Proyecto 1. Introducción a Python

Luis Habacuq Vallejo Domínguez

CONTENIDO

INTRODUCCIÓN 3
DEFINICIÓN DEL CÓDIGO 4
PLANTEAMIENTO DEL PROBLEMA 4
LOGIN DE USUARIO 4
PRIMERA PARTE (INGRESO GENERAL) 5
SEGUNDA PARTE (MENÚ DE ADMINISTRADOR) 7
SOLUCIÓN A LAS CONSIGNAS 11
LISTADOS CON LOS PRODUCTOS MÁS BUSCADOS Y MÁS VENDIDOS 13
LISTADOS POR CATEGORÍA 16
LISTADOS DE MEJORES Y PEORES RESEÑAS 18
INFORMACIÓN FINANCIERA 20
SOLUCIÓN AL PROBLEMA 23
CONCLUSIÓN 25

INTRODUCCIÓN

No es un secreto que la programación y más específicamente, la ciencia de datos, son términos que han adquirido popularidad en los últimos años, del mismo modo que conceptos como: Inteligencia Artificial, Machine Learning, Big Data, Análisis de Datos, etcétera. A sabiendas de la creciente demanda de profesionales en estas áreas, EMTECH desarrolló un curso introductorio a la Ciencia de Datos usando como herramienta el lenguaje de programación Python. Dicho curso llegó a mí gracias a una beca que me fue otorgada por Santander Universidades y a la que estoy muy agradecido.

Dicho esto, este documento es un reporte, mi reporte personal, del proyecto final de la primera fase del curso, la cual consistió en explorar el mundo de la programación y más específicamente de los conceptos más esenciales de Python (definición y uso de variables y listas, operadores lógicos, control de flujo), es decir, no se hace uso de paqueterías o funciones avanzadas creadas por los propios usuarios: es un programa relativamente complejo construido con las bases más puras de Python.

La forma en la que está estructurado es la siguiente: dentro de la definición del código, se presenta una breve descripción del planteamiento del problema, y posteriormente se describe cada línea de código según la funcionalidad o problema que se resuelve. Así, puede acudirse al índice si se desea consultar únicamente un punto específico dentro del código, o ir directamente a la sección que presenta el informe y soluciones para la tienda ficticia.

El programa completo está disponible para su descarga en un repositorio de mi cuenta de GitHub https://github.com/Habacuq/Curso-de-Python. Así mismo, me gustaría recibir retroalimentación de cualquier tipo por parte de cualquier lector de este reporte, misma que estaré agradecido de leer en mi correo electrónico haba29@outlook.com.

DEFINICIÓN DEL CÓDIGO

En esta sección se explica el código describiendo sus procesos y variables.

PLANTEAMIENTO DEL PROBLEMA

Nos piden hacer un análisis de la rotación de productos para una tienda de artículos de electrónica LifeStore. Tenemos que identificar los siguientes elementos:

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 ingreso y ventas mensuales.

Todo lo necesario para realizar estas tareas se encuentra en el propio archivo PROYECTO-01-VALLEJO-LUISHABACUQ.py

LOGIN DE USUARIO

Lo primero que haremos es declarar las variables y listas que nos serán de utilidad dentro del mecanismo de inicio de sesión.

```
usuarios = ["Habacuq", "Javier", "Ana", "Patricia"]
contraseñas = ["lps31k","mAr0z", "l3trm", "ppal2x"]
continuar = False
admin = False
salir = False
```

Las primeras dos listas contienen los nombres de usuario y las contraseñas predeterminadas que carga el programa. Cada entrada de la lista *usuarios* es el nombre de usuario, y la misma entrada correspondiente en la lista de *contraseñas* es su respectiva clave de ingreso.

Usuario	**Contraseña**
Habacuq	lps31k
Javier	mAr0z
Ana	l3trm
Patricia	ppal2x

Por otro lado, las variables *continuar*, *admin* y *salir* son del tipo binarias. Son esas tres variables las que sirven para darle funcionamiento a todo el sistema de inicio de sesión: la primera es la variable principal y su función es permitir acceder a los usuarios que ingresen exitosamente su usuario y contraseña, la segunda nos permite acceder a un menú especial para el administrador de las cuentas de inicio de sesión, y la tercera está para poder salir del menú en cualquier momento.

PRIMERA PARTE (INGRESO GENERAL)

Posteriormente, el programa pide al usuario que inserte su nombre de usuario, y se dividen en tres los casos posibles; si escribe *0*, entonces el programa entiende que el usuario quiere salir y detiene la ejecución del programa (como así se especifica en la función *input*), si escribe *admin* da pie a que se inicien las funciones especiales programadas únicamente para el administrador de las cuentas, las cuáles serán especificadas posteriormente, y por último, está el caso si escribe cualquier otra cosa diferente. Es importante destacar que, de esta forma, al tener una categoría especial para "todo lo demás" es como el programa está correctamente validado, pues se contemplan todos los casos posibles. La estructura general es la siguiente.

```
while True:
    usuario = input("Por favor, ingrese su nombre de usuario o escriba '0' para salir: ")
    if usuario == "admin":
        #bloque de código
    elif usuario == "0":
        #bloque de código
    else:
        #bloque de código
```

Lo interesante en ese código es la primera línea, la cuál pareciera que forma un ciclo infinito, sin embargo, después se hace uso del comando *break* para salir de ese ciclo. Veamos a detalle cada uno de los tres casos.

```
if usuario == "admin":
    contraseña = input("Ingrese la clave de acceso: ")
    if contraseña == "LifeStore2020":
        admin = True
        break
```

En este bloque de código, una vez que se determina que se introdujo "admin" como nombre de usuario, se despliega otro mensaje que nos pide una contraseña, la cuál es *LifeStore2020* (súper realista). Si se introduce la contraseña correcta, la variable binaria *admin* pasa a ser verdadera y el comando *break* nos saca del bucle que parecía infinito. Notemos que, si se introduce una contraseña incorrecta, simplemente no sucede nada y el código vuelve a empezar (se vuelve a ejecutar el ciclo *while* y vuelve a pedirte un nombre de usuario).

```
elif usuario == "0":
        print("¡Hasta luego!")
        salir = True
        break
```

Este es el más sencillo, simplemente de detectarse un "0" en la función *input*, imprime una despedida, cambia el valor de la variable *salir* a *True* (también nos va a servir más adelante) y rompe el ciclo principal a través del comando *break*.

```
else:
    numUsuario = -1
    for i in range(len(usuarios)):
        if usuarios[i] == usuario:
            numUsuario = i
            break #Este rompe el for
    if numUsuario >= 0:
        break #Este el while
    print("Usuario o contraseña incorrectos. Por favor, inténtelo de nuevo.")
```

Este último bloque es el más complejo de los tres. Lo primero que hace es iniciar un ciclo *for*, el cual se ejecuta sobre los elementos del vector usuarios, hacemos uso de la función *len* para determinar cuántos usuarios hay y tomar ese número como referencia para el ciclo. Notemos que, a pesar de que la función *range* empieza en cero, y termina en el valor n-1, sucede lo mismo con las listas por lo que está correctamente escrito.

Posteriormente, se compara si el i-ésimo elemento de la lista *usuarios* es igual al texto introducido previamente por el usuario. Si sucede una coincidencia, se guarda en una nueva variable el valor de la lista en el que se encontró la coincidencia, esto se hace para después usar ese valor cuando se verifica la contraseña. Una vez encontrada una coincidencia, se ejecuta el comando *break*, pero esta vez no es capaz de sacarnos del bucle infinito porque ya estamos dentro de otro bucle, así que ese break únicamente termina el ciclo *for*, por lo que posteriormente está otra línea que, de encontrar un valor en *numUsuario* (es decir, que hubo coincidencia) ejecuta otro *break* que ahora sí termina el bucle infinito. Finalmente, si no ocurre una coincidencia, se ejecuta un *print* indicando que el usuario o la contraseña son incorrectos y el bucle principal vuelve a ejecutarse desde cero.

Así es como termina la primera de las dos fases del inicio de sesión. La segunda es más compleja que la primera, pues llega a tener hasta seis ciclos anidados. Es un verdadero reto explicar en su totalidad cada línea de esta segunda parte, por lo que la explicación será menos explícita que en la primera, con la justificación de que los principales elementos ya fueron explicados, y para no dedicar la mitad de este reporte solo en esta sección.

SEGUNDA PARTE (MENÚ DE ADMINISTRADOR)

La estructura general de esta sección es la siguiente

```
while continuar == False:
   if salir: #Si previamente el usuario escribe '0'
      break
   if admin == False:
      #bloque de código
   else:
      #bloque de código
```

Hasta ahora no habíamos necesitado de la variable *continuar*, esta se usa por primera vez aquí, y es la forma en la que fácilmente se puede salir de este ciclo. Notemos que lo primero que hace el algoritmo es verificar si la variable salir se cambio a *True* en la primera parte. De ser así, se sale del bucle inmediatamente porque ya no necesita hacer nada más (incluso la despedida se hizo en la primera parte). Posteriormente compara el valor de la variable *admin*. Si previamente se accedió como administrador, entonces resulta *True* y se ejecuta el código de abajo, de otra forma, primer bloque que es el siguiente.

```
if admin == False:
      while continuar == False:
         contraseña = input("Ahora ingrese su contraseña: ")
         if contraseñas[numUsuario] == contraseña:
            continuar = True
            print("Inicio de sesión exitoso ¡Bienvenido " + usuarios[numUsuario] + "!")
         else:
            print("Contraseña incorrecta. Por favor inténtelo de nuevo.")
```

Es decir, si no se ingresó como administrador, pero se ingresó, lo que sigue es solicitar la contraseña, la cuál se verifica con la ayuda de la variable que previamente guardamos para almacenar la posición que nuestro usuario ocupa en la lista. Si después de comparar la contraseña ingresada coincide con la correspondiente en la lista de *contraseñas*, entonces la variable *continuar* pasa a ser *True*, lo cuál nos saca de ambos ciclos *while* y el programa continúa, no sin antes recibir un mensaje de bienvenida personalizado confirmando el ingreso exitoso. De otra forma, imprime que la contraseña es incorrecta y te la vuelve a pedir.

Soy consciente de que lo ideal sería volver a poner la opción de escribir ’0’ para salir de ahí (o no te dejará salir hasta que escribas correctamente la contraseña), o bien, meter un contador que empiece digamos en 5, y en cada intento programar algo como *contador -= 1*, y al llegar a cero te saque de ahí. Sin embargo, no quise hacer aún más complicado el código del inicio de sesión y, en mi lógica, los usuarios de la tienda normales no saben que puedes ingresar “admin” como usuario y desbloquear las opciones de administrador, así

que es como si tuvieras que usar dos claves que solo una persona debería conocer, por lo que limité esas opciones (no me parecieron realmente necesarias).

```
else:
   print("\nBienvenido al centro de administración de cuentas. \
         \n1. Agregar un usuario nuevo\n2. Modificar una contraseña\
         \n3. Eliminar usuario\n0. Salir")
   nMenu = input("Ingrese un número según la tarea que desee realizar: ")
   if nMenu == "1":
      #bloque de código
   elif nMenu == "2":
      #bloque de código
   elif nMenu == "3":
      #bloque de código
   elif nMenu == "0":
      #bloque de código
   else:
      print("Error. No ha ingresado un número correcto.")
```

La anterior es la estructura del código para cuando *admin* = *True*, es decir, el menú y las opciones de administrador. Aun siendo solo “el esqueleto”, ya se aprecia en qué consiste el menú de administrador. Importante el uso de *n* para que la función *print* funcione mejor. De las últimas dos líneas ya se aprecia que, en caso de introducir un carácter inválido, el menú de administrador vuelve a aparecer (el ciclo *while* principal vuelve a iterar), así que está correctamente validado. Antes de continuar explicando cada uno de los bloques de código para cada caso, se muestra cómo luce el menú de administrador.

```
Bienvenido al centro de administración de cuentas.
1. Agregar un usuario nuevo
2. Modificar una contraseña
3. Eliminar usuario
0. Salir

Ingrese un número según la tarea que desee realizar:
```

Es bastante claro a partir del menú, conocer qué facultades especiales tiene el administrador. Ahora sí, ya podemos indagar una por una las cuatro opciones que este tiene.

```
if nMenu == "1": #Agregar un usuario nuevo
   usuarios.append(input("Ingrese el nuevo nombre de usuario: "))
   contraseñas.append(input("Ingrese la contraseña asociada al usuario que ingresó: "))
   print("Nuevo usuario y contraseña registrados exitosamente.")
```

Esta es una de las más sencillas, únicamente hace uso de la función *append* para introducir los nuevos datos a las respectivas listas de *usuarios* y *contraseñas*. Una vez más, en una interfaz más amigable, si al introducir la contraseña pudiéramos censurar la captura digamos con algo como •••••••••, sería conveniente pedir que se capture dos veces la contraseña y que se compare una con otra para asegurar la correcta captura. En este caso, teniendo la captura completamente visible, no parece necesario esa incorporación.

```
elif nMenu == "2": #Modificar una contraseña
    while True:
        usuario = input("Ingrese el usuario al que le quiere modificar la contraseña: ")
        numUsuario = -1
        for i in range(len(usuarios)):
            if usuario == usuarios[i]:
                numUsuario = i
                break
        if numUsuario == -1:
            print("Error. Nombre de usuario no encontrado")
        else:
            break
    contraseñas[numUsuario] = input("Ingrese la nueva contraseña: ")
    print("Contraseña modificada exitosamente.")
```

Esta podría parecer la más complicada, y quizá lo es, pero son pocas las ideas nuevas porque se reciclan elementos previamente explicados. Por ejemplo, la idea del *while True:* no es nueva, lo mismo con todo el mecanismo de *numUsuario*. El algoritmo únicamente busca el usuario que se ingresa, si lo encuentra guarda la posición para luego usarla modificando la entrada correspondiente en la lista *contraseña*s, y si no lo encuentra, te solicita nuevamente el usuario.

```
elif nMenu == "3": #Eliminar un usuario
    while True:
        usuario = input("Ingrese el usuario que desea eliminar: ")
        numUsuario = -1
        for i in range(len(usuarios)):
            if usuario == usuarios[i]:
                numUsuario = i
                break
        if numUsuario == -1:
            print("Error. Nombre de usuario no encontrado")
        else:
            del(usuarios[numUsuario])
            del(contraseñas[numUsuario])
            print("Usuario eliminado exitosamente.")
            break
```

Más de lo mismo, lo único nuevo es la función *del* que permite eliminar elementos de una lista, y que se usa para borrar tanto al usuario como a su contraseña asociada (permitiendo así mantener la congruencia entre las dos listas).

Finalmente, está la opción del menú que nos permite salir de la ejecución.

```
elif nMenu == "0": #Salir
    print("¡Hasta luego!")
    continuar = False
    break
```

SOLUCIÓN A LAS CONSIGNAS

De manera general, el primer paso para cada uno de los problemas a resolver será la creación de una tabla maestra. ¿Por qué? Honestamente porque es la forma en la que yo puedo visualizar más fácilmente las soluciones. De esta forma, para cada uno de los problemas, únicamente se necesitará de una lista para solucionarlo (que no suele ser la misma para cada problema), en lugar de estar navegando entre las tres continuamente. Dicho esto, notemos la distribución de los campos en las listas.

Campos \ listas	lifestore_products	lifestore_sales	lifestore_searches
id_product	✘	✘	✘
name	✘		
price	✘		
category	✘		
stock	✘		
id_search			✘
id_sale		✘	
score (1 to 5)		✘	
date		✘	
refund (1 = True, 0 = False)		✘	

Es importante tener bien presente esa tabla, y notar que únicamente hay un elemento constante en las tres tablas, *id_product*, y será el elemento que nos permitirá crear las distintas tablas maestras para cada caso. Así mismo, la estructura de la interfaz es igual que la presentada para el caso del acceso para administrador (brevemente, es un ciclo *while* que adentro tiene muchos *elif* según lo que el usuario escriba), por lo que únicamente se mostrará el menú ya ejecutado para evitar ser repetitivo (así mismo, muchos conceptos de esta sección son idénticos a los de la parte anterior por lo que se evitará ser reiterativos en exceso, retomando únicamente los conceptos y no todo el código).

```
Bienvenido al sistema de análisis de datos de LifeStore
1. Listado con los 50 productos con mayores ventas netas
2. Listado de los 60 productos con mayores búsquedas
3. Listado con los productos con menores ventas netas por categoría
4. Listado con los productos con menores búsquedas por categoría
5. Listado de productos con las mejores reseñas
6. Listado de productos con las peores reseñas
7. Información financiera
0. Salir

Ingrese un número según la acción que desee realizar:
```

Una vez más, *nMenu* será la variable asignada a la función *input* que captura el número que el usuario introduce y, la validación por si presiona algo inválido se conserva (solo despliega un mensaje de error y vuelve a ejecutar el menú), así como el mecanismo de salida (en caso de introducir ‘0’, se muestra una despedida y se ejecuta un *break* que finaliza la ejecución).

LISTADOS CON LOS PRODUCTOS MÁS BUSCADOS Y MÁS VENDIDOS

En este primer caso, seré particularmente minucioso con la explicación del comienzo del algoritmo porque se repite para absolutamente todos los casos. Lo primero es crear la tabla principal sobre la que vamos a trabajar, la que previamente llamábamos "tabla maestra" y que en el código se llama *tablamaestra.* Bien, el primer paso es asignarle los mismos valores que tiene la lista *lifestore_products* ¿Por qué? Porque esa lista es la única que contiene el nombre del producto y el id, así que nos será de utilidad tomarla como base, porque para la mayoría de las consultas lo más recomendable es imprimir el nombre del producto en lugar del id que sería lo más fácil.

Posteriormente el "truco" consiste en meter un ciclo *for* adentro de otro ciclo *for*. El primero de ellos itera sobre cada elemento de *tablamaestra*, y para cada entrada, hace una comparación por medio del otro ciclo, con la tabla *lifestore_sales*, específicamente compara el único campo presente en ambas listas *id_product*. Con esto, podemos asociarle a cada producto su número de ventas y devoluciones, lo cuál de hecho haremos para posteriormente agregar esas entradas en los elementos de *tablamaestra*, junto con una extra que es la diferencia de estos, es decir, las ventas netas (ventas menos devoluciones).

```
elif nMenu == "1": #caso de la lista con productos más vendidos
    tablamaestra = lifestore_products #igualamos para empezar
    for j in range(len(tablamaestra)):
        ventas = 0 #cada interación comienza el conteo
        devoluciones = 0
        for i in range(len(lifestore_sales)):
            if tablamaestra[j][0] == lifestore_sales[i][1]: #compara id_product
                ventas += 1
                if lifestore_sales[i][4] == 1:
                    devoluciones += 1
        tablamaestra[j].append(ventas)
        tablamaestra[j].append(devoluciones)
        tablamaestra[j].append(ventas - devoluciones)
```

Es decir, hasta ahora lo que tenemos es una nueva lista, que contiene una lista para cada producto, con la siguiente estructura:

$$[id\_product, name, price, category, stock, ventas, devoluciones, ventasNetas]$$

Después, la idea es usar la función que Python ofrece para ordenar datos, sin ayuda de paqueterías extra o conceptos avanzados. Desafortunadamente, no hay forma (o no encontré) de aplicar esa función para cambiar el orden de listas adentro de una lista siguiendo un criterio en específico. La solución es ordenar las ventas en otra lista y seguir usando comparaciones sobre cada dato de *tablamaestra* como se muestra a continuación.

```
ventasOrdenadas = []
    for i in range(len(tablamaestra)):
        ventasOrdenadas.append(tablamaestra[i][7])
    ventasOrdenadas.sort(reverse = True) #para que ordene de mayor a menor
```

Notemos que el vector *ventasOrdenadas* se conforma de las ventas netas de cada producto en *tablamaestra*, y luego usamos la función *sort* para ordenar esos datos, con el importante apunte del parámetro *reverse = True*, pues de manera predeterminada, la función ordena los datos de forma ascendente y eso no es de nuestro interés.

Luego, creamos una nueva lista llamada *topVendidas*, cuya intención es que se llena justo con lo que se nos está pidiendo, los productos ordenados por ventas netas. Para esto, luego de crearse vacía y evitar problemas con Python, iniciamos un ciclo *for* que haga 50 iteraciones (porque solo necesitamos los 50 más vendidos). Dentro de este ciclo, se declara una variable auxiliar *j* igual a cero, después de abre un ciclo *while* el cuál una vez más parece infinito, pero tiene correctamente un *break*, y dentro de este ciclo, se comparan los valores almacenados en la lista que tiene las ventas ordenadas, con la lista que tiene a todos los productos y si son iguales, se añaden a nuestra lista final.

```
topVendidas = []
for i in range(50): #porque solo queremos el top 50
    j = 0
    while True:
        if tablamaestra[j][7] == ventasOrdenadas[i]:
            topVendidas.append(tablamaestra[j])
            del(tablamaestra[j]) #para evitar que se guarde dos veces el mismo producto
            break
        else:
            j += 1
```

Hacemos una pausa importante para explicar por qué cada que añade un elemento, lo elimina de la lista previa. Esta idea surgió a prueba y error, si el código se ejecuta sin esa línea lo que sucede es que, si varios productos tienen el mismo número de ventas netas, siempre aparecerá el mismo en cada entrada de la lista final, porque la variable *j* hace el mismo recorrido y siempre tomaría la primera que encuentra. Así mismo, como consecuencia de ese comando es que se condiciona sobre sumar j, porque si permitimos que siempre sume, podría saltarse valores (por ejemplo, captura el valor 5, luego lo elimina, pero el que antes era 6 ahora es el 5 y por lo tanto si *j* sube en una unidad, iría a checar que 6 que antes era 7 y omite por completo un valor). De nuevo, esta idea será recurrente por lo que solo se explicará en esta sección.

Luego de esto, ya tenemos lo que nos solicitan, sin embargo, si decidimos simplemente imprimir el vector, no se despliega de una manera adecuada, por lo que le daremos una estructura a lo que queremos imprimir a través del siguiente código, en donde además usamos la función *str* para que los datos de tipo *integer* se muestren adecuadamente. Con esto finalizamos el primer caso.

```
for i in range(len(topVendidas)):
    print("Producto: " + str(topVendidas[i][1]) + ". \nVentas netas " +
          "(ventas - devoluciones) totales: " + str(topVendidas[i][7]))
```

Para el siguiente caso, el de mostrar los productos más buscados, funciona exactamente igual, son los mismos conceptos y casi el mismo código, con la única excepción de que en lugar de la lista *lifestore_sales*, se toma *lifestore_searches* para contar las búsquedas en lugar de las ventas. Dicho esto, se omite la explicación para continuar con el siguiente caso.

LISTADOS POR CATEGORÍA

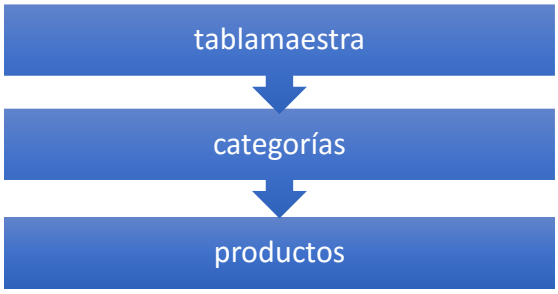
Esta sección funciona muy similar a la anterior con dos sutiles diferencias: la primera es que inicialmente se crea una lista con cada una de las categorías sobre las que se clasifican los artículos, y la segunda, es la estructura de la tabla base *tablamaestra*, la que hasta ahora había sido una lista con listas contenidas en ella, y en este caso subirá una dimensión al tratarse ahora de una lista que contiene listas que contienen a otras listas. Veamos la primera diferencia.

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

Primero se declara la lista *categorías*, posteriormente en un sencillo ciclo *for* se inspeccionan cada uno de los productos de la tienda y por medio del operador *not* se facilita la comparación de categorías para no repetirlas, la entrada 3 de *producto* es la que incluye la categoría. Después, se crea la lista *tablamaestra*.

```
tablamaestra=[]
for i in range(len(categorias)):
    tablamaestra.append([])
    for j in range(len(lifestore_products)):
        if categorias[i] == lifestore_products[j][3]:
            tablamaestra[i].append(lifestore_products[j])
```

Notemos la estrategia de meter un ciclo *for* dentro de otro. El primero de ellos, el exterior, itera sobre cada una de las categorías que se encontraron previamente y crea una lista vacía para cada categoría, posteriormente, el otro ciclo que itera sobre cada uno de los productos, compara las categorías con la categoría sobre la que se encuentre el primer ciclo y, de hallar coincidencias, la añade a lista correspondiente con la categoría a la que pertenece, conformando la siguiente estructura.



Lo siguiente en ese algoritmo es aplicar, sobre cada lista, las mismas ideas de ordenamiento que para el primer problema de los productos más vendidos, la única diferencia es que ahora hay un ciclo *for* externo que se añade y es el encargado de iterar sobre las categorías. Y naturalmente, también cambió la forma en la que se imprimen los resultados como sigue.

```
print("\nCategoría " + str(categorias[k]) + ".")
for i in range(len(topVendidas)):
    print("Producto: " + str(topVendidas[i][1]) + ". \nVentas netas " +
          "(ventas - devoluciones) totales: " + str(topVendidas[i][5]))
```

Cabe mencionar que el código simplemente ordena lo productos clasificados para cada categoría. No muestra “los n menos buscadas” o “los n menos vendidas”, el motivo de esto es que cada categoría tiene distintos productos por lo que no está claro cuántos solicitar de cada una. Sin embargo, fácilmente puede solucionarse esto dentro del código, cambiando el tamaño de la lista auxiliar *ventasOrdenadas*, de la misma forma que se hizo para las primeras dos listas. Por ejemplo, así se vería únicamente el producto menos vendido por categoría (solo se muestran tres categorías para no saturar).

```
Categoría tarjetas de video.
Producto: Tarjeta de Video EVGA NVIDIA GeForce GT 710, 2GB 64-bit GDDR3, PCI Express 2.0.
Ventas netas (ventas - devoluciones) totales: 0

Categoría discos duros.
Producto: SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2.
Ventas netas (ventas - devoluciones) totales: 0

Categoría memorias usb.
Producto: Kit Memoria RAM Corsair Vengeance LPX DDR4, 2400MHz, 32GB, Non-ECC, CL16.
Ventas netas (ventas - devoluciones) totales: 0
```

La opción ‘4’ del menú es similar, la única diferencia es la misma que hay entre las opciones ‘1’ y ‘2’: que se usa *lifestore_searches* en lugar de *lifestore_sales* para llenar las listas de cada producto, pero toda la estructura es la misma.

LISTADOS DE MEJORES Y PEORES RESEÑAS

Esta sección corresponde a las secciones ‘5’ y ‘6’ del menú, las cuales tienen el mismo mecanismo de funcionamiento excepto por únicamente una línea de código, la cual cambia el orden de ascendente a descendente y ya se ha explicado con anterioridad.

```
tablamaestra = lifestore_products
for j in range(len(tablamaestra)):
    ventas = 0
    sumaEstrellas = 0
    for i in range(len(lifestore_sales)):
        if tablamaestra[j][0] == lifestore_sales[i][1]:
            ventas += 1
            sumaEstrellas += lifestore_sales[i][2]
    tablamaestra[j].append(ventas)
    tablamaestra[j].append(sumaEstrellas)
```

Notemos que en el bloque de código mostrado arriba, no hay ningún concepto nuevo, solo que en este caso se añaden las ventas y va sumando el número de estrellas que tiene cada producto, con la intención de posteriormente añadir el cociente que representa el promedio de calificación. Sin embargo, no se puede hacer directamente porque si se topa con un producto que no se vendió, ocurre una indeterminación matemática consecuencia de dividir entre cero. Así, solucionamos de la siguiente manera.

```
j = 0
while j < len(tablamaestra):
    if tablamaestra[j][5] == 0:
        del(tablamaestra[j]) #eliminamos registros sin ventas
    else:
        tablamaestra[j].append(tablamaestra[j][6] / tablamaestra[j][5])
        j += 1
```

Observemos que se hace una inspección por sobre todos los productos y va eliminando aquellos que no tienen ventas. El razonamiento detrás es que no tiene sentido considerarlos en este campo, pues la reseña promedio es en realidad nula, y no sabemos nada sobre ese producto, salvo quizá que no se vende, pero eso es asunto de otra lista y no de esta. Una vez que eliminamos los elementos sin reseñas, ya podemos libremente sacar el promedio de valoración, el cuál será la entrada 7 esa lista particular. Después, hacemos lo habitual, extraer los promedios y guardarlos en otra lista, ordenarlos, y formar una nueva lista que guarde los productos completos correctamente ordenados.

```
estrellasOrdenadas = []
for i in range(len(tablamaestra)):
    estrellasOrdenadas.append(tablamaestra[i][7])
estrellasOrdenadas.sort() #Única diferencia entre ‘5’ y ‘6’
topEstrellas = []
```

```
for i in range(20):
    j = 0
    while True:
        if tablamaestra[j][7] == estrellasOrdenadas[i]:
            topEstrellas.append(tablamaestra[j])
            del(tablamaestra[j])
            break
        else:
            j += 1
```

Como vemos, no hay conceptos nuevos para esta sección, son las mismas ideas que en los listados anteriores. Veamos finalmente el código que despliega las listas, y un ejemplo de su visualización.

```
for i in range(len(topEstrellas)):
    print("Producto: " + str(topEstrellas[i][1]) + ". \nCalificación " + "promedio: " +
          str(topEstrellas[i][7]) + " estrellas luego de " + str(topEstrellas[i][5]) + " reseña(s).")
```

```
Producto: Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache.
Calificación promedio: 5.0 estrellas luego de 2 reseña(s).
Producto: Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na.
Generación Coffee Lake).
Calificación promedio: 5.0 estrellas luego de 3 reseña(s).
Producto: Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na.
Generación Coffee Lake).
Calificación promedio: 5.0 estrellas luego de 7 reseña(s).
```

Se omitieron los otros 17 resultados. No obstante, es suficiente para visualizar que se desglosa el nombre del producto, su calificación promedio y el número de reseñas que obtuvo. Este último dato es importante para que el usuario final tenga más información y pueda juzgar la fiabilidad de la valoración. También pudo haberse implementado una desviación estándar para cuantificar datos atípicos y quitar cierto sesgo, sin embargo para los relativamente pocos datos que se nos presentan, el promedio contrastado con el número de reseñas parece lo más adecuado.

INFORMACIÓN FINANCIERA

En esta sección se analizan con más detalle las ventas de la empresa, en particular estudiamos el horizonte temporal de estas. Cabe destacar que hay dos entradas en lista *lifestore_sales* que difieren de las demás en el año de compra, las cuáles reportan ventas en e 2019 y 2002. Si bien parece que fueron errores de captura, especialmente por lo fácil que es escribir “2002” en lugar de “2020”, o el típico error de creer que estás en el año anterior, he decidido dejar esos datos porque permiten visualizar funciones que solo se aprecian si hay registros en más de un año.

Dicho esto, comenzamos con lo más sencillo, dos listas que solo servirán para guardar los nombres de los meses y los días que tiene cada uno, la primera solo se usará con fines estéticos en el despliegue de resultados, mientras que la segunda será auxiliar en el proceso de sacar promedios de ventas diarias. Por último, se hace la tradicional copia de la lista original, para evitar modificarla y que afecte el funcionamiento general.

```
meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio","Julio",
         "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"]
dias = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
tablamaestra = lifestore_sales.copy()
```

Posteriormente, lo que haremos será añadir entradas a las listas de ventas, esto se lleva a cabo en un proceso en el que simultáneamente añadimos entradas, las cuales son número enteros extraídos mediante *slicing* y convertidos con la función *int*. Además de añadir el mes y el año en formato entero, también añadimos el precio del producto, la clave es comparar los id de la lista de productos con la de ventas, y de coincidir, se añade esa entrada extra y el ciclo se detiene para evitar que tarde más de lo necesario.

```
for venta in tablamaestra:
    venta.append(int(venta[3][3:5])) #mete el mes
    venta.append(int(venta[3][6:10])) #año
    for producto in lifestore_products:
        if producto[0] == venta[1]:
            venta.append(producto[2]) #añade el precio
            break #ya no es necesario seguir buscando
```

Después, sigue la creación de un vector que almacenará los años para los cuáles hay registros, esto es lo que solo sería posible con los datos atípicos. Veamos la forma del ciclo *for* que permite el llenado, una vez más usamos el operador *not* para no añadir elementos repetidos a la lista años. Finalmente los ordenamos de forma ascendente para manipularlos mejor.

```
años = []
for venta in tablamaestra:
    if venta[6] not in años:
        años.append(venta[6])
años.sort()
```

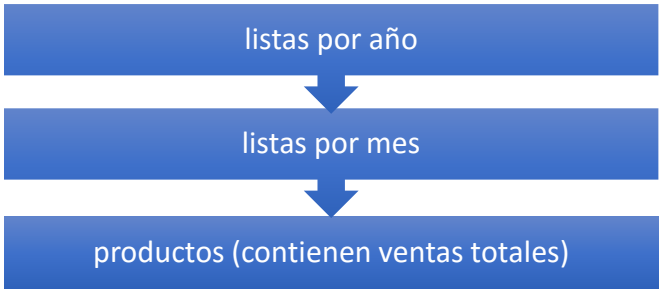
Después es donde empieza complicarse. Ahora creamos una lista y luego un ciclo *for* que itere en cada elemente del vector *años*, dentro de ese ciclo la primera instrucción será añadir una lista dentro de la recién creada *listaOrdenada*, y luego otro ciclo va añadiendo aquellos productos de *tablamaestra* que correspondan en el año sobre el que esté iterando el primer ciclo.

```
listaOrdenada = []
for año in años:
    listaOrdenada.append([])
    for venta in tablamaestra:
        if venta[6] == año:
            listaOrdenada[-1].append(venta)
```

Finalmente, viene el código más complicado quizá del proyecto. Lo que hace ahora es primero crear una lista *ventasAnuales*, la cuál nos servirá para guardar la suma de las ventas en cada año. Luego, itera sobre cada año de la *listaOrdenada* (en este caso serán tres), y luego crea otra variable *pormes*, que tendrá una estructura de 12 listas (para cada mes), que dentro contendrán todas las ventas de ese mes. Las variables *sumadelaño* y *sumadelmes* son de apoyo, son para contar e ir sumando los montos de las ventas, primero al mes y luego al año. Uno de los último ciclos hace el caso especial de los años bisiestos para febrero, haciendo uso del módulo (los años múltiplos de 4 son bisiestos). La última línea añade nuestra nueva estructura a la lista original. Todo el código que no se explicó es porque siguen siendo los conceptos ya explicados previamente.

```
ventasAnuales = []
for k in range(len(listaOrdenada)):
    pormes = []
    sumadelaño = 0
    ventasAnuales.append(0)
    for i in range(1, 13): #1, 2, ..., 12, los meses
        pormes.append([])
        sumadelmes = 0
        for producto in listaOrdenada[k]: #itera por cada mes
            if producto[5] == i: #si coincide el mes
                pormes[-1].append(producto) #lo añade su lista
                sumadelmes += producto[-1] #guarda el precio
        if sumadelmes != 0: #No nos interesan los meses sin ventas
            pormes[-1].append(sumadelmes) #guarda las ventas del mes
            ventasAnuales[k] += sumadelmes
            if i == 2 and años[k] % 4 == 0: #si el año es bisiesto y es febrero
                pormes[-1].append(sumadelmes / 29)
            else:
                pormes[-1].append(sumadelmes / dias[i-1])
    listaOrdenada[k] = pormes
```

Solo es importante notar que la estructura de nuestra lista base quedará primero siendo una lista que contiene una lista para cada año de ventas, y cada una de esas listas a su vez contiene doce listas que corresponden a los meses de ese año, por último, a esas listas que contienen listas sobre las ventas mensuales, se añadió como elemento además de los propios productos, el monto de las ventas de ese mes, esta estructura es la más complicada y quizá solo en mi cabeza tiene sentido, pero espero que el siguiente diagrama lo deje más claro.



Y el último bloque de código de este reporte, resulta ser el despliegue de los datos financieros, que para nada resultó cosa sencilla. Una vez más se retoma la idea de ciclos *for* anidados, uno por años y otro por meses. Como se tratan cantidades financieras, le damos el formato de dinero usando *{0:.2f}.format*, y es aquí en donde se usa por fin la lista que definimos al principio de esta sección, en la que están los nombres de los meses.

```
#Comienza la parte de impresión de datos.
for k in range(len(años)):
   print("\n\nInformación del año " + str(años[k]) + ".\nVentas totales: $" +
       "{0:.2f}".format(ventasAnuales[k]) + "\nPromedio mensual: $" +
       "{0:.2f}".format(ventasAnuales[k]/12))
   for i in range(12):
      if listaOrdenada[k][i] != []:
         print("\n" + meses[i] + "\nVentas: $" +
             "{:.2f}".format(listaOrdenada[k][i][-2]) +
             "\nPromedio diario: $" +
             "{:.2f}".format(listaOrdenada[k][i][-1]))
```

Y para terminar con esta sección, así es como luce.

```
Información del año 2002.
Ventas totales: $259.00
Promedio mensual: $21.58
Mayo
Ventas: $259.00
Promedio diario: $8.35

Información del año 2019.
Ventas totales: $4209.00
Promedio mensual: $350.75
```

SOLUCIÓN AL PROBLEMA

Respecto al problema inicial. En mi opinión, para poder hacer recomendaciones en forma, es necesario tener más datos, prácticamente no tenemos nada de información, únicamente contamos con datos sobre ventas. Necesitaríamos conocer también los costos (cuánto le cuesta a la tienda cada producto), gastos (como rentas, salarios, impuestos), incluso sería de ayuda conocer información sobre la competencia, quizá no estamos vendiendo porque hay alternativas simplemente mejores a LifeStore, o porque son productos que apenas van a comenzar a venderse o que ya pasaron de moda (información sobre el mercado). Incluso cuestiones macroeconómicas como la coyuntura actual por la pandemia son importantes al considerar recomendaciones para el negocio. Aun así, contar con estos datos es muy valioso, especialmente contrastando con no tener ningún dato. Dicho esto, se presentan a continuación algunos puntos relevantes a considerar para los encargados de la tienda.

Respecto a los productos más vendidos, notemos que son en este orden, un disco duro en estado sólido, dos procesadores y una tarjeta madre, por lo cuál mi recomendación es que se considere **vender** paquetes, ya sea **kits de actualización** (paquetes de tres o cuatro componentes para optimizar una computadora de escritorio o portátil) o directamente ofrecer **computadoras ensambladas** en venta. Algo mejor que tener un producto estrella, es tener un paquete de productos estrella.

Sobre los productos más buscados, en general coinciden con los más vendidos, hay congruencia en ese sentido. Sin embargo, existen productos como por ejemplo 'Logitech Audífonos Gamer G635 7.1, Alámbrico, 1.5 Metros, 3.5mm, Negro/Azul', el cual fue el sexto artículo más buscado con 35 búsquedas y únicamente se vendió dos veces. ¿Por qué sucede esta situación? Quizá nuestro motor de búsqueda lo está mostrando erróneamente cuando el usuario quiere otra cosa, o también puede pasar que nuestros precios no sean competitivos y el cliente lo encuentra a mejor precio en otro lugar. **Deberíamos bajar el precio** de este producto y todos aquellos en los que haya muchas búsquedas y pocas ventas, de esta forma, aunque perdamos utilidad neta, estamos apostando por aquellos productos que lo tienen todo para ser de los más vendidos, mejor vender cien productos, aunque a cada uno le ganes solo $100, a vender dos y que ganes $500 (por decir algo).

Ahora, cuando observamos las listas de ventas por categorías, notamos que muchos productos no se han vendido, y hay categorías realmente pobres como bocinas, audífonos y televisiones. Sobre esta última, si bien no hice ningún tratamiento o consideración especial con el stock de cada producto, directamente mirando de la lista notamos que hay mucho más stock que las ventas totales, esto siempre es peligroso para cualquier compañía, pero en particular aquí, la sugerencia es rematar o simplemente **dejar de comprar televisiones**, porque tienen al mismo tiempo la combinación de pocas ventas, mucho inventario, y bastante más costoso su almacenamiento (en comparación con los demás productos que LifeStore maneja), por si fuera poco, tampoco están siendo buscadas, y el giro de la tienda parece más enfocado a computación por lo que en definitiva mi sugerencia sería abandonar las televisiones.

Sobre las búsquedas por categoría, lo mismo, poca interacción de bocinas, pantallas, audífonos y memorias. La sugerencia sigue siendo evitar estos productos en la tienda y centrarse en componentes de computadora como tal, no gadgets.

Por otro lado, algo que podría implementarse con las reseñas de los productos, es considerar esos datos en el algoritmo de búsqueda de la tienda, para posicionar mejor en las búsquedas a los productos mejor calificados. Más interesante por otro lado, es conocer a los peor calificados, y la sugerencia para estos productos es evitar seguir vendiéndolos, claro sin dejarnos llevar por un dato atípico como lo podría ser con un producto con un 1 y un 5 de reseñas.

Finalmente, en el aspecto financiero, la primera sugerencia es cambiar el sistema de captura de ventas por uno automático, para que de esta forma se **eviten errores de captura** como los ocurridos con las compras del 2002 o 2019. Es fácil para nosotros distinguir que esos registros en particular tienen error, sin embargo, así como detectamos fácilmente estos, nada nos garantiza que haya otros errores en los meses de venta o en otros datos específicos. Recordemos que **lo más importante** para hacer un análisis cuantitativo de cualquier tipo **son los datos**.

La segunda sugerencia sería comenzar con ofertas o reconsiderar las estrategias de mercadeo o selección de productos, porque desde mayo de 2020 ha comenzado una gran caída en los ingresos, quizá consecuencia de la situación con la pandemia, que ha afectado en la economía de las personas. En ese sentido, una tercera sugerencia sería buscar convenios con los principales bancos para ofrecer compras a pagos mensuales con tarjeta de crédito, esta medida puede también solucionar el problema del inventario, pues presenta una nueva alternativa para que el consumidor compre y apuntar hacia un nuevo mercado (quienes no puede permitirse eso pagos de contado).

CONCLUSIÓN

La principal conclusión que me llevo luego de este trabajo es que es realmente difícil e incluso dudoso, hacer el trabajo interpretativo del científico de datos cuando no se conoce la información suficiente. Me queda claro que los científicos de datos deben conocer muy bien los negocios que analizan, o trabajar en conjunto con los expertos de cada área, porque si solo tenemos a nuestro alcance cifras limitadas, es cuando empezamos a hacer suposiciones sobre el negocio y su desempeño.

Otro aspecto al que nunca me había enfrentado antes, es precisamente esta labor de redacción y presentación de un código explicado a detalle. Espero que mi experiencia consultando libros de programación haya rendido frutos a través de este mi primer reporte de un programa. No hay duda de que tuve fallas como nulos aspectos gráficos, pero fue así porque en realidad no conozco una buena forma de presentarlos adecuadamente.

Espero honestamente que cualquier persona que en algún momento se tope con este documento y el programa asociado, haya aprendido o le haya parecido interesante. Este es solo el inicio de una probable carrera ciencia de datos, los invito a revisar periódicamente mi repositorio en GitHub, así como de conectar en LinkedIn https://www.linkedin.com/in/luis-habacuq-vallejo-dom%C3%ADnguez-08109313b/. Aún me falta mucho por aprender y espero seguirlo haciendo por los próximos años.

Educate yourself. When a question about a certain topic pops up, google it. Watch movies and documentaries. When something sparks your interest, read about it. Read read read. Study, learn, stimulate your brain. Don't just rely on the school system, educate that beautiful mind of yours.