EMTECH INSTITUTE

PROYECTO 1
INTRODUCCIÓN A PYTHON

REALIZADO POR
EMMANUEL ARTURO BOLAÑOS ESPINOZA

FECHA DE ENTREGA
6 DE SEPTIEMBRE DEL 2020

ÍNDICE

INTRODUCCIÓN

La tienda virtual LifeStore maneja una amplia gama de artículos. Recientemente, la Gerencia de Ventas ha reportado una importante acumulación en el inventario. De igual forma, se identificó una reducción en búsquedas de un grupo de productos, lo cuál ha disminuido las ventas del último trimestre.

Debido a lo anterior, se ha realizado este reporte, donde se explica de forma general el procedimiento que se siguió para analizar y clasificar la información entregada por LifeStore dentro del archivo "lifestore_file.py" por medio del lenguaje de programación Python.

Luego de la descripción del código, se mostrarán los resultados a los cuales se llegaron después de hacer el análisis de:

1. Productos más vendidos y rezagados.
2. Productos por reseña en el servicio.
3. Ingresos y ventas mensuales.

Finalmente, se propondrá una estrategia para retirar del mercado algunos productos y lograr reducir la acumulación de inventario.

DEFINICIÓN DEL CÓDIGO

El código completo se encuentra en el *ANEXO 1*. A continuación, se procede a explicar de forma general el proceso que sigue el mismo. Todo el código se encuentra comentado y dentro del mismo se explica de forma detallada el funcionamiento de este.

El usuario con acceso a la información (así como un ejemplo de usuario sin permisos de administrador) se ha escrito en forma de comentario en las primeras líneas del código para que sea más fácil de identificar.

Primero se importan las listas desde el archivo entregado por LifeStore a nuestro proyecto. Renombramos los archivos dentro del código para que sea más cómodo el acceso a ellos.

A continuación, se declaran algunas funciones que nos serán útiles más adelante:

- *quantify_list(sorted_list)*: esta función cuenta las veces que un elemento se repite dentro de una lista. El parámetro *sorted_list* debe se una lista en orden ascendente o descendente. La función regresa una lista con sublistas donde el primer elemento es el nombre del objeto y el segundo las veces que se repitió.
- *sort_by_n_element(unsorted_list, n, keep_col=10, rev=False)*: esta función ordena una lista con sublistas de menor a mayor. El parámetro *unsorted_list* requiere una lista con sublistas que puede encontrarse en desorden, *n* es la columna respecto a la cuál queremos ordenar la lista, *keep_col* tiene 10 por default, este valor se usa para indicar que solo se desea que la función regrese una sola columna. Sin embargo, si cambiamos el parámetro a un valor *k* (que sea válido para la sublista ingresada) podemos obtener de regreso la columna *n* ordenada y su correspondiente columna *k.*

Finalmente, el parámetro *rev* ordena la lista en el orden contrario cuando es igual a *True*.

- *save_to_csv(file_name, list_to_save)*: guarda una lista con sublistas en formato csv para que facilitar su futuro acceso. El parámetro *file_name* es una cadena de caracteres que contiene el nombre del archivo y *list_to_save* indica cual es la lista que se desea guardar.
- *didnt_appear(a,x,b,y)*: revisa si el elemento de una sublista en una lista se repite en otra. El parámetro *a* es la lista con la columna *x* que contiene todos los elementos. El parámetro *b* es la lista con la columna *b* que contiene menos elementos.
- *prod_name(names)*: revisa si el parámetro *names* (que debe ser una cadena de caracteres, contiene una coma. De ser así, regresa el contenido antes de llegar a la coma. En caso contrario, regresa el mismo parámetro.

Ahora, se pide el ingreso del usuario y contraseña. En caso de ingresar unas credenciales incorrectas, se le darán 3 oportunidades más para ingresar un usuario y contraseña válidos. En caso contrario, se le sacará del programa. De igual forma, ingresar un usuario y contraseña que no cuente con permisos de administrador le sacará del programa.

Una vez ingresadas las credenciales de administrador, damos la bienvenida al programa y mostramos el menú de opciones disponibles. Se le pide un número al usuario, se repite el proceso hasta que el número ingresado este dentro del rango. Posteriormente, se le pregunta al usuario si desea guardar los resultados de la visualización, esto permitirá acceder a ellos después desde una hoja de cálculo como Excel si el usuario así lo desea.

A continuación, se realiza el filtrado de los 50 productos más vendidos. Primero ordenamos de menor a mayor los id de la lista de productos vendidos (*lifestore_sales*) con ayuda de la función *sort_by_n_element* y usando la columna 1 (la que contiene los id) como segundo parámetro. Luego, usamos la función *quantify_list* con la lista ordenada que acabamos de conseguir para conocer la cantidad de veces que se repiten los id. Con *sort_by_n_element* y el parámetro *rev* como verdadero, reordenamos la lista para que ahora se encuentre de mayor a menor con respecto a la columna de repeticiones. Ahora usamos *didnt_appear* con la lista de productos y ventas para almacenar en una variable el producto que no se vendió. Agregamos la lista de producto no vendido al final de la lista que habíamos obtenido anteriormente.

A partir de este punto, revisamos si la operación seleccionada por el usuario solicita ver los resultados. Si es así, entonces recorremos la lista obtenida y la lista de productos para cambiar los id por el nombre del artículo al mismo tiempo que acortamos el nombre para que la lista se pueda leer cómodamente haciendo uso de la función *prod_name*. En caso de que la lista final tenga menos de 50 elementos, le hacemos saber esto al usuario antes de imprimir los valores. En caso contrario, recortamos la lista para que solo contenga 50 valores. Finalmente, recorremos las sublistas e imprimimos los valores para el usuario. Si el usuario pidió guardar la información, se agrega un encabezado a la lista y se guarda en la carpeta *outputs* en archivo csv con el nombre *50_mas_vendidos.csv*.

El proceso para los 100 productos más buscados es muy similar solo que la realizamos con la lista de búsquedas (*ls_searches)*. Ordenamos los id con *sort_by_n_element*, contamos las repeticiones con *quantify_list*, ahora ordenamos respecto a las repeticiones con *sort_by_n_element.* Agregamos los productos no buscados con *didnt_appear*. Finalmente, si el usuario pidió ver la información, cambiamos los id por nombres, revisamos que la lista contenga más de 100 artículos, en caso contrario se muestra la advertencia. Imprimimos la lista de productos buscados de mayor a menor y guardamos si el usuario así lo requirió, esta vez con el nombre de *100_mas_buscados.csv*.

Antes de comenzar con el siguiente objetivo. Construimos una nueva lista, esta lista contendrá la categoría y una sublista con todos los id que corresponden a esa categoría. Para obtener esta lista primero recuperamos las categorías existentes de la lista de productos, observamos que las categorías se repiten más de una vez. Creamos una nueva lista a la cual solo agregamos elementos si no se han repetido y usamos esta para recorrer la lista de productos y crear una nueva lista que solo tenga la categoría y sus correspondientes id.

Haciendo uso de la lista obtenida anteriormente, ahora podemos usarla para obtener el siguiente objetivo de forma más sencilla. Usamos esta lista para comparar cada elemento con la lista de ventas. Si un id de la lista esta igual en la lista de ventas, aumentamos un contador para conocer la cantidad de veces que un producto de la categoría se Vendió. Finalmente, ordenamos la lista obtenida de menor a mayor usando *sort_by_n_element*, teniendo como referencia el contador de ventas. Si el usuario lo pidió, mostramos la lista de ventas por categoría y guardamos la información en el archivo *cat_con_menos_ventas.csv*. Para encontrar la categoría con menos búsquedas se sigue el mismo proceso, solo que, en lugar de usar la lista de ventas, usamos la lista de búsquedas (lifestore_searches). Esta lista se guarda como *cat_con_menos_busquedas.csv*.

Lo siguiente es obtener los 20 productos con mejores y peores reseñas. El primer paso es construir una lista con los id. La creamos con *sort_by_n_element* usando la lista de productos debido a que contiene los id de todos los productos. Ahora usamos esa lista y recorremos la lista de ventas. Cuando el id y el id del producto vendido son iguales, anexamos esa calificación al id. En cada bucle se crea una sublista que contiene el id y todas sus reseñas. A continuación, recorremos esta lista para encontrar el promedio de calificación, en cada sublista que contiene las reseñas, encontramos la suma de esta y la dividimos entre su número de elementos (usamos *len* en la sublista). Ahora contamos con una lista que esta compuesta por un id y el promedio de su reseña. Usamos *sort_by_n_element* para acomodar cada elemento de menor a mayor respecto a su promedio de reseña. Cambiamos los id a nombres siguiendo el mismo procedimiento que antes y creamos dos listas. Una que va del índice 0 al 20 para las peores reseñas y otra que va de -1 a -21 en el sentido contrario para las mejores reseñas. Si el usuario lo pidió se muestra la lista con las mejores calificaciones y se guarda en *mejor_calificacion.csv*. De igual forma, cuando el usuario pide las peores calificaciones estas se muestran y se guardan en *peor_calificacion.csv.*

El último objetivo consiste en mostrar los ingresos y ventas promedio mensuales, el total anual y meses con más ventas al año. Primero creamos una lista con id, mes y año de venta.

También creamos una lista con los años con ventas registrados. Nos damos cuenta de que solo tres años están registrados y, además que los años 2002 y 2019 solo cuentan con una venta cada uno. Debido a esto, dejamos de lado estos datos pues parecen ser valores atípicos. Ahora trabajaremos solo con la lista que contiene ventas del 2020. Creamos una lista de meses con venta en orden. Usando esta lista y la lista de ventas del 2020, creamos una nueva, en la nueva lista cada sublista tiene los id que se vendieron, la cantidad de id vendidos y el mes que corresponde a las ventas que llamaremos org_ids_2020.

Hacemos otra lista que contenga los id y sus precios usando la lista de productos. A continuación, obtendremos los ingresos por mes, ingreso anual y el total ventas. Iniciamos un for para recorrer cada mes de la lista org_ids_2020, luego otro for para recorrer la lista de id y precios y finalmente un for más para usar como puntero. Finalmente, usamos un if que compara si el id del mes al cual apunta el puntero es igual al id de la lista de precios, si es así, se agrega a una variable llamada *income* que se reinicia cada que se regresa al primer for, el precio se guarda en otra variable que servirá para conocer el ingreso total del año y se aumenta el contador de ventas para conocer el numero de ventas del mes. De otra forma, se compara con el siguiente id, hasta tener todos los id de cada mes.

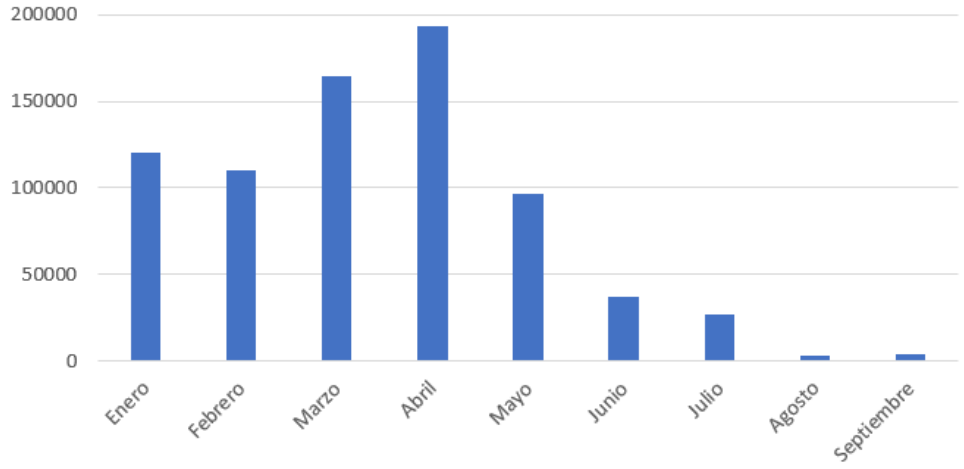
El promedio de ventas mensuales se obtiene al dividir el total de ventas entre la cantidad de meses en la lista de meses. Para obtener los meses con mejores ventas (las que están por encima del promedio), anexamos los meses con ventas por encima del promedio a una nueva lista y luego la ordenamos de mayor a menor para mejorar la presentación.

Finalmente, si el usuario pidió ver y/o guardar algún resultado, este se muestra y guarda. Los totales de venta e ingresos anuales se guardan en *ingresos_ventas_anuales.csv*, el promedio mensual de ventas se guarda en *promedio_mensual.csv*, las ventas e ingresos por mes se guardan en *ingresos_por_mes.csv* y los meses con mejores ventas con su respectivo ingreso están en *mejores_meses.csv*. Después de terminar con la última línea del programa, este volverá a mostrar el menú y pedir una vez más al usuario que seleccione la acción que desee realizar.

SOLUCIÓN AL PROBLEMA

Luego de correr el código podemos realizar algunas gráficas con la información obtenida que nos ayudaran a observar de forma más clara la situación de la empresa.

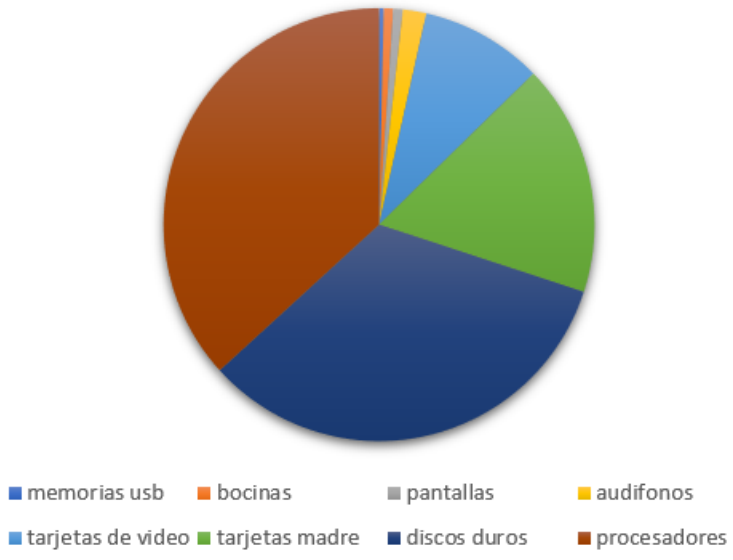
En la *Gráfica 1* nos damos cuenta de que durante los últimos cuatro meses los ingresos se han reducido bastante.



Gráfica 1. Ingresos mensuales.

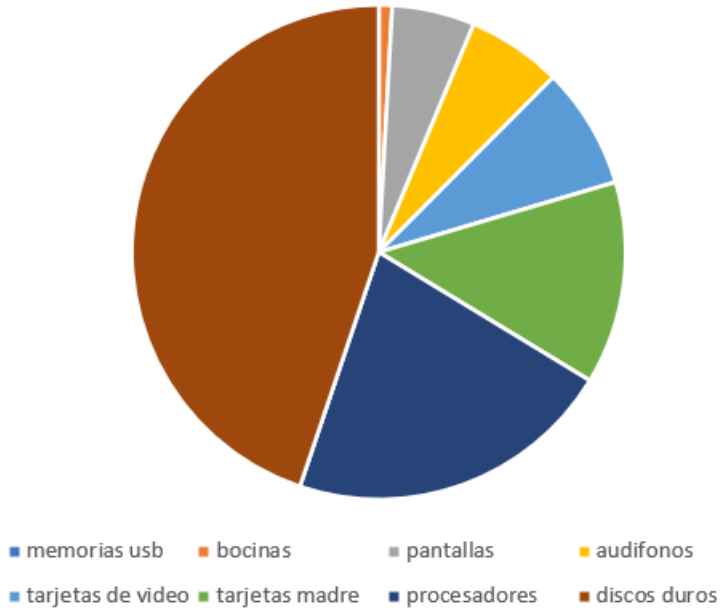
En la *Gráfica 2* podemos observar que hay cuatro categorías con muy pocas ventas:

- Memorias USB
- Bocinas
- Pantallas
- audífonos



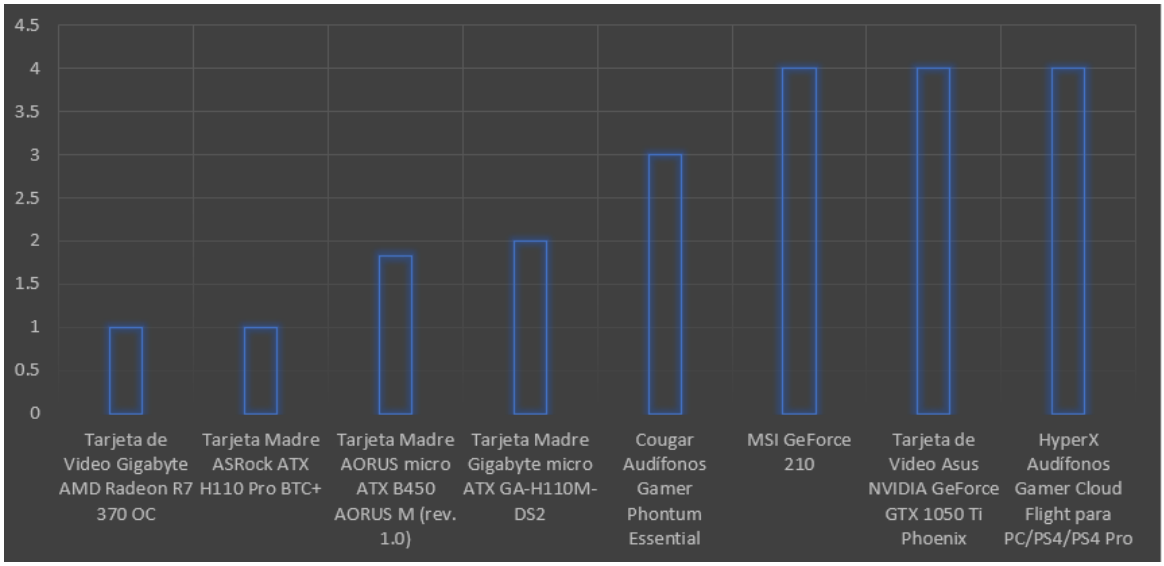
Gráfica 2. Ventas por categoría.

En la *Gráfica 3* podemos observar que la categoría de memorias USB y bocinas se encuentran, una vez más, en uno de los últimos lugares.



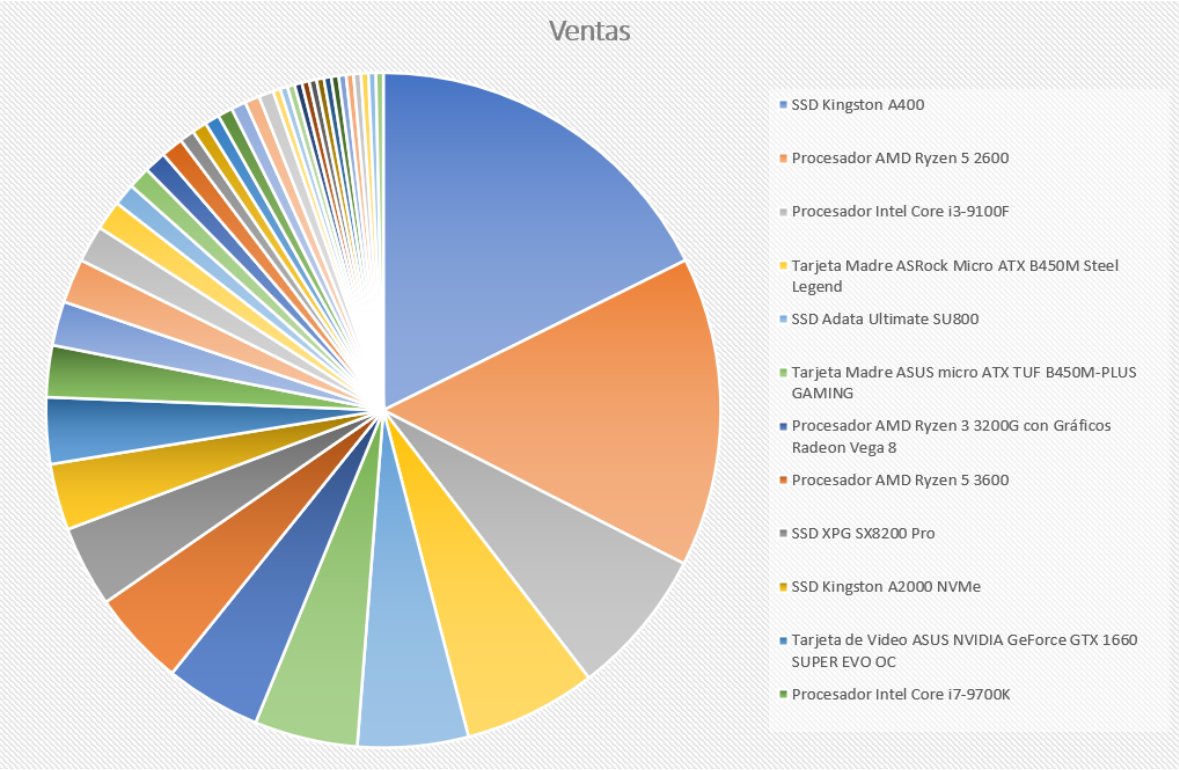
Gráfica 3. Búsquedas por categoría.

En la *Gráfica 4* podemos observar los productos con peores reseñas.



Gráfica 4. Peores reseñas.

Finalmente, en la *Gráfica 5* tenemos los productos que han acumulado la mayor cantidad de ventas. La mayoría son procesadores, discos duros y tarjetas madre. Lo cual coincide con la información reportada en la *Gráfica 2*.



Usando la información anteriormente presentada. Sería conveniente retirar los siguientes productos del mercado:

Producto a retirar	Razón principal
Categoría: Memorias USB	No se buscan y solo han tenido una venta
Tarjeta de Video Gigabyte AMD Radeon R7 370 OC	Pésimas reseñas
Tarjeta Madre ASRock ATX H110 Pro BTC+	Pésimas reseñas
Tarjeta Madre AORUS micro ATX B450 AORUS M (rev. 1.0)	Pésimas reseñas
Tarjeta Madre Gigabyte micro ATX GA-H110M-DS2	Pésimas reseñas
Cougar Audífonos Gamer Phontum Essential	Malas reseñas y pocas ventas

CONCLUSIÓN

Realizar este proyecto me ha ayudado a mejorar mis bases en programación, además, me he dado cuenta de que en ocasiones es importante centrarse en terminar el proyecto y ponerse objetivos claros desde el inicio para no alargar el código con características que quizás no sean necesarias y al final no sean incluidas. Esta experiencia igual me ha enseñado que además de poder programar, para un científico de datos es muy importante poder comprender la información que regresa el programa. Una de las estrategias simples pero que me ayudo a entender mejor la información fue cambiar los id por nombres de artículos, pues de esa forma me permitió identificarlos fácilmente. Finalmente, de no haber usado Excel para poder graficar los resultados obtenidos, me hubiera sido muy complicado poder examinar de forma precisa la relación que había entre los productos y proponer una estrategia concreta, esto me hace comprender la importancia de poder presentar los resultados obtenidos de forma correcta.

ANEXO 1

"""

CÓDIGO ESCRITO POR EMMANUEL BOLAÑOS

USUARIO CON PERMISOS:

USUARIO: admin

Contraseña: 123

EJEMPLO DE USUARIO SIN PERMISOS:

USUARIO: cliente

Contraseña: abc

"""

```
# importa las listas con los datos
from lifestore_file import lifestore_products as ls_products
from lifestore_file import lifestore_sales as ls_sales
from lifestore_file import lifestore_searches as ls_searches


# Función que cuenta las veces que un elemento se repite
# List ingresada debe estar en orden
# Regresa el elemento y las repeticiones
def quantify_list(sorted_list):
    # Declara una variable para contar repeticiones
    # y una lista en blanco
    reps = 1
```

```
    quantified_list = []
    # Para cada elemento de la lista ingresada
    # asigna un contador y lee el elemento
    for counter, element in enumerate(sorted_list):
        # Si el contador es del tamaño de la lista
        # menos uno, sal del bucle
        if counter == len(sorted_list) - 1:
            break
        # Si el elemento actual es igual al siguiente
        # aumenta el numero de repeticiones
        elif element == sorted_list[counter + 1]:
            reps = reps + 1
        # De lo contrario, agrega el elemento y
        # sus repeticiones a la lista y regresa
        # la variable que cuenta repeticiones a 1
        else:
            quantified_list.append([element, reps])
            reps = 1
    # Agrega el elemento final a la lista
    quantified_list.append([element, reps])
    return quantified_list


# Función para ordenar una lista de acuerdo a su columna n
# Regresa solo la columna ordenada
# Opcional: agrega otra columna con keep_col
# ordena en de mayor a menor con rev = True
def sort_by_n_element(unsorted_list, n, keep_col=10, rev=False):
```

```
    # Declara una lista en blanco
    clean_list = []
    # Para cada sublista de la lista ingresada
    for sub_list in unsorted_list:
        # Para cada categoría dentro de la sublista
        # Se asigna un contador para cada categoría
        for counter_2, category in enumerate(sub_list):
            # Si el contador es igual a n,
            # estamos en la columna correcta
            if counter_2 == n:
                # Si el valor keep_col fue cambiado,
                # entonces agrega la columna a la lista
                # Sino, ordena la lista de forma normal
                if keep_col != 10:
                    clean_list.append([category, sub_list[keep_col]])
                else:
                    clean_list.append(category)
    # Ordena la lista de menor a mayor a menos de que rev sea verdadero
    if not rev:
        clean_list.sort()
    else:
        clean_list.sort(reverse=True)
    return clean_list


# Función para guardar outputs en formato csv
def save_to_csv(file_name, list_to_save):
    with open("outputs/" + file_name + ".csv", "w") as f:
```

```
        for sublist in list_to_save:
            for item in sublist:
                f.write(str(item) + ",")
            f.write("\n")


# Función para regresar ids que no han sido comprados o buscados
def didnt_appear(a, x, b, y):

    new_list = []
    # Recorremos las sublistas de a
    for sblst in a:
        c = 0
        # Recorremos las sublistas de b
        for sblst_2 in b:
            # Si el elemento de la columna x de a coincide
            # con la columna y de b, aumenta el contador
            if sblst[x] == sblst_2[y]:
                c = c + 1
        # Si el contador es cero, no hubieron coincidencias!
        if c == 0:
            new_list.append([0, sblst[0]])
    return new_list


# Función para regresar solo el nombre del producto
def prod_name(names):
    # Encontrar indice que contiene la coma
```

```
    product_name_0 = names.find(",")
    # Si no hay coma, regresar el nombre sin modificar
    if product_name_0 == -1:
        return names
    # Regresar el str antes de la coma
    return names[:product_name_0]



# Pedimos sus credenciales al usuario (estan al principio del código)
print("INGRESE USUARIO Y CONTRASEÑA:")
usr = input("USUARIO:")
pwrd = input("CONTRASEÑA:")
kick = 4

# Mientras el usuario y contra sean incorrectas...
while usr != "admin" or pwrd != "123":
    if usr == "cliente" and pwrd == "abc":
        # Si no tiene login de admin, se le saca del programa
        print("Lo siento, no tienes permiso para acceder")
        quit()
    # El contador nos permite conocer la cantidad de intentos del usuario
    kick -= 1
    # Si te acabas los intentos, afuera
    if kick == 0:
        print("\n\nINTENTOS AGOTADOS\nSaliendo de programa...")
        quit()
    # Aviso cuando solo queda 1 intento
    elif kick == 1:
```

```
        print("\nUSUARIO Y CONTRASEÑA INCORRECTOS\n", kick, "intento restante")
    # Avisa que queda más de un intento
    else:
        print("\nUSUARIO Y CONTRASEÑA INCORRECTOS\n", kick, "intentos restantes")
    # Pregunta si desea volver a intentarlo
    _try = input("\n¿Volver a intentar? (si/no): ")
    # Entra en bucle para revisar si el usuario quiere volver a intentar
    # el ingreso de credenciales
    while 1:
        # Sal del bucle y vuelve a intentarlo
        if _try == "si":
            usr = input("USUARIO:")
            pwrd = input("CONTRASEÑA:")
            break
        # Sal del programa, no tiene las credenciales
        elif _try == "no":
            print("Saliendo de programa...")
            quit()
        # Si el usuario ingresa un valor no valido, repitele que debe hacer
        else:
            _try = input("Solo escriba 'si' o 'no', por favor:")

# Si llega a este punto, sabemos que es el admin
print("""
        ~~~~~BIENVENIDO ADMIN~~~~~
    """)
# Buble infinito para ejecutar el programa
while 1:
```

```
# Menú que muestra todas las opciones disponibles
print("""
            MENÚ:
1)    VER TODO
2)    VER LOS 50 PRODUCTOS MÁS VENDIDOS
3)    VER LOS 100 PRODUCTOS CON MÁS BÚSQUEDAS
4)    VER LAS CATEGORÍAS CON MENOS VENTAS
5)    VER LAS CATEGORÍAS CON MENOS BÚSQUEDAS
6)    VER LOS 20 PRODUCTOS CON MEJORES RESEÑAS
7)    VER LOS 20 PRODUCTOS CON PEORES RESEÑAS
8)    VER INGRESOS Y VENTAS DEL AÑO
9)    VER EL PROMEDIO MENSUAL DE VENTAS
10)   VER VENTAS E INGRESOS POR MES
11)   VER MEJORES MESES DEL AÑO
12)   SALIR
""")

# Pide al admin que ingrese un numero
op = int(input("INGRESE SOLO EL NÚMERO DE LA OPCIÓN QUE DESEA VER\n"))

# Si el numero está fuera de rango, vuelve a pedir input
while op >= 13 or op <= 0:
    op = int(input("INGRESE SOLO EL NÚMERO DE LA OPCIÓN QUE DESEA VER\n"))

# Instrucción para salir del programa
if op == 12:
    quit()
```

```
# Preguntar al admin si desea guardar la información
save = input("¿DESEA GUARDAR LOS RESULTADOS VISUALIZADOS? (si/no)\n")

# Bucle para asegurar que se ingresa un valor válido
while 1:
    if save == "si":
        break
    elif save == "no":
        break
    else:
        save = input("INGRESE SOLAMENTE 'si' O 'no'\n")

# Los 50 + vendidos
# Crea una nueva lista con ids de los productos vendidos
# ordenados de menor a mayor
productos_vendidos = sort_by_n_element(ls_sales, 1)
# Lista que muestra cuanto se vendió cada producto
cantidad_vendidos = quantify_list(productos_vendidos)
# Ordena la lista de mayor a menor
total_vendidos_y_ids = sort_by_n_element(cantidad_vendidos, 1, 0, rev=True)
# Lista con productos que no se vendieron
producto_no_vendido = didnt_appear(ls_products, 0, ls_sales, 1)
# Agregar productos no vendidos al final de la lista
total_vendidos_y_ids = total_vendidos_y_ids + producto_no_vendido
if op == 1 or op == 2:
    # OUTPUT
    # Cambiar ids a nombres para hacer amigable a la lectura
    for id_lst in total_vendidos_y_ids:
```

```
            for prd_lst in ls_products:
                if id_lst[1] == prd_lst[0]:
                    id_lst[1] = prod_name(prd_lst[1])
        # Crea la lista final con solo 50 valores
        if len(total_vendidos_y_ids) <= 49:
            print("\nNo se cuenta con 50 productos o más, mostrando", len(total_vendidos_y_ids), "productos")
            final_total_vendidos_y_ids = total_vendidos_y_ids
        else:
            final_total_vendidos_y_ids = total_vendidos_y_ids[0:50]
        print("\n\nLista de productos vendidos (mayor a menor)")
        for out_vendidos_ids in final_total_vendidos_y_ids:
            if out_vendidos_ids[0] >= 2:
                print(out_vendidos_ids[1], "se ha vendido", out_vendidos_ids[0], "veces")
            elif out_vendidos_ids[0] == 1:
                print(out_vendidos_ids[1], "se ha vendido", out_vendidos_ids[0], "vez")
            else:
                print(out_vendidos_ids[1], "no se ha vendido")
        if save == "si":
            # Guarda resultado en outputs/test_0.csv
            final_total_vendidos_y_ids = [["Ventas", "Nombre del producto"]] + final_total_vendidos_y_ids
            save_to_csv("50_mas_vendidos", final_total_vendidos_y_ids)


    # 100 + buscados
    # Lista con ids de productos buscados en orden
    productos_buscados = sort_by_n_element(ls_searches, 1)
    # Cantidad de búsquedas por producto
    cantidad_busquedas = quantify_list(productos_buscados)
```

```
    # Ordena la lista de mayor a menor
    total_busqueda_y_ids = sort_by_n_element(cantidad_busquedas, 1, 0, rev=True)
    # Lista de producto no buscado
    producto_no_buscado = didnt_appear(ls_products, 0, ls_searches, 1)
    total_busqueda_y_ids = total_busqueda_y_ids + producto_no_buscado
    if op == 1 or op == 3:
        # OUTPUT
        # Cambiar ids a nombres para hacer amigable a la lectura
        for id_lst in total_busqueda_y_ids:
            for prd_lst in ls_products:
                if id_lst[1] == prd_lst[0]:
                    id_lst[1] = prod_name(prd_lst[1])
        # Crea la lista final con solo 100 valores
        if len(total_busqueda_y_ids) <= 99:
            print("\nNo se cuenta con 100 productos o más, mostrando", len(total_busqueda_y_ids), "productos")
            final_total_buscados_y_ids = total_busqueda_y_ids
        else:
            final_total_buscados_y_ids = total_busqueda_y_ids[0:100]
        print("\nLista de productos buscados (mayor a menor)")
        for out_buscados_ids in final_total_buscados_y_ids:
            if out_buscados_ids[0] >= 2:
                print(out_buscados_ids[1], "se ha buscado", out_buscados_ids[0], "veces")
            elif out_buscados_ids[0] == 1:
                print(out_buscados_ids[1], "se ha buscado", out_buscados_ids[0], "vez")
            else:
                print(out_buscados_ids[1], "no se ha buscado")
        if save == "si":
            # Guarda resultado en outputs/
```

```
        final_total_buscados_y_ids = [["Búsquedas", "Nombre del producto"]] + final_total_buscados_y_ids
        save_to_csv("100_mas_buscados", final_total_buscados_y_ids)


    # 1.2) Por categoría
    # Obtén la lista de categorías
    list_of_categories_shuffled = sort_by_n_element(ls_products, 3)
    list_of_categories = []
    for cat in list_of_categories_shuffled:
        if cat not in list_of_categories:
            list_of_categories.append(cat)
    list_of_categories.sort()
    # Lista para almacenar categoría y sus ids
    cat_and_id = []
    # Recorrer la lista de categorías
    for category in list_of_categories:
        # Lista temporal para almacenar los ids de cada categoría
        temp = []
        # Recorre sublistas en la lista de productos
        for sublist_2 in ls_products:
            # Si la categoría de la sublista es igual a la
            # categoría actual, almacena en lista temporal
            if sublist_2[3] == category:
                temp.append(sublist_2[0])
        # Agrega la categoría y sus ids a una nueva lista
        cat_and_id.append([category, temp])


    # Categorías con menos ventas
    sale_per_cat = []
```

```
    # Recorrer sublistas de ids de categorías
    for ids in cat_and_id:
        tmp = 0
        # Recorre lista de ventas
        for sell in ls_sales:
            # Si id se encuentra en lista de ventas
            # aumentar contador
            if sell[1] in ids[1]:
                tmp += 1
        # Agregar categoría y cuenta de ventas a la lista
        # reiniciar contador
        sale_per_cat.append([ids[0], tmp])
        tmp = 0
    # Ordenar de categorías con ventas de menor a mayor
    sale_per_cat = sort_by_n_element(sale_per_cat, 1, 0)
    if op == 1 or op == 4:
        # OUTPUT
        print("\nLista de ventas por categoría (menor a mayor)")
        for ventas_x_cat in sale_per_cat:
            if ventas_x_cat[0] >= 2:
                print("La categoría de", ventas_x_cat[1], "se ha vendido", ventas_x_cat[0],
"veces")
            elif ventas_x_cat[0] == 1:
                print("La categoría de", ventas_x_cat[1], "se ha vendido", ventas_x_cat[0], "vez")
            else:
                print("La categoría de", ventas_x_cat[1], "no se ha vendido")
        if save == "si":
            # Guarda resultados en csv
            sale_per_cat = [["Ventas", "Categoría"]] + sale_per_cat
```

```
        save_to_csv("cat_con_menos_ventas", sale_per_cat)


# Categorías con menos búsquedas
search_per_cat = []
# Recorrer sublistas de ids y categorías
for ids in cat_and_id:
    tmp = 0
    # Recorre lista de ventas
    for search in ls_searches:
        # Si id se encuentra en lista de ventas
        # aumentar contador
        if search[1] in ids[1]:
            tmp += 1
    # Agregar categoría y cuenta de búsquedas a la lista
    # reiniciar contador
    search_per_cat.append([ids[0], tmp])
    tmp = 0
search_per_cat = sort_by_n_element(search_per_cat, 1, 0)
if op == 1 or op == 5:
    # OUTPUT
    print("\nLista de búsquedas por categoría (menor a mayor)")
    for busq_x_cat in search_per_cat:
        if busq_x_cat[0] >= 2:
            print("La categoría de", busq_x_cat[1], "se ha buscado", busq_x_cat[0], "veces")
        elif busq_x_cat[0] == 1:
            print("La categoría de", busq_x_cat[1], "se ha buscado", busq_x_cat[0], "vez")
        else:
            print("La categoría de", busq_x_cat[1], "no se ha buscado")
```

```
    if save == "si":
        # Guarda resultados en csv
        search_per_cat = [["Búsqueda", "Categoría"]] + search_per_cat
        save_to_csv("cat_con_menos_búsquedas", search_per_cat)

# 2)
# 20 mejores/peores reseñas
# Obtén ids para cada producto
product_ids = sort_by_n_element(ls_products, 0)

ids_packed_reviews = []
# Para cada id en la lista de productos
for ids in product_ids:
    # Lista usada para almacenar datos temporalmente
    tmp = []
    # Recorre lista de ventas
    for sell in ls_sales:
        # Si el id de la venta es el mismo que
        # el id actual
        if sell[1] == ids:
            # Agrega la calificación de la venta a
            # la lista temporal
            tmp.append(sell[2])
    # Agrega el id y las calificaciones a una
    # sola sublista de ids_packed_reviews
    ids_packed_reviews.append([ids, tmp])
# Calificación promedio para cada producto
ids_avg_revs = []
```

```
for c, review_pack in enumerate(ids_packed_reviews):
    if len(review_pack[1]) != 0:
        avg = sum(review_pack[1])/len(review_pack[1])
        ids_avg_revs.append([review_pack[0], avg])
# Ordenar calificación de productos de menor a mayor
ids_avg_revs = sort_by_n_element(ids_avg_revs, 1, 0)
# Cambiar ids a nombres
for id_lst in ids_avg_revs:
    for prd_lst in ls_products:
        if id_lst[1] == prd_lst[0]:
            id_lst[1] = prod_name(prd_lst[1])
# Ordenar ids para cada output
worst_revs = ids_avg_revs[0:20]
best_revs = ids_avg_revs[-1:-21:-1]
if op == 1 or op == 6:
    # OUTPUT
    print("\nLista de mejores calificaciones")
    for rev_1 in best_revs:
        print(rev_1[1], "tiene una calificación de", rev_1[0])
    best_revs = [["Calificación", "Producto"]] + best_revs
    if save == "si":
        # Guarda resultado en outputs/
        save_to_csv("mejor_calificación", best_revs)
if op == 1 or op == 7:
    print("\nLista de peores calificaciones")
    for rev_2 in worst_revs:
        print(rev_2[1], "tiene una calificación de", rev_2[0])
    worst_revs = [["Calificación", "Producto"]] + worst_revs
```

```
    if save == "si":
        # Guarda resultado en outputs/
        save_to_csv("peor_calificación", worst_revs)


# 3)
# Lista de ids, con mes y año de venta
ids_and_dates = []
for sublist_3 in ls_sales:
    ids_and_dates.append([sublist_3[1], int(sublist_3[3][3:5]), int(sublist_3[3][6:10])])
# Lista de los años con ventas
list_of_years_shuffled = sort_by_n_element(ids_and_dates, 2)
list_of_years = []
for yr in list_of_years_shuffled:
    if yr not in list_of_years:
        list_of_years.append(yr)
# Lista de ventas para cada año
ids_2002 = []
ids_2019 = []
ids_2020 = []
for year in ids_and_dates:
    if year[2] == list_of_years[0]:
        ids_2002.append(year)
    elif year[2] == list_of_years[1]:
        ids_2019.append(year)
    else:
        ids_2020.append(year)
# Lista de meses con ventas
list_of_months_shuffled = sort_by_n_element(ids_2020, 1)
```

```
list_of_months = []
for mt in list_of_months_shuffled:
    if mt not in list_of_months:
        list_of_months.append(mt)
# Lista con ids contabilizados y ordenados por mes
org_ids_2020 = []
for n in list_of_months:
    counter = 0
    ids_temp = []
    for i in ids_2020:
        if i[1] == n:
            counter = counter + 1
            ids_temp.append(i[0])
    org_ids_2020.append([ids_temp, counter, n])
# Lista de ids con precios
ids_prices = sort_by_n_element(ls_products, 0, 2)
# Obtenemos los ingresos por mes, los ingresos anuales
# y el total de ventas
income_per_month = []
anual_income = 0
total_sells = 0
for month in org_ids_2020:
    income = 0
    for i_id_price in ids_prices:
        for n_id in range(len(month[0])):
            if month[0][n_id] == i_id_price[0]:
                income = income + int(i_id_price[1])
                anual_income = anual_income + int(i_id_price[1])
```

```
                total_sells += 1
        income_per_month.append([income, month[1], month[2]])
    # Promedio mensual
    prom_mensual = int(total_sells/len(list_of_months))
    # Obtener lista de meses con mejores ventas (arriba del promedio)
    highlight_month = []
    for list_i in income_per_month:
        if list_i[1] > prom_mensual:
            highlight_month.append(list_i)
    # Ordenar mejores meses de mayor a menor
    highlight_sorted = sort_by_n_element(highlight_month, 1, rev=True)
    sort_highlight_month = []
    for i in highlight_sorted:
        for n in highlight_month:
            if i == n[1]:
                sort_highlight_month.append(n)
    if op == 1 or op == 8:
        print("\nIngreso anual:", anual_income, "\nVentas totales:",  total_sells)
        if save == "si":
            # Guardar a csv
            anual_totals = [["Ingreso anual", anual_income]] +  [["Venta total", total_sells]]
            save_to_csv("ingresos_ventas_anuales", anual_totals)
    if op == 1 or op == 9:
        print("\nPromedio mensual de ventas:", prom_mensual)
        if save == "si":
            # Guardar a csv
            prom_mensual = [["Promedio mensual de ventas", prom_mensual]]
            save_to_csv("promedio_mensual", prom_mensual)
```

```
    if op == 1 or op == 10:
        print("\nVentas por mes:")
        for element_list in income_per_month:
            if element_list[1] != 1:
                print("En el mes", element_list[2], "se tuvo una venta de",
                      element_list[1], "productos y un ingreso de", element_list[0])
            elif element_list[1] == 0:
                print("En el mes", element_list[2], "no se vendió producto")
            else:
                print("En el mes", element_list[2], "se tuvo una venta de",
                      element_list[1], "producto y un ingreso de", element_list[0])
        if save == "si":
            # Guardar a csv
            income_per_month = [["Ingreso", "Productos vendidos", "Mes"]] + income_per_month
            save_to_csv("ingresos_por_mes", income_per_month)
    if op == 1 or op == 11:
        print("\nMeses con ventas por encima del promedio:")
        for element_list in sort_highlight_month:
            print("En el mes", element_list[2], "se tuvo una venta de",
                  element_list[1], "productos y un ingreso de", element_list[0])
        if save == "si":
            # Guardar a csv
            sort_highlight_month = [["Ingreso", "Productos vendidos", "Mes"]] + sort_highlight_month
            save_to_csv("mejores_meses", sort_highlight_month)
```