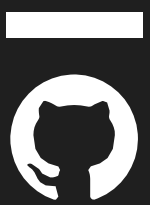
REPORTE-01

PROPUESTO A:
Historial de Ventas

REALIZADO POR:
Gómez Castañeda Juan Daniel



Fundamentos de Programación con Python
EmTech Institute
Emerging Technologies Institute

REPORTE - 01

Gómez Castañeda Juan Daniel

Índice

1. Objetivo

Poner en práctica las bases de programación en Python para análisis y clasificación de datos mediante la creación de programas de entrada de usuario y validaciones, uso y definición de variables y listas, operadores lógicos y condicionales para la clasificación de información.

2. Desarrollo

2.1. Productos más vendidos

Generar un listado de los 5 productos con mayores ventas

Para este primer punto, lo primero que se creó fue una lista donde únicamente se guarda el ID del producto que se vendió de la lista `lifestore_sales`. A continuación, se creó una lista anidada donde se guardaba por fila el ID del producto, el nombre del producto, la cantidad de veces que se vendió y la categoría a la que pertenece. Para crearla se utilizaron dos ciclos `for` donde el primero recorría todos los productos que existen en la lista `lifestore_products` y el segundo iba guardando la información del producto junto con el conteo del número de veces que se vendió con la función `.count()` sobre la lista `soldproduct`. Todo esto se muestra en el código 1

```
soldproduct = [sale[1] for sale in lifestore_sales]

for sale in lifestore_products: #Creating a nested list
    nested =[]
    timesold.append(nested)
    for k in range(1):
        nested.append(sale[0]) #Saving ID
        nested.append(sale[1]) #Saving Name
        nested.append(soldproduct.count(sale[0])) #Counting times sold
        nested.append(sale[-2]) #Saving Category
```

Código 1: Que crea una lista anidada con el número de veces vendido

Posteriormente buscamos ordenar de manera ascendente la lista con respecto al número de ventas que tuvo. Donde definimos una función anónima y aplicamos la función `.sort()` sobre la tercera columna de la lista `timesold` tal como se muestra en el código 2

```
def Sort(timesold): #timesold is [id_product, name, time_sold, category]
    timesold.sort(key = lambda x: x[2])
    return timesold

timesold = Sort(timesold) # Ascendant order
```

Código 2: Que ordena de forma ascendente la lista `timesold`

Generar un listado con los 10 productos con mayores búsquedas

Para este punto se repitió el mismo proceso que en los códigos 1 y 2, con la única diferencia de que la lista independiente que se creo fue a partir de la lista `lifestore_searches` por lo que el conteo de la función `.count()` fue de las búsquedas en lugar de las ventas.

```
searchedproduct = [search[1] for search in lifestore_searches]

for search in lifestore_products: #Creating a nested list
    nested =[]
    timesearched.append(nested)
    for k in range(1):
        nested.append(search[0]) #Saving ID
        nested.append(search[1]) #Saving Name
        nested.append(searchedproduct.count(search[0])) #Counting times searched
        nested.append(search[-2]) #Saving Category

def Sort(timesearched):
    timesearched.sort(key = lambda x: x[2])
    return timesearched

timesearched = Sort(timesearched) # Ascendant order
```

Código 3: Que crea una lista anidada con el número de veces buscado

Finalmente, para imprimir ambos listados únicamente hay que imprimir los últimos 5 elementos de la lista, ya que son los más vendidos o los más buscados dependiendo la lista que se esté imprimiendo.

```
print("\n MOST SOLD PRODUCTS")
for i in [-1,-2,-3,-4,-5]:
    print( F'ID: {timesold[i][0]}\t NAME: {timesold[i][1]}\t SALES: {timesold[i
    ][2]}' )

print("\n MOST SEARCHED PRODUCTS")
for i in [-1,-2,-3,-4,-5]:
    print( F'ID: {timesearched[i][0]}\t NAME: {timesearched[i][1]}\t SEARCHES: {
    timesearched[i][-1]}' )
```

Código 4: Que imprime los productos más vendidos y buscados

2.2. Productos rezagados

Por categoría, generar un listado con los 5 productos con menores ventas

Para esta sección se empezó por crear un diccionario para obtener el número exacto de categorías que existen y una lista que los incluya a todas. A continuación, se creó una lista vacía por cada categoría que existe en la lifestore, en dicha lista se guardará el producto y el número de veces vendido. Únicamente hizo falta un ciclo `for` para pasar por toda la lista `timesold` e ir acomodando cada producto en su lista correspondiente.

```
categories= [item[-2] for item in lifestore_products]
categories = list(dict.fromkeys(categories))

for item in timesold:
    if categories[0] in item:
        processors.append(item[:3])
    elif categories[1] in item:
        gpus.append(item[:3])
    elif categories[2] in item:
        motherboards.append(item[:3])
    elif categories[3] in item:
        drives.append(item[:3])
    elif categories[4] in item:
        usb.append(item[:3])
    elif categories[5] in item:
        screens.append(item[:3])
    elif categories[6] in item:
        speakers.append(item[:3])
    elif categories[7] in item:
        headphones.append(item[:3])
```

Código 5: Que guarda en listas la cantidad de veces que fue vendido, clasificando por categoría

Por categoría, generar un listado con los 10 productos con menores búsqueda

Para este punto se realizó el mismo procedimiento que en el código 5 para clasificar los productos por categoría. La única diferencia fue que el ciclo `for` en lugar de pasar por la lista `timesold` pasó por la lista `timessearched`.

```
for item in timesearched:
    if categories[0] in item:
        processors.append(item[:3])
    elif categories[1] in item:
        gpus.append(item[:3])
    elif categories[2] in item:
        motherboards.append(item[:3])
    elif categories[3] in item:
        drives.append(item[:3])
    elif categories[4] in item:
        usb.append(item[:3])
    elif categories[5] in item:
        screens.append(item[:3])
    elif categories[6] in item:
        speakers.append(item[:3])
    elif categories[7] in item:
        headphones.append(item[:3])
```

Código 6: Que guarda en listas la cantidad de veces que fue buscado, clasificando por categoría

Anteriormente en el código 2 los valores ya se habían acomodado de manera ascendente, por lo que queda imprimir los primeros 10 productos de cada lista que son los menos buscados o los menos vendidos dependiendo la lista. En el código 7 se imprimirá solamente las tarjetas de vídeo menos buscadas.

```
print("\n LEAST SEARCHED GPUS")
for i in range(10):
    print(f"ID: {gpus[i][0]}\t NAME: {gpus[i][1]}\t TIMES SEARCHED: {gpus[i][2]}")
```

Código 7: Que imprime las 10 tarjetas de vídeo menos buscadas

2.3. Productos por reseña en el servicio

Mostrar un listado de 5 productos con las mejores reseñas y otra con las peores

Para esta sección lo primero fue obtener calificación total por producto, con ayuda de dos ciclos `for` y la variable temporal `tempsum` para almacenar la suma por producto en la lista `timesold`.

```
for product in lifestore_products:
    for review in lifestore_sales:
        if product[0]==review[1]:
            tempsum += review[2] #Adding up review scores

    totalscore.append(tempsum)
    tempsum=0
```

Código 8: Que suma y guarda la calificación total obtenida por producto

Posteriormente se repitió el mismo proceso de los códigos 1 y 3 con la única diferencia de que en la lista anidada se realizó la operación `totalscore/timesold` para obtener la calificación promedio.

```
for review in timesold: #Creating a nested list
    nested=[]
    if review[2] > 0: #Obtaining reviews from products sold at least once
        averagescore.append(nested)
        for k in range(1):
            nested.append(review[0])
            nested.append(review[1])
            nested.append(review[2]) #Times reviewed
            nested.append(totalscore[review[0]-1]/review[2]) #Obtaining average

def Sort(averagescore):
    averagescore.sort(key = lambda x: x[-1])
    return averagescore

averagescore = Sort(averagescore) # Ascendant order
```

Código 9: Que crea una lista anidada con la calificación promedio de cada producto

2.4. Total, de ingresos y ventas

Para esta sección se crearon dos listas, una con las ventas ordenadas por orden cronológico con la ayuda de la función `date.sort()` y la segunda que contiene únicamente los nombres de los meses.

```
from datetime import datetime; import calendar
date = [sale[3] for sale in lifestore_sales]
date.sort(key=lambda date: datetime.strptime(date,'%d/%m/%Y')) #Sort dates in order

month =calendar.month_name[1:] #Creating a list with the months
```

Código 10: Que ordena las ventas por fecha y crea una lista con los nombres de los meses

Posteriormente se hizo una suma tanto de las ventas que hubo como de los ingresos por mes, con ayuda de un ciclo `for` y la condicionante `if` pudimos preguntar si el mes deseado se encontraba dentro de la venta e ir sumandola a la lista correspondiente. La primera lista llamada `monthprofit` guarda los ingresos por mes mientras que la lista `monthsales` guarda la cantidad de ventas en el mes. Adicionalmente también se calculó la cantidad de dinero perdido por devoluciones.

```
refundeditem = []; totalrefunds = 0; averageticket = []
totalsales = 0; monthsales = [0]*12;   monthprofit = [0]*12

soldproduct = [sale[1] for sale in lifestore_sales]

for i in range(0,len(lifestore_sales)):
    if int(lifestore_sales[i][-1]) == 0:                    #Verifying it was not a
    refund
        totalsales+=lifestore_products[soldproduct[i]][2]    #Adding up the total
    sales
        if "/01/" in date[i]:
            monthprofit[0]+=lifestore_products[soldproduct[i]][2]
            monthsales[0]+=1
        elif "/02/" in date[i]:
            monthprofit[1]+=lifestore_products[soldproduct[i]][2]
            monthsales[1]+=1
        elif "/03/" in date[i]:
            monthprofit[2]+=lifestore_products[soldproduct[i]][2]
            monthsales[2]+=1
        elif "/04/" in date[i]:
            monthprofit[3]+=lifestore_products[soldproduct[i]][2]
            monthsales[3]+=1
        elif "/05/" in date[i]:
            monthprofit[4]+=lifestore_products[soldproduct[i]][2]
            monthsales[4]+=1
        elif "/06/" in date[i]:
            monthprofit[5]+=lifestore_products[soldproduct[i]][2]
            monthsales[5]+=1
        elif "/07/" in date[i]:
            monthprofit[6]+=lifestore_products[soldproduct[i]][2]
            monthsales[6]+=1
        elif "/08/" in date[i]:
            monthprofit[7]+=lifestore_products[soldproduct[i]][2]
            monthsales[7]+=1
        elif "/09/" in date[i]:
```

```
34                monthprofit[8]+=lifestore_products[soldproduct[i]][2]
35                monthsales[8]+=1
36            elif "/10/" in date[i]:
37                monthsales[9]+=lifestore_products[soldproduct[i]][2]
38                monthsales[9]+=1
39            elif "/11/" in date[i]:
40                monthprofit[10]+=lifestore_products[soldproduct[i]][2]
41                monthsales[10]+=1
42            elif "/12/" in date[i]:
43                monthprofit[11]+=lifestore_products[soldproduct[i]][2]
44                monthsales[11]+=1
45        else:
46            refundeditem.append(soldproduct[i])                    #ID product refunded
47            totalrefunds+=lifestore_products[soldproduct[i]][2] #Total lost in refunds
```

Código 11: Que guarda y suma la cantidad de ingresos y ventas por mes

Para calcular el ticket promedio por mes se creó una lista más llamada `averageticket` y finalmente se juntaron las tres listas en una sola con la función `.zip()` para facilitar su impresión.

```
1 for i in range(12):
2     if monthsales[i] > 0:
3         averageticket.append(monthprofit[i]/monthsales[i]) #Obtaining average
4     else:
5         averageticket.append(0)
6
7 salesxmonth = [list(l) for l in zip(month, monthprofit, monthsales, averageticket)]
```

Código 12: Que calcula el ticket promedio por mes y junta 4 listas

Finalmente se tuvo que ordenar e imprimir los valores según el mes con mayores ingresos (código 13), el mes con mayores ventas (código 14) o el mes con el más alto ticket promedio (código 15).

```
1 def Sort(salesxmonth): #Sort for profit
2     salesxmonth.sort(key = lambda x: x[1])
3     return salesxmonth
4 salesxmonth = Sort(salesxmonth)
5
6 print("\n MOST PROFITABLE MONTHS")
7 for i in [-1,-2,-3,-4,-5]:
8     print( F'MONTH: {salesxmonth[i][0]}\t PROFIT: {"${:,.2f}".format(salesxmonth[i
      ][1])}
9     \t SALES: {salesxmonth[i][2]}\t AVERAGE: {salesxmonth[i][-1]}' )
```

Código 13: Que ordena e imprime la lista por ingresos

```
1 def Sort(salesxmonth): #Sort for sales
2     salesxmonth.sort(key = lambda x: x[2])
3     return salesxmonth
4 salesxmonth = Sort(salesxmonth)
5
6 print("\n MOST SALES PER MONTHS")
7 for i in [-1,-2,-3,-4,-5]:
8     print( F'MONTH: {salesxmonth[i][0]}\t PROFIT: {"${:,.2f}".format(salesxmonth[i
      ][1])}\t SALES: {salesxmonth[i][2]}\t AVERAGE: {salesxmonth[i][-1]}' )
```

Código 14: Que ordena e imprime la lista por ventas

```
def Sort(salesxmonth): #Sort for average ticket
    salesxmonth.sort(key = lambda x: x[-1])
    return salesxmonth
salesxmonth = Sort(salesxmonth)

print("\n HIGHEST AVERAGE TICKET PER MONTH")
for i in [-1,-2,-3,-4,-5]:
    print( F'MONTH: {salesxmonth[i][0]}\t PROFIT: {"${:,.2f}".format(salesxmonth[i
    ][1])}\t SALES: {salesxmonth[i][2]}\t AVERAGE: {salesxmonth[i][-1]}' )
```

Código 15: Que ordena e imprime la lista por el ticket promedio más alto

2.5. Login de usuario

Finalmente se agregó un login que tiene por usuario `Manager123` y por contraseña `lif3stor3` y un limite máximo de tres intentos. Con ayuda de un `while not` y de la variable booleana `Acces` permitiremos el acceso o lo negaremos.

```
username = "Manager123"
password = "lif3stor3"
tries = 0; Acces = False

while not Acces:
    tries += 1
    if tries == 4:
        exit()
    if input('Username: ') == username and input('Password: ') == password:
        Acces = True
        print('Acces Granted')
    else:
        print(f'You have {3 - tries} tries lefts')
```

Código 16: Que requiere ingresar un usuario y contraseña para poder correr el programa

3. Resultados y Análisis

Repositorio de GitHub

MAIN PRODUCT

SSD KINGSTON A400, 120GB

4.7

MOST SOLD PRODUCT

50 times sold

17% of the total products sold

MOST SEARCHED PRODUCT

263 times searched

25% of the searches

TOP 5 SOLD PRODUCTS

01 SSD Kingston A400, 120GB
Category: Discos Duros

02 Procesador AMD Ryzen 5 2600
Category: Procesadores

03 Procesador Intel Core i3-9100F
Category: Procesadores

04 Tarjeta Madre ASRock Micro ATX B450M Steel Legend
Category: Tarjetas madre

05 SSD Adata Ultimate SU800, 256GB
Category: Discos Duros

TOP 10 SEARCHED PRODUCTS

01 SSD Kingston A400, 120GB
Category: Discos Duros

02 SSD Adata Ultimate SU800, 256GB
Category: Discos Duros

03 Tarjeta Madre ASUS micro ATX TUF B450M-PLUS GAMING
Category: Tarjetas madre

04 Procesador AMD Ryzen 5 2600
Category: Procesadores

05 Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8
Category: Procesadores

06 Logitech Audífonos Gamer G635 7.1
Category: Audifonos

07 TV Monitor LED 24TL520S-PU 24
Category: Pantallas

08 Procesador Intel Core i7-9700K
Category: Procesadores

09 Procesador Intel Core i3-9100F
Category: Procesadores

10 SSD XPG SX8200 Pro, 256GB
Category: Discos Duros

BEST RATED PRODUCTS

01 Procesador Intel Core i7-9700K
Score: 5.0 Time Reviewed: 7

02 Procesador Intel Core i5-9600K
Score: 5.0 Time Reviewed: 4

03 Kit SSD Kingston KC600
Score: 5.0 Time Reviewed: 3

04 ASUS AMD Radeon RX 570
Score: 5.0 Time Reviewed: 3

05 Procesador Intel Core i9-9900K
Score: 5.0 Time Reviewed: 3

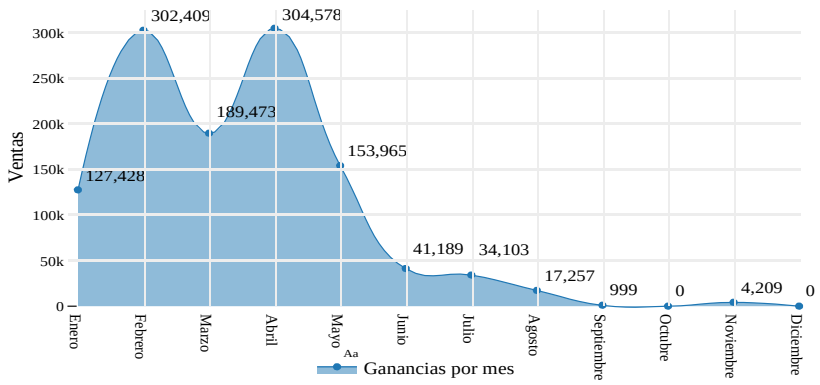


Figura 2: Histograma de las ganancias mes a mes

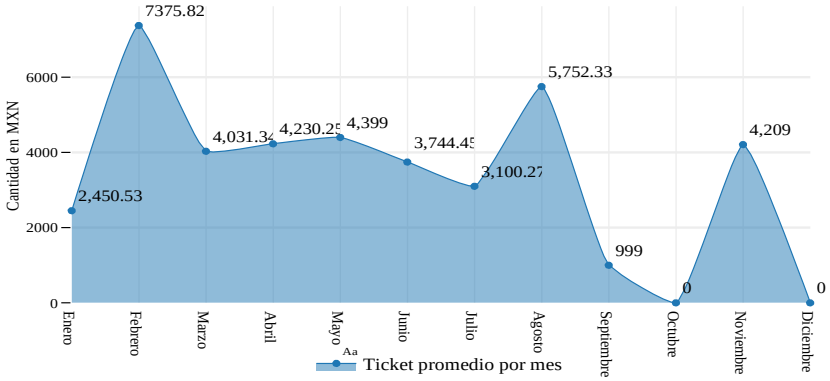


Figura 3: Histograma del ticket promedio mes a mes

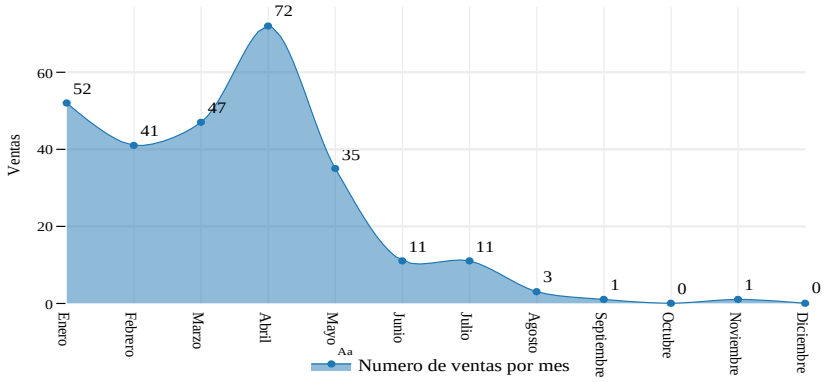


Figura 4: Histograma del numero de ventas mes a mes

3.1. Análisis

Obteniendo todos los puntos solicitados por la gerencia de ventas, pudimos resaltar que el producto más vendido y buscado es el *SSD KINGSTON A400 y* se muestra en 3. Representó el 25 % de todas las búsquedas realizadas este año y 17 % del número de ventas hechas. Además, obtuvo una calificación promedio de satisfacción del 4.72.

Analizando el resto de los productos más vendidos y buscados, en 3 podemos observar que solamente las categorías de *Procesadores* y la de *Discos Duros* representan el 70 % de las ventas de lifestore (104 procesadores vendidos y 94 discos duros vendidos) dejando únicamente el 30 % (85 ventas) para las 6 categorías restantes.

De 18 *tarjetas madre* diferentes con las que cuenta lifestore únicamente 8 se han vendido al menos una vez y solamente 2 se han vendido más de diez veces. De igual manera de las 18 diferentes *tarjetas de vídeo* con las que cuenta lifestore únicamente 8 se han vendido al menos una vez y ninguna ha sobrepasado las 10 ventas. No solamente estas categorías no se están vendiendo, dentro de los productos con las calificaciones más bajas por los usuarios, se encuentran ambas, las *tarjetas de vídeo* y las *tarjetas madre*, con calificaciones menores a 2 en una escala del 1 a 5. A diferencia de los *procesadores* que ocupan tres de las cinco posiciones de los productos mejores votados (ver 3).

Otro tema que sobresale dentro del último punto del proyecto fue el del análisis mes a mes de las ventas, ganancias y ticket promedio. Como se puede observar en los histogramas 2 y 4 los mejores meses en ventas y ganancias son abril y febrero, a diferencia del segundo semestre del año donde las ventas caen súbitamente. De julio a diciembre únicamente hay registradas 16 ventas, es decir un 5 % de las ventas de todo el año.

4. Conclusión

Python es un lenguaje de alto nivel, esto significa que es fácil de escribir, leer y entender. Tiene una gran variedad y un sin fin de usos, cuenta con una comunidad muy activa, lo que garantiza que el lenguaje se mantendrá actualizado con el paso del tiempo, y que surgirán nuevas librerías que nos permitirán ahorrar tiempo y trabajo. Para 2019 ya contaba con más de 145.000 librerías en su repositorio en línea, cubriendo casi cualquier tipo de necesidad. No es casualidad que Python se haya convertido en la opción principal para los científicos de datos de todo el mundo.

Como conclusión para lifestore están en un punto muy delicado donde tienen una gran variedad de productos, pero los clientes solo compran una pequeña variedad de ellos. Por otro lado, la cantidad de inventario que tienen para muchos de sus productos que no venden es demasiada. Finalmente, está el enorme problema de no vender nada durante el segundo semestre del año. Por lo que mi sugerencia sería priorizar mantener ventas continuas a lo largo del año, y después intentar modificar el inventario de productos para que se relacionen con los productos que mejor se venden.

5. Evidencias

Al correr el programa completo lo primero que ocurre es que la terminal pide tanto el nombre de usuario como la contraseña para poder acceder al resto de la información. Tienes 3 intentos hasta que el programa se detenga. En nuestro ejemplo el nombre de usuario fue: *Manager123* y la contraseña fue: *lif3stor3*.

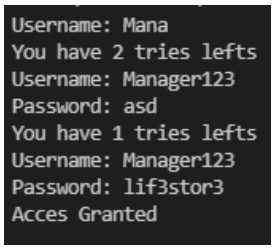


Figura 5: Terminal mostrando el funcionamiento del Login

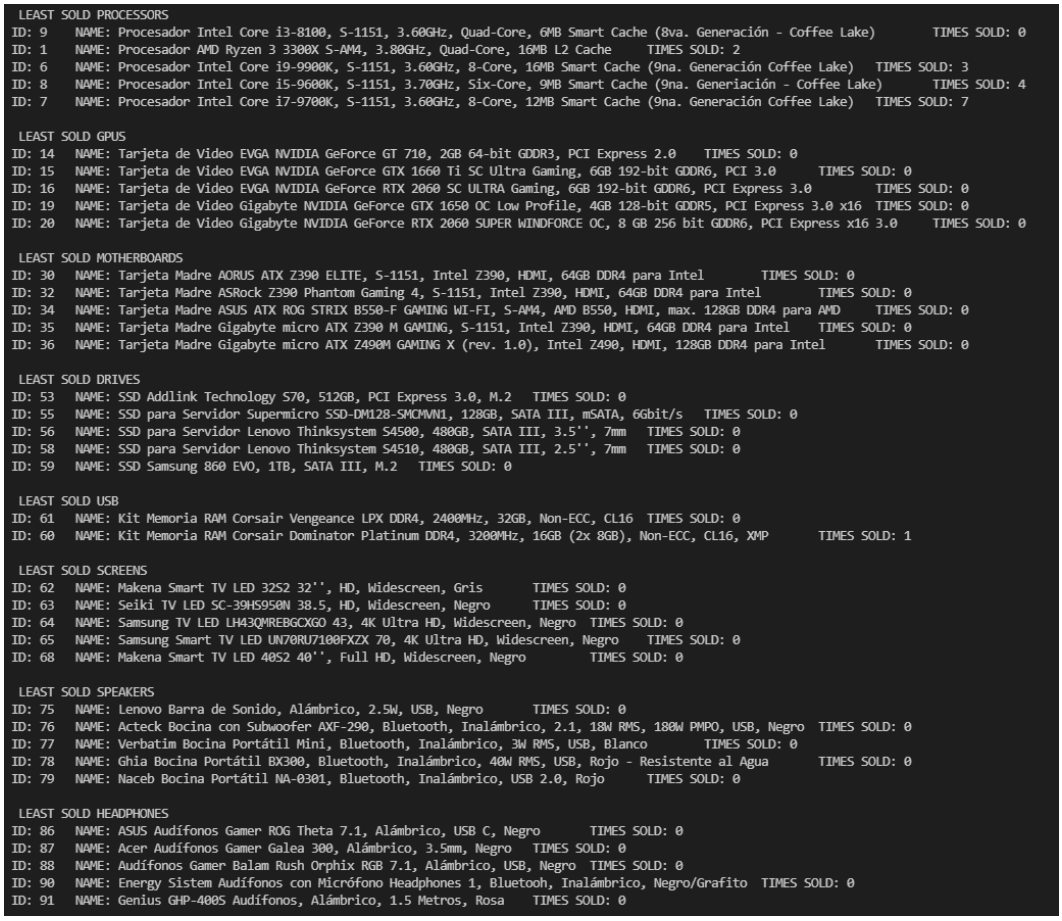


Figura 6: Terminal mostrando los productos menos vendidos por categoría

Después de presentar los 5 productos más vendidos la terminal nos arroja los 10 productos menos vendidos por categoría como se muestra en la figura 6. Mientras que en la figura 7 la terminal nos arroja los 10 productos menos buscados clasificados por categoría.

```
 LEAST SEARCHED GPUS
ID: 14   NAME: Tarjeta de Video EVGA NVIDIA GeForce GT 710, 2GB 64-bit GDDR3, PCI Express 2.0    TIMES SEARCHED: 0
ID: 16   NAME: Tarjeta de Video EVGA NVIDIA GeForce RTX 2060 SC ULTRA Gaming, 6GB 192-bit GDDR6, PCI Express 3.0        TIMES SEARCHED: 0
ID: 19   NAME: Tarjeta de Video Gigabyte NVIDIA GeForce GTX 1650 OC Low Profile, 4GB 128-bit GDDR5, PCI Express 3.0 x16  TIMES SEARCHED: 0
ID: 20   NAME: Tarjeta de Video Gigabyte NVIDIA GeForce RTX 2060 SUPER WINDFORCE OC, 8 GB 256 bit GDDR6, PCI Express x16 3.0     TIMES SEARCHED: 0
ID: 23   NAME: Tarjeta de Video MSI Radeon X1550, 128MB 64 bit GDDR2, PCI Express x16    TIMES SEARCHED: 0
ID: 24   NAME: Tarjeta de Video PNY NVIDIA GeForce RTX 2080, 8GB 256-bit GDDR6, PCI Express 3.0         TIMES SEARCHED: 0
ID: 10   NAME: MSI GeForce 210, 1GB GDDR3, DVI, VGA, HDCP, PCI Express 2.0      TIMES SEARCHED: 1
ID: 27   NAME: Tarjeta de Video VisionTek AMD Radeon HD5450, 2GB GDDR3, PCI Express x16  TIMES SEARCHED: 1
ID: 13   NAME: Tarjeta de Video Asus NVIDIA GeForce GTX 1050 Ti Phoenix, 4GB 128-bit GDDR5, PCI Express 3.0     TIMES SEARCHED: 2
ID: 17   NAME: Tarjeta de Video Gigabyte AMD Radeon R7 370 OC, 2GB 256-bit GDDR5, PCI Express 3.0       TIMES SEARCHED: 3

 LEAST SEARCHED MOTHERBOARDS
ID: 30   NAME: Tarjeta Madre AORUS ATX Z390 ELITE, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel       TIMES SEARCHED: 0
ID: 32   NAME: Tarjeta Madre ASRock Z390 Phantom Gaming 4, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel       TIMES SEARCHED: 0
ID: 33   NAME: Tarjeta Madre ASUS ATX PRIME Z390-A, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel      TIMES SEARCHED: 0
ID: 34   NAME: Tarjeta Madre ASUS ATX ROG STRIX B550-F GAMING WI-FI, S-AM4, AMD B550, HDMI, max. 128GB DDR4 para AMD     TIMES SEARCHED: 0
ID: 36   NAME: Tarjeta Madre Gigabyte micro ATX Z490M GAMING X (rev. 1.0), Intel Z490, HDMI, 128GB DDR4 para Intel      TIMES SEARCHED: 0
ID: 37   NAME: Tarjeta Madre ASRock ATX Z490 STEEL LEGEND, S-1200, Intel Z490, HDMI, 128GB DDR4 para Intel       TIMES SEARCHED: 0
ID: 38   NAME: Tarjeta Madre Gigabyte Micro ATX H310M DS2 2.0, S-1151, Intel H310, 32GB DDR4 para Intel         TIMES SEARCHED: 0
ID: 41   NAME: Tarjeta Madre ASUS micro ATX Prime H370M-Plus/CSM, S-1151, Intel H370, HDMI, 64GB DDR4 para Intel        TIMES SEARCHED: 0
ID: 43   NAME: Tarjeta Madre ASUS ATX ROG STRIX Z390-E GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel    TIMES SEARCHED: 0
ID: 35   NAME: Tarjeta Madre Gigabyte micro ATX Z390 M GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel    TIMES SEARCHED: 1

 LEAST SEARCHED DRIVES
ID: 53   NAME: SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2   TIMES SEARCHED: 0
ID: 55   NAME: SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s   TIMES SEARCHED: 0
ID: 58   NAME: SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5'', 7mm   TIMES SEARCHED: 0
ID: 59   NAME: SSD Samsung 860 EVO, 1TB, SATA III, M.2   TIMES SEARCHED: 1
ID: 56   NAME: SSD para Servidor Lenovo Thinksystem S4500, 480GB, SATA III, 3.5'', 7mm   TIMES SEARCHED: 2
ID: 52   NAME: SSD Western Digital WD Blue 3D NAND, 2TB, M.2     TIMES SEARCHED: 5
ID: 50   NAME: SSD Crucial MX500, 1TB, SATA III, M.2     TIMES SEARCHED: 7
ID: 49   NAME: Kit SSD Kingston KC600, 1TB, SATA III, 2.5, 7mm   TIMES SEARCHED: 10
ID: 51   NAME: SSD Kingston UV500, 480GB, SATA III, mSATA        TIMES SEARCHED: 11
ID: 48   NAME: SSD Kingston A2000 NVMe, 1TB, PCI Express 3.0, M2         TIMES SEARCHED: 27

 LEAST SEARCHED USB
ID: 60   NAME: Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB), Non-ECC, CL16, XMP       TIMES SEARCHED: 0
ID: 61   NAME: Kit Memoria RAM Corsair Vengeance LPX DDR4, 2400MHz, 32GB, Non-ECC, CL16  TIMES SEARCHED: 0

 LEAST SEARCHED SCREENS
ID: 62   NAME: Makena Smart TV LED 32S2 32'', HD, Widescreen, Gris       TIMES SEARCHED: 0
ID: 64   NAME: Samsung TV LED LH43QMREBGCXGO 43, 4K Ultra HD, Widescreen, Negro  TIMES SEARCHED: 0
ID: 65   NAME: Samsung Smart TV LED UN70RU7100FXZX 70, 4K Ultra HD, Widescreen, Negro    TIMES SEARCHED: 0
ID: 68   NAME: Makena Smart TV LED 40S2 40'', Full HD, Widescreen, Negro         TIMES SEARCHED: 0
ID: 69   NAME: Hisense Smart TV LED 40H5500F 39.5, Full HD, Widescreen, Negro    TIMES SEARCHED: 0
ID: 71   NAME: Samsung Smart TV LED UN32J4290AF 32, HD, Widescreen, Negro        TIMES SEARCHED: 0
ID: 72   NAME: Hisense Smart TV LED 50H8F 49.5, 4K Ultra HD, Widescreen, Negro   TIMES SEARCHED: 0
ID: 70   NAME: Samsung Smart TV LED 43, Full HD, Widescreen, Negro       TIMES SEARCHED: 1
ID: 63   NAME: Seiki TV LED SC-39HS950N 38.5, HD, Widescreen, Negro      TIMES SEARCHED: 4
ID: 73   NAME: Samsung Smart TV LED UN55TU7000FXZX 55, 4K Ultra HD, Widescreen, Negro/Gris       TIMES SEARCHED: 4

 LEAST SEARCHED SPEAKERS
ID: 75   NAME: Lenovo Barra de Sonido, Alámbrico, 2.5W, USB, Negro       TIMES SEARCHED: 0
ID: 77   NAME: Verbatim Bocina Portátil Mini, Bluetooth, Inalámbrico, 3W RMS, USB, Blanco       TIMES SEARCHED: 0
```

Figura 7: Terminal mostrando los productos menos buscados por categoría

Por último, la terminal en la figura 8 nos muestra los 5 meses con más ganancias, los 5 meses con más ventas y finalmente los meses con el mayor ticket promedio.

```
 MOST PROFITABLE MONTHS
MONTH: April     PROFIT: $304,578.00     SALES: 72       AVERAGE: 4230.25
MONTH: February  PROFIT: $302,409.00     SALES: 41       AVERAGE: 7375.829268292683
MONTH: March     PROFIT: $189,473.00     SALES: 47       AVERAGE: 4031.340425531915
MONTH: May       PROFIT: $153,965.00     SALES: 35       AVERAGE: 4399.0
MONTH: January   PROFIT: $127,428.00     SALES: 52       AVERAGE: 2450.5384615384614

 MOST SALES PER MONTHS
MONTH: April     PROFIT: $304,578.00     SALES: 72       AVERAGE: 4230.25
MONTH: January   PROFIT: $127,428.00     SALES: 52       AVERAGE: 2450.5384615384614
MONTH: March     PROFIT: $189,473.00     SALES: 47       AVERAGE: 4031.340425531915
MONTH: February  PROFIT: $302,409.00     SALES: 41       AVERAGE: 7375.829268292683
MONTH: May       PROFIT: $153,965.00     SALES: 35       AVERAGE: 4399.0

 HIGHEST AVERAGE TICKET PER MONTH
MONTH: February  PROFIT: $302,409.00     SALES: 41       AVERAGE: 7375.829268292683
MONTH: August    PROFIT: $17,257.00      SALES: 3        AVERAGE: 5752.333333333333
MONTH: May       PROFIT: $153,965.00     SALES: 35       AVERAGE: 4399.0
MONTH: April     PROFIT: $304,578.00     SALES: 72       AVERAGE: 4230.25
MONTH: November  PROFIT: $4,209.00       SALES: 1        AVERAGE: 4209.0
```

Figura 8: Terminal mostrando los meses con mayores ganancias, ventas y ticket promedio

6. Anexos

```
"""
lifestore_searches = [id_search, id product]
lifestore_sales = [id_sale, id_product, score (from 1 to 5), date, refund (1 for
    true or 0 to false)]
lifestore_products = [id_product, name, price, category, stock]
"""

from lifestore_file import lifestore_products, lifestore_sales, lifestore_searches

#--------------------LOGIN-----------------------------------#
username = "Manager123"
password = "lif3stor3"
tries = 0; Acces = False

while not Acces:
    tries += 1

    if tries == 4:
        exit()
    if input('Username: ') == username and input('Password: ') == password:
        Acces = True
        print('Acces Granted')
    else:
        print(f'You have {3 - tries} tries lefts')

#-------------------------SALES-----------------------------------#
soldproduct = []; timesold = []

soldproduct = [sale[1] for sale in lifestore_sales]

for sale in lifestore_products: #Creating a nested list
    nested =[]
    timesold.append(nested)
    for k in range(1):
        nested.append(sale[0])
        nested.append(sale[1])
        nested.append(soldproduct.count(sale[0])) #Counting times sold
        nested.append(sale[-2])

def Sort(timesold): #timesold is [id_product, name, time_sold, category]
    timesold.sort(key = lambda x: x[2])
    return timesold

timesold = Sort(timesold) # Ascendant order

print("\n MOST SOLD PRODUCTS")
for i in [-1,-2,-3,-4,-5]:
    print( F'ID: {timesold[i][0]}\t NAME: {timesold[i][1]}\t SALES: {timesold[i
    ][2]}')

#print("\n LEAST SOLD PRODUCTS")

#for i in range(0,5):
```

```
#     print( F'ID: {timesold[i][0]}\t NAME: {timesold[i][1]}\t SALES: {timesold[i][2]}' )

#-------------------------SALES PER CATERGORY-----------------------------------#
categories= []
processors = []; gpus = []; motherboards = []; drives = []; usb = []; screens = []; speakers =[];headphones = []

categories= [item[-2] for item in lifestore_products]

categories = list(dict.fromkeys(categories))

for item in timesold:
    if categories[0] in item:
        processors.append(item[:3])
    elif categories[1] in item:
        gpus.append(item[:3])
    elif categories[2] in item:
        motherboards.append(item[:3])
    elif categories[3] in item:
        drives.append(item[:3])
    elif categories[4] in item:
        usb.append(item[:3])
    elif categories[5] in item:
        screens.append(item[:3])
    elif categories[6] in item:
        speakers.append(item[:3])
    elif categories[7] in item:
        headphones.append(item[:3])

print("\n LEAST SOLD PROCESSORS")
for i in range(5):
    print(f"ID: {processors[i][0]}\t NAME: {processors[i][1]}\t TIMES SOLD: {processors[i][-1]}")

print("\n LEAST SOLD GPUS")
for i in range(5):
    print(f"ID: {gpus[i][0]}\t NAME: {gpus[i][1]}\t TIMES SOLD: {gpus[i][-1]}")

print("\n LEAST SOLD MOTHERBOARDS")
for i in range(5):
    print(f"ID:{motherboards[i][0]}\tNAME:{motherboards[i][1]}\tTIMES SOLD:{motherboards[i][-1]}")

print("\n LEAST SOLD DRIVES")
for i in range(5):
    print(f"ID: {drives[i][0]}\t NAME: {drives[i][1]}\t TIMES SOLD: {drives[i][-1]}")

print("\n LEAST SOLD USB")
for i in range(2):
    print(f"ID: {usb[i][0]}\t NAME: {usb[i][1]}\t TIMES SOLD: {usb[i][-1]}")

print("\n LEAST SOLD SCREENS")
for i in range(5):
```

```
102     print(f"ID: {screens[i][0]}\t NAME: {screens[i][1]}\t TIMES SOLD: {screens[i
        ][-1]}")
103
104 print("\n LEAST SOLD SPEAKERS")
105 for i in range(5):
106     print(f"ID: {speakers[i][0]}\t NAME: {speakers[i][1]}\t TIMES SOLD: {speakers[i
        ][-1]}")
107
108 print("\n LEAST SOLD HEADPHONES")
109 for i in range(5):
110     print(f"ID:{headphones[i][0]}\tNAME:{headphones[i][1]}\tTIMES SOLD:{
        headphones[i][-1]}")
111
112 #-------------------------SEARCHES-----------------------------------#
113 searchedproduct = []; timesearched = []
114
115 searchedproduct = [search[1] for search in lifestore_searches]
116
117 for search in lifestore_products: #Creating a nested list
118     nested =[]
119     timesearched.append(nested)
120     for k in range(1):
121         nested.append(search[0]) #Saving ID
122         nested.append(search[1]) #Saving Name
123         nested.append(searchedproduct.count(search[0])) #Counting times searched
124         nested.append(search[-2]) #Saving Category
125
126 def Sort(timesearched):
127     timesearched.sort(key = lambda x: x[2])
128     return timesearched
129
130 timesearched = Sort(timesearched)
131
132 print("\n MOST SEARCHED PRODUCTS")
133 for i in [-1,-2,-3,-4,-5]:
134     print(F'ID:{timesearched[i][0]}\t NAME:{timesearched[i][1]}\t SEARCHES:{
        timesearched[i][-2]}')
135
136 #print("\n LEAST SEARCHED PRODUCTS")
137 #for i in range(0,10):
138 #    print(F'ID:{timesearched[i][0]}\t NAME:{timesearched[i][1]}\t SEARCHES:{
        timesearched[i][-1]}')
139
140 #-------------------------SEARCHES PER CATERGORY---------------------------------#
141 processors2 = []; gpus2 = []; motherboards2 = []; drives2 = []; usb2 = []; screens2
        = []; speakers2 = []; headphones2 = []
142
143 for item in timesearched:
144     if categories[0] in item:
145         processors2.append(item[:3])
146     elif categories[1] in item:
147         gpus2.append(item[:3])
148     elif categories[2] in item:
149         motherboards2.append(item[:3])
150     elif categories[3] in item:
```

```
151             drives2.append(item[:3])
152         elif categories[4] in item:
153             usb2.append(item[:3])
154         elif categories[5] in item:
155             screens2.append(item[:3])
156         elif categories[6] in item:
157             speakers2.append(item[:3])
158         elif categories[7] in item:
159             headphones2.append(item[:3])
160 
161 print("\n LEAST SEARCHED PROCESSORS")
162 for i in range(9):
163     print(f"ID: {processors2[i][0]}\t NAME: {processors2[i][1]}\t TIMES SEARCHED: {
        processors2[i][-1]}")
164 
165 print("\n LEAST SEARCHED GPUS")
166 for i in range(10):
167     print(f"ID: {gpus2[i][0]}\t NAME: {gpus2[i][1]}\t TIMES SEARCHED: {gpus2[i][2]}
        ")
168 
169 print("\n LEAST SEARCHED MOTHERBOARDS")
170 for i in range(10):
171     print(f"ID:{motherboards2[i][0]}\t NAME:{motherboards2[i][1]}\t TIMES
        SEARCHED:{motherboards2[i][-1]}")
172 
173 print("\n LEAST SEARCHED DRIVES")
174 for i in range(10):
175     print(f"ID: {drives2[i][0]}\t NAME: {drives2[i][1]}\t TIMES SEARCHED: {drives2[
        i][-1]}")
176 
177 print("\n LEAST SEARCHED USB")
178 for i in range(2):
179     print(f"ID: {usb2[i][0]}\t NAME: {usb2[i][1]}\t TIMES SEARCHED: {usb2[i][-1]}")
180 
181 print("\n LEAST SEARCHED SCREENS")
182 for i in range(10):
183     print(f"ID: {screens2[i][0]}\t NAME: {screens2[i][1]}\t TIMES SEARCHED: {
        screens2[i][-1]}")
184 
185 print("\n LEAST SEARCHED SPEAKERS")
186 for i in range(10):
187     print(f"ID: {speakers2[i][0]}\t NAME: {speakers2[i][1]}\t TIMES SEARCHED: {
        speakers2[i][-1]}")
188 
189 print("\n LEAST SEARCHED HEADPHONES")
190 for i in range(10):
191     print(f"ID:{headphones2[i][0]}\t NAME:{headphones2[i][1]}\t TIMES SEARCHED:{
        headphones2[i][-1]}")
192 
193 
194 #----------------------REVIEWS---------------------------#
195 tempsum=0; totalscore = []; averagescore = []
196 
197 for i in range(0,len(lifestore_products)):
198     for k in range(0,len(lifestore_sales)):
```

```
199         if lifestore_sales[k][1]==timesold[i][0]:
200             tempsum += lifestore_sales[k][2] #Adding up review scores
201 
202     totalscore.append(tempsum)
203     tempsum=0
204 
205 for product in lifestore_products:
206     for review in lifestore_sales:
207         if product[0]==review[1]:
208             tempsum += review[2] #Adding up review scores
209 
210     totalscore.append(tempsum)
211     tempsum=0
212 
213  #timesold is [id_product, name, time_sold, category]
214 
215 for review in lifestore_products: #Creating a nested list
216     nested=[]
217     if timesold[review[0]-1][2] > 0: #Obtaining reviews from products sold at least
      once
218         averagescore.append(nested)
219         for k in range(1):
220             nested.append(review[0])
221             nested.append(review[1])
222             nested.append(timesold[review[0]-1][2]) #Times reviewed
223             nested.append(totalscore[review[0]-1]/timesold[review[0]-1][2])
224             #Obtaining average
225 
226 for review in timesold: #Creating a nested list
227     nested=[]
228     if review[2] > 0: #Obtaining reviews from products sold at least once
229         averagescore.append(nested)
230         for k in range(1):
231             nested.append(review[0])
232             nested.append(review[1])
233             nested.append(review[2]) #Times reviewed
234             nested.append(totalscore[review[0]-1]/review[2]) #Obtaining average
235 
236 def Sort(averagescore):
237     averagescore.sort(key = lambda x: x[-1])
238     return averagescore
239 
240 averagescore = Sort(averagescore)
241 
242 print("\n BEST RATED PRODUCTS")
243 for i in range(-1,-11,-1):
244     print( F'ID: {averagescore[i][0]}\t NAME: {averagescore[i][1]}\t SCORE: {
    averagescore[i][-1]}\t TIMES REVIEWED:{averagescore[i][2]}')
245 
246 print("\n WORST RATED PRODUCTS")
247 for i in range(0,10):
248     print( F'ID: {averagescore[i][0]}\t NAME: {averagescore[i][1]}\t SCORE: {
    averagescore[i][-1]}\t TIMES REVIEWED:{averagescore[i][2]}')
249 
250 #----------------------SALES PER MONTH----------------------------#
```

```
from datetime import datetime; import calendar
date = [sale[3] for sale in lifestore_sales]
date.sort(key = lambda date: datetime.strptime(date, '%d/%m/%Y')) # Sort the dates in order

month =calendar.month_name[1:]

refundeditem = []; totalrefunds = 0; averageticket = []
totalsales = 0; monthsales = [0]*12;  monthprofit = [0]*12

soldproduct = [sale[1] for sale in lifestore_sales]

for i in range(0,len(lifestore_sales)):
    if int(lifestore_sales[i][-1]) == 0:                    #Verifying it was not a refund
        totalsales+=lifestore_products[soldproduct[i]][2]    #Adding up the total sales
        if "/01/" in date[i]:
            monthprofit[0]+=lifestore_products[soldproduct[i]][2]
            monthsales[0]+=1
        elif "/02/" in date[i]:
            monthprofit[1]+=lifestore_products[soldproduct[i]][2]
            monthsales[1]+=1
        elif "/03/" in date[i]:
            monthprofit[2]+=lifestore_products[soldproduct[i]][2]
            monthsales[2]+=1
        elif "/04/" in date[i]:
            monthprofit[3]+=lifestore_products[soldproduct[i]][2]
            monthsales[3]+=1
        elif "/05/" in date[i]:
            monthprofit[4]+=lifestore_products[soldproduct[i]][2]
            monthsales[4]+=1
        elif "/06/" in date[i]:
            monthprofit[5]+=lifestore_products[soldproduct[i]][2]
            monthsales[5]+=1
        elif "/07/" in date[i]:
            monthprofit[6]+=lifestore_products[soldproduct[i]][2]
            monthsales[6]+=1
        elif "/08/" in date[i]:
            monthprofit[7]+=lifestore_products[soldproduct[i]][2]
            monthsales[7]+=1
        elif "/09/" in date[i]:
            monthprofit[8]+=lifestore_products[soldproduct[i]][2]
            monthsales[8]+=1
        elif "/10/" in date[i]:
            monthsales[9]+=lifestore_products[soldproduct[i]][2]
            monthsales[9]+=1
        elif "/11/" in date[i]:
            monthprofit[10]+=lifestore_products[soldproduct[i]][2]
            monthsales[10]+=1
        elif "/12/" in date[i]:
            monthprofit[11]+=lifestore_products[soldproduct[i]][2]
            monthsales[11]+=1
    else:
        refundeditem.append(soldproduct[i])                 #ID product refunded
```

```
303         totalrefunds+=lifestore_products[soldproduct[i]][2] #Total lost in refunds
304 
305 for i in range(12):
306     if monthsales[i] > 0:
307         averageticket.append(monthprofit[i]/monthsales[i]) #Obtaining average
308     else:
309         averageticket.append(0)
310 
311 salesxmonth = [list(l) for l in zip(month, monthprofit, monthsales, averageticket)]
312 
313 def Sort(salesxmonth): #Sort for profit
314     salesxmonth.sort(key = lambda x: x[1])
315     return salesxmonth
316 salesxmonth = Sort(salesxmonth)
317 
318 print("\n MOST PROFITABLE MONTHS")
319 for i in [-1,-2,-3,-4,-5]:
320     print( F'MONTH: {salesxmonth[i][0]}\t PROFIT: {"${:,.2f}".format(salesxmonth[i
    ][1])}\t SALES: {salesxmonth[i][2]}\t AVERAGE: {salesxmonth[i][-1]}' )
321 
322 def Sort(salesxmonth): #Sort for sales
323     salesxmonth.sort(key = lambda x: x[2])
324     return salesxmonth
325 salesxmonth = Sort(salesxmonth)
326 
327 print("\n MOST SALES PER MONTHS")
328 for i in [-1,-2,-3,-4,-5]:
329     print( F'MONTH: {salesxmonth[i][0]}\t PROFIT: {"${:,.2f}".format(salesxmonth[i
    ][1])}\t SALES: {salesxmonth[i][2]}\t AVERAGE: {salesxmonth[i][-1]}' )
330 
331 def Sort(salesxmonth): #Sort for average ticket
332     salesxmonth.sort(key = lambda x: x[-1])
333     return salesxmonth
334 salesxmonth = Sort(salesxmonth)
335 
336 print("\n HIGHEST AVERAGE TICKET PER MONTH")
337 for i in [-1,-2,-3,-4,-5]:
338     print( F'MONTH: {salesxmonth[i][0]}\t PROFIT: {"${:,.2f}".format(salesxmonth[i
    ][1])}\t SALES: {salesxmonth[i][2]}\t AVERAGE: {salesxmonth[i][-1]}' )
339 
340 print(F'\n TOTAL PROFIT BETWEEN {date[0]} AND {date[-1]}: {"${:,.2f}".format(
    totalsales)}')
```

Código 17: Final del Proyecto-01