cdf['value'].apply( lambda x : '$' + str(x))
cdf.head(10)
```

	price_string	currency	value	cleaned_price_string
2	$19.95	Dollar	19.95	$19.95
3	$92.00	Dollar	92.00	$92.0
4	11.50	Dollar	11.50	$11.5
6	$24.99	Dollar	24.99	$24.99
7	$148.00	Dollar	148.00	$148.0
8	$89	Dollar	89.00	$89.0
9	$14.95	Dollar	14.95	$14.95
10	$55.00	Dollar	55.00	$55.0
11	32.99	Dollar	32.99	$32.99
12	$92.00	Dollar	92.00	$92.0

TASK 4: List out the categories with average price of product.

Step 1: for this task, I only used the columns “price_string” and “categrory”. Since we are going to calculate the average price, I dropped all the NAN values in price_string column.

Step 2: removed the dollar sign from the rows that have dollar sign in front and converted all the values into float.

```
nwdf["price_string"] = nwdf["price_string"].replace("[$,]", "", regex=True).astype(float)
```

```
nwdf.head(10)
```

	price_string	category
2	19.95	SGVhbHRo
3	92.00	YmVhdXR5IGFuZCBwZXJzb25hbCBjYXJl
4	11.50	a2l0Y2hpbmcgYW5kIGRpbmluZw
6	24.99	SG91c2Vob2xkIGFuZCBDbGVhbmluZw
7	148.00	Q2xvdGhpbmcgYW5kIEFjY2Vzc29yaWVz
8	89.00	Q2xvdGhpbmcgYW5kIEFjY2Vzc29yaWVz
9	14.95	VG95cyBhbmQgR2FtZXM
10	55.00	QXJ0IHN1cHBsaWVz
11	32.99	a2l0Y2hpbmcgYW5kIGRpbmluZw
12	92.00	YmVhdXR5IGFuZCBwZXJzb25hbCBjYXJl

Step 3: calculated the mean by grouping category column.

```
: nwdf.groupby(['category'])['price_string'].mean()
```

```
: category
  Q2xvdGhpbmcgYW5kIEFjY2Vzc29yaWVz       125.413871
  QXJ0IHN1cHBsaWVz                        19.471053
  QmFieWNhcmU                            163.688889
  RWxlY3Ryb25pY3M                         60.258182
  SG91c2Vob2xkIGFuZCBDbGVhbmluZw          76.199000
  SGVhbHRo                                32.557586
  VG95cyBhbmQgR2FtZXM                     32.448947
  VG9vbHMgYW5kIGhvbWUgaW1wcm92ZW1lbnQ     82.223500
  YmVhdXR5IGFuZCBwZXJzb25hbCBjYXJl        34.745067
  Z3JvY2VyaWVz                            10.541667
  a2l0Y2hpbmcgYW5kIGRpbmluZw              16.703235
  b2ZmaWNlIHByb2R1Y3Rz                    20.480000
  cGV0IHN1cHBsaWVz                        14.181875
  Name: price_string, dtype: float64
```