

LIFESTORE

Informe anual de rendimientos

2020

ÍNDICE

ÍNDICE

INTRODUCCIÓN

DEFINICIÓN DEL CÓDIGO.

Funciones

clear()
login()
printing_dict()
printing_list()
bubble_sort()
groupby()

Código principal

Cinco productos con mayores ventas.
Diez productos con mayores búsquedas.
Cinco productos con menores ventas por categoría.
Diez productos con menores búsquedas por categoría.
Por categoría, cinco productos con mejores y peores reseñas.
Total de ingresos y ventas promedio mensuales, total anual y meses con más ventas al año.

SOLUCIÓN AL PROBLEMA

Productos con mejor tasa de conversión (búsqueda/venta)

Relación búsqueda/venta

CONCLUSIÓN

LIFESTORE | Informe Anual de rendimientos 2020

INTRODUCCIÓN

La tienda LifeStore tiene el el último trimestre una pérdida en el número de ventas y se presume que está correlacionado con una reducción en el número de búsquedas de algunos artículos. En el año 2020 frente a la pandemia el crecimiento de las ventas en línea favoreció a muchas tiendas que no necesitaban de un lugar físico para comercializar. Sin embargo, se debe considerar los problemas económicos que causa a nivel mundial una catástrofe de salud pública en los negocios relacionados con el comercio.

A continuación se presenta un análisis de la rotación de productos considerando:

01. Productos con mayores ventas y búsquedas.
02. Productos con menores ventas y búsquedas por categoría.
03. Por categoría, 5 productos con mejores y peores reseñas.
04. Total de ingresos y ventas promedio mensuales.
05. Total anual y meses con más ventas al año.

DEFINICIÓN DEL CÓDIGO.

[Link a repositorio de GitHub](#)

Importar datasets y librerías necesarias.

```
from lifestore_file import lifestore_products, lifestore_sales, lifestore_searches
from datetime import datetime
import random, os
import statistics
```

Funciones

clear()

La función **clear()** tiene como objetivo permitir limpiar la terminal donde se está corriendo el código para facilitar la lectura de los datos a mostrar, para ello debe identificar el sistema operativo y determina cuál es la que debe usar. En el caso de los usuarios de sistemas operativos como Unix/Linux/MacOS/BSD utiliza `os.system("clear")`, mientras que para usuarios DOS/Windows utiliza `os.system("cls")`.

```
def clear():
    if os.name == "posix":
        os.system("clear")
    elif os.name == ("ce", "nt", "dos"):
        os.system("cls")
```

login()

La función **login()** decoradora que ejecuta o no la [función main\(\)](#), realiza una autenticación del usuario pidiéndole con un usuario y una contraseña establecida. Cuenta los intentos de ingreso y si el usuario intenta ingresar 3 veces y su usuario o su contraseña son incorrectos, a la cuarta vez lo saca directamente del programa con el comando `exit()`.

```

def login(func):
    def wrapper():
        usuarioAccedio = False
        intentos = 0
        # Bienvenida!
        print('Bienvenido al sistema!\nInicio de sesión')

        # Recibo constantemente sus intentos
        while not usuarioAccedio:
            # Primero ingresa Credenciales
            usuario = input("Usuario: ")
            contras = input("Contraseña: ")
            intentos += 1
            # Reviso si el par coincide
            if usuario == 'daniel' and contras == 'daniel':
                usuarioAccedio = True
                clear()
                print(f'Hola de nuevo {usuario}!\n\nDatos del análisis. \n\n')
                func() #Correr función.
            else:
                clear()
                print(f'Tienes {3 - intentos} intentos restantes')
                if usuario == 'daniel':
                    print('Te equivocaste en la contraseña')
                else:
                    print(f'El usuario: "{usuario}" no esta registrado')

            if intentos == 3:
                exit()
    return wrapper

```

printing_dict()

La función **printing_dict()** recibe como variables de entrada un diccionario con la estructura {id del producto: variable a mostrar}, el título que debe aparecer en la parte de arriba de la tabla en la terminal, y una palabra que hace alusión a la variable a mostrar, por ejemplo: búsquedas o ventas. La función empareja el id del producto con su nombre comercial iterando cada elemento del diccionario de entrada, alojados en la lista lifestore_products.

```

def printing_dict(dict, title, word):
    string= '\n'+title
    for key, value in dict.items():
        string += '\n' +'El producto ' +'con ID ' + str(key) + ' con ' + str(value) + ' ' + word + ' es ' +
str(lifestore_products[int(key)-1][1])
    return print(string)

```

printing_list()

La función **printing_list()** tiene como variables de entrada la lista con estructura ['categoría del producto', {id del producto: variable a mostrar}], el título que quiere el usuario que aparezca en la

parte superior de la tabla y una palabra que es la que identifica los datos mostrados en la tabla, por ejemplo: búsquedas o ventas. Dicha función itera cada elemento de la lista de entrada para así poder emparejar el id del producto con su nombre comercial, separados en las diferentes categorías que tiene los productos de la tienda LifeStore alojados en la lista de listas lifestore_products. Para lograr esto, la función se vale de la función `printing_dict()` para poder mostrar en terminal los datos.

```
def printing_list(list, title, word):
    print('\n\n'+title)
    for i in list:
        printing_dict(i[1], title=str(i[0]).capitalize(), word=word)
```

bubble_sort()

La función `bubble_sort()` se utiliza para ordenar los elementos de menor a mayor de un diccionario utilizando el algoritmo de ordenamiento por burbuja. Realiza primero una lista con las llaves y los valores del diccionario. El algoritmo ordena en función de los valores, o la variable que se quiere ordenar, por ejemplo: las búsquedas o ventas de un producto, pero a su vez también ordena las llaves para que con esto se devuelva de la función un diccionario con los elementos con el nuevo orden descendente.

```
def bubble_sort(dict):
    values = list(dict.values())
    keys = list(dict.keys())
    n = len(values)

    for i in range(n):
        for j in range(0, n - i - 1):
            if values[j] < values[j+1]:
                values[j], values[j+1] = values[j+1], values[j]
                keys[j], keys[j+1] = keys[j+1], keys[j]

    sorted_dict = {keys[i]: values[i] for i in range(n)}
    return sorted_dict
```

groupby()

La función `groupby()` tiene como variable de entrada una lista de categorías, una lista de elementos a agrupar con formato [id del producto, valor de interés ventas/búsqueda, categoría del producto] y los primeros números de la lista que queremos que arroje. Esta función genera una lista temporal en la que va almacenando [id de los productos, valores de interés ventas/búsquedas] por categoría, para posteriormente agregarlo una lista con la estructura [categoría, [id de los productos, valores de interés ventas/búsquedas]]. La última lista utiliza la función `bubble_sort()` para ordenarlos. La función arroja una lista por categoría organizada de menor a mayor y solo los n primeros números.

```
def groupby(categories, sorted_categories, n):
    less_cate=[]
```

```

for j in categories:
    temp=[]
    for i in range(0, len(sorted_categories)):
        if sorted_categories[i][2] == j:
            temp.append([sorted_categories[i][0], sorted_categories[i][1]])
    temp_sorted = dict(list(bubble_sort(dict(temp)).items())[::-1][0:n])
    less_cate.append([j, temp_sorted])
return less_cate

```

Código principal

comienzo de la función principal y decorador de la función `login()`.

```

@login #Decorador para el login
def main():

```

Cinco productos con mayores ventas.

La lista `products` contiene el id de cada producto de la tienda LifeStore contenido en el dataset `lifestore_products`, mientras que la lista `id_prod_sales` contiene el id de los productos vendidos. Para poder contar el número de veces vendidos se usa un dictionary comprehension que genera el diccionario `count_sales` con la estructura `{id del producto: número de veces vendido}`. Se utiliza el método `.count()` contar las veces que aparece el producto en `id_prod_sales`. Posteriormente se usa la función `bubble_sort()` para ordenarlos de mayor a menor y generar el diccionario `sales_sorted`.

```

products = [a[0] for a in lifestore_products]
id_prod_sales = [a[1] for a in lifestore_sales]

count_sales = {int(product): id_prod_sales.count(product) for product in products}
sales_sorted = bubble_sort(count_sales)

```

Se genera una lista con los primeros 5 productos más vendidos de nombre `head_sales` y se imprime en terminal con la función `printing_dict()`.

```

head_sales = dict(list(sales_sorted.items())[0:5])
printing_dict(head_sales, title='\nLos 5 productos con mayores ventas son:', word= 'ventas')

```

Diez productos con mayores búsquedas.

La lista `id_prod_searches` contiene el id de cada producto buscado. Con dictionary comprehension se cuenta las veces que aparece el producto en la lista `id_prod_searches` y se guarda en `count_searches`. Con la función `bubble_sort()` se ordena de mayor a menor el diccionario `count_searches` por el número de búsquedas. Posteriormente, se genera el diccionario `head_searches` que contiene los 10 productos con mayores búsquedas y es mostrado en terminal con la función `printing_dict()`.

```

id_prod_searches = [a[1] for a in lifestore_searches]
count_searches = {int(product): id_prod_searches.count(int(product)) for product in products}
searches_sorted = bubble_sort(count_searches)

#Primeros 10 números de búsquedas
head_searches = dict(list(searches_sorted.items())[0:10])

```

```
printing_dict(head_searches, title='\nLos 10 productos con mayores búsquedas:',  
word='búsquedas')
```

Cinco productos con menores ventas por categoría.

La lista categories contiene todas las categorías de los productos de la tienda LifeStore, la lista category con el formato [[id del producto, categoría de dicho producto]] se obtiene de la iteración del dataset lifestore_products.

```
categories = ['procesadores', 'tarjetas de video', 'tarjetas madre', 'discos duros',  
             'memorias usb', 'pantallas', 'bocinas', 'audifonos']  
category = [[a[0], a[3]] for a in lifestore_products]
```

La lista sorted_sales_cate con el formato [[id del producto, número de ventas, categoría]] se categoriza con la función `groupby()` y se guarda en una nueva lista llamada less_sales_cate que junta los productos según su número de ventas de menor a mayor por su categoría. Estos datos son mostrados en la terminal con la función `printing_list()`.

```
sorted_sales_cate = [[key, value, category[key-1][1]]  
                     for key, value in count_sales.items()]  
less_sales_cate = groupby(categories, sorted_sales_cate, 5)  
  
printing_list(less_sales_cate, title='Los 5 productos con menos ventas por categoría son:',  
word='ventas')
```

Diez productos con menores búsquedas por categoría.

La lista sorted_sales_cate con el formato [[id del producto, número de ventas, categoría]] se categoriza con la función `groupby()` y se guarda en una nueva lista llamada less_sales_cate que junta los productos según su número de ventas de menor a mayor por su categoría. Estos datos son mostrados en la terminal con la función `printing_list()`.

```
sorted_searches_cate = [[key, value, category[key-1][1]]  
                        for key, value in count_searches.items()]  
less_searches_cate = groupby(categories, sorted_searches_cate, 10)  
  
printing_list(less_searches_cate, title='Los 10 productos con menos búsquedas por categoría  
son:', word='búsquedas')
```

Por categoría, cinco productos con mejores y peores reseñas.

En la lista sales_reviewed con la estructura [[id del producto, score de reseña]] se almacena cada una de las reseñas de las ventas hechas por la tienda LifeStore.

```
#Arreglo con id_producto y score de reseña.  
sales_reviewed = [[a[1], a[2]] for a in lifestore_sales]
```

Para generar la lista average_scores se itera cada id de los productos con un for, dentro se itera cada elemento de la lista sales_reviewed en donde se compara con un if si el id del producto de la lista sales_reviewed es igual al id de la primera iteración, si es igual lo almacena el score de la

venta de ese producto en una lista temporal. Como se pide que se omitan los productos que no fueron reseñados no se toman en cuenta con una estructura if. Se saca un promedio de la lista temporal para que average_score quede con la estructura [[id del producto, promedio de scores de reseña]].

```
#Agrupa de scores según id_product y sacar promedio de scores.  
average_scores=[]  
for j in range(1, 97):  
    temp=[]  
    for i in range(0, len(sales_reviewed)):  
        if str(sales_reviewed[i][0]) == str(j):  
            temp.append(sales_reviewed[i][1])  
    #Para no considerar los productos sin reseñas.  
    if temp != []:  
        average_scores.append([j, round(statistics.mean(temp), 2)])
```

Para organizar los productos según su score de manera descendente se utiliza la función **bubble_sort()** y se almacena en el diccionario sorted_scores.

```
#Ordenamiento descendente de id_producto y score promedio por producto.  
sorted_scores = bubble_sort(dict(average_scores))
```

Para mostrar en la terminal la lista con los 5 productos con mejores reseñas y la lista con los 5 productos con peores reseñas se usa la función **printing_dict()**

```
printing_dict(dict(list(sorted_scores.items())[0:5]), title='\nLa lista con los 5 productos con mejores  
reseñas son:', word='de score')  
printing_dict(dict(list(sorted_scores.items())[::-1][0:5]), title='\nLa lista con los 5 productos con  
peores reseñas son:', word='de score')
```

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

La lista date_count_sales con el formato [[precio del producto, fecha de venta]] se crea a partir de recorrer el dataset lifestore_sales y agregar el precio del producto contenido en el dataset lifestore_products y la fecha de la realización de la venta que es convertido a formato <día>/<mes>/<año> con datetime. Con if se omite los productos que son devueltos (returned).

```
date_count_sales =[[  
    lifestore_products[int(id_product)-1][2],  
    datetime.strptime(date, '%d/%m/%Y')]  
    for i, id_product, score, date, returned in lifestore_sales  
    if int(returned) != 1]
```

La lista sales_per_month con la estructura [[mes, número de ventas, ingreso mensual]] se crea a partir la iteración de los meses del año con un ciclo for y de una lista temporal (temp) que recorre los elementos de la lista date_count_sales y solo agrega los precios de los productos solo si coincide con el mes del ciclo for. Posteriormente al momento de agregar la lista temporal a sales_per_month suma todas las ventas que se hayan hecho en el mes con la función sum().

Par sacar la suma anual, en la lista sum_annual se recorre sólo las ingresos mensuales alojados en sales_per_month y se suman con la función sum().

```
sales_per_month=[]
```

```

for j in range(1, 13):
    temp=[i[0] for i in date_count_sales if int(i[1].month) == j]
    sales_per_month.append([j, len(temp), sum(temp)])

sum_annual= sum([a[2] for a in sales_per_month])

```

Para poder organizarlos se crea un diccionario months_sales con la estructura {mes:número de ventas}, que se ingresa a la función `bubble_sort()` y arroja los meses con más ventas en orden descendente en la lista month_highest_sales.

```

months_sales = {i[0]:i[1] for i in sales_per_month}
month_highest_sales = bubble_sort(months_sales)

```

En la lista m se alojan los meses del año en string. Para poder mostrarlos en terminal los meses con más ventas del año se tiene que crear una variable llamada string que contiene el título a la cual se le van a ir agregando las sentencias con el contenido, por ejemplo: “Abril con 74 ventas.”.

```

m = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre',
'Noviembre', 'Diciembre']
string= '\n\nLos meses con más ventas del año 2020 son:'
for key, value in month_highest_sales.items():
    string += '\n' + m[int(key)-1] +' con ' + str(value) + ' ventas.'
print(string)

```

Para mostrar en terminal la suma anual solo se utiliza `print()` de la variable `sum_annual`

```
print('\n\nEn ingreso anual del 2020 es:\n$' + str(sum_annual))
```

Para mostrar en terminal el ingreso neto y el número de ventas por mes se genera una variable string que contiene el título y se recorre la lista `sales_per_month` con un ciclo for en donde se van agregando los datos, por ejemplo: “Enero con 52 ventas y \$117738 de ingreso neto.”.

```

#Recorrer lista sales_per_month para mostrar en pantalla su contenido.
string= '\n\nEl ingreso neto y el número de ventas por mes del 2020:'
for i in sales_per_month:
    string += '\n' + m[int(i[0])-1] +' con ' + str(i[1]) + ' ventas ' + 'y $' + str(i[2]) + ' de ingreso neto.'
print(string)

```

[Link a repositorio de GitHub](#)

SOLUCIÓN AL PROBLEMA

Los 5 productos con mayores ventas.

| ID del producto | Ventas | Nombre comercial |
|-----------------|--------|----------------------------------------------------------------------------------------------------|
| 54 | 50 | Disco Duro SSD Kingston A400, 120GB, SATA III, 2.5", 7mm |
| 3 | 42 | Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth |

| | | |
|----|----|-------------------------------------------------------------------------------------------------------|
| 5 | 20 | Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake) |
| 42 | 18 | Tarjeta Madre ASRock Micro ATX B450M Steel Legend, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD |
| 57 | 15 | Disco Duro SSD Adata Ultimate SU800, 256GB, SATA III, 2.5", 7mm |

Los 10 productos con mayores búsquedas.

| ID del producto | Búsquedas | Nombre comercial |
|-----------------|-----------|------------------------------------------------------------------------------------------------------------------------|
| 54 | 263 | Disco Duro SSD Kingston A400, 120GB, SATA III, 2.5", 7mm |
| 57 | 107 | Disco Duro SSD Adata Ultimate SU800, 256GB, SATA III, 2.5", 7mm |
| 29 | 60 | Tarjeta Madre ASUS micro ATX TUF B450M-PLUS GAMING, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD |
| 3 | 55 | Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth |
| 4 | 41 | Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire |
| 85 | 35 | Logitech Audífonos Gamer G635 7.1, Alámbrico, 1.5 Metros, 3.5mm, Negro/Azul |
| 67 | 32 | Pantalla TV Monitor LED 24TL520S-PU 24, HD, Widescreen, HDMI, Negro |
| 7 | 31 | Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake) |
| 5 | 30 | Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake) |
| 47 | 30 | Disco Duro SSD XPG SX8200 Pro, 256GB, PCI Express, M.2 |

Los meses con más ventas del año 2020 y sus ingresos mensuales promedio.

| Mes | Número de ventas | Ingreso mensual promedio |
|---------|------------------|--------------------------|
| Abril | 74 | \$191066 |
| Enero | 52 | \$117738 |
| Marzo | 49 | \$162931 |
| Febrero | 40 | \$107270 |
| Mayo | 34 | \$91936 |
| Junio | 11 | \$36949 |
| Julio | 11 | \$26949 |
| Agosto | 3 | \$3077 |

En los meses septiembre, octubre, noviembre, diciembre no hubo ventas y por lo tanto tampoco ingresos netos mensuales. Para realizar este análisis no se consideraron las devoluciones.

Ingreso anual

\$737,916

Productos con mejor tasa de conversión (búsqueda/venta)

54

50 veces vendido
263 veces buscado

Disco Duro SSD Kingston A400,
120GB, SATA III, 2.5", 7mm

15 veces vendido
107 veces buscado

Disco Duro SSD Adata Ultimate
SU800, 256GB, SATA III, 2.5", 7mm

57

3

42 veces vendido
45 veces buscado

Procesador AMD Ryzen 5 2600, S-AM4,
3.40GHz, Six-Core, 16MB L3 Cache,
con Disipador Wraith Stealth



El producto con el id de producto 31 con el nombre comercial **Tarjeta Madre AORUS micro ATX B450 AORUS M (rev. 1.0), S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD'**, 2229 tiene muchas devoluciones y un gran número de reseñas negativas.

Relación búsqueda/venta

Existen productos que fueron buscados en la página de LifeStore, pero no fueron comprados, lo que da a entender que el usuario sabe de la existencia del producto pero algo no lo hace comprarlo, las razones pueden ser alguna de las siguientes:

- El precio del producto no es competitivo respecto a otros vendedores.
- La experiencia de usuario genera mucha carga cognitiva y no facilita el proceso de selección y compra.
- No existe una campaña de marketing que después de ser buscado el producto le dé seguimiento al usuario para recordárselo, ya sea por correo electrónico o con anuncios personalizados en los diferentes canales.

Posibles soluciones:

- Abaratar costos con ofertas como cupones o mejorar costos de envío.
- Mejorar la experiencia de usuario y agilizar los botones que trasladan al usuario desde la lista de productos hasta la página de pago de una manera más ágil para que el proceso de compra sea más involuntario.
- Generar campañas automatizadas que recuerden a los usuarios los productos buscados en la página.

Productos por categoría que fueron buscados pero no comprados.

| Id del producto | | Nombre comercial |
|-------------------------|--|------------------------------------------------------------------------------------------------------------|
| Procesadores | | |
| 1 | | Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache |
| 6 | | Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake) |
| 8 | | Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generación - Coffee Lake) |
| 7 | | Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake) |
| 9 | | Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake) |
| Tarjeta de video | | |
| 27 | | Tarjeta de Video VisionTek AMD Radeon HD5450, 2GB GDDR3, PCI Express x16 |
| Discos duros | | |
| 59 | | Disco Duro SSD Samsung 860 EVO, 1TB, SATA III, M.2 |
| 56 | | Disco Duro SSD para Servidor Lenovo ThinkSystem S4500, 480GB, SATA III, 3.5", 7mm |
| Pantallas | | |
| 70 | | Pantalla Samsung Smart TV LED 43, Full HD, Widescreen, Negro |
| Bocina | | |
| 80 | | Ghia Bocina Portátil BX800, Bluetooth, Inalámbrico, 2.1 Canales, 31W, USB, Negro |
| Audífonos | | |

| | |
|----|---------------------------------------------------------------------------------------------|
| 93 | Ginga Audífonos con Micrófono GI18ADJ01BT-RO, Bluetooth, Alámbrico/Inalámbrico, 3.5mm, Rojo |
| 91 | Genius GHP-400S Audífonos, Alámbrico, 1.5 Metros, Rosa |

Existen productos que no fueron buscados en la página de LifeStore y tampoco fueron comprados, lo que da a entender que el usuario no sabe de la existencia del producto, las razones pueden ser alguna de las siguientes:

- Los productos que se tienen en stock no están en tendencia o no son necesitados por los usuarios.
- Los modelos de los productos pueden no ser los más recientes.
- Al ser productos especializados para fabricantes de computadoras puede que el usuario se abruma con términos difíciles y no conozca dichos productos.
- El SEO de la página puede que no esté optimizado y no aparezca en los primeros lugares en los principales motores de búsqueda, por lo tanto se pierda entre otras tantas tiendas.

Posibles soluciones:

- Es necesario generar la necesidad de compra de los productos con campañas personalizadas en redes sociales o con influencer marketing para que conozcan la página y los productos, principalmente los no buscados ni vendidos.
- Tener opciones de productos compatibles preseleccionados para facilitar la selección y la compra al usuario.
- Generar contenido en la propia página que funcione como explicaciones para el usuario y una breve reseña que explique qué hace y cómo se puede usar con otros dispositivos que se venden en la página.
- Identificar y priorizar en la lista de sugeridos a la hora de agregar al carrito productos “puente” o productos que inciten a comprar más de una cosa, especialmente de los productos sin ventas ni búsquedas.
- Optimizar el SEO de la página para que aparezca en las primeras páginas de los motores de búsqueda.

Productos que no han sido buscados ni tampoco tiene ventas.

| Id del producto | | Nombre comercial |
|--------------------------|--|---------------------------------------------------------------------------------------------------------------|
| Tarjetas de video | | |
| 24 | | Tarjeta de Video PNY NVIDIA GeForce RTX 2080, 8GB 256-bit GDDR6, PCI Express 3.0 |
| 23 | | Tarjeta de Video MSI Radeon X1550, 128MB 64 bit GDDR2, PCI Express x16 |
| 20 | | Tarjeta de Video Gigabyte NVIDIA GeForce RTX 2060 SUPER WINDFORCE OC, 8 GB 256 bit GDDR6, PCI Express x16 3.0 |
| Tarjetas madre | | |
| 43 | | Tarjeta Madre ASUS ATX ROG STRIX Z390-E GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel |
| 41 | | Tarjeta Madre ASUS micro ATX Prime H370M-Plus/CSM, S-1151, Intel H370, HDMI, 64GB DDR4 para Intel |

| | |
|---------------------|-------------------------------------------------------------------------------------------------------|
| 38 | Tarjeta Madre Gigabyte Micro ATX H310M DS2 2.0, S-1151, Intel H310, 32GB DDR4 para Intel |
| 37 | Tarjeta Madre ASRock ATX Z490 STEEL LEGEND, S-1200, Intel Z490, HDMI, 128GB DDR4 para Intel |
| Discos duros | |
| 58 | Disco Duro SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5", 7mm |
| 55 | Disco Duro SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s |
| 53 | Disco Duro SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2 |
| Memorias usb | |
| 61 | Memoria USB Kit Memoria RAM Corsair Vengeance LPX DDR4, 2400MHz, 32GB, Non-ECC, CL16 |
| Pantallas | |
| 72 | Pantalla Hisense Smart TV LED 50H8F 49.5, 4K Ultra HD, Widescreen, Negro |
| 71 | Pantalla Samsung Smart TV LED UN32J4290AF 32, HD, Widescreen, Negro |
| 69 | Pantalla Hisense Smart TV LED 40H5500F 39.5, Full HD, Widescreen, Negro |
| Bocinas | |
| 83 | Ghia Bocina Portátil BX500, Bluetooth, Inalámbrico, 10W RMS, USB, Gris |
| 82 | Ghia Bocina Portátil BX400, Bluetooth, Inalámbrico, 8W RMS, USB, Negro |
| 81 | Ghia Bocina Portátil BX900, Bluetooth, Inalámbrico, 2.1 Canales, 34W, USB, Negro - Resistente al Agua |
| 79 | Naceb Bocina Portátil NA-0301, Bluetooth, Inalámbrico, USB 2.0, Rojo |
| Audífonos | |
| 96 | Klip Xtreme Audífonos Blast, Bluetooth, Inalámbrico, Negro/Verde |
| 92 | Getttech Audífonos con Micrófono Sonority, Alámbrico, 1.2 Metros, 3.5mm, Negro/Rosa |

CONCLUSIÓN

A lo largo de este curso se logró usar el lenguaje de programación Python para lograr un análisis de datos de una baja en ventas y búsquedas de la tienda en línea LifeStore. Como medidas a tomar en cuenta se recomienda mejorar la experiencia de usuario en la página para facilitar el proceso de compra, generar estrategias que cautivan al consumidor como cupones o descuentos de envío y automatizar una campaña que recuerde de manera automática los productos buscados en la página LifeStore.

En el caso particular de los productos que no se vendieron y fueron buscados, es necesario generar *awareness* con campañas en redes o diferentes canales apropiados para el consumidor meta, tener productos preseleccionados que se relacionen entre sí para que generen una cadena

de consumo a la hora de revisar el carrito de compra. No estaría de más generar contenido propio para que los usuarios no expertos en computadora puedan saber qué hace el producto y cómo les puede beneficiar. Y por último, optimizar el SEO de la página para que aparezca en los primeros lugares de búsqueda en los principales motores de búsqueda.

Los productos no vendidos ni buscados deben primero rematarse para recuperar la inversión y después de aplicar las medidas para generar *awareness* en los usuarios se debe volver a hacer una valoración para ver si conviene o no retirarlos de stock. No se recomienda retirar el mercado sin aplicar las medidas antes mencionadas.