
REPORTE 01
**PROYECTO FINAL DE FUNDAMENTOS DE
PROGRAMACIÓN CON PYTHON**

BECAS SANTANDER TECNOLOGÍA:

**DESARROLLA COMPETENCIAS PARA LA REVOLUCIÓN
DIGITAL**

Óscar Hernández Terán

[https://github.com/oscarteran/
EmTech_Reporte01](https://github.com/oscarteran/EmTech_Reporte01)

Índice

1. Objetivo	3
2. Introducción	3
2.1. Descripción del caso	3
2.2. Bussines Intelligence	3
3. Desarrollo	3
3.1. Primera inspección	3
3.2. Login	5
4. Resultados	8
4.1. Productos vendidos y rezagados	8
4.1.1. 50 productos con mayores ventas.	8
4.1.2. Productos con mayores búsquedas.	8
4.1.3. 50 productos con menores ventas, por categoría.	9
4.1.4. Productos con menores búsquedas, por categoría.	10
4.2. Productos por reseña en el servicio	11
4.2.1. Mejores reseñas	12
4.2.2. Peores reseñas	12
4.3. Ventanas de tiempo	12
4.3.1. Ingresos por mes	13
4.3.2. Ingresos totales	14
4.3.3. Ingresos promedios	14
4.3.4. Año 2019	15
4.3.5. Año 2020	15
4.3.6. Meses con mayores ventas	15
5. Gráficos	16
6. Conclusiones	18
6.1. Recomendaciones	18
6.2. Conclusiones generales	19
7. Códigos	20

Índice de figuras

1. BD de búsquedas.	3
2. BD de ventas.	4
3. BD de Productos.	4
4. Menú de identificación.	5
5. Acceso permitido.	6
6. Interrupción del programa.	7
7. Top 5 en ventas.	8

8. Top 5 en búsquedas.	9
9. Menos ventas en Procesadores.	9
10. Menos ventas en Discos Duros	10
11. Menos búsquedas en Procesadores.	10
12. Menos búsquedas en Discos Duros	11
13. 10 productos mejor calificados.	12
14. 10 productos peor calificados.	12
15. Ingresos Mensuales	13
16. Ingreso total	14
17. Ingresos promedio mensual	14
18. Ingreso anual 2019	15
19. Ingreso anual 2020	15
20. Meses de mayores ventas	15
21. Ingresos netos por mes.	16
22. Ingresos promedios por mes.	17
23. Ventas por mes.	17
24. Menos ventas en Tarjetas de video.	18

Listings

1. Impresión de las BD	4
2. Impresión de las BD	6
3. Función <i>sorted</i>	8
4. Función <i>sorted</i>	8
5. Obtención de categorías	9
6. Estructura del diccionario.	9
7. Búsquedas por categoria	10
8. Lista de meses	13
9. Código completo	20

1. Objetivo

Poner en práctica las bases de programación en Python para análisis y clasificación de datos mediante la creación de programas de entrada de usuario y validaciones, uso y definición de variables y listas, operadores lógicos y condicionales para la clasificación de información.

2. Introducción

2.1. Descripción del caso

LifeStore es una tienda virtual que maneja una amplia gama de artículos, recientemente, la Gerencia de ventas, se percató que la empresa tiene una importante acumulación de inventario. Asimismo, se ha identificado una reducción en las búsquedas de un grupo importante de productos, lo que ha redundado en una disminución sustancial de sus ventas del último trimestre.

2.2. Bussines Intelligence

El problema aquí presentado es un claro ejemplo de lo que en el campo de la Inteligencia Artificial y Data Science se conoce como *Bussines Intelligence*, es decir, un conjunto de estrategias de tecnológicas, estadística, visualización de datos y algoritmos que ayudan a mejorar el comportamiento y rendimiento de las empresas a partir del estudio detallado de los datos existentes.

3. Desarrollo

3.1. Primera inspección

Como primer paso, realizaremos una inspección visual de las listas que funcionan como bases de datos. Esta inspección se va a realizar con ayuda de la función `print()`, dándole un formato legible para la correcta interpretación.

Tenemos un total de 3 bases de datos (BD), la primera correspondiente a la búsqueda de los productos (1), la segunda a las ventas totales realizadas (2), y por último la que contiene la información propia de cada producto (3). A continuación se muestran 3 figuras generadas para la impresión con formato legible.

```
-----  
Estructura de la BD de búsqueda:  
  
id_search: 1 | id_product: 1  
id_search: 2 | id_product: 1  
id_search: 3 | id_product: 1  
id_search: 4 | id_product: 1  
id_search: 5 | id_product: 1
```

Figura 1: BD de búsquedas.

```
-----
Estructura de la BD de ventas:
```

```
id_sale: 1 | id_product: 1 | score: 5 | date: 24/07/2020 | refund: 0
id_sale: 2 | id_product: 1 | score: 5 | date: 27/07/2020 | refund: 0
id_sale: 3 | id_product: 2 | score: 5 | date: 24/02/2020 | refund: 0
id_sale: 4 | id_product: 2 | score: 5 | date: 22/05/2020 | refund: 0
id_sale: 5 | id_product: 2 | score: 5 | date: 01/01/2020 | refund: 0
```

Figura 2: BD de ventas.

```
-----
Estructura de la BD de Productos:
```

```
id_sale: 1
name: Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache
price: 3019 | category: procesadores | stock: 16 |

id_sale: 2
name: Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth
price: 4209 | category: procesadores | stock: 182 |

id_sale: 3
name: Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth
price: 3089 | category: procesadores | stock: 987 |

id_sale: 4
name: Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire
price: 2209 | category: procesadores | stock: 295 |

id_sale: 5
name: Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)
price: 1779 | category: procesadores | stock: 130 |
```

Figura 3: BD de Productos.

Estas impresiones por terminal fueran generadas con las siguientes líneas de código:

```
1 # -----
2 # -----Primera inspección de la estructura de la Base de datos (BD)-----
3 #
4 print('*'*100)
5 print('Estructura de la BD de b quedó:\n')
6 for i in range(5):
7     print('id_search: ', lifestore_searches[i][0], ' | ', 'id_product: ',
8           lifestore_searches[i][1])
9
10 print('*'*100)
11 print('Estructura de la BD de ventas:\n')
12 for i in range(5):
13     print('id_sale: ', lifestore_sales[i][0], ' | ', 'id_product: ',
14           lifestore_sales[i][1], ' | ',
15           'score: ', lifestore_sales[i][2], ' | ', 'date: ',
16           lifestore_sales[i][3], ' | ',
17           'refund: ', lifestore_sales[i][4]
18
19 print('*'*100)
20 print('Estructura de la BD de Productos:\n')
21 for i in range(5):
22     print('id_sale: ', lifestore_products[i][0], '\n',
23           'name: ', lifestore_products[i][1], '\n',
```

```

22     'price:  ', lifestore_products[i][2], ' | ', 'category:  ',
23     lifestore_products[i][3], ' | ',
24     'stock:  ', lifestore_products[i][4], ' | \n',
)

```

Listing 1: Impresión de las BD

3.2. Login

Para la correcta ejecución del programa, se solicitó la verificación de la identidad del usuario a través de una cuenta y contraseña. El menú de acceso se despliega de la siguiente manera:

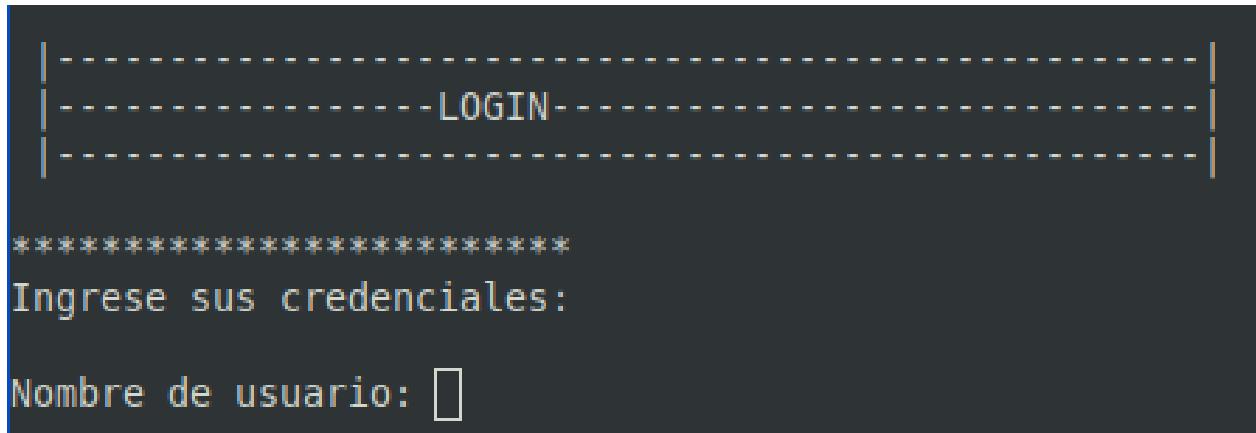


Figura 4: Menú de identificación.

Para este caso, se utilizaron las siguientes claves:

- **Usuario:** Oscar
- **Contraseña:** asdfgh

Como es de esperarse, en caso que las credenciales fueran correctas, la ejecución del programa continua para acceder a los siguientes bloques de código, de no ser este el caso, la ejecución del programa solicita un total de 5 veces las credenciales, si al 5to intento no se consigue, se interrumpe el programa y se vuelve imposible acceder a las siguientes instrucciones. El caso 1 se muestra en 5, mientras el 2 en 6.

```

-----LOGIN-----
*****
Ingrese sus credenciales:

Nombre de usuario: Oscar
Contraséña: asdfgh

*****
SUS DATOS SON CORRECTOS.
*****

Presione "Enter" para continuar:█

```

Figura 5: Acceso permitido.

El código para la correcta realización del **login** se muestra a continuación:

```

1 # -----
2 # -----Login-----
3 #
4 def login():
5     """
6         Esta función permite correr el resto del programa en
7         caso que el acceso sea el correcto.
8         En caso contrario, detendrá toda la ejecución.
9     """
10    print(' |-----|\n',
11        ' |-----LOGIN-----|\n',
12        ' |-----|\n')
13
14    user_C = 'Oscar'
15    pass_C = 'asdfgh'
16    Intentos = 0
17
18    while True:
19        print('*'*25)
20        print('Ingrese sus credenciales: \n')
21        user = input('Nombre de usuario: ')
22        passw = input('Contraseña: ')
23
24        Intentos += 1
25
26        if Intentos == 5:
27            print('*'*25)
28            print('Permiso denegado.')
29            print('Interrumpiendo ejecución del programa')

```

```

30         print('*'*25)
31         exit()
32     if user == user_C and passw == pass_C:
33         print()
34         print('*'*25)
35         print('SUS DATOS SON CORRECTOS.')
36         print('*'*25, '\n')
37         input('Presione "Enter" para continuar:')
38         break
39     else:
40         print('Intento fallido No. {} \n \n'.format(Intentos))
41
42 # Llamamos a la funcion
43 login()

```

Listing 2: Impresión de las BD

```

*****  

Ingrese sus credenciales:  

Nombre de usuario: Oscar  

Contraseña: sdfv  

Intento fallido No. 4  

*****  

Ingrese sus credenciales:  

Nombre de usuario: Oscar  

Contraseña: f  

*****  

Permiso denegado.  

Interrumpiendo ejecucion del programa  

*****

```

Figura 6: Interrupción del programa.

4. Resultados

4.1. Productos vendidos y rezagados

4.1.1. 50 productos con mayores ventas.

Para el punto número 1 se realizó, una lista que extrae los **id product** de la BD de ventas, posteriormente, a través de un ciclo se llena una nueva lista con la estructura = [id, nombre, ventas], dicha lista se ordena haciendo uso de la función *sorted* (3)

```
1 ventas_total = sorted(ventas_total, key=lambda x:x['total'], reverse=True)
```

Listing 3: Función *sorted*

Al únicamente pedir los primeros 50 productos, podemos hacer una asignación directa a través de la facilidad de acceder a los índices de una lista, como se muestra en 6

```
1 top_50 = ventas_total[:50]
```

Listing 4: Función *sorted*

Para no extender demasiado este documento con capturas muy amplias, se imprimen únicamente el top 5 de ventas (7), sin embargo, la lista ciertamente contiene los primeros 50.

```
-----Productos mas vendidos-----
Ventas : 50
Producto: SSD Kingston A400, 120GB, SATA III, 2.5'', 7mm

Ventas : 42
Producto: Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth

Ventas : 20
Producto: Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)

Ventas : 18
Producto: Tarjeta Madre ASRock Micro ATX B450M Steel Legend, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD

Ventas : 15
Producto: SSD Adata Ultimate SU800, 256GB, SATA III, 2.5'', 7mm
```

Figura 7: Top 5 en ventas.

4.1.2. Productos con mayores búsquedas.

Para acceder a los productos con mayores búsquedas, esencialmente se implementó la misma lógica que el el punto anterior. El resultado, nuevamente, de únicamente los primeros 5 productos, se puede ver en 8.

```
-----Productos mas buscados-----
Busquedas: 263
Producto: SSD Kingston A400, 120GB, SATA III, 2.5'', 7mm

Busquedas: 107
Producto: SSD Adata Ultimate SU800, 256GB, SATA III, 2.5'', 7mm

Busquedas: 60
Producto: Tarjeta Madre ASUS micro ATX TUF B450M-PLUS GAMING, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD

Busquedas: 55
Producto: Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth

Busquedas: 41
Producto: Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire
```

Figura 8: Top 5 en búsquedas.

4.1.3. 50 productos con menores ventas, por categoría.

En este punto, se trabajó con ayuda de diccionarios, las llaves de este fueron , precisamente las categorías en las que estaban clasificados los productos.

Para poder recuperar de manera automática una lista con los nombres de todas las categorías se utilizó el siguiente bloque de código:

```
1 categorias = [catego[3] for catego in lifestore_products]
2 unique_categorias = list(dict.fromkeys(categorias))
```

Listing 5: Obtención de categorías

El diccionario mencionado tiene la siguiente estructura:

```
1 categorias = {
2     'procesadores' : [] ,
3     'tarjetas de video' : [] ,
4     'tarjetas madre' : [] ,
5     'discos duros' : [] ,
6     'memorias usb' : [] ,
7     'pantallas' : [] ,
8     'bocinas' : [] ,
9     'audifonos' : [] ,
10 }
```

Listing 6: Estructura del diccionario.

A continuación, mostramos 2 de las 8 categorías, enseñando que productos se han vendido menos.

```
-----Productos con menos ventas-----
CATEGORIA: procesadores
Nombre: Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)
Ventas: 0
Nombre: Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache
Ventas: 2
Nombre: Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake)
Ventas: 3
Nombre: Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generación - Coffee Lake)
Ventas: 4
Nombre: Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)
Ventas: 7
Nombre: Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire
Ventas: 13
```

Figura 9: Menos ventas en Procesadores.

```

CATEGORIA: discos duros
Nombre: SSD Crucial MX500, 1TB, SATA III, M.2
Ventas: 1
Nombre: SSD Western Digital WD Blue 3D NAND, 2TB, M.2
Ventas: 2
Nombre: SSD Kingston UV500, 480GB, SATA III, mSATA
Ventas: 3
Nombre: Kit SSD Kingston KC600, 1TB, SATA III, 2.5, 7mm
Ventas: 3
Nombre: SSD Kingston A2000 NVMe, 1TB, PCI Express 3.0, M2
Ventas: 9
Nombre: SSD XPG SX8200 Pro, 256GB, PCI Express, M.2
Ventas: 11

```

Figura 10: Menos ventas en Discos Duros

4.1.4. Productos con menores búsquedas, por categoría.

Este apartado se logró de manera similar al anterior, recorremos tanto una lista que contiene las búsquedas, así como las lista de los productos, y a través de un condicional, en el momento que ambos iteradores coinciden en el nombre de producto, llenamos una lista que contienen el nombre del producto, y el número de búsquedas de cada uno. Véase 7.

```

1 # Organizamos por categoría
2 for busqueda in busquedas_total_i:
3     for productos in lifestore_products:
4         if busqueda['Name'] == productos[1]:
5             categorias_busqueda[productos[3]].append(
6                 [busqueda['Name'],
7                  busqueda['Busquedas']])

```

Listing 7: Búsquedas por categoría

El resultado, únicamente mostrando los primeros 5 productos, para las mismas dos categorías mostradas en 9 y 10 es el siguiente:

```

-----
-----Productos con menos búsquedas-----
-----
CATEGORIA: procesadores
Nombre: Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)
Busquedas: 1
Nombre: Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake)
Busquedas: 10
Nombre: Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache
Busquedas: 10
Nombre: Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generación - Coffee Lake)
Busquedas: 20
Nombre: Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth
Busquedas: 24
Nombre: Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)
Busquedas: 30

```

Figura 11: Menos búsquedas en Procesadores.

```
CATEGORIA: discos duros
Nombre: SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5'', 7mm
Busquedas: 0
Nombre: SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s
Busquedas: 0
Nombre: SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2
Busquedas: 0
Nombre: SSD Samsung 860 EVO, 1TB, SATA III, M.2
Busquedas: 1
Nombre: SSD para Servidor Lenovo Thinksystem S4500, 480GB, SATA III, 3.5'', 7mm
Busquedas: 2
Nombre: SSD Western Digital WD Blue 3D NAND, 2TB, M.2
Busquedas: 5
```

Figura 12: Menos búsquedas en Discos Duros

4.2. Productos por reseña en el servicio

Para completar este punto, se hizo uso de 8 listas auxiliares (Veáse [9](#)), separando variables en distintas listas, para tener Nombres únicos, id únicos, poder contar la cantidad de veces que se calificó un producto, y poder extraer el promedio de cada uno de éstos. Es necesario mencionar, que ciertos productos presentaban un promedio de tipo flotante, por lo que se cortó su parte decimal con la función de tipado *int*. Por último se tuvo un diccionario con la siguiente estructura:

- **Llave:** Nombre del producto
- **Valor:** Calificación

Este diccionario contiene todos los productos únicos con su correspondiente calificación, y está listo para poder ser ordenado y separado en sus 20 mejores y peores.

Como se ha venido trabajando, se muestran únicamente los primeros 10 productos por impresión para no saturar el documento.

4.2.1. Mejores reseñas

```
-----10 mejores calificados-----
Producto: Logitech Audifonos Gamer G635 7.1, Alámbrico, 1.5 Metros, 3.5mm, Negro/Azul
Valoracion: 5
Producto: Logitech Audifonos Gamer G332, Alámbrico, 2 Metros, 3.5mm, Negro/Rojo
Valoracion: 5
Producto: TV Monitor LED 24TL520S-PU 24, HD, Widescreen, HDMI, Negro
Valoracion: 5
Producto: TCL Smart TV LED 55S425 54.6, 4K Ultra HD, Widescreen, Negro
Valoracion: 5
Producto: Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB), Non-ECC, CL16, XMP
Valoracion: 5
Producto: SSD Western Digital WD Blue 3D NAND, 2TB, M.2
Valoracion: 5
Producto: SSD Crucial MX500, 1TB, SATA III, M.2
Valoracion: 5
Producto: Kit SSD Kingston KC600, 1TB, SATA III, 2.5, 7mm
Valoracion: 5
Producto: Tarjeta Madre Gigabyte XL-ATX TRX40 Designare, S-STRX4, AMD TRX40, 256GB DDR4 para AMD
Valoracion: 5
Producto: Tarjeta de Video Zotac NVIDIA GeForce GTX 1660 Ti, 6GB 192-bit GDDR6, PCI Express x16 3.0
Valoracion: 5
```

Figura 13: 10 productos mejor calificados.

4.2.2. Peores reseñas

```
-----10 peores calificados-----
Producto: Tarjeta de Video Gigabyte AMD Radeon R7 370 OC, 2GB 256-bit GDDR5, PCI Express 3.0
Valoracion: 1
Producto: Tarjeta Madre AORUS micro ATX B450 AORUS M (rev. 1.0), S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD
Valoracion: 1
Producto: Tarjeta Madre ASRock ATX H110 Pro BTC+, S-1151, Intel H110, 32GB DDR4, para Intel
Valoracion: 1
Producto: Tarjeta Madre Gigabyte micro ATX GA-H110M-DS2, S-1151, Intel H110, 32GB DDR4 para Intel
Valoracion: 2
Producto: Cougar Audifonos Gamer Phontum Essential, Alámbrico, 1.9 Metros, 3.5mm, Negro.
Valoracion: 3
Producto: Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth
Valoracion: 4
Producto: Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth
Valoracion: 4
Producto: Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire
Valoracion: 4
Producto: Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)
Valoracion: 4
Producto: MSI GeForce 210, 1GB GDDR3, DVI, VGA, HDCP, PCI Express 2.0
Valoracion: 4
```

Figura 14: 10 productos peor calificados.

4.3. Ventanas de tiempo

El primer paso en este punto fue ordenar las fechas, para esto nos ayudamos de la biblioteca *datetime*. Para trabajar con mayor tranquilidad se realizó una copia de la base de datos. Comenzamos cambiando el formato de la fecha de string a un tipo de dato propio de fechas, para poder trabajar más cómodamente. Una vez hecho esto, ordenamos nuestra BD. Principalmente nos auxiliamos creando una lista con estos valores como estructura:

- id, nombre, fecha, precio

Se extrajo el número correspondiente a cada mes del año de manera única, se contó la cantidad de operaciones realizadas por mes, y posterior a esto se creó una lista que contiene los límites de los índices entre los cuales se separa cada operación realizada por mes. Se define una lista que contiene desde el primer hasta el último mes de los cuales se tiene registro. (8)

```

1 llaves_mes = ['Noviembre', 'Diciembre', 'Enero', 'Febrero', 'Marzo',
2                 'Abril', 'Mayo', 'Junio', 'Julio',
3                 'Agosto', 'Septiembre']
```

Listing 8: Lista de meses

A partir, principalmente de estas variables y algunas listas auxiliares (Veáse 9) se logró obtener los resultados solicitados. Se muestran a continuación.

4.3.1. Ingresos por mes

```

-----Ingresos por mes-----
Mes: Noviembre
Ingreso: 4209
Mes: Diciembre
Ingreso: 0
Mes: Enero
Ingreso: 120237
Mes: Febrero
Ingreso: 110139
Mes: Marzo
Ingreso: 164729
Mes: Abril
Ingreso: 193295
Mes: Mayo
Ingreso: 96394
Mes: Junio
Ingreso: 36949
Mes: Julio
Ingreso: 26949
Mes: Agosto
Ingreso: 3077
Mes: Septiembre
Ingreso: 4199
```

Figura 15: Ingresos Mensuales

4.3.2. Ingresos totales

```
-----  
----- Ingresos totales -----  
-----  
Ingresos totales: 760177
```

Figura 16: Ingreso total

4.3.3. Ingresos promedios

```
-----  
----- Ingresos promedio -----  
-----  
Mes: Noviembre  
Ingreso: 4209.0  
Mes: Diciembre  
Ingreso: 0.0  
Mes: Enero  
Ingreso: 2932.609756097561  
Mes: Febrero  
Ingreso: 2159.5882352941176  
Mes: Marzo  
Ingreso: 2196.3866666666668  
Mes: Abril  
Ingreso: 5369.305555555556  
Mes: Mayo  
Ingreso: 8763.09090909091  
Mes: Junio  
Ingreso: 3359.0  
Mes: Julio  
Ingreso: 8983.0  
Mes: Agosto  
Ingreso: 3077.0
```

Figura 17: Ingresos promedio mensual

4.3.4. Año 2019

```
-----  
----- Ingresos 2019 -----  
-----  
Ingreso en el 2019: 4209
```

Figura 18: Ingreso anual 2019

4.3.5. Año 2020

```
-----  
----- Ingresos 2020 -----  
-----  
Ingreso en el 2020: 755968
```

Figura 19: Ingreso anual 2020

4.3.6. Meses con mayores ventas

```
-----  
----- Top 3 meses -----  
-----  
Mes: Abril  
Ingreso: 193295  
Mes: Marzo  
Ingreso: 164729  
Mes: Enero  
Ingreso: 120237
```

Figura 20: Meses de mayores ventas

5. Gráficos

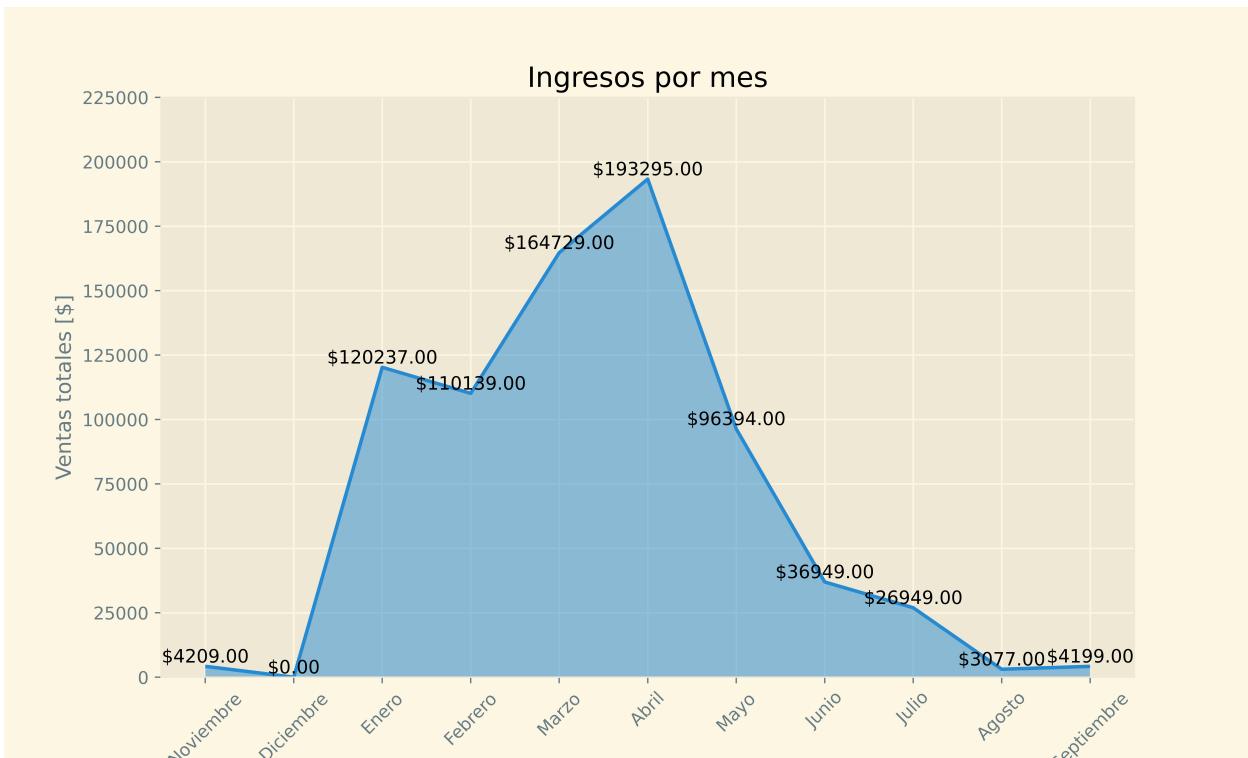


Figura 21: Ingresos netos por mes.

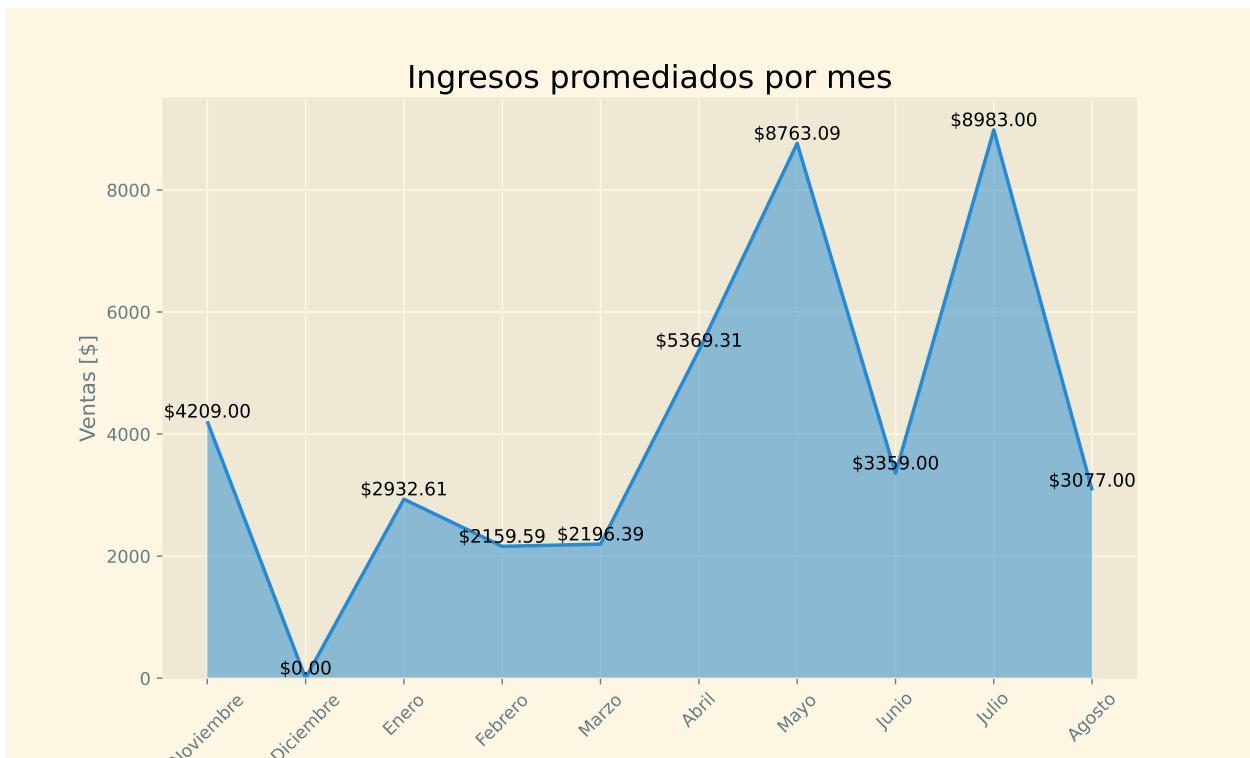


Figura 22: Ingresos promedios por mes.

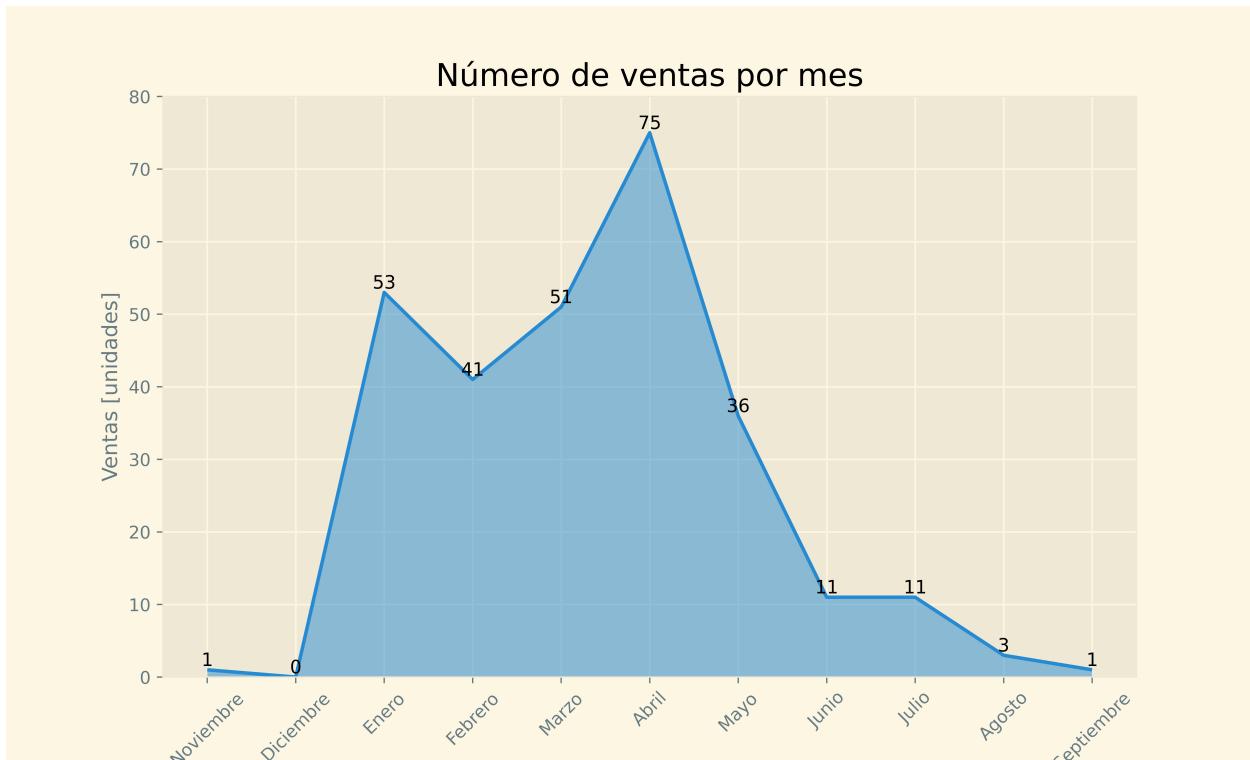


Figura 23: Ventas por mes.

6. Conclusiones

6.1. Recomendaciones

A través de la información recuperada a lo largo del código, se pueden hacer inferencias que ayuden a mejorar el funcionamiento de **LifeStore**.

1. Los productos que se muestran en [9](#) y [10](#), así como los contenidos en la variable *categorias* del código [9](#) tienen el potencial de ser retirados del mercado por sus bajas ventas.
2. Revisando la lista *stockorder* es posible observar concretamente que artículos presentan un sobre inventario.

Siendo más concretos en los puntos 1 y 2 antes mencionados, como sugerencia, se deben eliminar del mercado todos aquellos productos que presentan 0, 1, 2 y 3 ventas, son tan pocos demandados que representan una pérdida para la empresa. Por dar algunos ejemplos tenemos:

- Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)
- SSD Crucial MX500, 1TB, SATA III, M.2
- Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB), Non-ECC, CL16, XMP

Donde más se presenta este comportamiento en los productos es en la categoría de tarjetas de vídeo ([24](#)), aquí se sugiere dejar únicamente las 3 que presenten mayores ventas.

```
CATEGORIA: tarjetas de video
Nombre: Tarjeta de Video PNY NVIDIA GeForce RTX 2080, 8GB 256-bit GDDR6, PCI Express 3.0
Ventas: 0
Nombre: Tarjeta de Video MSI Radeon X1550, 128MB 64 bit GDDR2, PCI Express x16
Ventas: 0
Nombre: Tarjeta de Video Gigabyte NVIDIA GeForce RTX 2060 SUPER WINDFORCE OC, 8 GB 256 bit GDDR6, PCI Express x16 3.0
Ventas: 0
Nombre: Tarjeta de Video Gigabyte NVIDIA GeForce GTX 1650 OC Low Profile, 4GB 128-bit GDDR5, PCI Express 3.0 x16
Ventas: 0
Nombre: Tarjeta de Video EVGA NVIDIA GeForce RTX 2060 SC ULTRA Gaming, 6GB 192-bit GDDR6, PCI Express 3.0
Ventas: 0
Nombre: Tarjeta de Video EVGA NVIDIA GeForce GTX 1660 Ti SC Ultra Gaming, 6GB 192-bit GDDR6, PCI 3.0
Ventas: 0
```

Figura 24: Menos ventas en Tarjetas de video.

Para el punto 2, la sugerencia es que aquellos productos que presentan pocas ventas, se haga un análisis para encontrar el punto de equilibrio a futuro, que quiere decir esto, expliquemos con un ejemplo:

El producto '*Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth*' esta en existencias 987 veces. Este mismo producto presentó en, al rededor de 10 meses, 42 ventas, lo que da un aproximado de 4 ventas al mes. Esto significa que se necesitan más menos **246 meses** para vender todo el inventario.

Así entonces, se propone tener inventario para 1 año, tomando en cuenta las ventas promedio de cada artículo por mes para calcular cuantas unidades serán necesarias.

6.2. Conclusiones generales

Con el análisis de datos es posible ver comportamientos y variables que con una simple inspección pasan desapercibidos. Python es un lenguaje multipropósito, de lato nivel, fácil de aprender y sumamente versátil, además cuenta con un amplio campo de aplicación al Data Science.

Este trabajo presenta la manera en la que sin uso de paqueterías especializadas es posible generar información que nos ayuda a la toma de decisiones y a mejorar el rendimiento de las empresas a nivel de eficiencia.

Más específicamente, este trabajo nos muestra el potencial que tiene el uso de la Ciencia de datos a los negocios, pero también nos permite ver que realmente en cualquier área de la vida en la que se tenga un conjunto de datos *ad hoc* es posible mejorar el rendimiento, o bien ver patrones oscuros a primera vista.

7. Códigos

```

1 """
2 Autor:
3     OSCAR HERNANDEZ TERAN
4
5 Instrucciones del programa:
6     1. Categorias con menores ventas y categorias con menores busquedas
7     2. Categorias con mayores ventas y categorias con mayores busquedas
8     3. Sugerir una estrategia de productos a retirar del mercado asi
9         como sugerencias de como reducir la acumulacion de inventario
10        considerando los datos mensuales.
11
12
13
14 -----
15 -----INFORMACION A CERCA DE LA BASE DE DATOS-----
16 -----
17 This is the LifeStore_SalesList data:
18
19 lifestore_searches = [id_search, id_product]
20 lifestore_sales = [id_sale, id_product, score (from 1 to 5), date, refund
21     (1 for true or 0 to false)]
22 lifestore_products = [id_product, name, price, category, stock]
23
24 -----
25 -----NOTA-----
26 ----- Por buena practica, y para que el codigo pueda ser -----
27 ----- plasmado de manera correcta en LATEX, este programa -----
28 ----- sera comentado sin acentos-----
29
30
31 """
32
33 # Importamos la Base de datos
34 from lifestore_file import lifestore_products, lifestore_sales,
35     lifestore_searches
36
37
38 # -----
39 # -----Primera inspeccion de la estructura de la Base de datos (BD)-----
40 # -----
41 print('*'*100)
42 print('Estructura de la BD de b quedas:\n')
43 for i in range(5):
44     print('id_search: ', lifestore_searches[i][0], ' | ', 'id_product: ',
45         lifestore_searches[i][1])
46
47 print('*'*100)
48 print('Estructura de la BD de ventas:\n')
49 for i in range(5):
50     print('id_sale: ', lifestore_sales[i][0], ' | ', 'id_product: ',
51         lifestore_sales[i][1], ' | ',
52         lifestore_sales[i][2], ' | ',
53         lifestore_sales[i][3], ' | ',
54         lifestore_sales[i][4])

```

```
50         'score: ', lifestore_sales[i][2], ' | ', 'date: ',
51         lifestore_sales[i][3], ' | ',
52         'refund: ', lifestore_sales[i][4]
53     )
54
55 print('-'*100)
56 print('Estructura de la BD de Productos:\n')
57 for i in range(5):
58     print('id_sale: ', lifestore_products[i][0], '\n',
59           'name: ', lifestore_products[i][1], '\n',
60           'price: ', lifestore_products[i][2], ' | ', 'category: ',
61           lifestore_products[i][3], ' | ',
62           'stock: ', lifestore_products[i][4], ' | \n',
63           )
64
65 # -----Login-----
66 #
67 def login():
68     """
69     Esta función permite correr el resto del programa en
70     caso que el acceso sea el correcto.
71     En caso contrario, detendrá toda la ejecución.
72     """
73     print(' |-----| \n',
74           ' |-----LOGIN-----| \n',
75           ' |-----| \n')
76
77 user_C = 'Oscar'
78 pass_C = 'asdfgh'
79 Intentos = 0
80
81 while True:
82     print('*'*25)
83     print('Ingrese sus credenciales: \n')
84     user = input('Nombre de usuario: ')
85     passw = input('Contraseña: ')
86
87     Intentos += 1
88
89     if Intentos == 5:
90         print('*'*25)
91         print('Permiso denegado.')
92         print('Interrumpiendo ejecución del programa')
93         print('*'*25)
94         exit()
95     if user == user_C and passw == pass_C:
96         print()
97         print('*'*25)
98         print('SUS DATOS SON CORRECTOS.')
99         print('*'*25, '\n')
100        input('Presione "Enter" para continuar: ')
101        break
102    else:
103        print('Intento fallido No. {} \n'.format(Intentos))
```

```
104
105 # Llamamos a la funcion
106 login()
107
108
109 # -----
110 # ----- Productos mas vendidos -----
111 #
112
113 # Listamos los 50 productos mas vendidos
114 id_s = []
115 ventas_total = []
116
117 # id
118 for ventas in lifestore_sales:
119     id_s.append(ventas[1])
120
121 # id, nombre, numero de ventas
122 for idx, produc in enumerate(lifestore_products):
123     ventas_total.append({
124         'id': produc[0],
125         'name': produc[1],
126         'total': id_s.count(idx+1)
127     })
128
129 # Ordenamos a partir de una llave
130 ventas_total = sorted(ventas_total, key=lambda x:x['total'], reverse=True)
131
132 # 50 articulos mas vendidos
133 top_50 = ventas_total[:50]
134
135
136 # Impresion de los primeros 5 productos
137 print('-----')
138 print('-----Productos mas vendidos-----')
139 print('-----')
140 for i in range(5):
141     print(
142         'Ventas : ', top_50[i]['total'], '\n',
143         'Producto: ', top_50[i]['name'], '\n',
144     )
145
146
147 # -----
148 # ----- Productos mas buscados -----
149 #
150 _ids = []
151 busquedas_t = []
152
153 # id
154 for busq in lifestore_searches:
155     _ids.append(busq[1])
156
157 # id, nombre, numero de busquedas
158 for produc in lifestore_products:
159     busquedas_t.append(
```

```

160     {'id':produc[0],
161      'Name':produc[1],
162      'Busquedas':_ids.count(produc[0])
163    }
164  )
165
166 busquedas_total = sorted(busquedas_t, key=lambda x:x['Busquedas'], reverse
167 =True)
168
169 # Impresion de los primeros 5 productos
170 print('-----')
171 print('-----Productos mas buscados-----')
172 print('-----')
173 for i in range(5):
174     print(
175         ' Busquedas: ', busquedas_total[i]['Busquedas'], '\n',
176         'Producto: ', busquedas_total[i]['Name'], '\n',
177     )
178
179
180 # -----
181 # ----- 50 menores ventas por categoria -----
182 #
183
184 # Usamos compresion de listas
185 # categorias
186 categorias = [catego[3] for catego in lifestore_products]
187 unique_categorias = list(dict.fromkeys(categorias))
188
189
190 categorias = {
191     'procesadores' :[],
192     'tarjetas de video':[],
193     'tarjetas madre' :[],
194     'discos duros' :[],
195     'memorias usb' :[],
196     'pantallas' :[],
197     'bocinas' :[],
198     'audifonos' :[]
199 }
200
201 # Invertimos el orden de venta
202 ventas_total_50 = ventas_total[:50]
203 ventas_total_i = ventas_total_50[::-1]
204
205 # Organizamos por categoria
206 for ventas in ventas_total_i:
207     for productos in lifestore_products:
208         if ventas['name'] == productos[1]:
209             categorias[productos[3]].append([ventas['name'], ventas['total'],
210             ]])
211
212 # Impresion de los primeros 5 productos por categoria
213 print('-----')

```

```

214 print('-----Productos con menos ventas-----')
215 print('-----')
216 for name in categorias:
217     print('CATEGORIA: ', name)
218     for i in range(len(categorias[name])):
219         if i > 5:
220             break
221         else:
222             print(' Nombre: ', categorias[name][i][0], '\n',
223                   'Ventas: ', categorias[name][i][1] )
224     print()
225
226 # -----
227 # ----- 100 con menores busquedas por categoria-----
228 #
229 categorias_busqueda = {
230     'procesadores' : [] ,
231     'tarjetas de video' : [] ,
232     'tarjetas madre' : [] ,
233     'discos duros' : [] ,
234     'memorias usb' : [] ,
235     'pantallas' : [] ,
236     'bocinas' : [] ,
237     'audifonos' : []
238 }
239
240
241 # Invertimos la lista de busqueda
242 busquedas_total_i = busquedas_total[::-1]
243
244 # Organizamos por categoria
245 for busqueda in busquedas_total_i:
246     for productos in lifestore_products:
247         if busqueda['Name'] == productos[1]:
248             categorias_busqueda[productos[3]].append(
249                 [busqueda['Name'],
250                  busqueda['Busquedas']])
251
252
253 # Impresion de los primeros 5 productos por categoria
254 print('-----')
255 print('-----Productos con menos busquedas-----')
256 print('-----')
257 for name in categorias_busqueda:
258     print('CATEGORIA: ', name)
259     for i in range(len(categorias_busqueda[name])):
260         if i > 5:
261             break
262         else:
263             print(' Nombre: ', categorias_busqueda[name][i][0], '\n',
264                   'Busquedas: ', categorias_busqueda[name][i][1] )
265     print()
266
267
268 # -----
269 # -----Productos por calificacion en el servicio-----

```

```
270 # -----
271 # id, Nombre, Calificación
272 valoraciones = []
273 for ventas in lifestore_sales:
274     for produ in lifestore_products:
275         if ventas[1] == produ[0]:
276             valoraciones.append([ventas[1], produ[1], ventas[2]])
277
278 # id
279 id_produc = []
280 for i in range(len(valoraciones)):
281     id_produc.append(valoraciones[i][0])
282
283 # Nombres
284 ventas_nombres = []
285 for i in range(len(valoraciones)):
286     ventas_nombres.append(valoraciones[i][1])
287
288 # id único
289 id_unico = list(dict.fromkeys(id_produc))
290
291 # Nombres únicos
292 ventas_unicas = list(dict.fromkeys(ventas_nombres))
293
294 # Nombres, # de valoraciones
295 repeticiones = []
296 for repe in ventas_unicas:
297     repeticiones.append([repe, ventas_nombres.count(repe)])
298
299 # Lista de listas de valoraciones
300 lista_score = []
301 for id in id_unico:
302     aux = []
303     for ventas in lifestore_sales:
304         if ventas[1] == id:
305             aux.append(ventas[2])
306     lista_score.append(aux)
307
308 # Valores promedios
309 promedios = []
310 for idx, lista in enumerate(lista_score):
311     aux = int(sum(lista)/repeticiones[idx][1])
312     promedios.append(aux)
313
314 # Nombre, valoración
315 valoraciones_finales = dict(zip(ventas_unicas, promedios))
316
317 # Ordenamos el diccionario
318 valoraciones_finales = sorted(valoraciones_finales.items(), key=lambda x:x[1])
319 valoraciones_finales_i = valoraciones_finales[::-1]
320
321
322 top_20_valores = valoraciones_finales_i[:20]
323 bottom_20_valores = valoraciones_finales[:20]
324
```

```
325 print('-----')
326 print('-----10 mejores calificados-----')
327 print('-----')
328 for i in range(10):
329     print(' Producto: ', top_20_valores[i][0], '\n',
330           'Valoracion: ', top_20_valores[i][1] )
331
332 print('-----')
333 print('-----10 peores calificados-----')
334 print('-----')
335 for i in range(10):
336     print(' Producto: ', bottom_20_valores[i][0], '\n',
337           'Valoracion: ', bottom_20_valores[i][1] )
338
339 # -----
340 # -----Ingresos y ventas en ventanas de tiempo-----
341 #
342 # Total de ingresos y ventas promedio mensuales,
343 # total anual y meses con más ventas al año
344
345 # Importamos las librerías necesarias
346 from datetime import datetime
347
348 # Creamos una copia para trabajar
349 lifestore_sales_copy = lifestore_sales.copy()
350
351 # Cambiamos el tipo de dato
352 for value in range(len(lifestore_sales_copy)):
353     lifestore_sales_copy[value][3] = datetime.strptime(
354         lifestore_sales_copy[value][3], '%d/%m/%Y')
355
356 # Nueva lista
357 # id, nombre, fecha, precio
358 ingresos = []
359 for idx,lista in enumerate(lifestore_sales_copy):
360     for produc in lifestore_products:
361         if lista[1] == produc[0]:
362             ingresos.append([produc[0],
363                             produc[1],
364                             lista[3],
365                             produc[2]
366                         ])
367
368 # Ordenamos en función de la fecha
369 ingresos_ord = sorted(ingresos, key=lambda x:x[2])
370
371 # Extraemos los meses de cada fecha de venta
372 meses = []
373 for i in range(len(ingresos_ord)):
374     meses.append(datetime.strftime(ingresos_ord[i][2], '%m'))
375
376 # Creamos una lista con los meses sin repetir
377 meses_unicos = list(dict.fromkeys(meses))
378
379 # mes, # de ventas
```

```
380 longitudes = []
381 for mes in meses_unicos:
382     longitudes.append(meses.count(mes))
383
384 # Definimos una lista para los límites
385 limites = []
386 for i in range(len(longitudes)):
387     if i == 0:
388         limites.append(longitudes[i])
389     else:
390         limites.append(longitudes[i]+limites[i-1])
391
392 """
393 Dado que tenemos las longitudes y los límites,
394 podemos separar la lista original por mes
395 Lista de listas que contienen los meses de diciembre a septiembre
396 Cada lista interna tiene la forma:
397     id, nombre, fecha, precio
398 """
399
400 lista_meses = [[], [], [], [], [], [], [], [], [], []]
401
402 for i in range(len(limites)):
403     if i == 0:
404         lista_meses[i].append(ingresos_ord[0:limites[i]])
405     else:
406         lista_meses[i].append(ingresos_ord[limites[i-1]:limites[i]])
407
408 # Lista con los ingresos por mes
409 ingresos_por_mes = []
410 for i in range(len(longitudes)):
411     if i == 1:
412         ingresos_por_mes.append(0)
413     aux = 0
414     for lon in range(longitudes[i]):
415         aux += lista_meses[i][0][lon][-1]
416     ingresos_por_mes.append(aux)
417
418 llaves_mes = ['Noviembre', 'Diciembre', 'Enero', 'Febrero', 'Marzo',
419                 'Abril', 'Mayo', 'Junio', 'Julio',
420                 'Agosto', 'Septiembre']
421
422 # Creamos un diccionario
423 # mes:ingreso
424 ingresos_final = dict(zip(llaves_mes, ingresos_por_mes))
425
426
427 print('-----')
428 print('-----Ingresos por mes-----')
429 print('-----')
430 for key, value in ingresos_final.items():
431     print(' Mes: ', key, '\n',
432           'Ingreso: ', value)
433
434
435 # Ingresos totales
```

```
436 print('-----')
437 print('----- Ingresos totales -----')
438 print('-----')
439 print('Ingresos totales:', sum(ingresos_por_mes))
440
441
442 # Ventas promedio por mes
443 promedios = []
444 for i in range(len(longitudes)):
445     promedios.append(ingresos_por_mes[i]/longitudes[i])
446
447
448 # Diccionario
449 # mes: promedio
450 ingresos_promedios = dict(zip(llaves_mes, promedios))
451
452
453 print('-----')
454 print('----- Ingresos promedio -----')
455 print('-----')
456 for key, value in ingresos_promedios.items():
457     print(' Mes: ', key, '\n',
458           'Ingreso: ', value)
459
460 # A o 2019
461 print('-----')
462 print('----- Ingresos 2019 -----')
463 print('-----')
464 print('Ingreso en el 2019:', 4209)
465
466 # A o 2020
467 print('-----')
468 print('----- Ingresos 2020 -----')
469 print('-----')
470 print('Ingreso en el 2020:', sum(ingresos_por_mes) - 4209)
471
472 # Top 3 meses en venta
473 meses_ordenados = sorted(ingresos_final.items(), key=lambda x:x[1],
474                           reverse=True)
475
476 print('-----')
477 print('----- Top 3 meses -----')
478 print('-----')
479 for i in range(3):
480     print(' Mes: ', meses_ordenados[i][0], '\n',
481           'Ingreso:', meses_ordenados[i][1])
482
483 # Numero de ventas
484 tickets = []
485 for i in range(len(llaves_mes)):
486     if i == 1:
487         tickets.append(0)
488     else:
489         tickets.append(longitudes[i-1])
490
```

```

491
492 # -----
493 # -----EXTRA: Visualizaciones-----
494 #
495 import matplotlib.pyplot as plt
496 plt.style.use('Solarize_Light2')
497
498
499 # Ingresos por mes
500 plt.figure(figsize=(10,6))
501 plt.plot(ingresos_final.keys(), ingresos_final.values())
502 plt.fill_between(ingresos_final.keys(), ingresos_final.values(), alpha
503 =0.5)
504 for x, y in zip(ingresos_final.keys(), ingresos_final.values()):
505     plt.text(x, y, '$%.2f'%y, ha='center', va='bottom', fontsize=10.5)
506 plt.xlabel('Mes de venta')
507 plt.ylabel('Ventas totales [$]')
508 plt.title('Ingresos por mes', fontsize=16, c='k')
509 plt.xticks(rotation = 45)
510 plt.ylim(0, 225000)
511 plt.savefig('Ingresos_Por_Mes.jpg', dpi=900)
512 #plt.show()
513
514
515 # Ingresos promedios por mes
516 plt.figure(figsize=(10,6))
517 plt.plot(ingresos_promedios.keys(), ingresos_promedios.values())
518 plt.fill_between(ingresos_promedios.keys(), ingresos_promedios.values(),
519 alpha=0.5)
520 for x, y in zip(ingresos_promedios.keys(), ingresos_promedios.values()):
521     plt.text(x, y, '$%.2f'%y, ha='center', va='bottom', fontsize=10.5)
522 plt.xlabel('Mes de venta')
523 plt.ylabel('Ventas [$]')
524 plt.title('Ingresos promediados por mes', fontsize=18, c='k')
525 plt.xticks(rotation = 45)
526 plt.ylim(0, 9500)
527 plt.savefig('Ingresos_Promedio.jpg', dpi=900)
528 #plt.show()
529
530
531 # Numero de ventas por mes
532 plt.figure(figsize=(10,6))
533 plt.plot(ingresos_final.keys(), tickets)
534 plt.fill_between(ingresos_final.keys(), tickets, alpha=0.5)
535 for x, y in zip(ingresos_final.keys(), tickets):
536     plt.text(x, y, '%.0f'%y, ha='center', va='bottom', fontsize=10.5)
537 plt.xlabel('Mes de venta')
538 plt.ylabel('Ventas [unidades]')
539 plt.title('Número de ventas por mes', fontsize=18, c='k')
540 plt.xticks(rotation = 45)
541 plt.ylim(0, 80)
542 plt.savefig('Tickets.jpg', dpi=900)
543 #plt.show()

```

Listing 9: Código completo