EMTECH INSTITUTE

PROYECTO 1

INTRODUCCIÓN A PYTHON

REALIZADO POR

EMMANUEL ARTURO BOLAÑOS ESPINOZA

FECHA DE ENTREGA

6 DE SEPTIEMBRE DEL 2020

ÍNDICE

[INTRODUCCIÓN 2](#_Toc50294058)

[DEFINICIÓN DEL CÓDIGO 2](#_Toc50294059)

[SOLUCIÓN AL PROBLEMA 2](#_Toc50294060)

[CONCLUSIÓN 2](#_Toc50294061)

[ANEXO 1 2](#_Toc50294062)

# 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.

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

# CONCLUSIÓN

# 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)