First Project

Alan I. Ruiz Olivares

February 2022

Contents

1 Introduction

In this study of case we will practice the programming basis in Python for data analysis and classification to create input users logins and validations, using and defining variables, list, logic operators, dictionaries and conditionals for the classification of information requested.

We will see the case of **LifeStore**, a virtual store that has a wide variety of articles, recently, Sales corporate, saw that the company has a huge amount of inventory, therefore, they had identified a reduction of group search in a really important segment of products, that has caused in a really important and substantial diminution of sales in it last trimester.

2 Problem Solution

2.1 Top Sales

The top sales with the maximum and minimum are as follow.

Item	Name	Times Sold
54	SSD Kingston A400	50
3	Procesador AMD Ryzen 5 2600	42
5	Procesador Intel Core i3-9100F	20
42	ASRock Micro ATX B450M Steel Legend	18
57	SSD Adata Ultimate SU800	15

Table 1: Top 5 most sale items

Item	Name	Times Sold
10	MSI GeForce 210, 1GB GDDR3	1
13	Asus NVIDIA GeForce GTX 1050	1
17	Gigabyte AMD Radeon R7 370 OC	1
22	MSI NVIDIA GeForce GTX 1050 Ti OC	1
28	Zotac NVIDIA GeForce GTX 1660 Ti	1

Table 2: Top 5 min sale items

We can note that the top sale were the item 54, 3, 5, 42 and 57. So the store should prioritize in having enough stock and analyse why are the top sales? See the conditions of every item and see if that conditions can be establish to all the items in the store or at least to the less sold items.

2.2 Top Searches

Item	Name	Times Searched
54	SSD Kingston A400	264
57	SSD Adata Ultimate SU800	107
29	ASUS micro ATX TUF B450M-PLUS GAMING	60
3	Procesador AMD Ryzen 5 2600	55
4	Procesador AMD Ryzen 3 3200G	41

Table 3: Top 5 most searches

Item	Name	Times Searched
9	Intel Core i3-8100	1
10	MSI GeForce 210, 1GB GDDR3	1
27	VisionTek AMD Radeon HD5450	1
35	Gigabyte micro ATX Z390 M GAMING	1
45	ASRock ATX H110 Pro BTC	1

Table 4: Top 5 min searches

The items 54, 57, 29, 3, 4 where the most searched items. It´s important to note that the **Item 54** is the most searched item and the best seller, and the **Item 3** is the second in the top sales and the 4th item most searched in the store. In the other hand the **Item 10** is the item lest searched and lest sold in the whole year, maybe because is an old model or maybe is not that commercial. Some variants to check out and see if there's something wrong with the item

2.3 Top Categories

Category	Avg Review	Total Profit
Memorias USB	5	$2,519
Pantallas	5	$11,278
Discos Duros	4.73	$93,496
Procesadores	4.70	$371,726
Tarjetas de Video	4.57	$136,224

Table 5: Top 5 max category

Category	Avg Review	Total Profit
Tarjetas Madre	4	$127,321
Audifonos	4.4	$9,135
Bocinas	4.5	$8,478
Tarjetas de Video	4.57	$136,224
Procesadores	4.70	$371,726

Table 6: Top 5 min category

It's important to see that the best calcification per category is obtained by the category's with only one sale, therefore we can exclude it and give the medal to the best reviewed category is **Discos duros** and **Procesadores** with the best reviews and they represent 12% and 49% respectively of the total annual sales, that's 61% of the sales in the whole year.

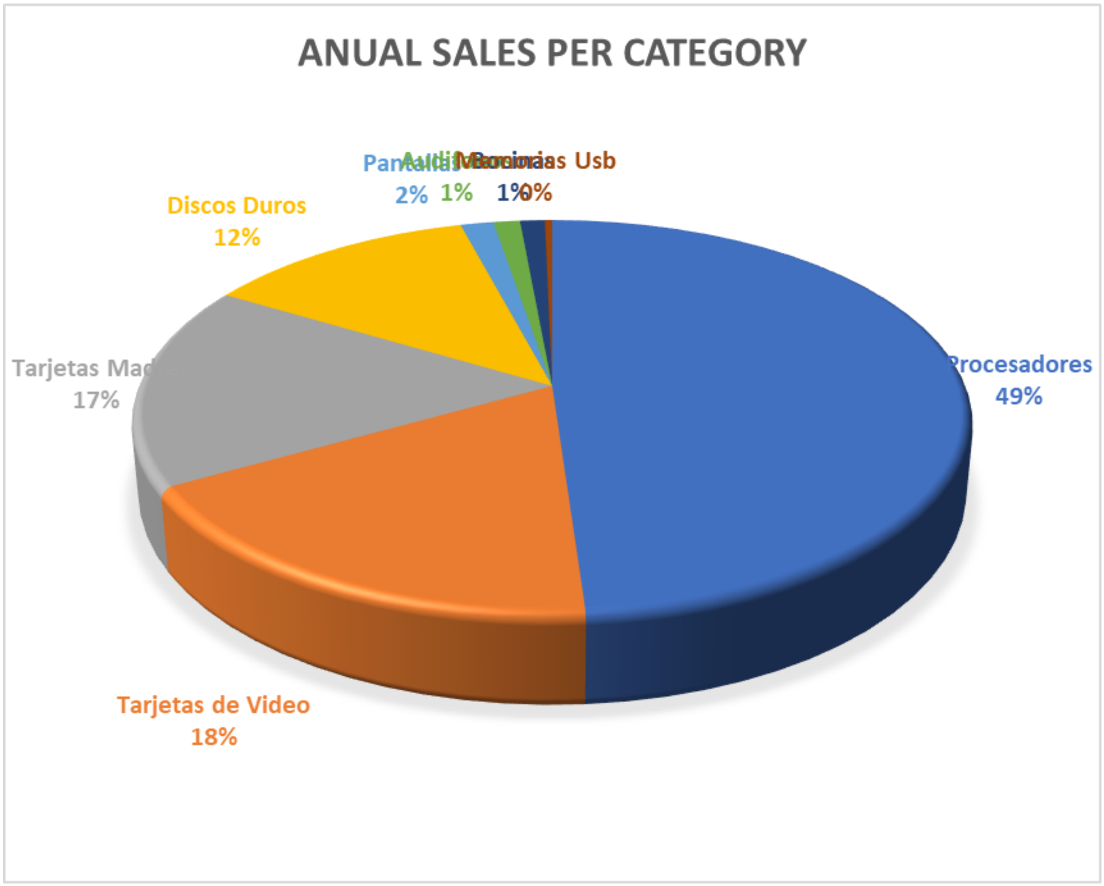

Figure 1: Pie chart of the categories and porcentual sales annualy

2.4 Top Stars

Item	Name	Average Stars
1	AMD Ryzen 3 3300X S-AM4	5
6	Intel Core i9-9900K, S-1151	5
7	Intel Core i7-9700K, S-1151	5
8	Intel Core i5-9600K, S-1151	5
11	ASUS AMD Radeon RX 570	5

Table 7: Top 5 max stars

Item	Name	Average Stars
31	AORUS micro ATX B450 AORUS M	2.6
89	Cougar Audífonos Gamer Phontum Essential	3
10	MSI GeForce 210	4
13	Asus NVIDIA GeForce GTX 1050 Ti Phoenix	4
94	HyperX Audífonos Gamer Cloud Flight para PC/PS4/PS4 Pro	4

Table 8: Top 5 min stars

The items with the best and worst reviews are described as follow in the tables

2.5 Sales per month

Month	Total items	Total Sales
January	52	$117,738
February	40	$107,270
March	49	$162,931
April	74	$191,066
May	34	$91,936
June	11	$36,949
July	11	$26,949
August	3	$3,077
September	0	$0
August	0	$0
October	0	$0
November	0	$0
December	0	$0

Table 9: Average sale per month

Even though in September and November was a sale, it was returned, that's why there are not sales reported in those months. March (22%), April (26%) and February (15%) represents the best sales months with 63% of the total sales of the whole year, with 49, 74, 40 sales per month respectively. It's important to analyze why those months represent more that the half of the total income in the whole year, such as the last quatri-mestter where the total sales were 0.

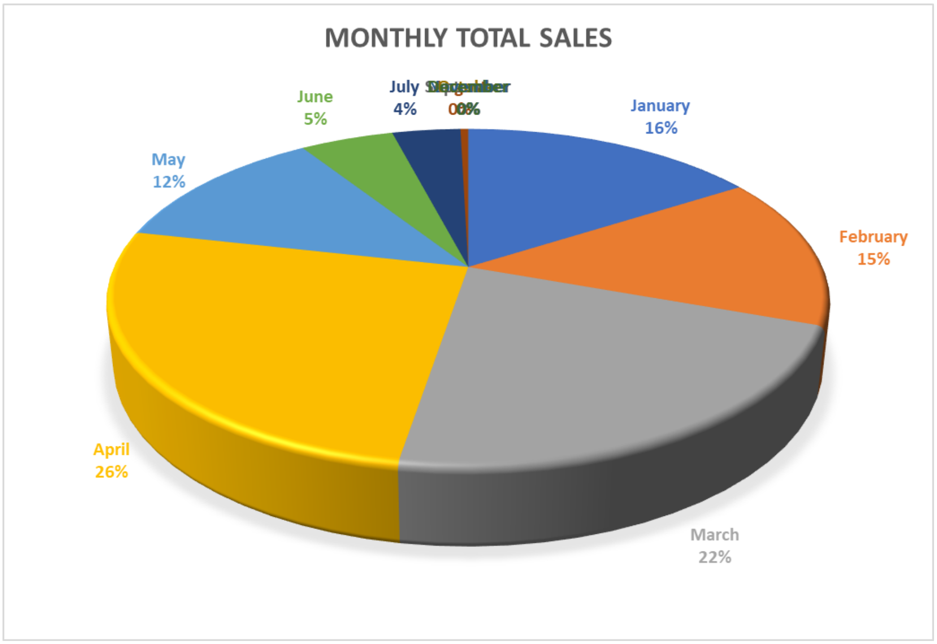

Figure 2: Pie chart of the total annual sales

3 Code Definition

In order for getting access to the data base and all the parameters calculated, an user and password is going to be required, therefore there's a login question written in the code as:

```
userAccess= False
trys = 0
#Welcome
welcome = "Welcome to the sistem\nPlease enter user and
    password: "
print(welcome)

#attemps that you try
while not userAccess :
  user= input("User: ")
  password=input("Password: ")
  trys +=1
  #Check that par coincides
  if user == "admin" and password == "admin" :
   userAccess = True
```

```
15   print ("Hello, welcome back! :)")
16  else :
17    print ("Error, you have " , 3 - trys, "trys left")
18    if user == "admin":
19     print ("Wrong password")
20    else :
21     print(f"The user : '{user}' is not register")
22  if trys == 3:
23    print ("Please contact us for further assitance :() \n
     Tel : (123) 456 7890  or by e-mail : contact@help.com")
24    exit()
```

Listing 1: Log-in request for a user and password

In this part, **line 4 - 5** is the *Welcome* message that is displayed when the program is running. In the **line 9 - 22** the parameter are set, the user and the password are asked, and if the user is not the definition in *line 13*, in this case *admin*, a message is going to be displayed as an error, and if the password is not the one defined on the same line, in this case *admin*, then an error is going to be displayed with a marker, the user will have 3 attempts to login, with the possibility of miss the user or the password, an a prompt will be displayed in each case, if the 3 attempts are reached, a personalized message for support will be displayed for more assistance, via e-,ail or phone number, and the program will be finalized. But if the user and password are right, it would give access to the user.

3.1 Part 1

3.1.1 Top Sales

Here all the tickets and items sold are exported from *lifestore_sales* as a list, *sales*. The dictionary *best_sale* is created, for every parameter in the list sales. Every parameter is valuated as *ticket* and *num_part*, for every ticket sold with a respective part number, so we write a new key to the dictionary if the ticket has a different part number and we add a list with every ticket sold per part number in **line 7 -9**. A print is suggested in line 10 in order to visually confirm that the dictionary is correct.

Therefore, a new dictionary is created, in here we will be able to know how many times the piece number was sold, because in *best_sale* we have the whole list of every ticket sold calculating the length of every list of the dictionary value per key in **line 14**. A print is suggested in *line 15* in order to have a visual confirmation of the dictionary.

The program in **line 18 - 21**, is suggested so the whole list, can be display. For every piece number how many times it was sold.

In **line 27**, the whole list is sorted from max to min, i.e. from the most sold piece to the less sold piece, but we do not need the whole list, at least no for now, then we create a variable called **top_sale** in order to ask for the length of the list as desired, so the input in *line 24* will be interpreted as a int value, so we can used it for slicing the whole list sorted, at the end of **line 27** and **line 37**, the max top is defined in *line 27-32* and the min top is defined in *line 35 - 39* in the same way as the max top. A couple of prints are suggested in *line 33 and 40* to a different format to have the results.

```
1  sales = [[producto [0], producto[1]] for producto in
      lifestore_sales ]
2
3  best_sale = {}
4  for par in sales:
5    ticket = par[0]
6    num_part = par[1]
7    if num_part not in best_sale.keys():
8      best_sale[num_part] = []
9    best_sale[num_part].append(ticket)
10 #print(best_sale)
11
12 times_sold = {} #We create this dictionary so we can know how
       many times a piece item was sold
13 for k,v in best_sale.items():
14     times_sold[k] = (len(v))
15 #print(times_sold)
16
17 """
18 #THIS WAS PRINTED TO SEE THE WHOLE LIST
19 for key in times_sold.keys():
20   print(f"The piece number: {key}")
21   print(f"was sold {times_sold[key]} times ")
22 """
23
24 top_sale = input("Plase tell me, how long would you like the
      Top to be? ") #This input is addded so we can know how
      long the list is going to be, we can ask for the top 3 or
      top 5 most or less sold
25
26 top_sale = int(top_sale)
27 sort_times_sold = sorted(times_sold.items(), key=operator.
    itemgetter(1), reverse = True)[:top_sale]
print("The highest sales per item is listed as follow:")

for sale in enumerate(sort_times_sold):
  print(f"The item number: {sale[1][0]} ")
  print(f"Has been sold {times_sold[sale[1][0]]} times ")
  #print(sale[1][0], 'category has', cat_sum[sale[1][0]], 'average sales')

min_top_sale = sorted(times_sold.items(), key=operator.itemgetter(1))[:top_sale]
print("The lowest sale per item are listed as follow:")
for sale in enumerate(min_top_sale):
  print(f"The item number: {sale[1][0]} ")
  print(f"Has been sold {times_sold[sale[1][0]]} times")
  #print(sale[1][0], 'category has', cat_sum[sale[1][0]], 'average sales')
```

Listing 2: Max top and Min top are defined.

3.1.2 Top 5 Search List

The first line is to define the a new list exporting the time 0 and 1 from *lifestore_sales*, then a empty new dictionary is created, then the parameters are define as follow, the id search and the id product, creating a new key for every id product with all the id sales register in the list *search* in the **line 2 - 8**, a print is suggested in line 9 to have a visual confirmation for the dictionary created.

A empty new empty list is created, *times_search*, so we can add every value that we are interested in, defining the parameters in every key in the list. In *line 14* the piece number and the times sold are added, and a print is suggested to confirm that is correct.

A command is created to do sort the list, from the smaller value to the higher and printed in **line 17 - 18**, with the same definition and process a new list is created to sort the values form the higher to smaller value in **line 20 -21**

```
1 search = [[producto [0], producto[1] ] for producto in lifestore_searches]
2 best_search = {}
3 for par in search:
4   id_search = par[0]
5   id_product = par[1]
```

```python
  if id_product not in best_search.keys():
    best_search[id_product] = []
  best_search[id_product].append(id_search)
#print(best_search)

times_search = []
for key, value in best_search.items():
    #print value
    times_search.append([key, len([item for item in value if item])]) #piece number and times sold are added to the list
#print(times_search)  #suggested print to confirm

times_search.sort(key=lambda x: x[1]) #sort from min to max
print(f"The lowest searches where (piece number, amount): {times_search[0:5]} ")

times_search.sort(key=lambda x: x[1],reverse=True) #invertimos el orden y ahora sera de mayor a menor
print(f"The highest searches where (piece number, amount): {times_search[0:5]} ")
```

Listing 3: Top search per item

3.2 Top reviews per category

A dictionary is created to add all the data wanted form *lifestore_sales* and giving values to every parameter as sale, then if a product id is not in the keys defined in the dictionary we will add the review to the item id in the dictionary in **line 4 - 10**

Then a new dictionary for all the categorized products is created, and for every product in the list *lifestore_products*, with parameters defined as product id and category, if the category is not in the keys, a new key will be created with every product id added.

For the sales category, for the category in the list *cat_prods* every product list is defined as a list with every category. Markers are created.

For every product number in the list, if every product number is not in the product reviews dictionary, then the iteration will continue, giving values to reviews, price, total sales, profits and review per category. Then the review average is calculated in **line 43**. At the end the category, review average and total profit per items sold are displayed

```python
from lifestore_file import lifestore_products,
    lifestore_sales

# Id review dictionary
prods_reviews = {}
for sale in lifestore_sales:
    prod_id = sale[1]
    review = sale[2]
    if prod_id not in prods_reviews.keys():
        prods_reviews[prod_id] = []
    prods_reviews[prod_id].append(review)

# id per cat product
cat_prods = {}
for prod in lifestore_products:
    prod_id = prod[0]
    cat = prod[3]
    if cat not in cat_prods.keys():
        cat_prods[cat] = []
    cat_prods[cat].append(prod_id)

# cateory sales
cat_ventas = {}
for cat in cat_prods.keys():
    # products per category list
    prods_list = cat_prods[cat]
    reviews_cat = []
    ganancias = 0
    ventas = 0

    # for every prodcut in the catehory
    for prod_id in prods_list:
        # acces reviews, times sold
        if prod_id not in prods_reviews.keys():
            continue
        reviews = prods_reviews[prod_id]
        precio = lifestore_products[prod_id-1][2]
        total_sales = len(reviews)
        ganancias += precio * total_sales
        ventas += total_sales
        reviews_cat += reviews

    #  review average per cat
    rev_prom_cat = sum(reviews_cat) / len(reviews_cat)
    # Guardo todo en mi diccionario
    cat_ventas[cat] = {
        'review_promedio': rev_prom_cat,
        'ganancias_totales': ganancias,
        'ventas_totales': ventas
    }

f'string'

for key in cat_ventas.keys():
    print(f"The category: {key} ")
    for llave, valor in cat_ventas[key].items():
        print(f'\t {llave}: {valor}')
```

Listing 4: Category information

3.3 Part 2

3.3.1 Top reviews

In this part, we are going to make a list with the reviews per item, in the list *lifestore_sales* we export every review per item in every ticket sold, therefore we export that information in **line 1**, creating a new dictionary *best_review*, defining the parameters as the star per item, so for every star given in each item will be added to a key defined as the piece number.

Then a new dictionary is created, as *avg_star* we will define every parameter as k, v in the *best_review* dictionary in order to operate it and get the average of all the items in the list. A print is suggested to have a visual confirmation of the new dictionary. A format print is defined in **line 15 - 17** so we can see the results in a better and more organized list.

```
1  review= [[sale[1], sale[2]]for sale in lifestore_sales if
      sale [4]==0]
2
3  best_review = {}
4  for par in review:
5    id_item=par[1]
6    id_star=par[0]
7    if id_star not in best_review.keys():
8      best_review[id_star]=[]
9    best_review[id_star].append(id_item)
10
11 #Here we calculate the average qualification or star of every
      item
```

```python
avg_star = {}
for k,v in best_review.items():
    avg_star[k] = sum(v)/ float(len(v))
#print(avg_star)
'''
for key in avg_star.keys():
  print(f"The Article: {key}")
  print(f"Has {avg_star[key]} average stars/review")
'''

top_stars = input ("How long would you like the top to be?: "
    )
print("The highest star per item is listed as follow:")

top_stars = int(top_stars)
sort_avg_star = sorted(avg_star.items(), key=operator.
    itemgetter(1), reverse = True)[:top_stars]
for star in enumerate(sort_avg_star):
  print(f"The item number: {star[1][0]} ")
  print(f"Has been review with {avg_star[star[1][0]]} stars "
    )

top_stars = int(top_stars)
sort_avg_star_min = sorted(avg_star.items(), key=operator.
    itemgetter(1))[:top_stars]
for star in enumerate(sort_avg_star_min):
  print(f"The item number: {star[1][0]} ")
  print(f"Has been review with {avg_star[star[1][0]]} stars "
    )
```

Listing 5: Best review sorted items

4 Part 3

4.1 Average income

A new list is defined importing the ticket and the date sold, excluding the items that where returned. A new empty dictionary is created *cat_months* for every item categorized per month, for every parameter in the date sold, we define every parameter with the ticket and the month sold, separating the whole date line by value (day, month, year) in this case we only use the month information, if the month is not in the key, then a new list will be created and it will be appended to the dictionary in **line 4 - 9**. A print is suggested to see the moth and sales per month in *linne 12 - 14*

For every key in the dictionary *cat_months* we will change the piece sold with the price of every item sold in every month, then it will be displayed in **line 17 - 25** as a line per month with the total amount of sales.

```python
 1 id_date = [[sale[0], sale[3]]for sale in lifestore_sales if
      sale [4]==0]
 2 #to categorize we use a dicctionary
 3 cat_months = {}
 4 for par in id_date:
 5   id = par[0]
 6   _, months, _ = par[1].split('/')
 7   if months not in cat_months.keys():
 8     cat_months[months] = []
 9   cat_months[months].append(id)
10
11 """
12 for key in cat_months.keys():
13   print(key)
14   print(cat_months[key])
15 """
16
17 for key in cat_months.keys():
18   months_list = cat_months[key]
19   months_sum = 0
20   for id_sales in months_list:
21     index = id_sales - 1
22     info_sale = lifestore_sales[index]
23     id_product = info_sale[1]
24     price = lifestore_products[id_product-1][2]
25     months_sum += price
26   print(f'In the month {key},the total income was {months_sum
      }, Total Sales in this month: {len(months_list)}')
```

Listing 6: Total income per month

5 Conclusion

The program created in python has as a job to make this analyses easier for the user and also to be protected in case someone not authorized wants to enter to the program. It has a scoreboard for every try attempted and a contact information in case the user forgets the credentials or if the program brakes.

The program helped us to mark the best and the worse items, categories, and even sales month. That can help *Lifestore* to manage and see what did they do different along the whole year, and analyze how to imporve sales in the lest sold items or to clean stock of items that aren't upgradable, such as HDD.

It's important to study the case of the fist four months. We can say that almost all the sales where made in those months and the last four we have loses, because there was no sales and 2 returned products, implying that money was lost in those returns, and continue with the stores and no sales.