

```
def request_seen(self, request):
    fp = self.request_fingerprint(request)
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + os.linesep)
    fingerprint(self, request):
```



PROYECTO 01: ANÁLISIS DE LA ROTACIÓN DE PRODUCTOS LIFESTORE

Edson Enrique Castañeda Mancillas

Grupo 3
Data Science

Tutor: Javier Ramírez



13 de Septiembre del 2021

Índice general

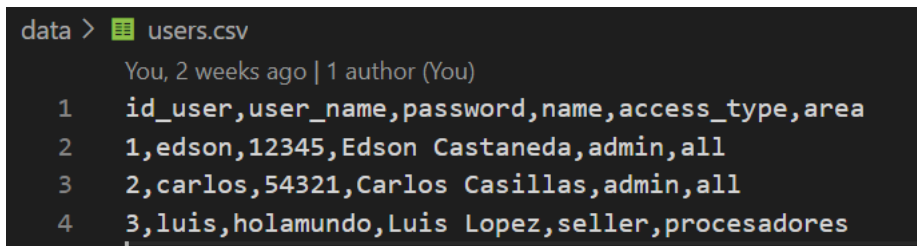
1	Introducción	2
2	Definición del código	3
2.1	Descripción del algoritmo	3
2.2	Implementación en Python	5
2.2.1	Login	5
2.2.2	Manejo inicial de datos	6
2.2.3	Procesamiento de datos	6
2.2.4	Librerías auxiliares	18
2.2.5	Función principal	22
2.3	Instrucciones de uso	28
2.3.1	Primer uso	28
2.3.2	Usos posteriores	28
3	Resultados	29
3.1	Demanda de productos	29
3.1.1	Productos con mayor demanda	29
3.1.2	Productos con menor demanda	33
3.2	Valoración del cliente a productos	37
3.2.1	Productos con mayor valoración	37
3.2.2	Productos con menor valoración	38
3.2.3	Productos sin valoración	39
3.3	Ventas e Ingresos en 2020	41
3.3.1	Resumen anual	41
3.3.2	Resumen mensual	41
3.4	Ventas e Ingresos por Categoría	44
4	Solución al problema	46
4.1	Opciones para mejorar venta de productos de menor demanda	46
4.2	Opciones de cambios de dirección que puede tomar la empresa para crecer su mercado	46
5	Conclusión	48

1. INTRODUCCIÓN

LifeStore es una tienda en línea de productos de computación. Recientemente, se ha detectado una acumulación de inventario, así como reducción en las búsquedas y ventas de muchos productos de su catálogo.

En el presente reporte, mediante el código en Python, se desarrolla un programa que permita al usuario acceder a registros de la empresa y poder hacer distintos tipos de consultas. Para poder realizar cualquier consultas, es necesario verificar que el usuario pueda ver dicha información, por lo que también se incluya una validación de ingreso.

Los usuarios permitidos se encuentran definidos en un archivo .CSV, donde se guardan datos como el nombre de la persona, su nombre de usuario y contraseña, como se muestra en la Fig. 1.1.



```
data > users.csv
You, 2 weeks ago | 1 author (You)
1 id_user,user_name,password,name,access_type,area
2 1,edson,12345,Edson Castaneda,admin,all
3 2,carlos,54321,Carlos Casillas,admin,all
4 3,luis,holamundo,Luis Lopez,seller,procesadores
```

Figura 1.1: Archivo CSV que incluye usuarios con acceso permitido.

Entre las consultas que el usuario puede realizar con este programa, se encuentran:

- Consultas por producto
- Consultas por valoración
- Consultas por fechas
- Consultas por categoría de producto

El repositorio del proyecto se encuentra en el siguiente enlace:
https://github.com/EdsonECM17/DS_Proyecto_01_LifeStore.git

2. DEFINICIÓN DEL CÓDIGO

2.1. DESCRIPCIÓN DEL ALGORITMO

De manera general, el algoritmo implementado permite a usuarios definidos realizar diferentes tipos de consultas con los datos que tiene registrados la empresa. El flujo de este proceso puede observarse en la Fig. 2.1. Para cada consulta que se desee realizar hay un resultado en consola que se muestra al usuario. Dentro del funcionamiento del programa, hay dos funciones importantes: el inicio de sesión y el uso de consulta para obtener información.

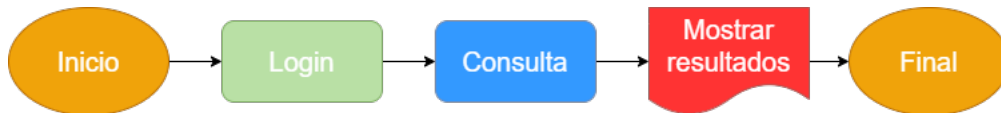


Figura 2.1: Diagrama de flujo de función principal.

El proceso principal del algoritmo son las consultas. Las consultas se apoyan de diferentes servicios, como se muestra en el diagrama de la Fig. 2.2. Mediante servicios se adquiere información procesada de los registros de la empresa y lo muestra resúmenes al usuario. Para realizar una consulta, primero se selecciona un tipo de consulta. Dependiendo del tipo de consulta, se llama a un servicio específico. Los servicios toman datos de entrada que se usaran para filtrar información. A partir de las tablas de registros de la empresa, se filtra la información con parámetros indicados y la tabla resultante es procesada para generar una respuesta. Esa respuesta es la salida de la función servicio. Posteriormente, la respuesta se procesa nuevamente para imprimirla de manera entendible para el usuario.

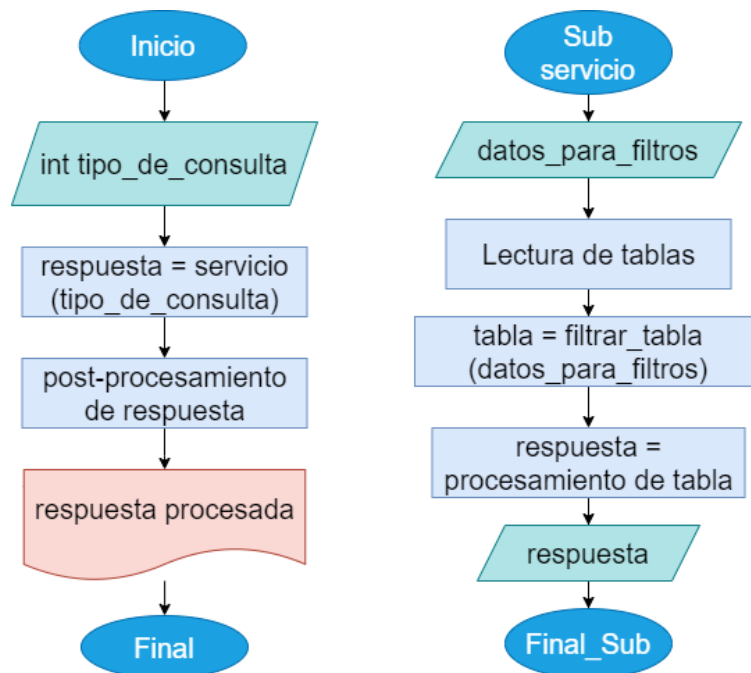


Figura 2.2: Diagrama de flujo de consultas.

Antes de poder acceder a las consultas, es necesario validar el acceso del usuario. Para el inicio de sesión, se tienen tres intentos de introducir un nombre de usuario y contraseña que se encuentren dentro de una lista válida para continuar al siguiente paso del algoritmo. Inicialmente, el acceso no es permitido. En la validación se usa un ciclo while mientras se tengan intentos y el acceso aun no sea permitido. Dentro de dicho ciclo, se introducen usuario y contraseña. Si la validación de los datos es correcta se permite el acceso a la siguiente etapa del código, o en caso contrario, se reduce un intento y se pasa a la siguiente iteración. Cuando terminen los intentos o el acceso sea permitido, termina el ciclo. Finalmente, se verifica el valor de la variable de acceso. En caso de ser verdadera se muestra el resto del programa, mientras que en caso de ser falsa aún se indica un mensaje de acceso negado. Este proceso se muestra en la Fig. 2.3.

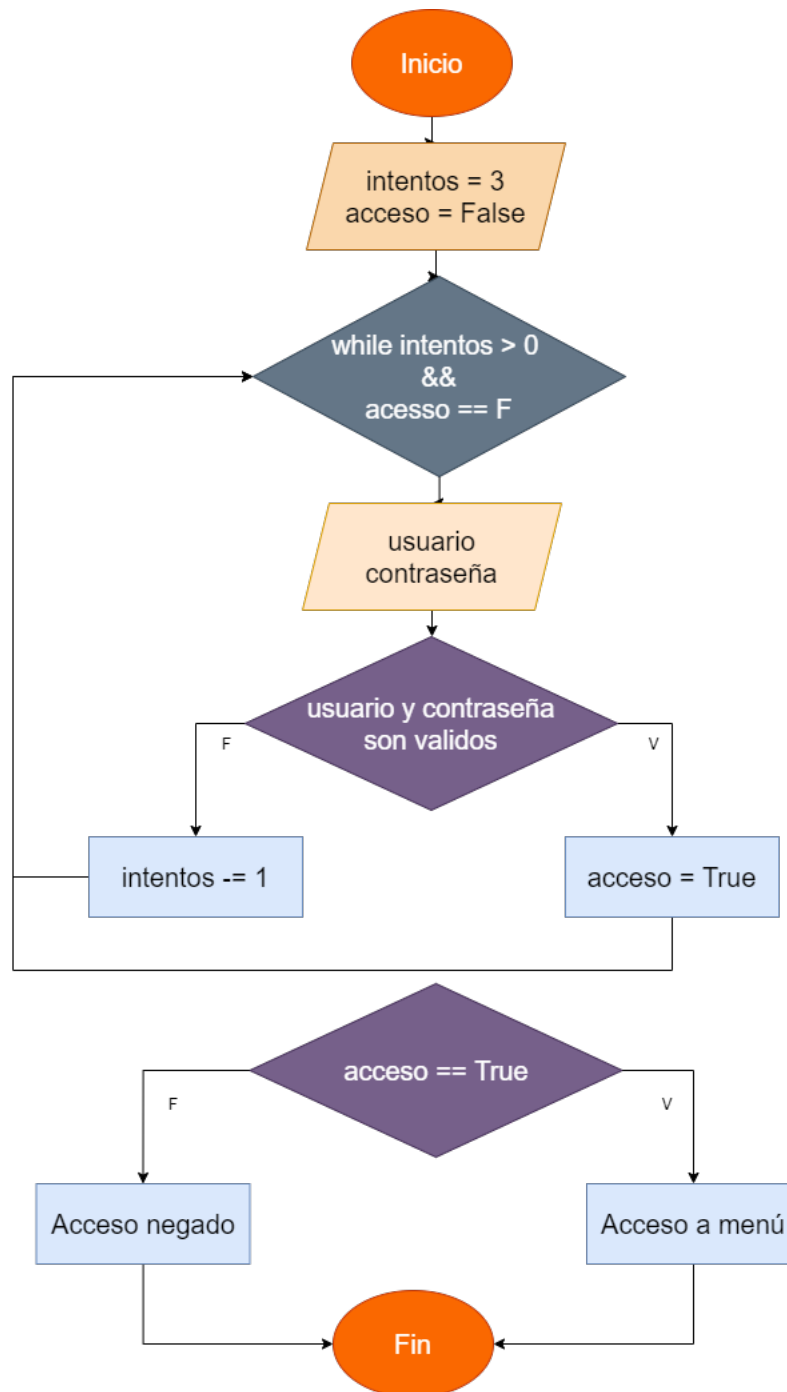


Figura 2.3: Diagrama de flujo de Inicio de Sesión.

2.2. IMPLEMENTACIÓN EN PYTHON

2.2.1. Login

Para el inicio de sesión, se definió un módulo auxiliar llamado *user_access.py*. Este módulo cuenta con una función que se encarga de validar el acceso comparando el dato con *users.csv*. Antes de definir la función, es necesario importar las librerías necesarias. En este caso se usa pandas para la manipulación de datos del archivo CSV. Por otro lado, se define como constante la ruta del archivo con los usuarios para que la función pueda encontrarlo y leer los datos.

```
1 import pandas as pd
2
3 USER_FILE_PATH = "data/users.csv"
```

Código 2.1: Librerías y constantes en módulo *user_access.py*

Una vez que se definieron las constantes, se creó la función de inicio de sesión. Esta función solo tiene un parámetro opcional, donde se puede variar el número de intentos. Por defecto, se tienen los tres intentos antes mencionados para el inicio de sesión. Mediante funciones input, un usuario introduce su nombre de usuario y contraseña. Si los datos concuerdan con uno de los usuarios registrados, la función retorna un True. En caso de error en todos los intentos, la función retorna un False.

```
1 def login(login_attempts:int = 3):
2     """
3     Args:
4         login_attempts (int, optional): Intentos de conexión que tiene
5         permitidos el usuario. Defaults to 3.
6     """
7     # Inicializar variables
8     # If true, accede a la información de LifeStore. False al empezar.
9     successful_login = False
10    # Leer tabla donde se encuentran los usuarios permitidos
11    user_table = pd.read_csv(USER_FILE_PATH)
12    # Iniciar intento de login
13    while successful_login is False and login_attempts > 0:
14        # Caso: todos los intentos de registro son fallidos.
15        # Terminar la función con un False.
16        if login_attempts == 0:
17            print("Demasiados intentos.\nEl acceso se ha deshabilitado "+
18                  "para este equipo.\nPara poder acceder nuevamente, "+
19                  "acercarse con el equipo de TI de Life Store.")
20            return False
21        # Descontar un intento.
22        login_attempts-=1
23        # Ingresar datos de entrada
24        user = input("Ingrese su usuario: ")
25        password = input("Ingrese su contraseña: ")
26        # Caso Usuario-Contraseña Validos
27        if user in user_table['user_name'].values:
28            # Conseguir resto de los datos para ese usuario especifico
29            user_data = user_table.loc[user_table[
30                                     'user_name'].str.contains(user)]
31            # Si la contraseña concuerda con el usuario, concede acceso
32            if password in user_data['password'].values:
33                successful_login = True
34                print("Bienvenido "+user_data['name'][0]+".\n")
35                return True
36
```

```

37     # Caso: usuario o contraseña son inválidos
38     if successful_login is False and login_attempts>0:
39         print("\nError con el usuario o contraseña proporcionados." +
40             "Revise e intente nuevamente.")

```

Código 2.2: Función de inicio de sesión en user_access.py

2.2.2. Manejo inicial de datos

Originalmente, los datos de entrada se encontraban en tres listas de listas. Para procesar los datos proporcionados de LifeStore con una mayor facilidad, se decide volver las diferentes listas de listas una tabla de pandas. Dichas tablas permiten el uso de filtros para obtener subtablas con datos de interés, por lo que se optó por aprovechar esta herramienta para las funciones relacionadas a consultas. Esta transformación a dataframes se hace en el módulo *lifestore_tables.py*. Primero se importa el módulo con las listas y la librería pandas. Posteriormente, se crea una tabla usando las listas y los nombres de columnas proporcionados en la documentación de *lifestore_file.py*. En el caso particular de los registros de ventas, casi todos los registros pertenecían al año 2020, por lo que se descartaron los años fuera de este caso.

```

1  import pandas as pd
2  import data.lifestore_file as lifestore
3
4  # Create dataframe of searches
5  lifestore_searches = pd.DataFrame(lifestore.lifestore_searches,
6      columns = ["id_search", "id_product"])
7
8  # Create dataframe of sales
9  lifestore_sales = pd.DataFrame(lifestore.lifestore_sales,
10     columns = ["id_sale", "id_product", "score", "date", "refund"])
11  lifestore_sales["date"] = pd.to_datetime(lifestore_sales["date"])
12  # Clean data to include only data from year (2020)
13  lifestore_sales = lifestore_sales[lifestore_sales["date"]>="1/1/2020"]
14
15  # Create dataframe of products
16  lifestore_products = pd.DataFrame(lifestore.lifestore_products,
17     columns = ["id_product", "name", "price", "category", "stock"])

```

Código 2.3: Código lifestore_tables.py

2.2.3. Procesamiento de datos

En general, todos los servicios de consulta internamente van a requerir filtrar las tablas inicialmente. Debido a esto, el proceso inicial requiere poder filtrar cada tabla con cualquiera de los parámetros que la componen. Los diferentes servicios van a usar las tablas filtradas para obtener datos que sea de interés del usuario, ya sea por año, mes, producto o categoría de producto.

Filtros

El filtrado es realizado por el módulo *filter_df.py*. Antes de poder implementar los filtros, es necesario importar las tablas generadas anteriormente y la clase DataFrame. Esta última permitirá validar la salida de los filtros.

```

1  from pandas.core.frame import DataFrame
2  from processing.lifestore_tables import lifestore_products
3  from processing.lifestore_tables import lifestore_sales
4  from processing.lifestore_tables import lifestore_searches

```

Código 2.4: Librerías necesarias en filters_df.py

Dentro de este módulo, se crea una clase llamada Filters. Esta clase será la encargada de filtrar los dataframe generados a partir de datos de entrada. La clase cuenta con tres métodos un correspondiente a cada tabla. Los argumentos de cada método son opcionales, y refieren a cada columna de la respectiva tabla. Si alguno de los argumentos se proporciona, la tabla se filtra para ese dato. Se va evaluando parámetro por parámetro. Después de filtrar, la función devuelve una subtabla. En caso de que no hubiera ningún parámetro para filtrar, se devuelve la tabla completa.

```

1 class Filters():
2     def __filter_products_df(self, id_product: int or None = None,
3         category: str or None = None, name: str or None = None,
4         price_min: int or None = None, price_max: int or None = None,
5         stock_min: int or None = None, stock_max: int or None = None
6     ) -> DataFrame:
7         """
8         Filtra el dataframe de productos de acuerdo con valores en las
9         columnas que tiene la tabla generada. Si no hay filtro, se
10        regresa un dataframe completo.
11
12        Args:
13            id_product (int or None, optional): Id de producto.
14            Defaults to None.
15            category (str or None, optional): Categoría de producto.
16            Defaults to None.
17            name (str or None, optional): Nombre del producto.
18            Defaults to None.
19            price_min (int or None, optional): Precio mínimo de producto.
20            Defaults to None.
21            price_max (int or None, optional): Precio máximo de producto.
22            Defaults to None.
23            stock_min (int or None, optional): Inventario mínimo de
24            producto. Defaults to None.
25            stock_max (int or None, optional): Inventario máximo de
26            producto. Defaults to None.
27        Returns:
28            DataFrame: Dataframe con columnas de la tabla que cumplen con
29            los filtros indicados.
30        """
31        product_df = lifestore_products
32        # Incluir filtro por id de producto si existe
33        if id_product is not None:
34            product_df=product_df[product_df["id_product"] == id_product]
35        # Incluir filtro por categoria de producto si existe
36        if category is not None:
37            product_df=product_df[product_df["category"] == category]
38        # Incluir filtro por nombre de producto si existe
39        if name is not None:
40            product_df=product_df[product_df["name"] == name]
41        # Incluir filtro por precio si existe
42        if price_min is not None:
43            product_df=product_df[product_df["price"] >= price_min]
44        if price_max is not None:
45            product_df=product_df[product_df["price"] <= price_max]
46        # Incluir filtro por stock si existe
47        if price_min is not None:
48            product_df=product_df[product_df["stock"] >= stock_min]
49        if price_max is not None:
50            product_df=product_df[product_df["stock"] <= stock_max]
51        return product_df

```



```

52     def __filter_sales_df(self, id_sale: int or None = None,
53                           id_product: int or None = None, score_min: int or None = None,
54                           score_max: int or None = None, start_date: str or None = None,
55                           end_date: str or None = None, refund: bool or None = None
56                           ) -> DataFrame:
57         """
58         Filtra el dataframe de ventas de acuerdo con valores en las
59         columnas que tiene la tabla generada. Si no hay filtro, se
60         regresa un dataframe completo.
61         Args:
62             id_sale (int or None, optional): Id de venta.
63             Defaults to None.
64             id_product (int or None, optional): Id de producto.
65             Defaults to None.
66             score_min (int or None, optional): Calificación mínima de
67             venta. Defaults to None.
68             score_max (int or None, optional): Calificación máxima de
69             venta. Defaults to None.
70             start_date (str or None, optional): Fecha de inicio de
71             periodo de ventas considerado. Defaults to None.
72             end_date (str or None, optional): Fecha de fin de
73             periodo de ventas considerado. Defaults to None.
74             refund (bool or None, optional): Ventas devueltas (True)
75             o no devueltas (False). Defaults to None.
76         Returns:
77             DataFrame: Dataframe con columnas de la tabla que cumplen
78             con los filtros indicados.
79         """
80         sales_df = lifestore_sales
81         # Incluir filtro por id de venta si existe
82         if id_sale is not None:
83             sales_df=sales_df[sales_df["id_sale"] == id_sale]
84         # Incluir filtro por id de producto si existe
85         if id_product is not None:
86             sales_df=sales_df[sales_df["id_product"] == id_product]
87         # Incluir filtro por calificación si existe
88         if score_min is not None:
89             sales_df=sales_df[sales_df["score"] >= score_min]
90         if score_max is not None:
91             sales_df=sales_df[sales_df["score"] <= score_max]
92         # Incluir filtro por fechas si existe
93         if start_date is not None:
94             sales_df=sales_df[sales_df["date"] >= start_date]
95         if end_date is not None:
96             sales_df=sales_df[sales_df["date"] <= end_date]
97         # Incluir filtro por devoluciones si existe
98         if refund is not None:
99             sales_df=sales_df[sales_df["refund"] == int(refund)]
100         return sales_df
101
102
103     def __filter_searches_df(self, id_search: int or None = None,
104                             id_product: int or None = None) -> DataFrame:
105         """
106         Filtra el dataframe de búsquedas de acuerdo con valores en las
107         columnas que tiene la tabla generada. Si no hay filtro, se
108         regresa un dataframe completo.
109

```

```

110     Args:
111         id_search (int or None, optional): Id de búsqueda.
112         Defaults to None.
113         id_product (int or None, optional): Id de producto.
114         Defaults to None.
115
116     Returns:
117         DataFrame: Dataframe con columnas de la tabla que cumplen
118         con los filtros indicados.
119     """
120     searches_df = lifestore_searches
121     # Incluir filtro por id de búsqueda si existe
122     if id_search is not None:
123         searches_df=searches_df[searches_df["id_search"] == id_search]
124     # Incluir filtro por id de producto si existe
125     if id_product is not None:
126         searches_df=searches_df[searches_df["id_product"]==id_product]
127
128     return searches_df

```

Código 2.5: Clase Filters.

Servicios

Las consultas son la parte principal del código implementado. Estas consultas se apoyan de la clase servicios en *lifestore_services.py*. La clase servicios se va a apoyar de las tablas antes creadas y de la clase filtros, de la cual va a heredar los métodos definidos anteriormente. Así mismo para el manejo de fechas se apoya del módulo calendar. Antes de definir la clase se importan dichas librerías.

```

1 import calendar
2
3 from processing.lifestore_tables import lifestore_products
4 from processing.lifestore_tables import lifestore_sales
5 from processing.lifestore_tables import lifestore_searches
6
7 from processing.filters_df import Filters

```

Código 2.6: Librerías utilizadas en lifestore_services.py

Se crea una clase *Services*, hija de la clase *Filters*. Esta clase agrupa diversas operaciones de consulta de datos de LifeStore, para apoyar en análisis de información. Dentro de los métodos de esta nueva clase se encuentran diferentes tipos de funciones: servicios generales, servicios por tiempo, servicios por producto, servicios por categoría.

Servicios generales:

- Conteo de ventas
- Calcular ingresos
- Conteo de búsquedas

Servicios por tiempo:

- Ventas anuales
- Ingresos anuales
- Ventas por mes
- Ingresos por mes

Servicios por producto:

- Ventas por producto
- Búsquedas por producto
- Nombre de producto
- Inventario de producto
- Valoración de producto

Servicios por categoría:

- Productos por categoría
- Ventas por categoría
- Ingresos por categoría

```

1 class Service(Filters):
2     # Métodos generales
3     def count_sales(self, id_product: int or None = None,
4                     start_date: str or None = None, end_date: str or None = None,
5                     score_min: int or None = None, score_max: int or None = None,
6                     refund_status: bool or None = None) -> int:
7         """
8         Consulta la tabla de ventas con una serie de filtros de producto
9         y/o tiempo, extrayendo solo las filas de interés. Mediante un
10        conteo de las ventas que cumplen con las condiciones
11        del
12        caso, se determina el número de ventas.
13
14        Args:
15            id_product (int or None, optional): Id de producto.
16            Defaults to None.
17            start_date (str or None, optional): Fecha de inicio de periodo
18            de ventas considerado. Defaults to None.
19            end_date (str or None, optional): Fecha de fin de periodo
20            de ventas considerado. Defaults to None.
21            score_min (int or None, optional): Calificación mínima de
22            venta. Defaults to None.
23            score_max (int or None, optional): Calificación máxima de
24            venta. Defaults to None.
25            refund_status (bool or None, optional): Ventas devueltas
26            (True) o no devueltas (False). Defaults to None.
27
28        Returns:
29            int: Número de ventas que tuvo el caso solicitado.
30        """
31        # Obtener subtabla de tabla de ventas
32        # Con filas que cumplan con los filtros indicados
33        sales_df = self._Filters__filter_sales_df(
34            id_product = id_product, start_date = start_date,
35            end_date = end_date, refund = refund_status,
36            score_min = score_min, score_max = score_max)
37        # Contar ventas mediante el número de filas de la tabla filtrada
38        sales_number = len(sales_df)
39        return sales_number
40

```

```

41     def calculate_income(self, id_product: int or None = None,
42                           start_date: str or None = None, end_date: str or None = None,
43                           score_min: int or None = None, score_max: int or None = None,
44                           refund_status: bool or None = None) -> int:
45         """
46         Obtiene el total de ingresos para un producto específico o todos
47         los productos que cumplan con los filtros en las entradas.
48
49         Args:
50             id_product (int or None, optional): Id de producto.
51             Defaults to None.
52             start_date (str or None, optional): Fecha de inicio de
53             periodo de ventas considerado. Defaults to None.
54             end_date (str or None, optional): Fecha de fin de
55             periodo de ventas considerado. Defaults to None.
56             score_min (int or None, optional): Calificación mínima de
57             venta. Defaults to None.
58             score_max (int or None, optional): Calificación máxima de
59             venta. Defaults to None.
60             refund_status (bool or None, optional): Ventas devueltas
61             (True) o no devueltas (False). Defaults to None.
62
63         Returns:
64             int: Total de ingresos para caso solicitado.
65         """
66         income = 0
67         # Si se incluyó un id_product en la solicitud,
68         # filtra la tabla generada ahora por producto
69         if id_product is not None:
70             product_sales = self.count_sales(
71                 id_product, start_date, end_date, score_min,
72                 score_max, refund_status)
73             income = product_sales * lifestore_products["price"][
74                 lifestore_products["id_product"] == id_product].item()
75         # Si no se indicó id_product, se itera cada id
76         # Se suman los ingresos de cada producto diferente
77         else:
78             for row in lifestore_products.iterrows():
79                 # Se obtiene id
80                 id_product = row[1]['id_product']
81                 # Se cuentan ventas del producto
82                 product_sales = self.count_sales(id_product,
83                     start_date, end_date, score_min,
84                     score_max, refund_status)
85                 # Se suman los ingresos de producto a los ingresos totales
86                 income += product_sales * lifestore_products["price"][
87                     lifestore_products["id_product"] == id_product].item()
88         return income
89
90     def count_searches(self, id_product: int or None):
91         """
92         Consulta la tabla de búsquedas con un filtro de producto y cuenta
93         la cantidad de búsquedas que cumplan con las condiciones indicadas
94
95         Args:
96             id_product (int or None, optional): Id de producto.
97             Defaults to None.
98

```

```

99     Returns:
100         int: número de búsquedas que tuvo el caso solicitado.
101     """
102     # Obtener tabla de búsquedas filtrada.
103     # Si no hay filtro, se obtiene completa
104     searches_df = self._Filters__filter_searches_df(
105         id_product = id_product)
106     # Contar búsquedas
107     searches_number = len(searches_df)
108     return searches_number
109
110 # Métodos relacionados a tiempo
111 def get_year_sales(self, year:int, id_product: int or None = None,
112     refund_status: bool or None = None) -> int:
113     """
114     Obtiene el número de ventas anuales. Cuenta con algunos filtros
115     opcionales que permiten considerar únicamente un producto o
116     descartar las ventas que terminaron en devolución.
117
118     Args:
119         year (int): Año seleccionado.
120         id_product (int or None, optional): Id de producto.
121         Defaults to None.
122         refund_status (bool or None, optional): Filtro de
123         devoluciones. Defaults to None.
124
125     Returns:
126         int: Número de ventas anuales.
127     """
128     # Definir fecha de inicio del año y fecha de fin
129     year_start= f"{year}-01-01"
130     year_end = f"{year}-12-31"
131     # Usar función get ventas para obtener ventas anuales
132     sales_number = self.count_sales(
133         start_date=year_start, end_date=year_end,
134         id_product=id_product, refund_status=refund_status)
135     return sales_number
136
137 def get_year_income(self, year:int, id_product: int or None = None,
138     refund_status: bool or None = None) -> int:
139     """
140     Obtiene los ingresos anuales. Cuenta con algunos filtros
141     opcionales que permiten considerar únicamente un producto o
142     descartar las ventas que terminaron en devolución.
143
144     Args:
145         year (int): Año seleccionado.
146         id_product (int or None, optional): Id de producto.
147         Defaults to None.
148         refund_status (bool or None, optional): Filtro de
149         devoluciones. Defaults to None.
150
151     Returns:
152         int: Ingresos anuales.
153     """
154     # Definir fecha de inicio del año y fecha de fin de año
155     year_start= f"{year}-01-01"
156     year_end = f"{year}-12-31"

```

```

157     # Usar función get ventas para obtener ingresos anuales
158     income = self.calculate_income(
159         start_date=year_start, end_date=year_end,
160         id_product=id_product, refund_status=refund_status)
161     return income
162
163
164     def get_monthly_sales(self, year: int, id_product: int or None = None,
165     refund_status: bool or None = None) -> dict:
166         """
167         Obtiene el número de ventas de cada mes. Cuenta con algunos
168         filtros opcionales que permiten considerar únicamente un
169         producto o descartar las ventas que terminaron en devolución.
170         Las ventas de cada mes se organizan en un diccionario.
171
172         Args:
173             year (int): Año seleccionado.
174             id_product (int or None, optional): Id de producto.
175             Defaults to None.
176             refund_status (bool or None, optional): Filtro de
177             devoluciones. Defaults to None.
178
179         Returns:
180             dict: Número de ventas de cada mes.
181         """
182         # Inicializa variables
183         sales_dict = {1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0,
184             11:0, 12:0} # month: sales_number_of_month
185         # Ciclo for para obtener el número de ventas de cada mes
186         for month in range(1,13):
187             # Obtener cantidad de días del mes para saber ultimo día
188             month_days = calendar.monthrange(year, month)[1]
189             # Definir fecha de inicio del año y fecha de fin de año
190             month_start = f"{year}-{month:02d}-01"
191             month_end = f"{year}-{month:02d}-{month_days:02d}"
192             # Obtener ventas de ese mes
193             sales_dict[month] = self.count_sales(
194                 start_date=month_start, end_date=month_end,
195                 id_product=id_product, refund_status=refund_status)
196         return sales_dict
197
198     def get_monthly_income(self, year: int, id_product: int or None = None,
199     refund_status: bool or None = None) -> dict:
200         """
201         Obtiene los ingresos mensuales. Cuenta con algunos
202         filtros opcionales que permiten considerar únicamente un
203         producto o descartar las ventas que terminaron en devolución.
204         Las ventas de cada mes se organizan en un diccionario.
205
206         Args:
207             year (int): Año seleccionado.
208             id_product (int or None, optional): Id de producto.
209             Defaults to None.
210             refund_status (bool or None, optional): Filtro de
211             devoluciones. Defaults to None.
212
213         Returns:
214             dict: Ingresos de cada mes.
215         """

```

```

215     # Inicializa variables
216     income_dict = {1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0,
217                   11:0, 12:0} # month: income_of_month
218     # Ciclo for para obtener los ingresos de cada mes
219     for month in range(1,13):
220         # Obtener cantidad de días del mes para saber ultimo día
221         month_days = calendar.monthrange(year, month)[1]
222         # Definir fecha de inicio del año y fecha de fin de año
223         month_start = f"{year}-{month:02d}-01"
224         month_end = f"{year}-{month:02d}-{month_days:02d}"
225         # Obtener ventas de ese mes
226         income_dict[month] = self.calculate_income(
227             start_date=month_start, end_date=month_end,
228             id_product=id_product, refund_status=refund_status)
229     return income_dict
230
231 def get_products_sales(self, refund_status: bool or None = None,
232 start_date: str or None = None, end_date: str or None = None
233 ) -> dict:
234     """
235     Obtiene el número de ventas de cada producto de la tienda.
236     Las ventas pueden filtrarse por fechas.
237     Las ventas que terminaron en devolución pueden considerarse
238     u omitirse.
239
240     Args:
241         refund_status (bool or None, optional): Ventas devueltas
242         (True) o no devueltas (False). Defaults to None.
243         start_date (str or None, optional): Fecha de inicio de periodo
244         de ventas considerado. Defaults to None.
245         end_date (str or None, optional): Fecha de fin de periodo
246         de ventas considerado. Defaults to None.
247
248     Returns:
249         dict: Ventas por producto. El key de cada elemento es el id
250         de producto, mientras que el valor corresponde al
251         número de ventas.
252     """
253     # Inicializar variables
254     products_sales = {}
255     # Ciclo for para revisar cada producto de la tabla productos
256     for row in lifestore_products.iterrows():
257         # Se obtiene id
258         id_product = row[1]['id_product']
259         # Se cuentan ventas del producto
260         sales_number = self.count_sales(
261             id_product=id_product, refund_status=refund_status,
262             start_date=start_date, end_date=end_date)
263         # Se almacena resultado en diccionario
264         products_sales[id_product] = sales_number
265     return products_sales
266
267 def get_products_searches(self) -> dict:
268     """
269     Obtiene el número de búsquedas de cada producto de la tienda.
270     Las búsquedas pueden filtrarse por fechas.
271     Las búsquedas que terminaron en devolución pueden considerarse
272     u omitirse.

```

```

273     Returns:
274         dict: Búsquedas por producto. El key de cada elemento es el id
275         de producto, mientras que el valor corresponde al
276         número de búsquedas.
277     """
278     # Inicializar variables
279     products_searches = {}
280     # Ciclo for para revisar cada producto de la tabla productos
281     for row in lifestore_products.iterrows():
282         # Se obtiene id
283         id_product = row[1]['id_product']
284         # Se cuentan búsquedas del producto
285         searches = self.count_searches(id_product=id_product)
286         # Se almacena resultado en diccionario
287         products_searches[id_product] = searches
288     return products_searches
289
290 def get_product_name(self, id_product: int) -> str:
291     """ Obtiene el nombre del producto de la tabla lifestore_products.
292
293     Args:
294         id_product (int): ID del producto.
295
296     Returns:
297         str: Nombre del producto.
298     """
299     name = lifestore_products.loc[lifestore_products[
300         'id_product'] == id_product, 'name'].item()
301     return name
302
303 def get_product_stock(self, id_product: int) -> int:
304     """
305     Obtiene la cantidad de unidades de un producto en inventario,
306     a partir de la tabla lifestore_products.
307
308     Args:
309         id_product (int): ID del producto.
310
311     Returns:
312         int: Unidades del producto en inventario.
313     """
314     stock = lifestore_products.loc[lifestore_products[
315         'id_product'] == id_product, 'stock'].item()
316     return stock
317
318 def get_product_grades(self, reviews_weight: float = 0.6,
319     refunds_weight: float = 0.4, start_date: str or None = None,
320     end_date: str or None = None) -> dict:
321     """
322     Calcula la valoración que tiene los productos vendidos
323     considerando las calificaciones de los clientes por cada venta y
324     la cantidad de devoluciones que tiene el producto.
325     Para usar ambos factores se asigna cierto peso a cada factor.
326     Estos factores cumplen con la siguiente expresión:
327     factor_calificaciones + factor_devoluciones = 1
328
329     Args:
330         reviews_weight (float, optional): Factor de calificaciones.
331         Defaults to 0.6.

```



```

331         refunds_weight (float, optional): Factor de devoluciones.
332         Defaults to 0.4.
333         start_date (str or None, optional): Fecha de inicio de periodo
334         de ventas considerado. Defaults to None.
335         end_date (str or None, optional): Fecha de fin de periodo
336         de ventas considerado. Defaults to None.
337
338     Returns:
339         dict: Valoración de cada producto.
340     """
341     product_grades = {}
342     # Ciclo for para revisar cada producto de la tabla productos
343     for row in lifestore_products.iterrows():
344         # Se obtiene id
345         id_product = row[1]['id_product']
346         # Para ese id, se obtiene tabla de ventas de producto
347         sales_df = self._Filters__filter_sales_df(
348             id_product = id_product, start_date = start_date,
349             end_date = end_date)
350         # Determinar total de ventas
351         total_sales = len(sales_df)
352         # Si las ventas son mayores a cero, revisa calificaciones
353         # del producto, sino calificación N.D.
354         if total_sales > 0:
355             # Caso hay ventas
356             # Obtener puntaje promedio de revisiones de clientes
357             reviews_mean = sales_df["score"].mean()
358             # Se normaliza puntaje de revisiones, entre 0 y 1
359             reviews_normalized = (reviews_mean-1)/(5-1)
360             # Contar devoluciones
361             total_refunds = len(sales_df[sales_df["refund"] > 0])
362             # Obtener relación productos no devueltos y ventas
363             refunds_pct = 1 - total_refunds/total_sales
364             # Se calcula calificación dándole pesos a
365             # las revisiones y la cantidad de productos devueltos
366             product_grades[id_product] = round((
367                 reviews_weight*reviews_normalized +
368                 refunds_weight*refunds_pct)*100, 2)
369         else:
370             # Caso no hubo ventas
371             product_grades[id_product] = 'N.D.'
372
373     # Obtener subtabla de tabla de ventas con filas que cumplan
374     # con los filtros indicados
375     return product_grades
376
377 def count_category_products(self) -> dict:
378     """
379     Cuenta productos existentes en cada categoría.
380
381     Returns:
382         dict: Diccionario con conteo de productos de cada categoría.
383         (category: amount_of_products)
384     """
385     # Inicializar variables
386     products_per_category = {}
387     # Se identifican las diferentes categorías de la tabla de productos
388     categories = lifestore_products.category.unique().tolist()

```

```

389     for category in categories:
390         # Filtra tabla de productos por la categoría
391         product_df = self._Filters__filter_products_df(
392             category = category)
393         # Cuenta elementos en tabla de productos de categoría
394         products_per_category[category] = len(product_df)
395     return products_per_category
396
397 def get_category_sales(self, id_product: int or None = None,
398     refund_status: bool or None = None, start_date: str or None = None,
399     end_date: str or None = None) -> dict:
400     """
401     Clasifica en las diferentes categorías de producto las
402     ventas de la empresa.
403
404     Args:
405         id_product (int or None, optional): Id de producto.
406         Defaults to None.
407         refund_status (bool or None, optional): Ventas devueltas
408         (True) o no devueltas (False). Defaults to None.
409         start_date (str or None, optional): Fecha de inicio de periodo
410         de ventas considerado. Defaults to None.
411         end_date (str or None, optional): Fecha de fin de periodo
412         de ventas considerado. Defaults to None.
413
414     Returns:
415         dict: Total de ventas por categoría de producto.
416     """
417     # Inicializar variables
418     sales_per_category = {}
419     # Se obtienen las ventas de cada producto
420     product_sales = self.get_products_sales(
421         refund_status=refund_status, start_date=start_date,
422         end_date=end_date)
423     # Se identifican las diferentes categorías de la tabla de producto
424     categories = lifestore_products.category.unique().tolist()
425     for category in categories:
426         sales_number = 0
427         # Filtra tabla de productos por la categoría
428         product_df = self._Filters__filter_products_df(
429             category = category)
430         # Se suman las ventas de cada producto en la tabla filtrada
431         for row in product_df.iterrows():
432             # Se obtiene id
433             id_product = row[1]['id_product']
434             # Si suman las ventas del producto a ventas de categoría
435             sales_number += product_sales[id_product]
436         # Cuenta elementos en tabla de productos de categoría
437         sales_per_category[category] = sales_number
438     return sales_per_category
439
440 def get_category_income(self, id_product: int or None = None,
441     refund_status: bool or None = None, start_date: str or None = None,
442     end_date: str or None = None) -> dict:
443     """
444     Clasifica en las diferentes categorías de producto los
445     ingresos de la empresa.
446

```

```

447     Args:
448         id_product (int or None, optional): Id de producto.
449         Defaults to None.
450         refund_status (bool or None, optional): Ventas devueltas
451         (True) o no devueltas (False). Defaults to None.
452         start_date (str or None, optional): Fecha de inicio de periodo
453         de ventas considerado. Defaults to None.
454         end_date (str or None, optional): Fecha de fin de periodo
455         de ventas considerado. Defaults to None.
456
457     Returns:
458         dict: Total de ingresos por categoría de producto.
459     """
460     income_per_category = {}
461     # Se identifican las diferentes categorías de la tabla de producto
462     categories = lifestore_products.category.unique().tolist()
463     for category in categories:
464         income = 0
465         # Filtra tabla de productos por la categoría
466         product_df = self._Filters__filter_products_df(
467             category = category)
468         # Se suman las ventas de cada producto en la tabla filtrada
469         for row in product_df.iterrows():
470             # Se obtiene id
471             id_product = row[1]['id_product']
472             # Se suman los ingresos del producto a las ventas
473             income += self.calculate_income(
474                 id_product=id_product, refund_status=refund_status,
475                 start_date=start_date, end_date=end_date)
476         # Cuenta elementos en tabla de productos de categoría
477         income_per_category[category] = income
478     return income_per_category

```

Código 2.7: Código lifestore_services.py

2.2.4. Librerías auxiliares

Una vez que se implementaron las subfunciones principales, se desarrollaron otros módulos que también apoyarían a la función principal. Estos otros módulos se relacionan al menú que se le presenta al usuario y a gráficas generadas, respectivamente.

Funciones de menú

Se desarrollaron funciones auxiliares para el menú que se muestra al usuario después de un inicio de sesión correcto en la función principal. En este módulo se definen dos funciones. La primera despliega un menú y obtiene la opción seleccionada dentro de las opciones de dicho menú. La segunda función permite obtener respuesta de preguntas con dos posibles respuestas.

```

1 def select_menu(menu_options: dict) -> int:
2     """ Función auxiliar para desplegar menú en consola.
3         Regresa unicamente la opción seleccionada.
4
5     Args:
6         menu_options (dict): Diccionario que tienen enteros como llaves
7                               y las diferentes opciones como valores.
8
9     Returns:
10         int: caso seleccionado del menú.
11     """

```

```

12     while True:
13         print("\n¿Qué desea consultar?:")
14         for key in menu_options.keys():
15             print(f"{key}. - {menu_options[key]}")
16         selected_option = input("Ingrese el número de la acción deseada:")
17         # Validar que el dato sea entero o volver a poner el menú
18         try:
19             selected_option = int(selected_option)
20         except:
21             # Si el dato ingresado no es un entero,
22             # Se indica error vuelve a mostrar menú.
23             print("El valor ingresado no es un número. Ingrese el dato "+
24                   "en el formato correcto.\n")
25             continue
26         # Validar que el dato se encuentre dentro de las opciones
27         if selected_option not in menu_options.keys():
28             # Si no existe la opción, se presenta mensaje de error
29             # Se vuelve a mostrar menú.
30             print("El número ingresado no se encuentra en el menú."+
31                   "Seleccione otra opción.\n")
32             continue
33         else:
34             # Si el numero está dentro del menú, se sale del while
35             break
36     return selected_option
37
38
39 def validate_question(question: str) -> bool:
40     """ Función para validar una pregunta de si o no.
41
42     Args:
43         question (str): Pregunta que se quiere validar
44
45     Returns:
46         bool: Indica la respuesta seleccionada para la pregunta
47     """
48     # Añadir opciones (Sí o No) al string de pregunta
49     question = question.replace("?", " (Y o N)?")
50     while True:
51         selected_option = input(question).upper()
52         # Validar si se seleccionó una de las dos opciones
53         if selected_option not in ["Y", "N"]:
54             print("El dato no fue ingresado correctamente, favor de "+
55                   "seleccionar una de las opciones especificadas.\n")
56             continue
57         else:
58             # Convertir a selección a boolean
59             if selected_option == 'Y':
60                 selected_option = True
61             elif selected_option == 'N':
62                 selected_option = False
63             break
64
65     return selected_option

```

Código 2.8: Código menu_utils.py

Funciones relacionadas a gráficas

En algunos casos, las consultas pueden beneficiarse de una presentación más visual que solo mostrar texto. Para esos casos se implementó un módulo que permita generar gráficas de la información procesada. Para generar las gráficas se usa Plotly. Antes de las funciones específicas, se hacen configuraciones requeridas por la librería para poder guardar los objetos que crea como una imagen estática, y se crea un método para dicha función. Para gráfica resultados de las consultas, se usa la clase *Summary_Chart* que herede el método anterior. Se configura un formato de fondo para las gráficas y una ruta donde guardarla. Los datos que recibe esta función son en forma de diccionario, y lo devuelve como gráfica de barras en la función *bar_summary*. El resultado de la función son dos archivos: un archivo *html* con un gráfico interactivo y un archivo *png* con la gráfica como imagen.

```

1 import os
2
3 import plotly.graph_objects as go
4
5 # Comando necesario para usar librerías de imágenes fijas
6 os.environ["PATH"] = os.environ["PATH"] +
7     f";{os.path.abspath('venv/lib/site-packages/kaleido/executable/')}"
8
9 class Chart:
10     """
11     Funciones generales para la generación de gráficas.
12     """
13     def save_as_image(self, fig, file_name: str):
14         """ Esta función guarda una gráfica como imagen.
15         Args:
16             fig (plotly.graph_objects.Figure): Objeto de la gráfica creada
17             file_name (str): Ruta donde almacenar el archivo.
18         """
19         fig.write_image(file_name, width=1350, height=730)
20
21
22 class Summary_Chart(Chart):
23     """
24     Funciones para gráfica resultados de las consultas realizadas
25     de las tablas de LifeStore.
26     """
27     def __init__(self, file_path: str) -> None:
28         """Establece parámetros default para objetos de la clase.
29
30         Args:
31             file_path (str): Ruta donde guardar gráficas.
32         """
33         super().__init__()
34         # Definir folder donde se ubicarán las gráficas
35         self.file_path = file_path
36         # Agregar atributos de formato de gráfica (visualización)
37         self.layout = go.Layout(
38             title=dict(y=0.99, x=0.5, xanchor='center', yanchor='top'),
39             xaxis=dict(showgrid=True, showline=True, linewidth=1,
40                       linecolor='black', mirror=True, gridwidth=0.4,
41                       gridcolor='rgb(204,209,208)',
42                       tickfont=dict(size=20, type='category')),
43             yaxis=dict(zeroLine=False, showline=True, linewidth=1,
44                       linecolor='black', mirror=True, gridwidth=0.4,
45                       gridcolor='rgb(204, 209, 208)',
46                       tickfont=dict(size=20)),

```

```

47         margin=dict(r=20, t=35),
48         plot_bgcolor='rgba(0,0,0,0)', width=1080, height=566,
49         font=dict(family='Arial, monospace', size=18))
50
51
52     def bar_summary(self, data: dict, plot_title: str, x_axis_name: str,
53                     y_axis_name: str, file_name: str, color:str = "blue") -> None:
54         """Gráfica de barras a partir de un diccionario.
55
56         Args:
57             data (dict): Datos a graficar.
58             plot_title (str): Título de la gráfica.
59             x_axis_name (str): Nombre de eje x.
60             y_axis_name (str): Nombre de eje y.
61             file_name (str): Nombre del archivo.
62             color (str, optional): Color del gráfico. Defaults to "blue".
63         """
64         # Se da de alta diccionario con colores disponibles
65         color_dict = {
66             "blue": "rgb(15, 78, 171)", "green": "rgb(15, 171, 72)",
67             "red": "rgb(232, 4, 0)", "purple": "rgb(109, 15, 171)",
68             "yellow": "rgb(199, 199, 18)", "orange": "rgb(232, 155, 0)"}
69         # Seleccionar color del gráfico
70         plot_color = color_dict[color]
71
72         # Separar diccionarios en dos listas, una para cada eje
73         x_data = list(data.keys())
74         y_data = list(data.values())
75
76         # Generar objeto de gráfica
77         plot_bar = go.Bar(x=x_data, y=y_data)
78         mydata = [plot_bar]
79
80         # Formato de la gráfica de barras
81         fig = go.Figure(data=mydata, layout=self.layout)
82
83         # Actualizar color del grafico
84         fig.update_traces(marker_color=plot_color,
85                           marker_line_color=plot_color)
86         # Actualizar gráfico con datos de entrada
87         fig.update_layout(
88             title_text=plot_title,
89             xaxis_title=x_axis_name,
90             yaxis_title=y_axis_name)
91
92         # Validar si folder existe, o si es necesario crearlo
93         if not os.path.exists(f"{self.file_path}"):
94             os.makedirs(f"{self.file_path}")
95         # Guardar grafico interactivo como html
96         fig.write_html(f"{self.file_path}/{file_name}.html")
97         # Guardar imagen statica en
98         self.save_as_image(fig, f"{self.file_path}/{file_name}.png")
99         # Indicar a usuario en consola
100        print(f"Resultado graficado en {self.file_path}/{file_name}.png")

```

Código 2.9: Código menu_utils.py

2.2.5. Función principal

La función principal llama a los módulos desarrollados, por lo que es necesario importarlos.

```
1 from services.lifestore_services import Service
2 from utils.graph_utils import Summary_Chart
3 from login.user_access import login
4 from utils.menu_utils import select_menu, validate_question
```

Código 2.10: Importar módulos a función principal.

Además de los módulos desarrollados, el programa va a utilizar algunas constantes globales. Entre estas constantes encontramos opciones de menús, preguntas de si o no, una lista de meses y una ruta para guardar las gráficas.

```
1 # MENUS Y SUBMENUS
2 # Definir menú principal
3 main_menu = {1: "Productos más vendidos", 2: "Productos rezagados",
4              3: "Productos por valoración", 4: "Ventas anuales",
5              5: "Ventas mensuales", 6: "Ventas por categoría",
6              7: "Consultas Avanzadas", 8: "Salir"}
7 # Definir submenú de consulta
8 menu_sales={1: "Número de ventas", 2: "Ingresos totales"}
9 # Definir submenú de año
10 menu_year = {1: "2020"}
11
12 # Preguntas de Sí o No utilizadas en el menú
13 continue_question = "&Desea realizar otra consulta?"
14 refunds_question = "&Desea descartar ventas que terminaron en devolución?"
15
16 # Variables auxiliares
17 month_list = ("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
18              "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre")
19
20 plot_path = "output" # Carpeta para gráficas
```

Código 2.11: Definición de constantes en función principal.

La ejecución pasa por diferentes etapas. La etapa inicial es crear objetos para la clase de servicios y la clase de gráficas. Posteriormente, se ejecuta la función de inicio de sesión. Esta función devuelve un booleano dependiendo de si se logró un acceso correcto o incorrecto. Posteriormente se ejecuta un while con el menú de consultas solamente si se tuvo un acceso correcto. En este menú se despliegan varias categorías de consulta. El usuario puede seleccionar una categoría. En algunos de los casos hay submenús que proporcionan más detalles de la consulta. Una vez que se tienen todos los detalles necesarios, se ejecuta el servicio. Este servicio regresa una respuesta, que se imprime de manera que sea fácil de interpretar para el usuario. Al termina la consulta se le pregunta al usuario si quiere realizar otra consulta. En caso de que el usuario responda que si, itera nuevamente el ciclo while, mientras que respondiendo que no, hay un break del ciclo y termina el programa.

```
1 # Crear objeto de la clase Services para las consultas
2 service = Service()
3 # Crear objeto de la clase Grafica
4 plot = Summary_Chart(plot_path)
5
6 # Llamar a función que valida inicio de sesión
7 data_access = login() # If true, continúes
8
9 # Acceso correcto (data_access = true)
10 while data_access:
11     # Desplegar menú principal y obtener opción seleccionada
12     main_menu_option = select_menu(main_menu)
```

```

13     # Caso para cada opción del menú principal
14     if main_menu_option == 1:
15         # Caso producto más vendidos
16         # Consulta las ventas de cada producto
17         product_sales = service.get_products_sales()
18         # Ordenar lista de productos de mayor a menor número de ventas
19         most_sold_products = sorted(product_sales, key=product_sales.get,
20                                     reverse=True)
21         # Filtrar y presentar los 50 productos más vendidos
22         print("Productos más vendidos de la tienda:")
23         for i in range(0, 50):
24             # Si no existen tantos elementos en lista, salir de bucle
25             if i >= len(most_sold_products):
26                 break
27             # Obtiene el id del producto en posición i
28             product_id = most_sold_products[i]
29             # Obtiene el nombre del producto
30             product_name = service.get_product_name(product_id)
31             # Obtiene la cantidad de unidades en inventario del producto
32             product_stock = service.get_product_stock(product_id)
33             # Muestra nombre del producto con ventas e inventario.
34             print(f"{i+1}.- {product_name} (ID: {product_id:02d}). "+
35                   f"Ventas: {product_sales[product_id]}")
36             print(f"Unidades en inventario: {product_stock}")
37         print('\n')
38         # Consulta las búsquedas de cada producto
39         product_searches = service.get_products_searches()
40         # Ordenar lista de productos de mayor a menor número de búsquedas
41         most_searched_products = sorted(product_searches,
42                                         key=product_searches.get, reverse=True)
43         # Filtrar y presentar los 50 productos más buscados
44         print("Productos más buscados de la tienda:")
45         for i in range(0, 50):
46             # Si no existieran tantos elementos en lista, salir de bucle
47             if i >= len(most_searched_products):
48                 break
49             # Obtiene el id del producto en posición i
50             product_id = most_searched_products[i]
51             # Obtiene el nombre del producto
52             product_name = service.get_product_name(product_id)
53             # Despliega nombre del producto y búsquedas al usuario.
54             print(f"{i+1}.- {product_name} (ID: {product_id: 02d}). "+
55                   f"Busquedas: {product_searches[product_id]}")
56
57     elif main_menu_option == 2:
58         # Caso productos rezagados
59         # Consulta las ventas de cada producto
60         product_sales = service.get_products_sales()
61         # Ordenar lista de productos de menor a mayor número de ventas
62         less_sold_products = sorted(product_sales, key=product_sales.get,
63                                     reverse=True)
64         # Filtrar y presentar los 50 productos menos vendidos
65         print("Productos menos vendidos de la tienda:")
66         for i in range(0, 50):
67             # Si no existieran tantos elementos en lista, salir de bucle
68             if i >= len(less_sold_products):
69                 break
70             # Obtiene el id del producto en posición i

```



```

71         product_id = less_sold_products[i]
72         # Obtiene el nombre del producto
73         product_name = service.get_product_name(product_id)
74         # Obtiene la cantidad de unidades en inventario del producto
75         product_stock = service.get_product_stock(product_id)
76         # Muestra nombre del producto con ventas e inventario
77         print(f"{i+1}.- {product_name} (ID: {product_id:02d}). "+ f"Ventas: {product_s
78         print(f"Unidades en inventario: {product_stock}.")
79     print('\n')
80     # Consulta las busquedas de cada producto
81     product_searches = service.get_products_searches()
82     # Ordenar lista de productos de menor a mayor número de búsquedas
83     less_searched_products = sorted(product_searches,
84                                     key=product_searches.get)
85     # Filtrar y presentar los 50 productos menos buscados
86     print("Productos menos buscados de la tienda:")
87     for i in range(0, 50):
88         # Si no existieran tantos elementos en lista, salir de bucle
89         if i>=len(less_searched_products):
90             break
91         # Obtiene el id del producto en posición i
92         product_id = less_searched_products[i]
93         # Obtiene el nombre del producto
94         product_name = service.get_product_name(product_id)
95         # Despliega nombre del producto y búsquedas al usuario
96         print(f"{i+1}.- {product_name} (ID: {product_id: 02d}). "+
97             f"Busquedas: {product_searches[product_id]}")
98
99     elif main_menu_option == 3:
100         # Caso de valoración de productos
101         product_grades = service.get_product_grades()
102         # Separar productos con calificación (ventas>0)
103         # de los productos no calificados ('N.D.')
104         non_graded_products = {id_product:grade for
105                                (id_product,grade) in product_grades.items()
106                                if isinstance(grade, str)}
107         graded_products = {id_product:grade for
108                             (id_product,grade) in product_grades.items()
109                             if isinstance(grade, float)}
110         # Filtrar productos por calificación de mayor a menor
111         most_valued_products = sorted(graded_products,
112                                       key=graded_products.get, reverse=True)
113         # Productos con mejor valoración
114         print("Los productos mejor valorados son:")
115         for i in range(0, 20):
116             # Si no existieran tantos elementos en lista, salir de bucle
117             if i>=len(most_valued_products):
118                 break
119             # Obtiene el id del producto en posición i
120             product_id = most_valued_products[i]
121             # Obtiene el nombre del producto
122             product_name = service.get_product_name(product_id)
123             # Muestra nombre del producto junto a su valoración
124             print(f"{i+1}.- {product_name} (ID: {product_id:02d}). "+
125                 f"Valoración: {product_grades[product_id]}")
126         print("\n")
127         # Productos con menor valoración
128         print("Los productos peor valorados son:")

```

```

129     for i in range(0, 20):
130         # Si no existieran tantos elementos en lista, salir de bucle
131         if i>=len(most_valued_products):
132             break
133         # Obtiene el id del producto en posición [-(1+i)]
134         product_id = most_valued_products[-(1+i)]
135         # Obtiene el nombre del producto
136         product_name = service.get_product_name(product_id)
137         # Muestra nombre del producto junto a su valoración
138         print(f"{i+1}.- {product_name} (ID: {product_id:02d}). " +
139               f"Valoración: {product_grades[product_id]}.")
140     print("\n")
141     print(f"Por otro lado, hay {len(non_graded_products)} equipos "+
142           "sin ventas y, en consecuencia, sin valoración.")
143     print("Los productos que no tienen valoración son:")
144     for product_id in non_graded_products.keys():
145         # Obtener nombre del producto
146         product_name = service.get_product_name(product_id)
147         # Imprime datos del producto
148         print(f"ID:{product_id:02d} - {product_name}.")
149
150     elif main_menu_option == 4:
151         # Caso: Ventas Anuales
152         # Desplegar menú de ventas y obtener opción seleccionada
153         menu_sales_option = select_menu(menu_sales)
154         # Desplegar menu para seleccionar año y obtener año como entero
155         year = int(menu_year[select_menu(menu_year)])
156         # Validar si se consideran o descartan devoluciones
157         if validate_question(refunds_question):
158             refunds_case = False
159         else:
160             refunds_case = None
161         if menu_sales_option == 1:
162             # Se obtiene número de ventas y se muestra resultado
163             sales_number = service.get_year_sales(year,
164             refund_status=refunds_case)
165             print(f"En {year}, se tuvieron un total de {sales_number}"+
166                   " ventas.")
167             print(f"En promedio, se tuvieron {round(sales_number/12)} "+
168                   "ventas al mes.")
169         elif menu_sales_option == 2:
170             # Se obtienen ingresos y se muestran resultados
171             income = service.get_year_income(year,
172             refund_status=refunds_case)
173             print(f"En {year}, los ingresos totales son ${income:,.2f}.")
174
175     elif main_menu_option == 5:
176         # Caso: Ventas Mensuales
177         # Desplegar menú de ventas y obtener opción seleccionada
178         menu_sales_option = select_menu(menu_sales)
179         # Desplegar menú para seleccionar año y obtener año como entero
180         year = int(menu_year[select_menu(menu_year)])
181         # Validar si se consideran o descartan devoluciones
182         if validate_question(refunds_question):
183             refunds_case = False
184         else:
185             refunds_case = None
186         if menu_sales_option == 1:

```

```

187     # Se obtiene número de ventas de cada mes
188     sales_month = service.get_monthly_sales(year,
189     refund_status=refunds_case)
190     # Se muestran los resultados de cada mes
191     print(f"Ventas mensuales del año {year}:")
192     for month in sales_month.keys():
193         print(f"{month}.- {month_list[month-1]}: "+
194         f"{sales_month[month]}")
195     # Se ordenan los meses de mayor a menor número de ventas
196     month_most_sales = sorted(sales_month, key=sales_month.get,
197     reverse=True)
198     # Se presentan los meses con mayor número de ventas
199     print("\nMeses con más ventas:")
200     for i in range(0,6):
201         month = month_most_sales[i]
202         print(f"{i+1}.- {month_list[month-1]}")
203     # Se presentan los meses con menor número de ventas
204     print("\nMeses con menos ventas:")
205     for i in range(0,6):
206         month = month_most_sales[11-i] # Del ultimo al primero
207         print(f"{i+1}.- {month_list[month-1]}")
208     print("")
209     # Graficar resultados
210     plot.bar_summary(sales_month, f"Ventas Mensuales en {year}",
211     "Mes", "No. de Ventas", "mes_ventas")
212 elif menu_sales_option == 2:
213     # Se obtienen ingresos y se muestran resultados
214     income_month = service.get_monthly_income(year,
215     refund_status=refunds_case)
216     # Se muestran los resultados de cada mes
217     print(f"Ingresos mensuales del año {year}:")
218     for month in income_month.keys():
219         print(f"{month}.- {month_list[month-1]}: "+
220         f"${income_month[month]:,.2f}")
221     # Se ordenan los meses en una lista de más a menos ingresos
222     month_most_income = sorted(income_month, key=income_month.get,
223     reverse=True)
224     # Se presentan los meses con mayor número de ingresos
225     print("\nMeses con más ingresos:")
226     for i in range(0,6):
227         month = month_most_income[i]
228         print(f"{i+1}.- {month_list[month-1]}")
229     # Se presentan los meses con menor número de ingresos
230     print("\nMeses con menos ingresos:")
231     for i in range(0,6):
232         month = month_most_income[11-i] # Del ultimo al primero
233         print(f"{i+1}.- {month_list[month-1]}")
234     print("")
235     # Graficar resultados
236     plot.bar_summary(income_month, f"Ingresos Mensuales en {year}",
237     "Mes", "Ingresos [$]", "mes_ingresos", "green")
238
239 elif main_menu_option == 6:
240     # Caso ventas por categorías
241     # Obtener no. de productos por categoría y mostrar
242     category_products = service.count_category_products()
243     print("Productos por categoria:")
244     for category in category_products.keys():

```

```

245         print(f"- {category}: {category_products[category]}")
246     print("")
247     # Graficar resultados
248     plot.bar_summary(category_products, "Productos por categoría",
249                     "Categoría", "No. de Producto", "categoria_productos",
250                     "red")
251     # Obtener ventas por categoría y mostrar
252     category_sales = service.get_category_sales(refund_status=False)
253     print("\nNúmero de ventas por categoría:")
254     for category in category_sales.keys():
255         print(f"- {category}: {category_sales[category]}")
256     print("")
257     # Graficar resultados
258     plot.bar_summary(category_sales, "Ventas por categoría",
259                     "Categoría", "No. de Ventas", "categoria_ventas")
260     # Obtener ingresos por categoría y mostrar
261     category_income = service.get_category_income(refund_status=False)
262     print("\nIngresos por categoría:")
263     for category in category_income.keys():
264         print(f"- {category}: ${category_income[category]:,.2f}")
265     print("")
266     # Graficar resultados
267     plot.bar_summary(category_income, "Ingresos por categoría",
268                     "Categoría", "Ingresos [$]", "categoria_ingresos",
269                     "green")
270
271     elif main_menu_option == 7:
272         print("Opción no disponible.")
273
274     elif main_menu_option == 8:
275         break
276     # Después de consultar caso, validar si desea continuar o salir.
277     print(',')
278     data_access = validate_question(continue_question)
279
280 print("\nHasta la próxima!")

```

Código 2.12: Ejecución de función principal.

2.3. INSTRUCCIONES DE USO

Al usar librerías externas, es necesario tenerlas instaladas en el ambiente donde se vaya a ejecutar el programa para garantizar una ejecución correcta. La primera condición para garantizar la ejecución es que el ambiente virtual cuente con Python instalado. A continuación se describen instrucciones para primer uso y cualquier uso posterior.

2.3.1. Primer uso

Después de descargar el repositorio, se accede por consola a la carpeta code (Fig. 2.4).

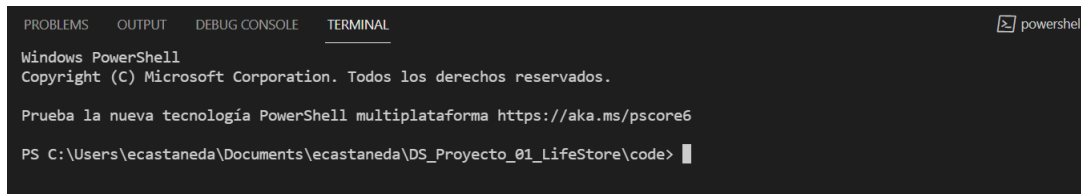


Figura 2.4: Usuario en carpeta code.

En la misma consola, se ejecuta el siguiente comando para instalar la herramienta virtualenv. Esta herramienta se usa para crear ambientes virtuales. Una vez que se tenga la librería, se ejecuta un comando para crear un ambiente virtual y otro comando para activarlo.

```
1 pip3 install virtualenv
2 python -m virtualenv venv
3 .\venv\Scripts\activate
```

A continuación, se instalan las librerías de terceros de Python que el programa necesita. Para eso, solo se tiene que ejecutar la siguiente línea de comando en la misma consola. El archivo *requirements.txt* cuenta con una lista de todas las librerías necesarias y va instalando cada una en el ambiente virtual.

```
1 pip install -r requirements.txt
```

Una vez que se tenga todo instalado, ya se puede ejecutar la función principal. Esto también se puede ejecutar desde la consola, con el comando.

```
1 python .\PROYECTO-01-CASTANEDA-EDSON.py
```

2.3.2. Usos posteriores

Para usos posteriores, no es necesario instalar librerías a menos que se hagan cambios importantes en el repositorio con nuevas herramientas. Para ejecutar el programa ya con todas las librerías, únicamente hay que activar primero el ambiente virtual desde la carpeta code, y ejecutar la función principal.

```
1 .\venv\Scripts\activate
```

```
1 python .\PROYECTO-01-CASTANEDA-EDSON.py
```

3. RESULTADOS

En esta sección del reporte, se presentan los resultados que un usuario puede consultar en el sistema implementado en Python.

3.1. DEMANDA DE PRODUCTOS

La demanda de cada producto para LifeStore puede medirse mediante dos métricas diferentes: la cantidad de veces que el producto se vendió, y la cantidad de veces que el producto se ha buscado. Mediante esto, se puede determinar en qué productos se está enfocando el interés de la clientela, y que productos no reciben nada de atención.

Para la métrica del número de ventas, se tuvieron que tomar ciertas consideraciones para el análisis. En este caso, se siguen contando como venta las transacciones que concluyeron en devolución. En el caso particular del no. de ventas, para el despliegue de resultados se decide mostrar también las unidades en existencias que se tienen de los productos, para poder distinguir si se está invirtiendo poco en productos con buena demanda, o invirtiendo mucho en productos con baja demanda.

A continuación, se presentan los resultados de las consultas para productos con mayor demanda y productos con menor demanda.

3.1.1. Productos con mayor demanda

Productos más vendidos

Los productos con un mayor número de ventas de la tienda se presentan en la Tabla 3.1. Del total de diferentes productos que tiene la tienda en sus registros, solo 42 fueron vendidos al menos 1 vez. En general en la tabla, y especialmente en las primeras diez posiciones, se distingue que los productos que más se venden son principalmente discos duros o procesadores. En casos particulares como en algunas tarjetas madre (por ejemplo, en la posición 4) el inventario no cubrirá futuras ventas que podría tener el producto. Por otro lado, se distingue que las unidades de pantallas en inventario son desproporcionadas respecto a la cantidad de estos productos que se venden en realidad.

Tabla 3.1: Productos más vendidos de LifeStore.

	ID	Nombre de Producto	Ventas	Stock
1	54	SSD Kingston A400, 120GB, SATA III, 2.5", 7mm	49	300
2	3	Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth	42	987
3	5	Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)	20	130
4	42	Tarjeta Madre ASRock Micro ATX B450M Steel Legend, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	18	0
5	57	SSD Adata Ultimate SU800, 256GB, SATA III, 2.5", 7mm	15	15
6	29	Tarjeta Madre ASUS micro ATX TUF B450M-PLUS GAMING, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	14	10

Tabla continua en siguiente hoja.

	ID	Nombre de Producto	Ventas	Stock
7	4	Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith	13	295
8	2	Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth	12	182
9	47	SSD XPG SX8200 Pro, 256GB, PCI Express, M.2	11	8
10	12	Tarjeta de Video ASUS NVIDIA GeForce GTX 1660 SUPER EVO OC, 6GB 192-bit GDDR6, PCI Express x16 3.0	9	0
11	48	SSD Kingston A2000 NVMe, 1TB, PCI Express 3.0, M2	9	50
12	7	Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)	7	114
13	31	Tarjeta Madre AORUS micro ATX B450 AORUS M (rev. 1.0), S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	6	120
14	44	Tarjeta Madre MSI ATX B450 TOMAHAWK MAX, S-AM4, AMD B450, 64GB DDR4 para AMD	6	0
15	18	Tarjeta de Video Gigabyte NVIDIA GeForce GT 1030, 2GB 64-bit GDDR5, PCI Express x16 3.0	5	5
16	8	Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generación - Coffee Lake)	4	8
17	6	Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake)	3	54
18	11	Tarjeta de Video ASUS AMD Radeon RX 570, 4GB 256-bit GDDR5, PCI Express 3.0	3	2
19	49	Kit SSD Kingston KC600, 1TB, SATA III, 2.5, 7mm	3	3
20	51	SSD Kingston UV500, 480GB, SATA III, mSATA	3	0
21	1	Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache	2	16
22	21	Tarjeta de Video MSI AMD Mech Radeon RX 5500 XT MECH Gaming OC, 8GB 128-bit GDDR6, PCI Express 4.0	2	0
23	25	Tarjeta de Video Sapphire AMD Pulse Radeon RX 5500 XT Gaming, 8GB 128-bit GDDR6, PCI Express 4.0	2	10
24	33	Tarjeta Madre ASUS ATX PRIME Z390-A, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	2	43
25	52	SSD Western Digital WD Blue 3D NAND, 2TB, M.2	2	13
26	74	Logitech Bocinas para Computadora con Subwoofer G560, Bluetooth, Inalámbrico, 2.1, 120W RMS, USB, negro	2	1
27	85	Logitech Audífonos Gamer G635 7.1, Alámbrico, 1.5 Metros, 3.5mm, Negro/Azul	2	39
28	10	MSI GeForce 210, 1GB GDDR3, DVI, VGA, HDCP, PCI Express 2.0	1	13
29	13	Tarjeta de Video Asus NVIDIA GeForce GTX 1050 Ti Phoenix, 4GB 128-bit GDDR5, PCI Express 3.0	1	1
30	17	Tarjeta de Video Gigabyte AMD Radeon R7 370 OC, 2GB 256-bit GDDR5, PCI Express 3.0	1	1
31	22	Tarjeta de Video MSI NVIDIA GeForce GTX 1050 Ti OC, 4GB 128-bit GDDR5, PCI Express x16 3.0	1	0
32	28	Tarjeta de Video Zotac NVIDIA GeForce GTX 1660 Ti, 6GB 192-bit GDDR6, PCI Express x16 3.0	1	3
33	40	Tarjeta Madre Gigabyte XL-ATX TRX40 Designare, S-sTRX4, AMD TRX40, 256GB DDR4 para AMD	1	1
34	45	Tarjeta Madre ASRock ATX H110 Pro BTC+, S-1151, Intel H110, 32GB DDR4, para Intel	1	25
35	46	Tarjeta Madre Gigabyte micro ATX GA-H110M-DS2, S-1151, Intel H110, 32GB DDR4 para Intel	1	49
36	50	SSD Crucial MX500, 1TB, SATA III, M.2	1	4
37	60	Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB), Non-ECC, CL16, XMP	1	10

Tabla continua en siguiente hoja.

	ID	Nombre de Producto	Ventas	Stock
38	66	TCL Smart TV LED 55S425 54.6, 4K Ultra HD, Widescreen, Negro	1	188
39	67	TV Monitor LED 24TL520S-PU 24, HD, Widescreen, HDMI, Negro	1	411
40	84	Logitech Audífonos Gamer G332, Alámbrico, 2 Metros, 3.5mm, Negro/Rojo	1	83
41	89	Cougar Audífonos Gamer Phontum Essential, Alámbrico, 1.9 Metros, 3.5mm, Negro.	1	4
42	94	HyperX Audífonos Gamer Cloud Flight para PC/PS4/PS4 Pro, Inalámbrico, USB, 3.5mm, Negro	1	12

Productos más buscados

Las búsquedas, presentadas en la Tabla 3.2, muestran una tendencia similar a la tabla de mayor número de ventas. Los productos que más se buscan también son principalmente procesadores y discos duros. En este caso, se distingue interés de cliente en Tarjetas Madre específicas. También se puede distinguir que hay productos que se han buscado al menos 1 vez, pero no comprado en la tienda. En el caso de las pantallas, la tabla de búsquedas muestra interés en algunos modelos particulares, como se puede ver en la posición 7 o 18 de esta tabla. En esta tabla se puede hablar también de una mayor aparición de diferentes modelos de tarjetas de vídeo y tarjetas madre.

Tabla 3.2: Productos más buscados de LifeStore.

	ID	Nombre de Producto	Búsquedas
1	54	SSD Kingston A400, 120GB, SATA III, 2.5", 7mm	263
2	57	SSD Adata Ultimate SU800, 256GB, SATA III, 2.5", 7mm	107
3	29	Tarjeta Madre ASUS micro ATX TUF B450M-PLUS GAMING, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	60
4	3	Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth	55
5	4	Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire	41
6	85	Logitech Audífonos Gamer G635 7.1, Alámbrico, 1.5 Metros, 3.5mm, Negro/Azul	35
7	67	TV Monitor LED 24TL520S-PU 24, HD, Widescreen, HDMI, Negro	32
8	7	Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)	31
9	5	Procesador Intel Core i3-9100F, S-1151, 3.60GHz, Quad-Core, 6MB Cache (9na. Generación - Coffee Lake)	30
10	47	SSD XPG SX8200 Pro, 256GB, PCI Express, M.2	30
11	48	SSD Kingston A2000 NVMe, 1TB, PCI Express 3.0, M2	27
12	44	Tarjeta Madre MSI ATX B450 TOMAHAWK MAX, S-AM4, AMD B450, 64GB DDR4 para AMD	25
13	2	Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth	24
14	42	Tarjeta Madre ASRock Micro ATX B450M Steel Legend, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	23
15	8	Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generación - Coffee Lake)	20
16	12	Tarjeta de Video ASUS NVIDIA GeForce GTX 1660 SUPER EVO OC, 6GB 192-bit GDDR6, PCI Express x16 3.0	15
17	21	Tarjeta de Video MSI AMD Mech Radeon RX 5500 XT MECH Gaming OC, 8GB 128-bit GDDR6, PCI Express 4.0	15
18	66	TCL Smart TV LED 55S425 54.6, 4K Ultra HD, Widescreen, Negro	15
19	18	Tarjeta de Video Gigabyte NVIDIA GeForce GT 1030, 2GB 64-bit GDDR5, PCI Express x16 3.0	11

Tabla continua en siguiente hoja.

	ID	Nombre de Producto	Búsquedas
20	51	SSD Kingston UV500, 480GB, SATA III, mSATA	11
21	1	Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache	10
22	6	Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake)	10
23	25	Tarjeta de Video Sapphire AMD Pulse Radeon RX 5500 XT Gaming, 8GB 128-bit GDDR6, PCI Express 4.0	10
24	31	Tarjeta Madre AORUS micro ATX B450 AORUS M (rev. 1.0), S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	10
25	40	Tarjeta Madre Gigabyte XL-ATX TRX40 Designare, S-sTRX4, AMD TRX40, 256GB DDR4 para AMD	10
26	49	Kit SSD Kingston KC600, 1TB, SATA III, 2.5, 7mm	10
27	84	Logitech Audífonos Gamer G332, Alámbrico, 2 Metros, 3.5mm, Negro/Rojo	10
28	50	SSD Crucial MX500, 1TB, SATA III, M.2	7
29	89	Cougar Audífonos Gamer Phontum Essential, Alámbrico, 1.9 Metros, 3.5mm, Negro.	7
30	74	Logitech Bocinas para Computadora con Subwoofer G560, Bluetooth, Inalámbrico, 2.1, 120W RMS, USB, negro	6
31	94	HyperX Audífonos Gamer Cloud Flight para PC/PS4/PS4 Pro, Inalámbrico, USB, 3.5mm, Negro	6
32	11	Tarjeta de Video ASUS AMD Radeon RX 570, 4GB 256-bit GDDR5, PCI Express 3.0	5
33	22	Tarjeta de Video MSI NVIDIA GeForce GTX 1050 Ti OC, 4GB 128-bit GDDR5, PCI Express x16 3.0	5
34	26	Tarjeta de Video VisionTek AMD Radeon HD 5450, 1GB DDR3, PCI Express x16 2.1	5
35	28	Tarjeta de Video Zotac NVIDIA GeForce GTX 1660 Ti, 6GB 192-bit GDDR6, PCI Express x16 3.0	5
36	52	SSD Western Digital WD Blue 3D NAND, 2TB, M.2	5
37	15	Tarjeta de Video EVGA NVIDIA GeForce GTX 1660 Ti SC Ultra Gaming, 6GB 192-bit GDDR6, PCI 3.0	4
38	46	Tarjeta Madre Gigabyte micro ATX GA-H110M-DS2, S-1151, Intel H110, 32GB DDR4 para Intel	4
39	63	Seiki TV LED SC-39HS950N 38.5, HD, Widescreen, Negro	4
40	73	Samsung Smart TV LED UN55TU7000FXZX 55, 4K Ultra HD, Widescreen, Negro/Gris	4
41	17	Tarjeta de Video Gigabyte AMD Radeon R7 370 OC, 2GB 256-bit GDDR5, PCI Express 3.0	3
42	39	ASUS T. Madre uATX M4A88T-M, S-AM3, DDR3 para Phenom II/Athlon II/Sempron 100	3
43	95	Logear Audífonos Gamer GHG601, Alámbrico, 1.2 Metros, 3.5mm, Negro	3
44	13	Tarjeta de Video Asus NVIDIA GeForce GTX 1050 Ti Phoenix, 4GB 128-bit GDDR5, PCI Express 3.0	2
45	56	SSD para Servidor Lenovo Thinksystem S4500, 480GB, SATA III, 3.5", 7mm	2
46	76	Acteck Bocina con Subwoofer AXF-290, Bluetooth, Inalámbrico, 2.1, 18W RMS, 180W PMPO, USB, Negro	2
47	91	Genius GHP-400S Audífonos, Alámbrico, 1.5 Metros, Rosa	2
48	9	Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)	1
49	10	MSI GeForce 210, 1GB GDDR3, DVI, VGA, HDCP, PCI Express 2.0	1
50	27	Tarjeta de Video VisionTek AMD Radeon HD5450, 2GB GDDR3, PCI Express x16	1

3.1.2. Productos con menor demanda

Productos menos vendidos

La tabla 3.3 muestra que hay muchos productos del catálogo de LifeStore que en realidad no se están vendiendo. Entre estos productos, destacan muchos modelos de pantallas, bocinas y audífonos. En muchos casos el inventario es bajo, siendo menos de 10 unidades las que se desean vender, pero en otros, las unidades en inventario son mayores a 30 y pueden llegar a valores muy altos. Un ejemplo claro de esto es el producto "Makena Smart TV LED 40S2 40", Full HD, Widescreen, Negro", que no ha registrado ninguna venta, pero hay 239 unidades en inventario. En menor medida, pero también con una cantidad importante de unidades en inventario pueden identificarse diversos modelos de tarjetas de vídeo. La presencia de modelos tarjetas madre, procesadores, o discos duros es baja, pero también hay casos de este tipo de productos que no presentan ventas.

Tabla 3.3: Productos menos vendidos de LifeStore.

	ID	Nombre de Producto	Ventas	Stock
1	9	Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake)	0	35
2	14	Tarjeta de Video EVGA NVIDIA GeForce GT 710, 2GB 64-bit GDDR3, PCI Express 2.0	0	36
3	15	Tarjeta de Video EVGA NVIDIA GeForce GTX 1660 Ti SC Ultra Gaming, 6GB 192-bit GDDR6, PCI 3.0	0	15
4	16	Tarjeta de Video EVGA NVIDIA GeForce RTX 2060 SC ULTRA Gaming, 6GB 192-bit GDDR6, PCI Express 3.0	0	10
5	19	Tarjeta de Video Gigabyte NVIDIA GeForce GTX 1650 OC Low Profile, 4GB 128-bit GDDR5, PCI Express 3.0 x16	0	8
6	20	Tarjeta de Video Gigabyte NVIDIA GeForce RTX 2060 SUPER WINDFORCE OC, 8 GB 256 bit GDDR6, PCI Express x16 3.0	0	10
7	23	Tarjeta de Video MSI Radeon X1550, 128MB 64 bit GDDR2, PCI Express x16	0	10
8	24	Tarjeta de Video PNY NVIDIA GeForce RTX 2080, 8GB 256-bit GDDR6, PCI Express 3.0	0	2
9	26	Tarjeta de Video VisionTek AMD Radeon HD 5450, 1GB DDR3, PCI Express x16 2.1	0	180
10	27	Tarjeta de Video VisionTek AMD Radeon HD5450, 2GB GDDR3, PCI Express x16	0	43
11	30	Tarjeta Madre AORUS ATX Z390 ELITE, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0	50
12	32	Tarjeta Madre ASRock Z390 Phantom Gaming 4, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0	10
13	34	Tarjeta Madre ASUS ATX ROG STRIX B550-F GAMING WI-FI, S-AM4, AMD B550, HDMI, max. 128GB DDR4 para AMD	0	2
14	35	Tarjeta Madre Gigabyte micro ATX Z390 M GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0	30
15	36	Tarjeta Madre Gigabyte micro ATX Z490M GAMING X (rev. 1.0), Intel Z490, HDMI, 128GB DDR4 para Intel	0	10
16	37	Tarjeta Madre ASRock ATX Z490 STEEL LEGEND, S-1200, Intel Z490, HDMI, 128GB DDR4 para Intel	0	60
17	38	Tarjeta Madre Gigabyte Micro ATX H310M DS2 2.0, S-1151, Intel H310, 32GB DDR4 para Intel	0	15
18	39	ASUS T. Madre uATX M4A88T-M, S-AM3, DDR3 para Phenom II/Athlon II/Sempron 100	0	98
19	41	Tarjeta Madre ASUS micro ATX Prime H370M-Plus/CSM, S-1151, Intel H370, HDMI, 64GB DDR4 para Intel	0	286
20	43	Tarjeta Madre ASUS ATX ROG STRIX Z390-E GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0	5

Tabla continua en siguiente hoja.

	ID	Nombre de Producto	Ventas	Stock
21	53	SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2	0	1
22	55	SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s	0	10
23	56	SSD para Servidor Lenovo Thinksystem S4500, 480GB, SATA III, 3.5", 7mm	0	3
24	58	SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5", 7mm	0	16
25	59	SSD Samsung 860 EVO, 1TB, SATA III, M.2	0	10
26	61	Kit Memoria RAM Corsair Vengeance LPX DDR4, 2400MHz, 32GB, Non-ECC, CL16	0	5
27	62	Makena Smart TV LED 32S2 32", HD, Widescreen, Gris	0	6
28	63	Seiki TV LED SC-39HS950N 38.5, HD, Widescreen, Negro	0	146
29	64	Samsung TV LED LH43QMREBGCXGO 43, 4K Ultra HD, Widescreen, Negro	0	71
30	65	Samsung Smart TV LED UN70RU7100FXZX 70, 4K Ultra HD, Widescreen, Negro	0	7
31	68	Makena Smart TV LED 40S2 40", Full HD, Widescreen, Negro	0	239
32	69	Hisense Smart TV LED 40H5500F 39.5, Full HD, Widescreen, Negro	0	94
33	70	Samsung Smart TV LED 43, Full HD, Widescreen, Negro	0	10
34	71	Samsung Smart TV LED UN32J4290AF 32, HD, Widescreen, Negro	0	3
35	72	Hisense Smart TV LED 50H8F 49.5, 4K Ultra HD, Widescreen, Negro	0	11
36	73	Samsung Smart TV LED UN55TU7000FXZX 55, 4K Ultra HD, Widescreen, Negro/Gris	0	4
37	75	Lenovo Barra de Sonido, Alámbrico, 2.5W, USB, Negro	0	11
38	76	Acteck Bocina con Subwoofer AXF-290, Bluetooth, Inalámbrico, 2.1, 18W RMS, 180W PMPO, USB, Negro	0	18
39	77	Verbatim Bocina Portátil Mini, Bluetooth, Inalámbrico, 3W RMS, USB, Blanco	0	1
40	78	Ghia Bocina Portátil BX300, Bluetooth, Inalámbrico, 40W RMS, USB, Rojo - Resistente al Agua	0	2
41	79	Naceb Bocina Portátil NA-0301, Bluetooth, Inalámbrico, USB 2.0, Rojo	0	31
42	80	Ghia Bocina Portátil BX800, Bluetooth, Inalámbrico, 2.1 Canales, 31W, USB, Negro	0	15
43	81	Ghia Bocina Portátil BX900, Bluetooth, Inalámbrico, 2.1 Canales, 34W, USB, Negro - Resistente al Agua	0	20
44	82	Ghia Bocina Portátil BX400, Bluetooth, Inalámbrico, 8W RMS, USB, Negro	0	31
45	83	Ghia Bocina Portátil BX500, Bluetooth, Inalámbrico, 10W RMS, USB, Gris	0	16
46	86	ASUS Audífonos Gamer ROG Theta 7.1, Alámbrico, USB C, Negro	0	20
47	87	Acer Audífonos Gamer Galea 300, Alámbrico, 3.5mm, Negro	0	8
48	88	Audífonos Gamer Balam Rush Orphix RGB 7.1, Alámbrico, USB, Negro	0	15
49	90	Energy Sistem Audífonos con Micrófono Headphones 1, Bluetooth, Inalámbrico, Negro/Grafito	0	1
50	91	Genius GHP-400S Audífonos, Alámbrico, 1.5 Metros, Rosa	0	16

Productos menos buscados

Muchos de los productos que vende la empresa tampoco están siendo buscados por los usuarios. Esto se muestra en la Tabla 3.4, donde la mayoría de los productos no han tenido ninguna búsqueda. Al igual que en los casos de mayor demanda, muchos de los productos que aparecieron en la tabla de menos ventas se presentan en esta tabla también. Tal como en la tabla de menos ventas, en la tabla de menos búsquedas se identifican muchos modelos de pantallas, bocinas y audífonos.

Tabla 3.4: Productos menos buscados de LifeStore.

	ID	Nombre de Producto	Búsquedas
1	14	Tarjeta de Video EVGA NVIDIA GeForce GT 710, 2GB 64-bit GDDR3, PCI Express 2.0	0
2	16	Tarjeta de Video EVGA NVIDIA GeForce RTX 2060 SC ULTRA Gaming, 6GB 192-bit GDDR6, PCI Express 3.0	0
3	19	Tarjeta de Video Gigabyte NVIDIA GeForce GTX 1650 OC Low Profile, 4GB 128-bit GDDR5, PCI Express 3.0 x16	0
4	20	Tarjeta de Video Gigabyte NVIDIA GeForce RTX 2060 SUPER WINDFORCE OC, 8 GB 256 bit GDDR6, PCI Express x16 3.0	0
5	23	Tarjeta de Video MSI Radeon X1550, 128MB 64 bit GDDR2, PCI Express x16	0
6	24	Tarjeta de Video PNY NVIDIA GeForce RTX 2080, 8GB 256-bit GDDR6, PCI Express 3.0	0
7	30	Tarjeta Madre AORUS ATX Z390 ELITE, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0
8	32	Tarjeta Madre ASRock Z390 Phantom Gaming 4, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0
9	33	Tarjeta Madre ASUS ATX PRIME Z390-A, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0
10	34	Tarjeta Madre ASUS ATX ROG STRIX B550-F GAMING WI-FI, S-AM4, AMD B550, HDMI, max. 128GB DDR4 para AMD	0
11	36	Tarjeta Madre Gigabyte micro ATX Z490M GAMING X (rev. 1.0), Intel Z490, HDMI, 128GB DDR4 para Intel	0
12	37	Tarjeta Madre ASRock ATX Z490 STEEL LEGEND, S-1200, Intel Z490, HDMI, 128GB DDR4 para Intel	0
13	38	Tarjeta Madre Gigabyte Micro ATX H310M DS2 2.0, S-1151, Intel H310, 32GB DDR4 para Intel	0
14	41	Tarjeta Madre ASUS micro ATX Prime H370M-Plus/CSM, S-1151, Intel H370, HDMI, 64GB DDR4 para Intel	0
15	43	Tarjeta Madre ASUS ATX ROG STRIX Z390-E GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	0
16	53	SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2	0
17	55	SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s	0
18	58	SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5", 7mm	0
19	60	Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB), Non-ECC, CL16, XMP	0
20	61	Kit Memoria RAM Corsair Vengeance LPX DDR4, 2400MHz, 32GB, Non-ECC, CL16	0
21	62	Makena Smart TV LED 32S2 32", HD, Widescreen, Gris	0
22	64	Samsung TV LED LH43QMREBGCXGO 43, 4K Ultra HD, Widescreen, Negro	0
23	65	Samsung Smart TV LED UN70RU7100FXZX 70, 4K Ultra HD, Widescreen, Negro	0
24	68	Makena Smart TV LED 40S2 40", Full HD, Widescreen, Negro	0

Tabla continua en siguiente hoja.

	ID	Nombre de Producto	Búsquedas
25	69	Hisense Smart TV LED 40H5500F 39.5, Full HD, Widescreen, Negro	0
26	71	Samsung Smart TV LED UN32J4290AF 32, HD, Widescreen, Negro	0
27	72	Hisense Smart TV LED 50H8F 49.5, 4K Ultra HD, Widescreen, Negro	0
28	75	Lenovo Barra de Sonido, Alámbrico, 2.5W, USB, Negro	0
29	77	Verbatim Bocina Portátil Mini, Bluetooth, Inalámbrico, 3W RMS, USB, Blanco	0
30	78	Ghia Bocina Portátil BX300, Bluetooth, Inalámbrico, 40W RMS, USB, Rojo - Resistente al Agua	0
31	79	Naceb Bocina Portátil NA-0301, Bluetooth, Inalámbrico, USB 2.0, Rojo	0
32	81	Ghia Bocina Portátil BX900, Bluetooth, Inalámbrico, 2.1 Canales, 34W, USB, Negro - Resistente al Agua	0
33	82	Ghia Bocina Portátil BX400, Bluetooth, Inalámbrico, 8W RMS, USB, Negro	0
34	83	Ghia Bocina Portátil BX500, Bluetooth, Inalámbrico, 10W RMS, USB, Gris	0
35	86	ASUS Audífonos Gamer ROG Theta 7.1, Alámbrico, USB C, Negro	0
36	87	Acer Audífonos Gamer Galea 300, Alámbrico, 3.5mm, Negro	0
37	88	Audífonos Gamer Balam Rush Orphix RGB 7.1, Alámbrico, USB, Negro	0
38	90	Energy Sistem Audífonos con Micrófono Headphones 1, Bluetooth, Inalámbrico, Negro/Grafito	0
39	92	Getttech Audífonos con Micrófono Sonority, Alámbrico, 1.2 Metros, 3.5mm, Negro/Rosa	0
40	96	Klip Xtreme Audífonos Blast, Bluetooth, Inalámbrico, Negro/Verde	0
41	35	Tarjeta Madre Gigabyte micro ATX Z390 M GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	1
42	45	Tarjeta Madre ASRock ATX H110 Pro BTC+, S-1151, Intel H110, 32GB DDR4, para Intel	1
43	59	SSD Samsung 860 EVO, 1TB, SATA III, M.2	1
44	70	Samsung Smart TV LED 43, Full HD, Widescreen, Negro	1
45	80	Ghia Bocina Portátil BX800, Bluetooth, Inalámbrico, 2.1 Canales, 31W, USB, Negro	1
46	93	Ginga Audífonos con Micrófono GI18ADJ01BT-RO, Bluetooth, Alámbrico/Inalámbrico, 3.5mm, Rojo	1

3.2. VALORACIÓN DEL CLIENTE A PRODUCTOS

Otro aspecto importante para poder medir y evaluar el éxito en ventas es mediante la valoración que los productos reciban de parte del cliente. El factor principal para medir dicha valoración son las reseñas que existen por cada venta, donde un producto se evalúa con una calificación de 1 a 5. Sin embargo, otro factor que debe considerarse en la valoración es la cantidad de ventas de un producto que concluyeron en devolución. Para determinar la valoración de los productos, se obtienen porcentajes de cada factor y se usan en una suma dando a cada uno de estos factores un peso que sumados sean iguales a 1 y se representó el valor final como un porcentaje (0 a 100). Los valores que se usaron por defecto fueron darle un peso de 0.6 a las reseñas y 0.4 a las devoluciones. Ambos factores requieren de registrar al menos una venta, por lo que productos sin ventas no pueden tener valoración.

3.2.1. Productos con mayor valoración

En la Tabla 3.5, la cantidad de veces que el producto se vendió no es un factor, solo la opinión de los clientes al presentarse al menos una venta. Muchos productos cuentan con valoración del 100 %, lo que se traduce en solo reseñas de 5 y ninguna devolución. Los productos que no tuvieron calificación perfecta se mantuvieron cercanos a esa valoración.

Tabla 3.5: Productos con mayor valoración de LifeStore.

	ID	Nombre del Producto	Valoración
1	1	Procesador AMD Ryzen 3 3300X S-AM4, 3.80GHz, Quad-Core, 16MB L2 Cache	100
2	6	Procesador Intel Core i9-9900K, S-1151, 3.60GHz, 8-Core, 16MB Smart Cache (9na. Generación Coffee Lake)	100
3	7	Procesador Intel Core i7-9700K, S-1151, 3.60GHz, 8-Core, 12MB Smart Cache (9na. Generación Coffee Lake)	100
4	8	Procesador Intel Core i5-9600K, S-1151, 3.70GHz, Six-Core, 9MB Smart Cache (9na. Generación - Coffee Lake)	100
5	11	Tarjeta de Video ASUS AMD Radeon RX 570, 4GB 256-bit GDDR5, PCI Express 3.0	100
6	21	Tarjeta de Video MSI AMD Mech Radeon RX 5500 XT MECH Gaming OC, 8GB 128-bit GDDR6, PCI Express 4.0	100
7	22	Tarjeta de Video MSI NVIDIA GeForce GTX 1050 Ti OC, 4GB 128-bit GDDR5, PCI Express x16 3.0	100
8	25	Tarjeta de Video Sapphire AMD Pulse Radeon RX 5500 XT Gaming, 8GB 128-bit GDDR6, PCI Express 4.0	100
9	28	Tarjeta de Video Zotac NVIDIA GeForce GTX 1660 Ti, 6GB 192-bit GDDR6, PCI Express x16 3.0	100
10	40	Tarjeta Madre Gigabyte XL-ATX TRX40 Designare, S-sTRX4, AMD TRX40, 256GB DDR4 para AMD	100
11	49	Kit SSD Kingston KC600, 1TB, SATA III, 2.5, 7mm	100
12	50	SSD Crucial MX500, 1TB, SATA III, M.2	100
13	52	SSD Western Digital WD Blue 3D NAND, 2TB, M.2	100
14	60	Kit Memoria RAM Corsair Dominator Platinum DDR4, 3200MHz, 16GB (2x 8GB), Non-ECC, CL16, XMP	100
15	66	TCL Smart TV LED 55S425 54.6, 4K Ultra HD, Widescreen, Negro	100
16	67	TV Monitor LED 24TL520S-PU 24, HD, Widescreen, HDMI, Negro	100
17	84	Logitech Audífonos Gamer G332, Alámbrico, 2 Metros, 3.5mm, Negro/Rojo	100
18	85	Logitech Audífonos Gamer G635 7.1, Alámbrico, 1.5 Metros, 3.5mm, Negro/Azul	100
19	57	SSD Adata Ultimate SU800, 256GB, SATA III, 2.5", 7mm	98
20	3	Procesador AMD Ryzen 5 2600, S-AM4, 3.40GHz, Six-Core, 16MB L3 Cache, con Disipador Wraith Stealth	97.14

3.2.2. Productos con menor valoración

Entre los productos vendidos, hay algunos que no se han percibido del todo bien por parte del cliente, tal como muestra la Tabla 3.6. Los dos productos con peor valoración registran una calificación menor a 0, lo que refiere a la peor calificación en las reseñas y que fueron devueltos. Así mismo hay otros productos con calificación baja. Los productos vendidos con valoración más baja son tarjetas madre y tarjetas de vídeo. La mayoría de los productos en la tabla cuenta con valoraciones mayores al 80 %, es decir, en general los productos comprados resultan del agrado del cliente.

Tabla 3.6: Productos con menor valoración de LifeStore.

	ID	Nombre del Producto	Valoración
1	45	Tarjeta Madre ASRock ATX H110 Pro BTC+, S-1151, Intel H110, 32GB DDR4, para Intel	0
2	17	Tarjeta de Video Gigabyte AMD Radeon R7 370 OC, 2GB 256-bit GDDR5, PCI Express 3.0	0
3	46	Tarjeta Madre Gigabyte micro ATX GA-H110M-DS2, S-1151, Intel H110, 32GB DDR4 para Intel	15
4	31	Tarjeta Madre AORUS micro ATX B450 AORUS M (rev. 1.0), S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	32.5
5	89	Cougar Audífonos Gamer Phontum Essential, Alámbrico, 1.9 Metros, 3.5mm, Negro.	70
6	29	Tarjeta Madre ASUS micro ATX TUF B450M-PLUS GAMING, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	84.29
7	94	HyperX Audífonos Gamer Cloud Flight para PC/PS4/PS4 Pro, Inalámbrico, USB, 3.5mm, Negro	85
8	13	Tarjeta de Video Asus NVIDIA GeForce GTX 1050 Ti Phoenix, 4GB 128-bit GDDR5, PCI Express 3.0	85
9	10	MSI GeForce 210, 1GB GDDR3, DVI, VGA, HDCP, PCI Express 2.0	85
10	2	Procesador AMD Ryzen 5 3600, S-AM4, 3.60GHz, 32MB L3 Cache, con Disipador Wraith Stealth	90
11	18	Tarjeta de Video Gigabyte NVIDIA GeForce GT 1030, 2GB 64-bit GDDR5, PCI Express x16 3.0	91
12	4	Procesador AMD Ryzen 3 3200G con Gráficos Radeon Vega 8, S-AM4, 3.60GHz, Quad-Core, 4MB L3, con Disipador Wraith Spire	91.92
13	74	Logitech Bocinas para Computadora con Subwoofer G560, Bluetooth, Inalámbrico, 2.1, 120W RMS, USB, negro	92.5
14	33	Tarjeta Madre ASUS ATX PRIME Z390-A, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel	92.5
15	47	SSD XPG SX8200 Pro, 256GB, PCI Express, M.2	93.18
16	42	Tarjeta Madre ASRock Micro ATX B450M Steel Legend, S-AM4, AMD B450, HDMI, 64GB DDR4 para AMD	93.33
17	51	SSD Kingston UV500, 480GB, SATA III, mSATA	95
18	48	SSD Kingston A2000 NVMe, 1TB, PCI Express 3.0, M2	95
19	44	Tarjeta Madre MSI ATX B450 TOMAHAWK MAX, S-AM4, AMD B450, 64GB DDR4 para AMD	95
20	54	SSD Kingston A400, 120GB, SATA III, 2.5", 7mm	95.2

3.2.3. Productos sin valoración

Los productos que no tuvieron ventas, y por ende no se pueden valorar, se muestran en la Tabla 3.7. En total hay 54 productos que no han tenido ventas. Muchos de productos presentados en las tablas de menor demanda aparecen en esta tabla. Hay una gran variedad de modelos de tarjetas de vídeo, tarjetas madre, pantallas y equipos de audio que no se están vendiendo.

Tabla 3.7: Productos sin valoración de LifeStore.

ID	Nombre del Producto
9	Procesador Intel Core i3-8100, S-1151, 3.60GHz, Quad-Core, 6MB Smart Cache (8va. Generación - Coffee Lake).
14	Tarjeta de Video EVGA NVIDIA GeForce GT 710, 2GB 64-bit GDDR3, PCI Express 2.0.
15	Tarjeta de Video EVGA NVIDIA GeForce GTX 1660 Ti SC Ultra Gaming, 6GB 192-bit GDDR6, PCI 3.0.
16	Tarjeta de Video EVGA NVIDIA GeForce RTX 2060 SC ULTRA Gaming, 6GB 192-bit GDDR6, PCI Express 3.0.
19	Tarjeta de Video Gigabyte NVIDIA GeForce GTX 1650 OC Low Profile, 4GB 128-bit GDDR5, PCI Express 3.0 x16.
20	Tarjeta de Video Gigabyte NVIDIA GeForce RTX 2060 SUPER WINDFORCE OC, 8 GB 256 bit GDDR6, PCI Express x16 3.0.
23	Tarjeta de Video MSI Radeon X1550, 128MB 64 bit GDDR2, PCI Express x16.
24	Tarjeta de Video PNY NVIDIA GeForce RTX 2080, 8GB 256-bit GDDR6, PCI Express 3.0 .
26	Tarjeta de Video VisionTek AMD Radeon HD 5450, 1GB DDR3, PCI Express x16 2.1.
27	Tarjeta de Video VisionTek AMD Radeon HD5450, 2GB GDDR3, PCI Express x16.
30	Tarjeta Madre AORUS ATX Z390 ELITE, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel.
32	Tarjeta Madre ASRock Z390 Phantom Gaming 4, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel .
34	Tarjeta Madre ASUS ATX ROG STRIX B550-F GAMING WI-FI, S-AM4, AMD B550, HDMI, max. 128GB DDR4 para AMD.
35	Tarjeta Madre Gigabyte micro ATX Z390 M GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel .
36	Tarjeta Madre Gigabyte micro ATX Z490M GAMING X (rev. 1.0), Intel Z490, HDMI, 128GB DDR4 para Intel.
37	Tarjeta Madre ASRock ATX Z490 STEEL LEGEND, S-1200, Intel Z490, HDMI, 128GB DDR4 para Intel.
38	Tarjeta Madre Gigabyte Micro ATX H310M DS2 2.0, S-1151, Intel H310, 32GB DDR4 para Intel .
39	ASUS T. Madre uATX M4A88T-M, S-AM3, DDR3 para Phenom II/Athlon II/Sempron 100.
41	Tarjeta Madre ASUS micro ATX Prime H370M-Plus/CSM, S-1151, Intel H370, HDMI, 64GB DDR4 para Intel.
43	Tarjeta Madre ASUS ATX ROG STRIX Z390-E GAMING, S-1151, Intel Z390, HDMI, 64GB DDR4 para Intel.
53	SSD Addlink Technology S70, 512GB, PCI Express 3.0, M.2.
55	SSD para Servidor Supermicro SSD-DM128-SMCMVN1, 128GB, SATA III, mSATA, 6Gbit/s.
56	SSD para Servidor Lenovo Thinksystem S4500, 480GB, SATA III, 3.5", 7mm.
58	SSD para Servidor Lenovo Thinksystem S4510, 480GB, SATA III, 2.5", 7mm.
59	SSD Samsung 860 EVO, 1TB, SATA III, M.2.
61	Kit Memoria RAM Corsair Vengeance LPX DDR4, 2400MHz, 32GB, Non-ECC, CL16.
62	Makena Smart TV LED 32S2 32", HD, Widescreen, Gris.

Tabla continua en siguiente hoja.

ID	Nombre del Producto
63	Seiki TV LED SC-39HS950N 38.5, HD, Widescreen, Negro.
64	Samsung TV LED LH43QMREBGCXGO 43, 4K Ultra HD, Widescreen, Negro.
65	Samsung Smart TV LED UN70RU7100FXZX 70, 4K Ultra HD, Widescreen, Negro.
68	Makena Smart TV LED 40S2 40", Full HD, Widescreen, Negro.
69	Hisense Smart TV LED 40H5500F 39.5, Full HD, Widescreen, Negro.
70	Samsung Smart TV LED 43, Full HD, Widescreen, Negro.
71	Samsung Smart TV LED UN32J4290AF 32, HD, Widescreen, Negro.
72	Hisense Smart TV LED 50H8F 49.5, 4K Ultra HD, Widescreen, Negro.
73	Samsung Smart TV LED UN55TU7000FXZX 55, 4K Ultra HD, Widescreen, Negro/Gris.
75	Lenovo Barra de Sonido, Alámbrico, 2.5W, USB, Negro.
76	Acteck Bocina con Subwoofer AXF-290, Bluetooth, Inalámbrico, 2.1, 18W RMS, 180W PMPO, USB, Negro.
77	Verbatim Bocina Portátil Mini, Bluetooth, Inalámbrico, 3W RMS, USB, Blanco.
78	Ghia Bocina Portátil BX300, Bluetooth, Inalámbrico, 40W RMS, USB, Rojo - Resistente al Agua.
79	Naceb Bocina Portátil NA-0301, Bluetooth, Inalámbrico, USB 2.0, Rojo.
80	Ghia Bocina Portátil BX800, Bluetooth, Inalámbrico, 2.1 Canales, 31W, USB, Negro.
81	Ghia Bocina Portátil BX900, Bluetooth, Inalámbrico, 2.1 Canales, 34W, USB, Negro - Resistente al Agua.
82	Ghia Bocina Portátil BX400, Bluetooth, Inalámbrico, 8W RMS, USB, Negro.
83	Ghia Bocina Portátil BX500, Bluetooth, Inalámbrico, 10W RMS, USB, Gris.
86	ASUS Audífonos Gamer ROG Theta 7.1, Alámbrico, USB C, Negro.
87	Acer Audífonos Gamer Galea 300, Alámbrico, 3.5mm, Negro.
88	Audífonos Gamer Balam Rush Orphix RGB 7.1, Alámbrico, USB, Negro.
90	Energy Sistem Audífonos con Micrófono Headphones 1, Bluetooth, Inalámbrico, Negro/Grafito.
91	Genius GHP-400S Audífonos, Alámbrico, 1.5 Metros, Rosa.
92	Getttech Audífonos con Micrófono Sonority, Alámbrico, 1.2 Metros, 3.5mm, Negro/Rosa.
93	Ginga Audífonos con Micrófono GI18ADJ01BT-RO, Bluetooth, Alámbrico/Inalámbrico, 3.5mm, Rojo.
95	logear Audífonos Gamer GHG601, Alámbrico, 1.2 Metros, 3.5mm, Negro.
96	Klip Xtreme Audífonos Blast, Bluetooth, Inalámbrico, Negro/Verde.

3.3. VENTAS E INGRESOS EN 2020

Para el conteo de ventas y el cálculo de ingresos presentado en esta sección se están omitiendo productos que terminaron en devoluciones.

3.3.1. Resumen anual

En 2020, se tuvieron un total de 273 ventas. En promedio, se tuvieron 23 ventas al mes. Las ventas equivalieron a ingresos totales de \$737,657.00.

3.3.2. Resumen mensual

Cada mes tuvo un comportamiento diferente, como se muestra en la Tabla 3.8. Más ventas no significa necesariamente más ingresos, ya que este último factor a su vez se relaciona con el tipo de productos que se vendan.

Tabla 3.8: Ventas e ingresos de LifeStore.

Mes	Ventas	Ingresos
Enero	37	\$72,983
Febrero	36	\$107,674
Marzo	41	\$131,649
Abril	71	\$169,319
Mayo	19	\$69,071
Junio	16	\$57,314
Julio	13	\$38,037
Agosto	5	\$7,655
Septiembre	14	\$36,916
Octubre	9	\$16,361
Noviembre	9	\$21,021
Diciembre	3	\$9,657

Las variación de las ventas por mes puede observarse en la Fig. 3.1. Al principio del año se observa un mayor número de ventas. Los meses con más y menos ventas se indican a continuación.

Meses con más ventas:

1. Abril
2. Marzo
3. Enero
4. Febrero
5. Mayo
6. Junio

Meses con menos ventas:

1. Diciembre
2. Agosto
3. Noviembre
4. Octubre
5. Julio
6. Septiembre

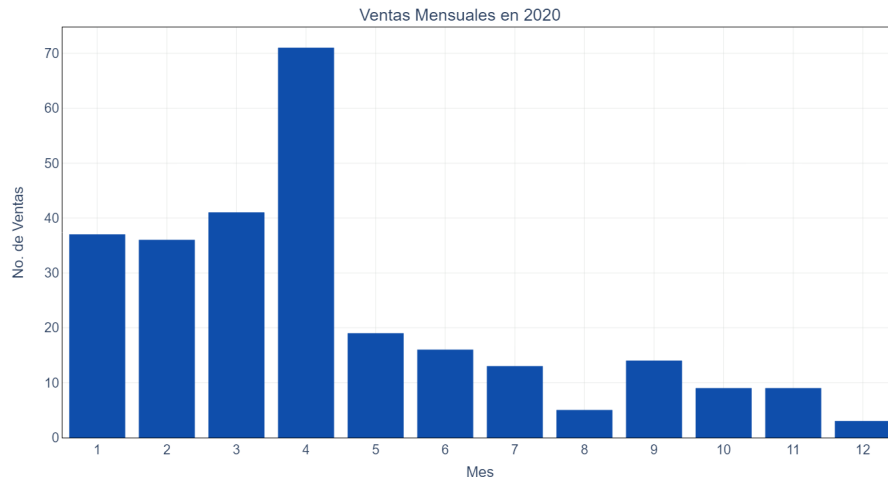


Figura 3.1: Ventas por mes.

En los ingresos (Fig. 3.2), como se mencionó anteriormente, la tendencia se mantiene, pero hay meses que con menos ventas tienen mayores ingresos, debido a los productos vendidos en esos meses. Los meses con más y menos ingresos se muestran a continuación.

Meses con más ingresos:

1. Abril
2. Marzo
3. Febrero
4. Enero
5. Mayo
6. Junio

Meses con menos ingresos:

1. Agosto
2. Diciembre
3. Octubre
4. Noviembre
5. Septiembre
6. Julio

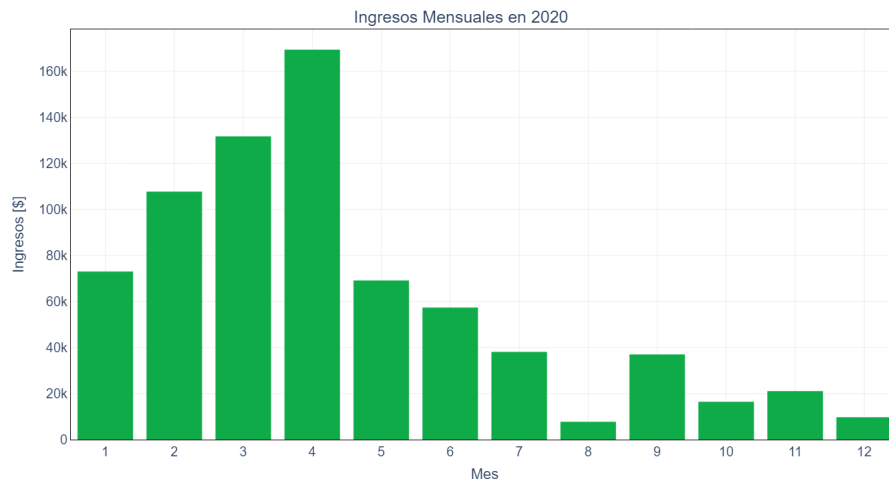


Figura 3.2: Ingresos por mes.

3.4. VENTAS E INGRESOS POR CATEGORÍA

La Tabla 3.9 muestra la cantidad de productos, así como de ventas e ingreso que tiene LifeStore por categoría. Esta consulta puede ayudar a visualizar en que área de productos se están teniendo buenos resultados, y que otras áreas necesitan mayor esfuerzo o una evaluación.

Tabla 3.9: Ventas e ingresos por categorías de LifeStore.

Categoría	No. Productos	Ventas	Ingresos
Procesadores	9	103	\$367,517
Tarjetas de vídeo	19	25	\$132,025
Tarjetas madre	18	43	\$113,727
Discos duros	13	92	\$92,978
Memorias USB	2	1	\$2,519
Pantallas	12	2	\$11,278
Bocinas	10	2	\$8,478
Audífonos	13	5	\$9,135

Tal como muestra la Fig. 3.3, la distribución de productos en cada categoría es mayormente uniforme. Los productos en los que hay una mayor cantidad de modelos son las tarjetas madre y tarjetas de vídeo, con 18 y 19 modelos respectivamente. Por otro lado, en la categoría de memorias USB, solo hay 2 modelos diferentes.

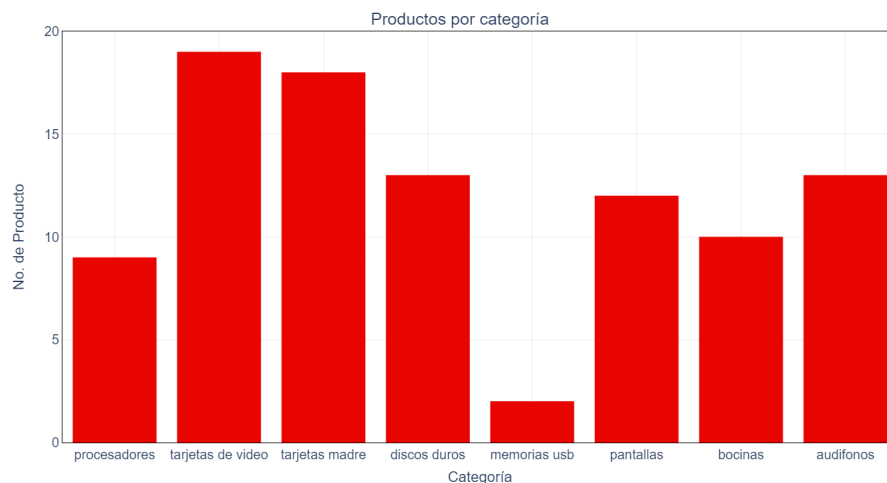


Figura 3.3: Producto por categoría.

Las ventas (Fig. 3.4) , por otro lado, se enfocan principalmente procesadores y discos duros, siendo esta más del 50 % de las ventas totales de LifeStore. Otras áreas con una cantidad considerable de ventas son las tarjetas madre y las tarjetas de vídeo.

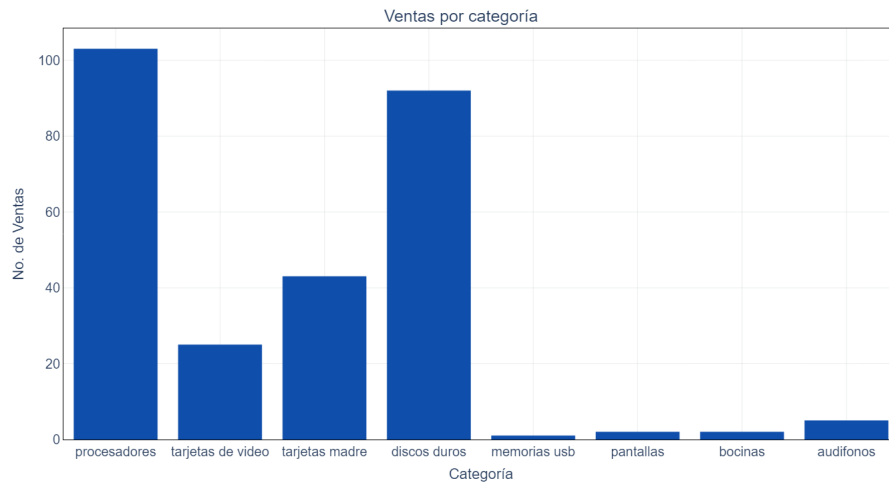


Figura 3.4: Ventas por categoría.

La Fig. 3.5 muestra que la mayor parte de ingresos que registra LifeStore proviene de sus procesadores. Aunque los discos duros se vendan más que las tarjetas de vídeo y tarjetas madre, estas últimas dos generan mayores ingresos.

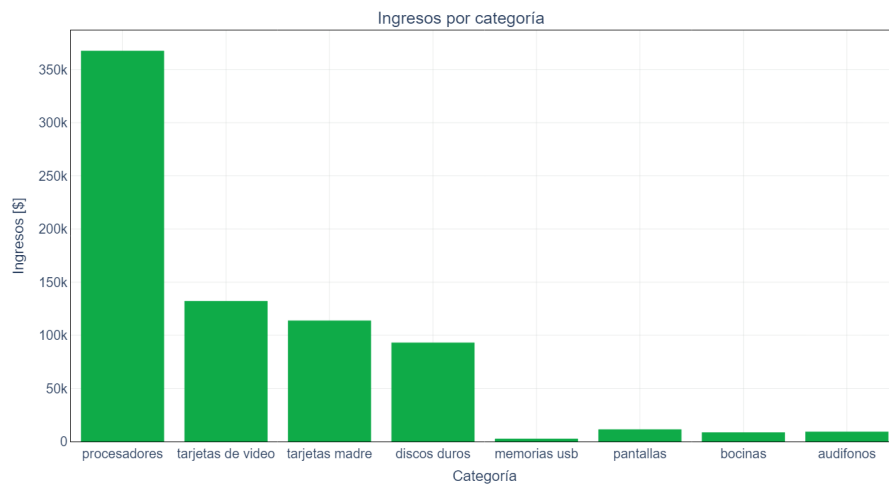


Figura 3.5: Ingresos por categoría.

4. SOLUCIÓN AL PROBLEMA

Después de revisar los resultados, se puede detectar que del catálogo de productos, hay muchos casos que no han presentado ventas, y eso ha generado a su vez bajas ventas en los últimos meses. Hay diferentes opciones por las que empresa puede optar para mejorar las ventas en general.

4.1. OPCIONES PARA MEJORAR VENTA DE PRODUCTOS DE MENOR DEMANDA

Los productos menos vendidos ocupan un mayor marketing para captar el interés de la clientela. El problema principal de la mayoría de los productos que no se están vendiendo son la cantidad de unidades con la que cuentan en inventario. Hay muchas unidades retenidas en algunos casos. Aunque una posibilidad es que simplemente el usuario no esté interesado, pero que tampoco aparezcan en búsquedas puede ser señal del que el usuario no sepa que esos productos estén disponibles. Para atacar este problema, se propone aumentar en específico la difusión de estos productos en los medios de la empresa. Así mismo, cuando se busquen los productos populares, agregar una sección de "También te podría interesar....^{en} el sitio web que difunda otros productos. Una mayor difusión de estos productos puede generar ventas en los próximos meses.

En los casos donde el inventario es amplio y no hay ventas, se propone una campaña de ofertas. Aunque la campaña de difusión puede aumentar las ventas de múltiples productos, en los casos de mucho inventario no es muy factible que todo se venda. Para estos casos, se propone hacer eventos donde por tiempo limitado los productos se pongan en ofertas en un rango del 10-30 %. Ya que un caso de productos que no se venden son Pantallas, podría hacerse un evento de una semana llamado "la Semana de las Pantallas", donde se encuentren ofertas de este tipo de productos. Esto permitirá aumentar las ventas más de esas unidades retenidas e ir vaciando el inventario.

4.2. OPCIONES DE CAMBIOS DE DIRECCIÓN QUE PUEDE TOMAR LA EMPRESA PARA CRECER SU MERCADO

Para solucionar los problemas en ventas y evitar que se vuelvan a presentar, hay también varias opciones a largo plazo que la empresa puede considerar para mejorar en su mercado. Una vez que se reduzca el problema del inventario retenido, la empresa puede considerar agregar o remover productos de su catálogo.

Lifestore puede reevaluar la diversificación de mercado que tiene. Actualmente, los 96 productos que maneja la empresa pertenecen a 8 categorías diferentes de electrónica. En la actualidad solo se venden 42 de esos 96 productos, que pertenecen principalmente a 4 categorías: tarjetas de vídeo, tarjetas madre, discos duros y procesadores. Sin embargo, otras áreas como pantallas o equipos de audio, no presentan ni ventas ni ingresos importantes para la empresa. Lifestore debería evaluar si son mercados que le interesen o bien, enfocarse en los 4 de mayor éxito y crecer en ese sector específico del mercado, especialmente en procesadores, donde presenta mayor éxito.

Tanto para las categorías que no se venden como las de mayores ventas, hay modelos que

deberían descontinuarse antes de agregar nuevos productos. Muchos modelos de televisiones no han tenido éxito, y al ser una categoría en la que tampoco la empresa ha tenido ventas importantes, debería reducir su variedad de productos después de liberarse del inventario. En el caso de las categorías exitosas, hay también algunos productos que no se venden como deberían. En el caso de procesadores hay un modelo particular que no se vende, y en cuanto a tarjetas de vídeo y tarjetas madre hay tanto modelos con muchas ventas y buenas valoraciones, como productos que se están devolviendo más de una ocasión. Se deben remover de catalogo productos que generen problemas por el tema de devoluciones y de ventas antes de agregar productos nuevos.

5. CONCLUSIÓN

Al analizar un conjunto de datos, es posible detectar que son múltiples los factores que hay que considerar para analizar cierto evento. Por ejemplo, la demanda, donde se consideraron el número de ventas de cada producto y el número de búsquedas que tuvieron. Ambos factores nos permiten identificar hasta qué punto un producto está captando el interés del cliente y se está comercializando como la empresa espera. Otro escenario con múltiples factores es generar una sola evaluación mediante dichos factores. Esto involucra asignar cierta importancia a cada factor, como se hizo para determinar la valoración de los productos vendidos, tomando en cuenta tanto reseñas como las devoluciones que el producto ha tenido. Al querer analizar cierta variable que se relacione a otras múltiples variables, el analista de datos debe establecer un criterio que permita la comparación y contraste de datos para una evaluación de cada caso.

La información puede dar muchas ideas al interesado de cómo afrontar cierta situación, pero todo depende de cómo se presente dicha información. El programa desarrollado, considerando su módulo de inicio de sesión y su módulo de consultas, tiene una función general de tomar la información de la empresa originalmente en tablas y desglosarla en detalles que sean más entendibles para el usuario. Un usuario no puede ver una tabla de muchas filas y ver que está mal con la empresa. En el mejor de los casos, se encontraría con algún factor, pero con mucho tiempo invertido. Es necesario traducir esa información a pocas líneas que informen al usuario del estado de sus ventas, ingresos, etc. Para mostrar esta información de manera más entendible, el desarrollador puede apoyarse de listas, tablas o gráficas, como sucedió en el reporte. Estas opciones resumen los datos a algo que el usuario puede interpretar y comprender en poco tiempo. La presentación es clave para entender el problema y las propuestas de soluciones.