EMTECH
Emerging Technologies Institute

Reporte Proyecto 1

Introducción a Python

Presentado por:
Flores Luna Luis Eduardo

Código asociado al código en:

https://github.com/eduar2flores/REPORTE-01-FLORES-LUNA---LUIS-EDUARDO

INDICE

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.

DESCRIPCIÓN DEL CASO

LifeStore es una tienda virtual que maneja una amplia gama de artículos, recientemente, la Gerencia de ventas, se percató que la empresa tiene una importante acumulación de inventario.

Asimismo, se ha identificado una reducción en las búsquedas de un grupo importante de productos, lo que ha redundado en una disminución sustancial de sus ventas del último trimestre.

CONSIGNA

1) Productos más vendidos y productos rezagados a partir del análisis de las categorías con menores ventas y categorías con menores búsquedas.

2) Productos por reseña en el servicio a partir del análisis de categorías con mayores ventas y categorías con mayores búsquedas.

3) Sugerir una estrategia de productos a retirar del mercado así como sugerencia de cómo reducir la acumulación de inventario considerando los datos de ingresos y ventas mensuales.

Productos más vendidos y productos rezagados

PRODUCTOS MÁS VENDIDOS

1. SSD Kingston A400, 120GB, SATA III, 2.5'', 7mm"
2. 'Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth'
3. 'Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)'
4. 'Tarjeta Madre ASRock Micro ATX B450M Steel Legend, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD'
5. "SSD Adata Ultimate SU800, 256GB, SATA III, 2.5'', 7mm"

PRODUCTOS MÁS BUSCADOS

1. "SSD Kingston A400, 120GB, SATA III, 2.5'', 7mm"
2. SSD Adata Ultimate SU800, 256GB, SATA III, 2.5'', 7mm"
3. Tarjeta Madre ASUS micro ATX TUF B450M-PLUS GAMING, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD'
4. Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth'
5. Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire'
6. Logitech Audífonos Gamer G635 7.1, Alámbrico, 1.5 Metros, 3.5mm, Negro/Azul'
7. TV Monitor LED 24TL520S-PU 24, HD, Widescreen, HDMI, Negro'
8. Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)'
9. Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)'
10. SSD XPG SX8200 Pro, 256GB, PCI Express, M.2'

Productos más vendidos y productos rezagados

PRODUCTOS MENOS VENDIDOS

Categoria: Procesadores

1. 'Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)'
2. Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache',
3. 'Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake)',
4. Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generiación - Coffee Lake)'
5. Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)'

Categoria: Tarjetas de vídeo

1. Tarjeta de Video EVGA NVIDIA GeForce GT 710, 2GB 64-bit GDDR3, PCI Express 2.0',
2. Tarjeta de Video EVGA NVIDIA GeForce GTX 1660 Ti SC Ultra Gaming, 6GB 192-bit GDDR6, PCI 3.0',
3. Tarjeta de Video EVGA NVIDIA GeForce RTX 2060 SC ULTRA Gaming, 6GB 192-bit GDDR6, PCI Express 3.0',
4. Tarjeta de Video Gigabyte AMD Radeon R7 370 OC, 2GB 256-bit GDDR5, PCI Express 3.0',
5. Tarjeta de Video Gigabyte NVIDIA GeForce GTX 1650 OC Low Profile, 4GB 128-bit GDDR5, PCI Express 3.0 x16'

Categoria: Tarjetas madre

1. Tarjeta Madre AORUS ATX Z390 ELITE, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel',
2. Tarjeta Madre ASRock Z390 Phantom Gaming 4, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel\xa0',
3. Tarjeta Madre ASUS ATX ROG STRIX B550-F GAMING WI-FI, S-AM4, AMD B550, HDMI, max. 128GB DDR4 para AMD'
4. Tarjeta Madre Gigabyte micro ATX Z390 M GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel\xa0'
5. Tarjeta Madre Gigabyte micro ATX Z490M GAMING X (rev. 1.0), Intel Z490, HDMI, 128GB DDR4 para Intel'

Categoria: Memorias USB

1. 'Kit Memoria RAM Corsair Vengeance LPX DDR4, 2400MHz, 32GB, Non-ECC, CL16',
2. 'Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB), Non-ECC, CL16, XMP'

Productos más vendidos y productos rezagados

PRODUCTOS MENOS VENDIDOS

Categoria: Discos duros

1. SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2'
2. SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s',
3. SSD para Servidor Lenovo Thinksystem S4500, 480GB, SATA III, 3.5'', 7mm",
4. SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5'', 7mm",
5. SSD Samsung 860 EVO, 1TB, SATA III, M.2']]

Categoria: Pantallas

1. Makena Smart TV LED 32S2 32'', HD, Widescreen, Gris"
2. 'Seiki TV LED SC-39HS950N 38.5, HD, Widescreen, Negro'
3. 'Samsung TV LED LH43QMREBGCXGO 43, 4K Ultra HD, Widescreen, Negro'
4. 'Samsung Smart TV LED UN70RU7100FXZX 70, 4K Ultra HD, Widescreen, Negro'
5. "Makena Smart TV LED 40S2 40'', Full HD, Widescreen, Negro"

Categoria: Bocinas

1. 'Lenovo Barra de Sonido, Alámbrico, 2.5W, USB, Negro',
2. 'Acteck Bocina con Subwoofer AXF-290, Bluetooth, Inalámbrico, 2.1, 18W RMS, 180W PMPO, USB, Negro'
3. Verbatim Bocina Portátil Mini, Bluetooth, Inalámbrico, 3W RMS, USB, Blanco'
4. 'Ghia Bocina Portátil BX300, Bluetooth, Inalámbrico, 40W RMS, USB, Rojo
5. Resistente al Agua', 'Naceb Bocina Portátil NA-0301, Bluetooth, Inalámbrico, USB 2.0, Rojo']]

Categoria: Audífonos

1. 'ASUS Audífonos Gamer ROG Theta 7.1, Alámbrico, USB C, Negro'
2. 'Acer Audífonos Gamer Galea 300, Alámbrico, 3.5mm, Negro'
3. 'Audífonos Gamer Balam Rush Orphix RGB 7.1, Alámbrico, USB, Negro'
4. 'Energy Sistem Audífonos con Micrófono Headphones 1, Bluetooh, Inalámbrico, Negro/Grafito'
5. 'Genius GHP-400S Audífonos, Alámbrico, 1.5 Metros, Rosa'], Rojo']]

Productos más vendidos y productos rezagados

PRODUCTOS MENOS BUSCADOS

Categoria: Procesadores

1. Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)'
2. 'Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache',
3. Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake)'
4. 'Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generiación - Coffee Lake)'
5. 'Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth'
6. Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)'
7. 'Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)'
8. 'Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire'
9. 'Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth']]

Categoria: Tarjetas de vídeo

1. ['Tarjeta de Video EVGA NVIDIA GeForce GT 710, 2GB 64-bit GDDR3, PCI Express 2.0'
2. 'Tarjeta de Video EVGA NVIDIA GeForce RTX 2060 SC ULTRA Gaming, 6GB 192-bit GDDR6, PCI Express 3.0',
3. 'Tarjeta de Video Gigabyte NVIDIA GeForce GTX 1650 OC Low Profile, 4GB 128-bit GDDR5, PCI Express 3.0 x16'
4. 'Tarjeta de Video Gigabyte NVIDIA GeForce RTX 2060 SUPER WINDFORCE OC, 8 GB 256 bit GDDR6, PCI Express x16 3.0'
5. 'Tarjeta de Video MSI Radeon X1550, 128MB 64 bit GDDR2, PCI Express x16'
6. 'Tarjeta de Video PNY NVIDIA GeForce RTX 2080, 8GB 256-bit GDDR6, PCI Express 3.0\xa0'
7. 'MSI GeForce 210, 1GB GDDR3, DVI, VGA, HDCP, PCI Express 2.0'
8. 'Tarjeta de Video VisionTek AMD Radeon HD5450, 2GB GDDR3, PCI Express x16'
9. 'Tarjeta de Video Asus NVIDIA GeForce GTX 1050 Ti Phoenix, 4GB 128-bit GDDR5, PCI Express 3.0']]

Categoria: Tarjetas madre

1. Tarjeta Madre AORUS ATX Z390 ELITE, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel'
2. 'Tarjeta Madre ASRock Z390 Phantom Gaming 4, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel\xa0',
3. 'Tarjeta Madre ASUS ATX PRIME Z390-A, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel\xa0'
4. 'Tarjeta Madre ASUS ATX ROG STRIX B550-F GAMING WI-FI, S-AM4, AMD B550, HDMI, max. 128GB DDR4 para AMD'
5. 'Tarjeta Madre Gigabyte micro ATX Z490M GAMING X (rev. 1.0), Intel Z490, HDMI, 128GB DDR4 para Intel',
6. 'Tarjeta Madre ASRock ATX Z490 STEEL LEGEND, S-1200, Intel Z490, HDMI, 128GB DDR4 para Intel'
7. 'Tarjeta Madre Gigabyte Micro ATX H310M DS2 2.0, S-1151, Intel H310, 32GB DDR4 para Intel\xa0'
8. 'Tarjeta Madre ASUS micro ATX Prime H370M-Plus/CSM, S-1151, Intel H370, HDMI, 64GB DDR4 para Intel',
9. 'Tarjeta Madre ASUS ATX ROG STRIX Z390-E GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel']]

Categoria: Memorias USB

1. 'Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB)
2. Non-ECC, CL16, XMP', 'Kit Memoria RAM Corsair Vengeance

Productos más vendidos y productos rezagados

PRODUCTOS MENOS BUSCADOS

Categoria: Discos duros

1. ['SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2'
2. 'SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s'
3. "SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5'', 7mm"
4. 'SSD Samsung 860 EVO, 1TB, SATA III, M.2'
5. "SSD para Servidor Lenovo Thinksystem S4500, 480GB, SATA III, 3.5'', 7mm"
6. 'SSD Western Digital WD Blue 3D NAND, 2TB, M.2'
7. 'SSD Crucial MX500, 1TB, SATA III, M.2'
8. 'Kit SSD Kingston KC600, 1TB, SATA III, 2.5, 7mm'
9. 'SSD Kingston UV500, 480GB, SATA III, mSATA']

Categoria: Pantallas

1. Makena Smart TV LED 32S2 32'', HD, Widescreen, Gris"
2. 'Samsung TV LED LH43QMREBGCXGO 43, 4K Ultra HD, Widescreen, Negro'
3. 'Samsung Smart TV LED UN70RU7100FXZX 70, 4K Ultra HD, Widescreen, Negro'
4. "Makena Smart TV LED 40S2 40'', Full HD, Widescreen, Negro"
5. 'Hisense Smart TV LED 40H5500F 39.5, Full HD, Widescreen, Negro'
6. 'Samsung Smart TV LED UN32J4290AF 32, HD, Widescreen, Negro'
7. 'Hisense Smart TV LED 50H8F 49.5, 4K Ultra HD, Widescreen, Negro'
8. 'Samsung Smart TV LED 43, Full HD, Widescreen, Negro'
9. 'Seiki TV LED SC-39HS950N 38.5, HD, Widescreen, Negro']

Categoria: Bocinas

1. ['Lenovo Barra de Sonido, Alámbrico, 2.5W, USB, Negro'
2. 'Verbatim Bocina Portátil Mini, Bluetooth, Inalámbrico, 3W RMS, USB, Blanco'
3. Ghia Bocina Portátil BX300, Bluetooth, Inalámbrico, 40W RMS, USB, Rojo - Resistente al Agua'
4. 'Naceb Bocina Portátil NA-0301, Bluetooth, Inalámbrico, USB 2.0, Rojo'
5. 'Ghia Bocina Portátil BX900, Bluetooth, Inalámbrico, 2.1 Canales, 34W, USB, Negro - Resistente al Agua'
6. Ghia Bocina Portátil BX400, Bluetooth, Inalámbrico, 8W RMS, USB, Negro'
7. 'Ghia Bocina Portátil BX500, Bluetooth, Inalámbrico, 10W RMS, USB, Gris'
8. 'Ghia Bocina Portátil BX800, Bluetooth, Inalámbrico, 2.1 Canales, 31W, USB, Negro'
9. 'Acteck Bocina con Subwoofer AXF-290, Bluetooth, Inalámbrico, 2.1, 18W RMS, 180W PMPO, USB, Negro'

Categoria: Audífonos

1. ASUS Audífonos Gamer ROG Theta 7.1, Alámbrico, USB C, Negro'
2. 'Acer Audífonos Gamer Galea 300, Alámbrico, 3.5mm, Negro'
3. Audífonos Gamer Balam Rush Orphix RGB 7.1, Alámbrico, USB, Negro'
4. 'Energy Sistem Audífonos con Micrófono Headphones 1, Bluetooh, Inalámbrico, Negro/Grafito'
5. 'Getttech Audífonos con Micrófono Sonority, Alámbrico, 1.2 Metros, 3.5mm, Negro/Rosa'
6. 'Klip Xtreme Audífonos Blast, Bluetooth, Inalámbrico, Negro/Verde'
7. 'Ginga Audífonos con Micrófono GI18ADJ01BT-RO, Bluetooth, Alámbrico/Inalámbrico, 3.5mm, Rojo'
8. 'Genius GHP-400S Audífonos, Alámbrico, 1.5 Metros, Rosa'
9. 'logear Audífonos Gamer GHG601, Alámbrico, 1.2 Metros, 3.5mm, Negro'

Productos por reseña en el servicio

PRODUCTOS CON MEJORES RESEÑAS

1. 'Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth'
2. 'Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)'
3. 'Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generiación - Coffee Lake)'
4. 'Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)'
5. 'Tarjeta de Video ASUS NVIDIA GeForce GTX 1660 SUPER EVO OC, 6GB 192-bit GDDR6, PCI Express x16 3.0']

PRODUCTOS CON PEORES RESEÑAS

1. 'Tarjeta Madre Gigabyte micro ATX GA-H110M-DS2, S-1151, Intel H110, 32GB DDR4 para Intel'
2. 'Tarjeta de Video Gigabyte NVIDIA GeForce GT 1030, 2GB 64-bit GDDR5, PCI Express x16 3.0'
3. 'Tarjeta Madre ASRock Z390 Phantom Gaming 4, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel\xa0'
4. 'SSD XPG SX8200 Pro, 256GB, PCI Express, M.2'
5. 'Energy Sistem Audífonos con Micrófono Headphones 1, Bluetooh, Inalámbrico, Negro/Grafito'

Total de ingresos y ventas promedio mensuales, total anual y meses con más ventas al año

Ventas promedio mensuales
22.83

Ingresos promedio mensuales
$61,493

Ventas anuales en total
274

Ingresos anuales en total
$737,916

Meses con más ventas

1. **Abril**
2. **Enero**
3. **Marzo**
4. **Febrero**
5. **Mayo**

Desarrollo del código asociado al análisis

Una de las peticiones es generar un programa para asegurarnos que sólo aquel que tenga un nombre de usuario y una contraseña válidas, podrá ser capaz de acceder a la información. Como la función debe recibir información del usuario, utilizaremos la función input().

Ahora, deberíamos tener una base de datos que tenga el nombre del usuario y la contraseña asociada, pero la relación debe ser uno a uno entre estas dos variables. Un diccionario sería ideal para establecer estas variables.

```
Credenciales={'Carlos':'carl02', 'Jimmy':'EmTech', 'Monse':'HIMYM'}
```

Ahora, vamos a solicitar los datos de ingreso más de una vez, por lo que un ciclo for o uno while podría ser de utilidad. En este caso, nos vamos a decidir por el ciclo While, el cual se va a repetir hasta que la condición sea falsa. Podemos establecer una variable binaria que cambie, y detenga el ciclo while si es que el usuario y la contraseña son válidas:

```
Acceso= False
while not Acceso:
    if ----------:
        Acceso= True
```

Para poder dar retroalimentación al usuario, vamos a utilizar mensajes dependiendo del error que cometa. Por ejemplo, primero vamos a solicitar el nombre de usuario, y si no es uno de los nombres válidos, devolveremos un mensaje de usuario incorrecto. Los nombres de usuario del diccionario está representado por la lista: Credenciales.keys()
Ahora, si el nombre de usuario es correcto, procedemos a solicitar la contraseña.

En cada una de las iteraciones vamos a ir aumentando un contador llamado "Intentos" e introducimos un condicional: "si la variable Intentos es mayor a 3, entonces el programa terminará con el programa mediante la función exit().

```
Acceso=False
Intentos=0

while not Acceso:
    Intentos+=1
    print("Favor de introducir el nombre de usuario")
    nombre_de_usuario=input("Nombre de usuario:")
    if nombre_de_usuario in Credenciales.keys():
        print("Favor de introducir la contraseña:")
        contraseña=input("Contraseña:")
        if contraseña == Credenciales[nombre_de_usuario]:
            Acceso=True
        else:
            print("Contraseña incorrecta, pruebe de nuevo")
            print(f"Lleva {Intentos} intentos, el límite es de 3")
    else:
        print("Usuario no encontrado, pruebe de nuevo")
        print(f"Lleva {Intentos} intentos, el límite es de 3")
    if Intentos > 3:
        exit()
```

Desarrollo del código asociado al análisis

Es primordial identificar los productos más vendidos y los productos rezagados a partir del análisis de las categorías con menores ventas y categorías con menores búsquedas.

En este caso, vamos a identificar los 5 productos con más ventas y los 10 productos más buscados.

- **Identificar a los 5 productos con más ventas**

En primer lugar, hay que distinguir aquellos productos que fueron realmente vendidos, es decir, que no sufrieron de una devolución. Esta información está plasmada en la variable **refund** en cada una de las transacciones representadas por una lista, dentro de la variable **lifestore_sales**. Si la variable refund es igual a 0, implica que no existió una devolución, y la venta fue llevada a cabo.

Ahora, para estimar los 5 productos más vendidos hay que tener conocimiento del número de ventas de cada producto; con este fin, vamos a iterar la lista de productos (**lifestore_products**), de modo que en cada iteración estemos trabajando con un producto distinto.

Utilizamos un ciclo for para iterar los elementos de lifestore_products; cada iteración nos provee de un producto a la vez, definido por la variable **producto** , de la cual podemos obtener el id del producto (**producto[0]**). Ahora, dentro de esta iteración, vamos a implementar otro *ciclo for* con la intención de iterar sobre la variable **lifestore_sales**, la cual contiene la información de todas las ventas realizadas (cada iteración representa una venta).

```
for producto in lifestore_products:
        id_del_producto= producto[0]
        for venta in lifestore_sales:
                #En este punto, estamos considerando a un producto y a una de las múltiples ventas.
```

En este punto, podemos implementar una condición: "si la id_product del producto actual, es igual a la id_product asociada a una de las ventas, y además, la venta no implicó una devolución, entonces podemos asegurar que ese producto tuvo una venta exitosa". Esto está definido por la siguiente condición:

```
if id_del_producto == venta[1] and venta[4]==0:
```

Ahora, hay que registrar que se realizó una venta de ese producto, por lo tanto, dentro de cada iteración de la lista lifestore_products vamos a establecer una variable que va a fungir como contador, el cual va a aumentar si se cumplen las condiciones anteriores:

```
for producto in lifestore_products:
        ventas=0
        for ----------:
                if ----------:
                        ventas+=1
```

Desarrollo del código asociado al análisis

Luego de recorrer cada una de las ventas de lifestore_sales, hay que registrar el id_product y su número de ventas asociadas en una variable que perdure antes de pasar a la siguiente iteración de lifestore_products. Así, es necesario establecer una lista que almacene todos estos valores, y establecerla antes del primer ciclo for:

```
ventas_por__producto=[]
for producto in lifestore_products:
        id_del_producto=producto[0]
        ventas=0
        for ----------:
            if ----------:
                ventas+=1
        ventas_por_producto.append([id_del_producto,ventas])
```

Así, al final, la lista ventas_por_producto estará conformada por varias lista, donde cada una tendrá el id de un producto y el número de ventas asociadas a ese producto.

Ahora, necesitamos ordenar la lista ventas_por_producto en función del número de ventas, es decir, del elemento con índice 1 de cada una de las listas que conforman ventas_por_producto. Esto puede realizarse fácilmente con la función sorted(), como se ve a continuación:

```
ventas_ordenadas= sorted(ventas_por_producto, key= lambda x: x[1], reverse=True)
```

El parámetro "key=" nos permite especificar una función que nos diga cómo queremos ordenar nuestra lista: en este caso, la función lambda definida nos retorna el valor con índice 1 de cada una de nuestras listas, de modo, que este valor se utiliza para ordenar la lista; el parámetro "reverse=True" hace que sea en orden descendente el acomodo de los valores, es decir, de mayor a menor.

Así, para obtener los 5 productos más vendidos basta con recuperar los 5 primeros productos en esta lista.

- **Identificar los 10 productos con mayores búsquedas.**

El enfoque para resolver este problema es exactamente el mismo; solamente que en este caso tenemos que cuantificar el número de búsquedas, por lo que necesitamos trabajar con la lista **lifestore_searches**, que contiene las búsquedas y el id del producto asociada a cada búsqueda.

Así, en lugar de iterar sobre lifestore_sales para trabajar con cada una de las ventas, vamos a iterar sobre lifestore_searches para trabajar con cada una de las búsquedas. Además, nuestro contador no será "ventas", será "búsquedas".

```
busquedas=0
for busqueda in lifestore_searches:
```

Desarrollo del código asociado al análisis

Otro de los cambios, es que ahora no hay que tener cuidado con las devoluciones, así, simplemente hay que asegurarnos que "el id del producto que está siendo considerado en una iteración, coincida con el id del producto asociado a cada una de las diferentes busquedas"; es decir:

```
if id_del_producto == busquedas[1]:
```

El resto del código es similar, solamente hay algunos cambios en los nombres que pueden recibir las variables. Así, para obtener a los 10 productos más buscados, basta con recuperar los primeros 10 elementos de la lista "busquedas_ordenadas".

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

En este caso, se piden los 5 productos con menores ventas por categoría, por lo que primero es determinar las categorias en las que se pueden agrupar todos los productos dentro de lifestore_products.

Para esto, generamos una lista vacía, luego vamos a iterar sobre la lista lifestore_products, de modo que en cada iteración estamos abarcando uno de los productos: así, podemos fijarnos en la categoría a la que pertenece, "si esa categoría no está en la lista vacía, la agregamos". De esta forma, al terminar de iterar ya tendremos todas las categorias posibles en la lista categorias.

```
categorias=[]
for producto in lifestore_products:
        categoria=producto[3]
        if not categoria in categorias:
                categorias.append(categoria)
```

Ahora tenemos todas las categorias únicas.

Además, sabemos cómo generar una lista que tenga los id del producto, y la cantidad de ventas o búsquedas asociadas a ese producto; esta lista se puede ordenar de forma descendente para obtener los productos más vendidos, buscados al fijarnos en los primeros elementos. De forma similar, podríamos ordenar esta lista de forma ascendente, para que los productos menos vendidos o menos búscados sean los primeros elementos de esa lista. ¿Cómo se realiza ese cambio? Sencillo, con el parámetro "reverse=":

```
sorted(lista_por_ordenar, key= lambda x: x[1], reverse=False)
```

Podemos aprovechar todo lo anterior para obtener los 5 productos con menos ventas, por categoria.

Desarrollo del código asociado al análisis

Podemos armar una lista vacía que va a almacenar el nombre de la categoria, y los 5 productos con menos ventas de dicha categoría.

```
productos_menos_vendidos_por_categoria=[]
for categoria in categorias:
    #Usamos los comandos  descritos en las secciones anteriores
    productos_menos_vendidos_por_categoria.append([categoria, productos menos vendidos])
```

Sin embargo, hay un problema, porque la categoría "memorias usb" únicamente cuenta con dos productos en los registros de lifestore_products, de modo que no podemos obtener 5 productos con menos ventas en esta categoría.

Por lo tanto, la categoria "memorias usb" solamente podrá recuperar 2 productos. Debido a este caso específico, necesitamos aplicar el mismo código pero en dos situaciones diferentes: usar el método de las secciones anteriores para recuperar sólamente 2 productos cuando se trate de la categoria "memorias usb", y usar el mismo método, pero recuperando 5 productos cuando se trate del resto de categorías. Eso lo podemos conseguir con un condicional:

```
for categoria in categorias:
    if categoria == "memorias usb":
        #Usamos los comandos ya descritos para obtener los 2 productos con menos ventas
    else:
        #Usamos los comandos ya descritos para obtener los 5 productos con menos ventas
```

Así, obtenemos los 5 productos con menos ventas por categoria.

- **Por categoría, generar un listado con los 9 productos con menores búsquedas**

En este caso, el procedimiento es exactamente igual al anterior, sin embargo, en lugar de iterar sobre las ventas, vamos iterar sobre las búsquedas almacenadas en la lista lifestore_searches. El procedimiento tienen las mismas restricciones para la categoria "memorias usb", y por lo tanto, sigue el esquema descrito anteriormente.

Desarrollo del código asociado al análisis

- **Mostrar dos listados de 5 productos cada una, un listado para productos con las mejores reseñas y otro para las peores, considerando los productos con devolución.**

En este caso buscamos los productos con mejores y peores reseñas; por lo tanto, hay que establecer una métrica para clasificar a los productos basados en sus reseñas. Una opción es utilizar el promedio de todas las reseñas asociadas a un producto, y es la métrica que usaremos.

Como necesitamos trabajar con las reseñas, utilizaremos la lista lifestore_sales, ya que el valor con índice 2 de cada elemento de esta lista es justamente la reseña asociada a cada una de las ventas.

Así, es necesario iterar sobre los diferentes productos presentes en la lista lifestore_products y luego iterar sobre la lista lifestore_sales para que a cada producto lo podamos comparar con las diferentes ventas y recuperar las reseñas únicamente de aquellas ventas que correspondan (usando un condicional, como lo hemos hecho anteriormente).

```
for producto in lifestore_products:
    id_producto=producto[0]
    for venta in lifestore_sales:
        if id_producto == venta[1]:
            reseña=venta[2]
```

Hay que notar, que debemos almacenar cada una de las reseñas de un producto, esto con la intención de poder promediar todas al final. Por lo tanto, en cada iteración de lifestore_products debemos generar una lista que los vaya almacenando, digamos: reseña_producto.

Así, en cada iteración de venta donde los id de los productos coincidan, vamos a recuperar la reseña y almacenarla en esta lista.

```
for producto in lifestore_products:
    id_producto=producto[0]
    reseña_producto=[]
    for venta in lifestore_sales:
        if id_producto == venta[1]:
            reseña=venta[2]
            reseña_producto.append(reseña)
```

Notese que a diferencia de los códigos anteriores, no estamos considerando una condición respecto a las devoluciones, ya que estamos considerando las reseñas de todos los productos: se hayan vendido o hayan sido devueltos.

Ahora, vamos a calcular el promedio de las reseñas almacenadas en reseña_producto para cada uno de los productos. Sin embargo, hay que tener cuidado, ya que existen algunos productos que no han sido vendidos nunca, y por lo tanto, no tienen alguna reseña asociada.

Desarrollo del código asociado al análisis

Los productos que no tienen ninguna venta asociada, no tendrán ninguna reseña y, por lo tanto, la lista reseña_producto estará vacía en este punto del código. Entonces podemos implementar un condicional relativo a la cantidad de elementos en la lista: "si no hay elementos en la lista continuemos con el siguiente producto; si tiene un elemento o más, calculemos el promedio de las reseñas y guardemos el id del producto y el promedio dentro de una lista.

```
if len(reseña_producto) >= 1:
        promedio=sum(reseña_producto)  /  len(reseña_producto)
        Reseñas_por_producto.append([id_producto, promedio])
else:
        continue
```

Así, luego de trabajar con todos los productos, tendremos una lista con listas cuyos elementos serán el ID del producto y el promedio de sus reseñas. Basta ordenarlo en orden descendente o ascendente para obtener los primeros 5 elementos y así adquirir los 5 productos con mejores y peores reseñas, respectivamente. Para eso, empleamos el mismo código de antes:

```
Puntuaciones_ordenadas=sorted(Reseñas_por_producto, key= lambda x: x[1], reverse=True)
```

- **Total de ingresos y ventas promedio mensuales, total anual y meses con más ventas al año**

Este punto es un poco polémico, pero entendemos que requieren:

- Calcular el promedio mensual de las ventas (número de ventas), y el promedio mensual del total de ingresos (dinero obtenido de esas ventas).
- Calcular el total anual de las ventas y de los ingresos.
- Obtener los meses con mayor número de ventas

- **Calcular el promedio mensual de las ventas.**

En este caso, necesitamos obtener las ventas realizadas en un determinado mes, en lugar de fijarnos de un producto determinado como en las secciones anteriores. Sin embargo, el enfoque es muy parecido. Lo primero es recuperar las fechas en las que cada una de las ventas fue efectuada.

Cada venta tiene una fecha asociada en su elemento con índice 3; sin embargo, esta fecha es una variable de tipo string con formato día/mes/año. Únicamente necesitamos el mes, así que podemos implementar un código que convierta cada string en una variable del tipo datetime, para luego extraer el mes de este tipo de variable utilizando el método *.month*.

El método para convertir un string a un datetime es la función datetime.strptime() de la librería datetime.

```
fecha=datetime.strptime(venta[3], '%d/%m/%Y')
```

Desarrollo del código asociado al análisis

El parámetro '%d/%m/%Y' indica el formato en el que el string representa la fecha.
Así, de cada venta podemos recuperar el mes de cada fecha:

```
month=fecha.month
```

De forma similar a lo que hemos hecho anteriormente, vamos a iterar sobre una lista que contenga los meses del año; luego iteramos cada una de las ventas contenidas en lifestore_sales y si el mes coincide con el mes bajo iteración y no hay devolución (refund=0), entonces aumentamos un contador definido en cada iteración asociada al mes. El código sería el siguiente:

```
meses=range(1,13) #Los meses son números del 1 al 12.
for mes in meses:
        ventas=0
        for venta in lifestore_sales:
            if venta[3]==mes and venta[4]==0:
                ventas+=1
```

Al final, únicamente agregamos el mes y el número de ventas en una lista que esté definida antes de empezar a iterar sobre meses.

```
Ventas_por_mes=[]
meses=range(1,13) #Los meses son números del 1 al 12.
for mes in meses:
        ventas=0
        for venta in lifestore_sales:
            if venta[3]==mes and venta[4]==0:
                ventas+=1
        Ventas_por_mes.append([mes,ventas])
```

Así, tenemos una lista que contiene el número de ventas asociado a su respectivo mes; sólo calculamos el promedio de estos números para obtener el promedio mensual de ventas.

- **Calcular el promedio mensual de los ingresos.**

Para esta tarea aplicamos el mismo enfoque de iterar sobre los diferentes meses y luego iterar lifestore_sales para asegurarnos que las ventas corresponden a determinado mes. Sin embargo, este método requiere un paso extra: "si ya sabemos que determinada venta corresponde a un mes, ahora iteramos sobre todos los productos posibles para buscar el ingreso que obtendremos del producto cuyo id corresponde con el id_product de la venta".

Esto lo logramos iterando la lista lifestore_products para irnos fijando en los productos, y "si el id del producto es igual al id_product de la venta, entonces recuperamos el precio de ese producto y lo guardamos en una lista".

Desarrollo del código asociado al análisis

Otra diferencia es que tenemos que generar una lista que almacene los precios de los productos adquiridos en cada venta en un determinado mes; es decir, tiene que estar definido dentro de cada iteración de la lista meses. Y además, hay que establecer una lista para que almacenemos el número de mes y la suma de todos los ingresos de ese mes.

```
Ingresos=[]
meses=range(1,13) #Los meses son números del 1 al 12.
for mes in meses:
    ingresos_por_mes=[]
    for venta in lifestore_sales:
        if venta[3]==mes and venta[4]==0:
            for producto in lifestore_products:
                if venta[1]==producto[0]:
                    ingresos_por_mes.append(producto[2])
    ingreso_total_por_mes=sum(ingresos_por_mes)
    número_de_ventas_por_mes=len(ingresos_por_mes)
    Ingresos.append([mes, ingreso_total_por_mes, número_de_ventas_por_mes])
```

En un punto del código anterior calculamos el número de transacciones por mes necesarias para obtener los respectivos ingresos: debería ser igual al número de ventas calculadas en la sección anterior.

Ahora, ya tenemos el ingreso por mes, de cada uno de los meses. Lo que falta, es calcular el promedio de estos valores y ya tendríamos el ingreso promedio por mes.

- **Calcular el total anual de las ventas y de los ingresos.**

Para calcular el total anual de las ventas y de los ingresos, basta con sumar las ventas e ingresos de todos los meses. Afortunadamente, tenemos dos listas que contienen el número de ventas e ingresos por mes, sólo hay que sumar estos valores para obtener lo solicitado.

- **Obtener los meses con mayor número de ventas**

Ahora, queremos los meses con mayores ventas. Afortunadamente, tenemos una lista con el número de mes, los números de ventas realizadas cada mes; basta con ordenar la lista en función del número de ventas como hicimos en los primeros ejercicios, y extraer los meses con mayores ventas, que serán los primeros de la lista que se obtiene al ordenar de forma descendente.

Código asociado al análisis

- **Identificar a los 5 productos con más ventas**

```
from lifestore import lifestore_products, lifestore_sales

Ventas_por_id_product= []

for producto in lifestore_products:

  id_producto=producto[0]
  Numero_ventas=0
  Nombre_producto=producto[1]

  for venta in lifestore_sales:
    if id_producto == venta[1] and venta[4] == 0:
      Numero_ventas += 1

  Ventas_por_id_product.append([id_producto, Numero_ventas,
  Nombre_producto])

Ventas_por_id_product_ordenadas= sorted(Ventas_por_id_product,
key= lambda x: x[1], reverse=True)

ID_más_vendidos=[]
for posicion in range(0,5):
  id_producto=Ventas_por_id_product_ordenadas[posicion][0]
  ID_más_vendidos.append(id_producto)

print(ID_más_vendidos)
```

Código asociado al análisis

- **Identificar a los 10 productos con más búsquedas**

```
from lifestore import lifestore_products, lifestore_searches

Busquedas=[]

for producto in lifestore_products:
  Número_de_busquedas=0
  id_producto=producto[0]
  for busqueda in lifestore_searches:
    if id_producto==busqueda[1]:
      Número_de_busquedas+=1
  Busquedas.append([id_producto,Número_de_busquedas])

Busquedas_ordenadas= sorted(Busquedas, key= lambda x: x[1],
reverse=True)

ID_más_buscados=[]
for posicion in range(0,10):
  id_producto=Busquedas_ordenadas[posicion][0]
  ID_más_buscados.append(id_producto)
print(ID_más_buscados)
```

Código asociado al análisis

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

```
categorias=[]
for producto in lifestore_products:
  if not producto[3] in categorias:
    categorias.append(producto[3])

Productos_menos_vendidos_por_categoria=[]

for categoria in categorias:


  if categoria == "memorias usb":
    Productos_menos_vendidos=[]
    Ventas=[]
    for producto in lifestore_products:
      if producto[3] == categoria:
        Número_de_ventas=0
        for venta in lifestore_sales:
          if producto[0]==venta[1] and venta[4]==0:
            Número_de_ventas+=1

        Ventas.append([producto[0],Número_de_ventas])

    Ventas_ordenadas = sorted(Ventas, key= lambda x: x[1],
    reverse=False)
    for posicion in range(0,2):
      id_producto=Ventas_ordenadas[posicion][0]
      Productos_menos_vendidos.append(id_producto)
    Productos_menos_vendidos_por_categoria.append([categoria,
    Productos_menos_vendidos])
  else:
    Productos_menos_vendidos=[]
    Ventas=[]
    for producto in lifestore_products:
      if producto[3] == categoria:
        Número_de_ventas=0
        for venta in lifestore_sales:
          if producto[0]==venta[1] and venta[4]==0:
            Número_de_ventas+=1

        Ventas.append([producto[0],Número_de_ventas])

    Ventas_ordenadas = sorted(Ventas, key= lambda x: x[1],
    reverse=False)
    for posicion in range(0,5):
      id_producto=Ventas_ordenadas[posicion][0]
      Productos_menos_vendidos.append(id_producto)
    Productos_menos_vendidos_por_categoria.append([categoria,
    Productos_menos_vendidos])
```

Código asociado al análisis

- **Por categoría, generar un listado con los 9 productos con menores búsquedas**

```
Productos_menos_buscados_por_categoria=[]
for categoria in categorias:
  if categoria == "memorias usb":
    Productos_menos_buscados=[]
    Busquedas=[]
    for producto in lifestore_products:
      if producto[3] == categoria:
        Número_de_busquedas=0
        for busqueda in lifestore_searches:
          if producto[0]==busqueda[1]:
            Número_de_busquedas+=1

        Busquedas.append([producto[0],Número_de_busquedas])

    Busquedas_ordenadas = sorted(Busquedas, key= lambda x: x[1], reverse=False)
    for posicion in range(0,2):
      id_producto=Busquedas_ordenadas[posicion][0]
      Productos_menos_buscados.append(id_producto)
    Productos_menos_buscados_por_categoria.append([categoria,Productos_menos_buscados])
  else:
    Productos_menos_buscados=[]
    Busquedas=[]
    for producto in lifestore_products:
      if producto[3] == categoria:
        Número_de_busquedas=0
        for busqueda in lifestore_searches:
          if producto[0]==busqueda[1]:
            Número_de_busquedas+=1

        Busquedas.append([producto[0],Número_de_busquedas])

    Busquedas_ordenadas = sorted(Busquedas, key= lambda x: x[1], reverse=False)
    for posicion in range(0,9):
      id_producto=Busquedas_ordenadas[posicion][0]
      Productos_menos_buscados.append(id_producto)
    Productos_menos_buscados_por_categoria.append([categoria,Productos_menos_buscados])


#Vamos a mostrar el ID de los productos menos buscados por categoría
print(Productos_menos_buscados_por_categoria)
```

Código asociado al análisis

- **Mostrar dos listados de 5 productos cada una, un listado para productos con las mejores reseñas y otro para las peores, considerando los productos con devolución**

```
from lifestore_file import lifestore_searches, lifestore_sales,lifestore_products

Puntuaciones=[]

#Iteramos sobre cada uno de los productos
for producto in lifestore_products:
  #Vamos a ir guardando las puntuaciones de las diferentes transacciones
  Puntuaciones_de_producto=[]
  for transaccion in lifestore_sales:
    if producto[0] == transaccion[1]:
      Puntuaciones_de_producto.append(transaccion[2])
  if len(Puntuaciones_de_producto) != 0:
    Puntuacion_promedio=(sum(Puntuaciones_de_producto))/(len(Puntuaciones_de_producto))
    Puntuaciones.append([producto[0],Puntuacion_promedio])
  else:
    continue
Puntuaciones_ordenadas=sorted(Puntuaciones, key= lambda x: x[1], reverse=True)
Mejores_resenas=[]
Peores_resenas=[]
IDs_resenas_ordenadas= [x[0] for x in Puntuaciones_ordenadas]
for i in range(0,5):
  ID=IDs_resenas_ordenadas[i]
  Mejores_resenas.append(lifestore_products[ID][1])
for i in range(len(Puntuaciones_ordenadas)-1,len(Puntuaciones_ordenadas)-6,-1):
  ID=IDs_resenas_ordenadas[i]
  Peores_resenas.append(lifestore_products[ID][1])

print(Mejores_resenas)
print(Peores_resenas)
```

Código asociado al análisis

- **Total de ingresos y ventas promedio mensuales, total anual y meses con más ventas al año**

```
from lifestore_file import lifestore_searches, lifestore_sales,lifestore_products
from datetime import datetime
from statistics import mean

meses=range(1,13)
Ventas=[]
for mes in meses:
  Ventas_del_mes=[]
  for venta in lifestore_sales:
    fecha=datetime.strptime(venta[3], '%d/%m/%Y')
    month=fecha.month
    if venta[4]==0 and mes==month:
      for producto in lifestore_products:
        if venta[1]==producto[0]:
          Ventas_del_mes.append(producto[2])
  Ventas.append([mes,len(Ventas_del_mes),sum(Ventas_del_mes)])

Numero_ventas_promedio_por_mes=mean([i[1] for i in Ventas])
print(Numero_ventas_promedio_por_mes)
Ingresos_promedio_por_mes=mean([i[2] for i in Ventas])
print(Ingresos_promedio_por_mes)

Ventas_anuales=sum([i[1] for i in Ventas])
Ingresos_anuales=sum([i[2] for i in Ventas])
print(Ventas_anuales)
print(Ingresos_anuales)

Ventas_ordenadas=sorted(Ventas,key= lambda x:x[1], reverse=True)
Meses_con_más_ventas=[x[0] for x in Ventas_ordenadas]
print(Meses_con_más_ventas)
```

Código asociado al análisis

- **Log In**

```
"""Tenemos que tener una base de datos con los nombre de los diferentes usuarios y sus respectivas contraseñas, pero con relaciones uno a uno. Un diccionario son la opción"""

Credenciales={'Carlos':'carl02', 'Jimmy':'EmTech', 'Monse':'HIMYM'}
#Ahora, podemos usar un ciclo while para repetir la solicitud de usuario y contraseña de forma indeterminada, o hasta que se alcance un número determinado de intentos.
Acceso=False
Intentos=0

while not Acceso:
  Intentos+=1
  print("Favor de introducir el nombre de usuario")
  nombre_de_usuario=input("Nombre de usuario:")
  if nombre_de_usuario in Credenciales.keys():
    print("Favor de introducir la contraseña:")
    contraseña=input("Contraseña:")
    if contraseña == Credenciales[nombre_de_usuario]:
      Acceso=True
    else:
      print("Contraseña incorrecta, pruebe de nuevo")
      print(f"Lleva {Intentos} intentos, el límite es de 3")
  else:
    print("Usuario no encontrado, pruebe de nuevo")
    print(f"Lleva {Intentos} intentos, el límite es de 3")
  #Si se superan los 3 intentos, se cierra el programa
  if Intentos > 3:
    exit()
print(f'Bienvenido {nombre_de_usuario}')
```

CONCLUSIÓN

El uso de la tecnología permite generar información de impacto sobre las decisiones que se tomarán en una empresa, de ahí la importancia de contar con personas que comprendan términos económicos, financieros, que dominen la matemática y sean capaces de acceder a los recursos que ofrece una computadora mediante un lenguaje de programación.

Utilizar un lenguaje de programación de alto nivel, como lo es Python, nos puede ayudar a resolver problemáticas que surgen en la industria. Conforme se amplien los conocimientos de Python de un individuo, será capaz de realizar análisis más sofisticados, además de que se podrá generar gráficos para visualizar la información procesada.