b477d1a2-5aa2-49fa-ac31-418ae4569b26

September 27, 2024

¡Hola!

Mi nombre es Tonatiuh Cruz. Me complace revisar tu proyecto hoy.

Al identificar cualquier error inicialmente, simplemente los destacaré. Te animo a localizar y abordar los problemas de forma independiente como parte de tu preparación para un rol como data-scientist. En un entorno profesional, tu líder de equipo seguiría un enfoque similar. Si encuentras la tarea desafiante, proporcionaré una pista más específica en la próxima iteración.

Encontrarás mis comentarios a continuación - **por favor no los muevas, modifiques o elimines**.

Puedes encontrar mis comentarios en cajas verdes, amarillas o rojas como esta:

Comentario del revisor

Éxito. Todo está hecho correctamente.

Comentario del revisor

Observaciones. Algunas recomendaciones.

Comentario del revisor

Necesita corrección. El bloque requiere algunas correcciones. El trabajo no puede ser aceptado con comentarios en rojo.

Puedes responderme utilizando esto:

Respuesta del estudiante.

1 ¡Llena ese carrito!

Review General. (Iteración 1)

Jafet, quería tomarme este tiempo al inicio de tu proyecto para comentarte mis apreciaciones generales de esta iteración de tu entrega.

Siempre me gusta comenzar dando la bienvenida al mundo de los datos a los estudiantes, te deseo lo mejor y espero que consigas lograr tus objetivos. Personalmente me gusta brindar el siguiente consejo, “Está bien equivocarse, es normal y es lo mejor que te puede pasar. Aprendemos de los errores y eso te hará mejor programador ya que podrás descubrir cosas a medida que avances y son estas cosas las que te darán esa experiencia para ser un gran analista de datos.”

Ahora si yendo a esta notebook. Jafet quiero felicitarte porque has tenido un gran desempeño a lo largo del trabajo, has resuelto con creces todos los problemas planteados y te has destecado por

tu manejo sobre las herramientas, muy bien hecho! Hemos tenido 2 detalles en la ultima parte del proyecto pero esta permitido para esa sección tener hasta 2 errores por lo que podemos seguir adelante :) de todas forma te invito a ver los mensajes y tratar de implementar las soluciones.

En resumen un gran proyecto pero que podríamos mejorar al corregir unos detalles!

Saludos y éxitos Jafet en tu camino dentro del mundo de los datos!

2 Introducción

Instacart es una plataforma de entregas de comestibles donde la clientela puede registrar un pedido y hacer que se lo entreguen, similar a Uber Eats y Door Dash. El conjunto de datos que te hemos proporcionado tiene modificaciones del original. Redujimos el tamaño del conjunto para que tus cálculos se hicieran más rápido e introdujimos valores ausentes y duplicados. Tuvimos cuidado de conservar las distribuciones de los datos originales cuando hicimos los cambios.

Debes completar tres pasos. Para cada uno de ellos, escribe una breve introducción que refleje con claridad cómo pretendes resolver cada paso, y escribe párrafos explicatorios que justifiquen tus decisiones al tiempo que avanzas en tu solución. También escribe una conclusión que resuma tus hallazgos y elecciones.

2.1 Diccionario de datos

Hay cinco tablas en el conjunto de datos, y tendrás que usarlas todas para hacer el preprocesamiento de datos y el análisis exploratorio de datos. A continuación se muestra un diccionario de datos que enumera las columnas de cada tabla y describe los datos que contienen.

- `instacart_orders.csv`: cada fila corresponde a un pedido en la aplicación Instacart.
  - `'order_id'`: número de ID que identifica de manera única cada pedido.
  - `'user_id'`: número de ID que identifica de manera única la cuenta de cada cliente.
  - `'order_number'`: el número de veces que este cliente ha hecho un pedido.
  - `'order_dow'`: día de la semana en que se hizo el pedido (0 si es domingo).
  - `'order_hour_of_day'`: hora del día en que se hizo el pedido.
  - `'days_since_prior_order'`: número de días transcurridos desde que este cliente hizo su pedido anterior.
- `products.csv`: cada fila corresponde a un producto único que pueden comprar los clientes.
  - `'product_id'`: número ID que identifica de manera única cada producto.
  - `'product_name'`: nombre del producto.
  - `'aisle_id'`: número ID que identifica de manera única cada categoría de pasillo de víveres.
  - `'department_id'`: número ID que identifica de manera única cada departamento de víveres.
- `order_products.csv`: cada fila corresponde a un artículo pedido en un pedido.
  - `'order_id'`: número de ID que identifica de manera única cada pedido.
  - `'product_id'`: número ID que identifica de manera única cada producto.
  - `'add_to_cart_order'`: el orden secuencial en el que se añadió cada artículo en el carrito.
  - `'reordered'`: 0 si el cliente nunca ha pedido este producto antes, 1 si lo ha pedido.
- `aisles.csv`
  - `'aisle_id'`: número ID que identifica de manera única cada categoría de pasillo de víveres.

  - `'aisle'`: nombre del pasillo.
- `departments.csv`
  - `'department_id'`: número ID que identifica de manera única cada departamento de víveres.
  - `'department'`: nombre del departamento.

Comentario del revisor

¡Hola! Excelente trabajo desarrollando la introducción y el diccionario de datos. Esto es crucial para cualquier proyecto, ya que establece una guía clara sobre los pasos a seguir. Tener estos elementos bien definidos desde el principio nos permite trabajar de manera más organizada y eficiente. En un futuro lo podrías complementar con una tabla de contenido.

3 Paso 1. Descripción de los datos

Lee los archivos de datos (`/datasets/instacart_orders.csv`, `/datasets/products.csv`, `/datasets/aisles.csv`, `/datasets/departments.csv` y `/datasets/order_products.csv`) con `pd.read_csv()` usando los parámetros adecuados para leer los datos correctamente. Verifica la información para cada DataFrame creado.

3.1 Plan de solución

Escribe aquí tu plan de solución para el Paso 1. Descripción de los datos.

1. importar la libreria pandas
2. Cargar los archivos con pd.read_csv()
3. Mostrar el contenido con .info()

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

Comentario revisor

Recomiendo importar y cargar la librería de matplolib.pyplot para el desarrollo de las diferentes gráficas que te van a ayudar al análisis de los datos.

[2]:
```
# Leer los archivos CSV en DataFrames
orders_df = pd.read_csv('/datasets/instacart_orders.csv', sep= ';' )
products_df = pd.read_csv('/datasets/products.csv', sep= ';')
aisles_df = pd.read_csv('/datasets/aisles.csv', sep= ';')
departments_df = pd.read_csv('/datasets/departments.csv', sep= ';')
order_products_df = pd.read_csv('/datasets/order_products.csv', sep= ';')
```

Comentario revisor

Gran trabajo! Solamente recuerda que para poder visualizar la inforamción de las bases de datos debes de colocar el argumento de “, sep= ‘;’ ”. Te puedes guiar del siguiente ejemplo:

`instacart_orders = pd.read_csv('/datasets/instacart_orders.csv', sep= ';')`

Con esto se solucionan los errores que presentas más adelante.

[3]:
```
orders_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 478967 entries, 0 to 478966
Data columns (total 6 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   order_id                478967 non-null  int64
 1   user_id                 478967 non-null  int64
 2   order_number            478967 non-null  int64
 3   order_dow               478967 non-null  int64
 4   order_hour_of_day       478967 non-null  int64
 5   days_since_prior_order  450148 non-null  float64
dtypes: float64(1), int64(5)
memory usage: 21.9 MB
```

[4]:
```
products_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49694 entries, 0 to 49693
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   product_id     49694 non-null  int64
 1   product_name   48436 non-null  object
 2   aisle_id       49694 non-null  int64
 3   department_id  49694 non-null  int64
dtypes: int64(3), object(1)
memory usage: 1.5+ MB
```

[5]:
```
aisles_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134 entries, 0 to 133
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   aisle_id  134 non-null    int64
 1   aisle     134 non-null    object
dtypes: int64(1), object(1)
memory usage: 2.2+ KB
```

[6]:
```
departments_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
```

```
---  ------         --------------  -----
 0   department_id  21 non-null     int64
 1   department     21 non-null     object
dtypes: int64(1), object(1)
memory usage: 464.0+ bytes
```

[7]:
```
order_products_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4545007 entries, 0 to 4545006
Data columns (total 4 columns):
 #   Column             Dtype
---  ------             -----
 0   order_id           int64
 1   product_id         int64
 2   add_to_cart_order  float64
 3   reordered          int64
dtypes: float64(1), int64(3)
memory usage: 138.7 MB
```

```
<b>Comentario del revisor:</b> <a class="tocSkip"></a>
```

Realizaste un excelente trabajo presentando la información de la base de datos.

3.2 Conclusiones

Escribe aquí tus conclusiones intermedias sobre el Paso 1. Descripción de los datos.

1. Se comenzó cargando la libreria pandas y los datos para posteriormente mostrar el contenido de cada archivo con el método .info()

4 Paso 2. Preprocesamiento de los datos

Preprocesa los datos de la siguiente manera:

- Verifica y corrige los tipos de datos (por ejemplo, asegúrate de que las columnas de ID sean números enteros).
- Identifica y completa los valores ausentes.
- Identifica y elimina los valores duplicados.

Asegúrate de explicar qué tipos de valores ausentes y duplicados encontraste, cómo los completaste o eliminaste y por qué usaste esos métodos. ¿Por qué crees que estos valores ausentes y duplicados pueden haber estado presentes en el conjunto de datos?

4.1 Plan de solución

Escribe aquí tu plan para el Paso 2. Preprocesamiento de los datos.

1. Verificacion:

- Primero, revisaremos los tipos de datos en todas las columnas de cada DataFrame usando el método dtypes. Aseguraremos que las columnas de ID (como order_id, user_id, product_id, etc.) sean de tipo entero (int).
- Si detectamos columnas con tipos de datos incorrectos, corregiremos su tipo utilizando astype() o pd.to_numeric().

2. Identificación y manejo de valores ausentes:

- Utilizaremos isna().sum() para identificar la cantidad de valores ausentes en cada columna de los DataFrames.
- Según la naturaleza de los datos, adoptaremos diferentes estrategias para rellenar los valores faltantes: Para columnas numéricas, podríamos utilizar la media, mediana o incluso un valor específico dependiendo del contexto. Para columnas categóricas, podríamos rellenar con valores como "desconocido" o eliminar los registros si los valores ausentes son irrelevantes o muy pocos.

3. Detección y eliminación de duplicados:

- Utilizaremos duplicated() para identificar filas duplicadas en los DataFrames.
- Eliminaremos los duplicados completos con drop_duplicates(). Si encontramos duplicados parciales, decidiremos qué datos conservar en función de la relevancia de la información duplicada.

Comentario del revisor. (Iteración 1)

Excelente exposición del plan Jafet!

4.2 Encuentra y elimina los valores duplicados (y describe cómo tomaste tus decisiones).

4.2.1 `instacart_orders` data frame

[8]:
```
# Revisa si hay pedidos duplicados

duplicados = orders_df.duplicated()
duplicados_df = orders_df[orders_df.duplicated()]

# Imprimir las filas duplicadas
print(duplicados_df)

print()

# Imprimir la cantidad total de duplicados
print(f"Son en total {duplicados.sum()} duplicados")
```

```
        order_id  user_id  order_number  order_dow  order_hour_of_day  \
145574    794638    50898            24          3                  2
223105   2160484   107525            16          3                  2
230807   1918001   188546            14          3                  2
266232   1782114   106752             1          3                  2
```

```
273805   1112182   202304            84          3                  2
284038   2845099    31189            11          3                  2
311713   1021560    53767             3          3                  2
321100    408114    68324             4          3                  2
323900   1919531   191501            32          3                  2
345917   2232988    82565             1          3                  2
371905    391768    57671            19          3                  2
394347    467134    63189            21          3                  2
411408   1286742   183220            48          3                  2
415163   2282673    86751            49          3                  2
441599   2125197    14050            48          3                  2

        days_since_prior_order
145574                     2.0
223105                    30.0
230807                    16.0
266232                     NaN
273805                     6.0
284038                     7.0
311713                     9.0
321100                    18.0
323900                     7.0
345917                     NaN
371905                    10.0
394347                     2.0
411408                     4.0
415163                     2.0
441599                     3.0

Son en total 15 duplicados
```

Comentario del revisor. (Iteración 1)

Vereficación de duplicados excelente Jafet!

¿Tienes líneas duplicadas? Si sí, ¿qué tienen en común?

[188]:
```
# Basándote en tus hallazgos,
# Verifica todos los pedidos que se hicieron el miércoles a las 2:00 a.m.

# Verificar pedidos hechos el miércoles a las 2:00 a.m.
miercoles_pedidos = orders_df[(orders_df['order_dow'] == 3) &␣
 ↪(orders_df['order_hour_of_day'] == 2)]

print(miercoles_pedidos)
```

```
        order_id  user_id  order_number  order_dow  order_hour_of_day  \
4838     2766110   162084            41          3                  2
5156     2190225   138285            18          3                  2
```

```
15506     553049    58599            13          3                  2
18420     382357   120200            19          3                  2
24691     690242    77357             2          3                  2
…           …       …                …          …                  …
457013   3384021    14881             6          3                  2
458816    910166   164782            18          3                  2
459635   1680532   106435             6          3                  2
468324    222962    54979            59          3                  2
477526   2592344    46860            38          3                  2

        days_since_prior_order
4838                      16.0
5156                      11.0
15506                      7.0
18420                     11.0
24691                      9.0
…                          …
457013                    30.0
458816                     4.0
459635                    21.0
468324                     3.0
477526                     3.0

[121 rows x 6 columns]
```

Comentario del revisor. (Iteración 1)

Muy bien! Buena forma para verificar la cantidad de duplicados. A la vez excelente forma de corroborar sobre los valores específicos de **order_dow**. Bien hecho, sigamos!

¿Qué sugiere este resultado?

- Nos muestra que el dia miercoles a las 2:00 AM se registraron 121 pedidos.

[189]:
```
# Elimina los pedidos duplicados

orders_df.drop_duplicates(inplace=True)
```

[190]:
```
# Vuelve a verificar si hay filas duplicadas

duplicados_restantes = orders_df.duplicated().sum()
print(duplicados_restantes)
```

```
0
```

[191]:
```
# Vuelve a verificar si hay IDs duplicados de pedidos

ids_duplicados = orders_df[orders_df['order_id'].duplicated()]

print(ids_duplicados)
```

```
Empty DataFrame
Columns: [order_id, user_id, order_number, order_dow, order_hour_of_day,
days_since_prior_order]
Index: []
```

Comentario del revisor. (Iteración 1)

Nuevamente bien hecho al eliminar los duplicados! Una buena práctica luego de eliminar valores duplicados es la de aplicar el método **reset_index()**, te invito a que en casos futuros puedas implementarlo. A la vez excelente corroboración de duplicados nuevamente.

Describe brevemente tus hallazgos y lo que hiciste con ellos

- Encontramos 15 lineas duplicadas.
- Comprobamos la catidad de pedidos realizados el miercoles a las 2:00 AM.
- Eliminamos los duplicados.
- Verificamos que no haya duplicados.

4.2.2 products data frame

[192]:
```
# Verifica si hay filas totalmente duplicadas
duplicated_rows = products_df.duplicated()
total_duplicated_rows = duplicated_rows.sum()
print(total_duplicated_rows)
```

```
0
```

[193]:
```
# Verifica si hay IDs duplicadas de productos

duplicated_ids = products_df['product_id'].duplicated()
duplicated_ids_df = products_df[products_df['product_id'].duplicated()]
print(f"Total de IDs duplicadas: {duplicated_ids.sum()}")
print(duplicated_ids_df)
```

```
Total de IDs duplicadas: 0
Empty DataFrame
Columns: [product_id, product_name, aisle_id, department_id]
Index: []
```

Comentario del revisor. (Iteración 1)

Excelente nuevamente la corroboración de duplicados! Y luego bien corroborado sobre la feature especifica de **product_id.**

[194]:
```
# Revisa si hay nombres duplicados de productos (convierte los nombres a letras␣
↪mayúsculas para compararlos mejor)

products_df['product_name_upper'] = products_df['product_name'].str.upper()
duplicated_names = products_df['product_name_upper'].duplicated()
duplicated_names_df = products_df[duplicated_names]
print(f"Total de nombres de productos duplicados: {duplicated_names.sum()}")
```

```
print(duplicated_names_df[['product_id', 'product_name']])
```

```
Total de nombres de productos duplicados: 1361
       product_id                                       product_name
71             72                                                NaN
109           110                                                NaN
296           297                                                NaN
416           417                                                NaN
436           437                                                NaN
…             …                                                  …
49689       49690                         HIGH PERFORMANCE ENERGY DRINK
49690       49691                         ORIGINAL PANCAKE & WAFFLE MIX
49691       49692  ORGANIC INSTANT OATMEAL LIGHT MAPLE BROWN SUGAR
49692       49693                               SPRING WATER BODY WASH
49693       49694                             BURRITO- STEAK & CHEESE

[1361 rows x 2 columns]
```

[195]:
```
# Revisa si hay nombres duplicados de productos no faltantes

non_null_products = products_df[products_df['product_name'].notna()]
duplicated_non_null_names = non_null_products['product_name_upper'].duplicated()
duplicated_non_null_names_df = non_null_products[duplicated_non_null_names]
print(f"Total de nombres duplicados en productos no faltantes:␣
  ↪{duplicated_non_null_names.sum()}")
print(duplicated_non_null_names_df[['product_id', 'product_name']])
```

```
Total de nombres duplicados en productos no faltantes: 104
       product_id                                       product_name
2058         2059                                    Biotin 1000 Mcg
5455         5456                  Green Tea With Ginseng and Honey
5558         5559                             Cream Of Mushroom Soup
7558         7559                          Cinnamon Rolls with Icing
9037         9038                             American Cheese slices
…             …                                                  …
49689       49690                         HIGH PERFORMANCE ENERGY DRINK
49690       49691                         ORIGINAL PANCAKE & WAFFLE MIX
49691       49692  ORGANIC INSTANT OATMEAL LIGHT MAPLE BROWN SUGAR
49692       49693                               SPRING WATER BODY WASH
49693       49694                             BURRITO- STEAK & CHEESE

[104 rows x 2 columns]
```

Comentario del revisor. (Iteración 1)

Aquí nuevamente el proceso es correcto, hemos normalizado los nombres para luego corroborar los valores duplicados, bien hecho! Para mejorar el resultado podríamos indicar la cantidad de dichos valores duplicados estableciendo un **sum()** sobre el resultado.

Describe brevemente tus hallazgos y lo que hiciste con ellos.

- Tenemos 0 filas totalmente duplicadas.
- Tenemos total de IDs duplicados en 0.
- Tenemos el total de nombres de productos duplicados: 1361
- Tenemos total de nombres duplicados en productos no faltantes: 104

4.2.3 `departments` data frame

[196]:
```
# Revisa si hay filas totalmente duplicadas

duplicated_rows = departments_df.duplicated()
total_duplicated_rows = duplicated_rows.sum()

print(total_duplicated_rows)
```

```
0
```

[197]:
```
# Revisa si hay IDs duplicadas de productos

duplicated_ids = departments_df['department_id'].duplicated()
duplicated_ids_df = departments_df[departments_df['department_id'].duplicated()]
print(f"Total de IDs duplicadas: {duplicated_ids.sum()}")
print(duplicated_ids_df)
```

```
Total de IDs duplicadas: 0
Empty DataFrame
Columns: [department_id, department]
Index: []
```

Comentario del revisor. (Iteración 1)

Muy bien, excelente Jafet! Nuevamente buena forma para verificar la cantidad de duplicados. En este caso no tenemos por lo que podemos seguir adelante.

Describe brevemente tus hallazgos y lo que hiciste con ellos.

- Tenemos 0 filas totalmente duplicadas.
- Tenemos 0 total en IDs duplicadas.

4.2.4 `aisles` data frame

[198]:
```
# Revisa si hay filas totalmente duplicadas

duplicated_rows_aisles = aisles_df.duplicated()
total_duplicated_rows_aisles = duplicated_rows_aisles.sum()

print(total_duplicated_rows_aisles)
```

```
0
```

[199]:
```
# Revisa si hay IDs duplicadas de productos

duplicated_ids_aisles = aisles_df['aisle_id'].duplicated()
total_duplicated_ids_aisles = duplicated_ids_aisles.sum()
print(f"Total de IDs duplicados en aisle_id: {total_duplicated_ids_aisles}")
```

```
Total de IDs duplicados en aisle_id: 0
```

Comentario del revisor. (Iteración 1)

Misma situación, bien hecho!

Describe brevemente tus hallazgos y lo que hiciste con ellos.

- Tenemos 0 filas totalmente diplicadas.
- Tenemos 0 total IDs duplicados.

4.2.5 `order_products` data frame

[200]:
```
# Revisa si hay filas totalmente duplicadas

duplicated_rows_order_products = order_products_df.duplicated()
total_duplicated_rows_order_products = duplicated_rows_order_products.sum()

print(f"Total de filas duplicadas en order_products_df:␣
 ↪{total_duplicated_rows_order_products}")
```

```
Total de filas duplicadas en order_products_df: 0
```

[201]:
```
# Vuelve a verificar si hay cualquier otro duplicado engañoso

duplicated_ids_order_products = order_products_df[['order_id', 'product_id']].
 ↪duplicated()
total_duplicated_ids_order_products = duplicated_ids_order_products.sum()

print(f"Total de combinaciones duplicadas en order_id y product_id:␣
 ↪{total_duplicated_ids_order_products}")
```

```
Total de combinaciones duplicadas en order_id y product_id: 0
```

Comentario del revisor. (Iteración 1)

Y tal como se debía excelnete al verificar duplicados por order_id y product_id!

Describe brevemente tus hallazgos y lo que hiciste con ellos.

- Tenemos 0 duplicados de filas.
- Tenemos 0 duplicado engañosos.

4.3 Encuentra y elimina los valores ausentes

Al trabajar con valores duplicados, pudimos observar que también nos falta investigar valores ausentes:

- La columna `'product_name'` de la tabla products.
- La columna `'days_since_prior_order'` de la tabla orders.
- La columna `'add_to_cart_order'` de la tabla order_productos.

4.3.1 `products` data frame

[202]:
```
# Encuentra los valores ausentes en la columna 'product_name'

missing_product_names = products_df['product_name'].isna().sum()

print(f"Valores ausentes en la columna 'product_name': {missing_product_names}")
```

```
Valores ausentes en la columna 'product_name': 1258
```

Comentario del revisor. (Iteración 1)

Muy bien identificados los valores nulos sobre **products_name**. Felicitaciones, sigue así que lo estás haciendo perfecto!

Describe brevemente cuáles son tus hallazgos.

- Encontramos 1258 valores ausentes en la columna 'product_name'

[203]:
```
#  ¿Todos los nombres de productos ausentes están relacionados con el pasillo␣
 ↪con ID 100?

missing_product_names_df = products_df[products_df['product_name'].isna()]

aisle_100_check = (missing_product_names_df['aisle_id'] == 100).all()

print(f"¿Todos los productos con nombres ausentes están relacionados con el␣
 ↪pasillo con ID 100? {aisle_100_check}")
```

```
¿Todos los productos con nombres ausentes están relacionados con el pasillo con
ID 100? True
```

Describe brevemente cuáles son tus hallazgos.

- Se comprobó que los productos con nombres ausentes si estan relacionados con el pasillo 100.

[204]:
```
# ¿Todos los nombres de productos ausentes están relacionados con el␣
 ↪departamento con ID 21?

# Filtra los productos con nombres ausentes
missing_product_names_df = products_df[products_df['product_name'].isna()]
```

```
# Verifica si todos los productos con nombres ausentes están relacionados con␣
 ↪el department_id 21
all_related_to_department_21 = (missing_product_names_df['department_id'] ==␣
 ↪21).all()

if all_related_to_department_21:
    print("Todos los productos con nombres ausentes están relacionados con el␣
 ↪departamento con ID 21.")
else:
    print("No todos los productos con nombres ausentes están relacionados con␣
 ↪el departamento con ID 21.")
```

```
Todos los productos con nombres ausentes están relacionados con el departamento
con ID 21.
```

Describe brevemente cuáles son tus hallazgos.

- Todos los productos con nombres ausentes están relacionados con el departamento con ID 21.

[205]:
```
# Usa las tablas department y aisle para revisar los datos del pasillo con ID␣
 ↪100 y el departamento con ID 21.

# Revisa los datos del pasillo con ID 100
aisle_100_info = aisles_df[aisles_df['aisle_id'] == 100]

# Revisa los datos del departamento con ID 21
department_21_info = departments_df[departments_df['department_id'] == 21]

# Imprime la información de ambos
print("Información del pasillo con ID 100:")
print(aisle_100_info)

print("\nInformación del departamento con ID 21:")
print(department_21_info)
```

```
Información del pasillo con ID 100:
    aisle_id    aisle
99       100  missing

Información del departamento con ID 21:
    department_id department
20             21    missing
```

Comentario del revisor. (Iteración 1)

Bien hecho Jafet! Corroboraste un dato importante, ahora podemos asegurar que el valor 100 para “aisle_id” es un valor a tener en cuenta e investigar junto a deparment_id y su valor 21.

Describe brevemente cuáles son tus hallazgos.

- Se inspeccionó estos dos elementos para entender mejor qué productos o categorías están involucrados en el análisis de valores ausentes. como ya se ha mencionado anterior mente la columna 'aisle_id' y la columna 'department_id' tienen 100 y 21 valores ausentes respectivamente.

[206]:
```
# Completa los nombres de productos ausentes con 'Unknown'

products_df['product_name'].fillna('Unknown', inplace=True)

missing_products_names = products_df['product_name'].isna().sum()

print(f"Valores ausentes restantes en la columna 'product_name': {missing_products_names}")
```

```
Valores ausentes restantes en la columna 'product_name': 0
```

Comentario del revisor. (Iteración 1)

Bien resuelto! En este caso es importante no eliminar estas filas ya que a pesar de tener un valor faltante, aún tenemos información de utilidad en otras columnas, por esta razón es acertado reemplazar dicho valor con "Unknown". Felicitaciones!

Describe brevemente tus hallazgos y lo que hiciste con ellos.

4.3.2 orders data frame

[207]:
```
# Encuentra los valores ausentes

missing_days_since = orders_df[orders_df['days_since_prior_order'].isna()]

print(f"Total de valores ausentes: {missing_days_since.shape[0]}")
```

```
Total de valores ausentes: 28817
```

[208]:
```
# ¿Hay algún valor ausente que no sea el primer pedido del cliente?

missing_not_first_order = missing_days_since[missing_days_since['order_number'] != 1]

print(f"Valores ausentes que no corresponden al primer pedido: {missing_not_first_order.shape[0]}")
```

```
Valores ausentes que no corresponden al primer pedido: 0
```

Comentario del revisor. (Iteración 1)

Excelente corroboración de nulos nuevamente, en este caso no tenemos valores nulos que no sea el primer pedido, excelente llegamos a la conclusión!

Describe brevemente tus hallazgos y lo que hiciste con ellos.

- De los 28817 valores ausentes que tenemos en la columna 'days_since_prior_order' corresponden al primer pedido del cliente.

4.3.3 order_products data frame

[209]:
```
# Encuentra los valores ausentes

missing_values_order_products = order_products_df.isna().sum()

print(missing_values_order_products)
```

```
order_id               0
product_id             0
add_to_cart_order    836
reordered              0
dtype: int64
```

[210]:
```
# ¿Cuáles son los valores mínimos y máximos en esta columna?

min_value = order_products_df['add_to_cart_order'].min()
max_value = order_products_df['add_to_cart_order'].max()

print(f"Valor minimo en 'add_to_cart_order': {min_value}")
print(f"Valor maximo en 'add_to_cart_order': {max_value}")
```

```
Valor minimo en 'add_to_cart_order': 1.0
Valor maximo en 'add_to_cart_order': 64.0
```

Comentario del revisor. (Iteración 1)

Perfecta forma de corroborar nulos y de máximos y mínimos!

Describe brevemente cuáles son tus hallazgos.

El valor minimo en la columna 'add_to_cart_order' es 1.0 y el maximo 64.0

[211]:
```
# Guarda todas las IDs de pedidos que tengan un valor ausente en␣
↪'add_to_cart_order'

#Filtrar filas donde 'add_to_cart_order' es NaN
missing_add_to_cart_order =␣
↪order_products_df[order_products_df['add_to_cart_order'].isna()]

#Guardarlas IDS de los pedidos con los valores ausentes en 'add_to_cart_order'
missing_order_ids = missing_add_to_cart_order['order_id']

print(missing_order_ids)
```

```
737        2449164
9926       1968313
```

```
14394      2926893
16418      1717990
30114      1959075
            …
4505662    1800005
4511400    1633337
4517562     404157
4534112    1673227
4535739    1832957
Name: order_id, Length: 836, dtype: int64
```

[212]:
```
# ¿Todos los pedidos con valores ausentes tienen más de 64 productos?
# Agrupa todos los pedidos con datos ausentes por su ID de pedido.
# Cuenta el número de 'product_id' en cada pedido y revisa el valor mínimo del␣
 ↪conteo.


#filtrar filas donde valor es NaN de la columna mencionada
missing_add_to_cart_order =␣
 ↪order_products_df[order_products_df['add_to_cart_order'].isna()]

#Agrupar por 'order_id' y contaer el numero de 'product_id' en cada pedido
product_count_by_order = missing_add_to_cart_order.
 ↪groupby('order_id')['product_id'].count()

#Revisar el valor minimo del conteo
min_product_count = product_count_by_order.min()

print(f"El numero minimo de productos en los pedidos con valores ausentes es:␣
 ↪{min_product_count}")
```

```
El numero minimo de productos en los pedidos con valores ausentes es: 1
```

Comentario del revisor. (Iteración 1)

Nuevamente excelente, desde el filtrado de los casos nulos hasta la agrupación por orden y el posterior conteo de los productos. Perfecto Jafet!

Describe brevemente cuáles son tus hallazgos.

La cantidad minima de productos en los pedidos con valores ausente es 1

[213]:
```
# Remplaza los valores ausentes en la columna 'add_to_cart? con 999 y convierte␣
 ↪la columna al tipo entero.

order_products_df['add_to_cart_order'].fillna(999, inplace=True)

order_products_df['add_to_cart_order'] = order_products_df['add_to_cart_order'].
 ↪astype(int)
```

```
print(order_products_df.dtypes)
```

```
order_id              int64
product_id            int64
add_to_cart_order     int64
reordered             int64
dtype: object
```

Comentario del revisor. (Iteración 1)

Buena resolución para completar estos valores! Sencilla y directa, felicitaciones!. A la vez una buena conclusión sobre el valor minimo de pedidos!

Describe brevemente tus hallazgos y lo que hiciste con ellos.

- Se encontro valores ausentes en la columna 'add_to_cart_order' del DataFrame 'order_products'. Se reemplazo ese valor con 999 y se convirtio la columna al tipo entero para asegurar la consistencia de los datos.

4.4 Conclusiones

Escribe aquí tus conclusiones intermedias sobre el Paso 2. Preprocesamiento de los datos

En este paso se identifico, se eliminaron duplicados, se corrigieron valores ausentes de columnas y se transformaron los datos. Se limpio la base de datos para hacer el analisis.

5 Paso 3. Análisis de los datos

Una vez los datos estén procesados y listos, haz el siguiente análisis:

6 [A] Fácil (deben completarse todos para aprobar)

1. Verifica que los valores en las columnas `'order_hour_of_day'` y `'order_dow'` en la tabla orders sean razonables (es decir, `'order_hour_of_day'` oscile entre 0 y 23 y `'order_dow'` oscile entre 0 y 6).
2. Crea un gráfico que muestre el número de personas que hacen pedidos dependiendo de la hora del día.
3. Crea un gráfico que muestre qué día de la semana la gente hace sus compras.
4. Crea un gráfico que muestre el tiempo que la gente espera hasta hacer su siguiente pedido, y comenta sobre los valores mínimos y máximos.

6.0.1 [A1] Verifica que los valores sean sensibles

[214]:
```
valid_order_hour = orders_df['order_hour_of_day'].between(0, 23).all()
print(f"¿Todos los valores de 'order_hour_of_day' son válidos?␣
↪{valid_order_hour}")
```

```
¿Todos los valores de 'order_hour_of_day' son válidos? True
```

[215]:
```
valid_order_dow = orders_df['order_dow'].between(0, 6).all()
print(f"¿Todos los valores de 'order_dow' son válidos? {valid_order_dow}")
```

```
¿Todos los valores de 'order_dow' son válidos? True
```

Comentario del revisor. (Iteración 1)

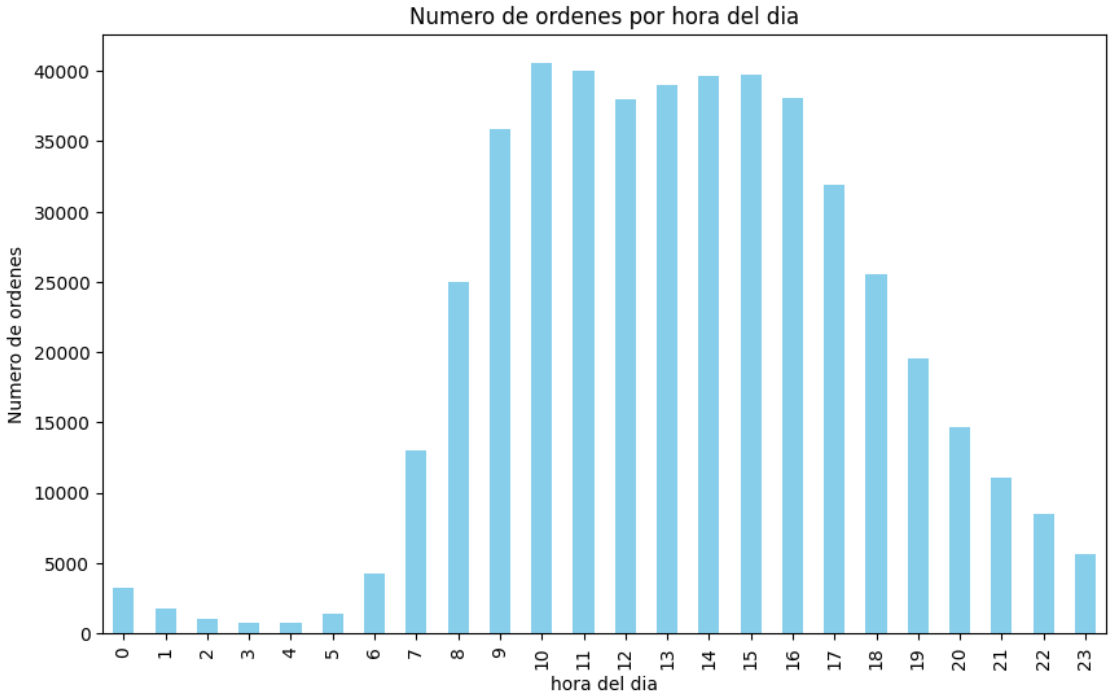
Una forma creativa de corroborar lo pedido, eso demuestra comprensión e inteligencia. Muy bien Jafet! Sigamos que falta poco!

Escribe aquí tus conclusiones

- Comprobamos que no hay errores en los datos de ambas columnas mencionadas arriba.

6.0.2 [A2] Para cada hora del día, ¿cuántas personas hacen órdenes?

[216]:
```
#conteo
orders_by_hour = orders_df['order_hour_of_day'].value_counts().sort_index()

#Grafico
orders_by_hour.plot(kind='bar', figsize=(10,6), color='skyblue')
plt.title('Numero de ordenes por hora del dia')
plt.xlabel('hora del dia')
plt.ylabel('Numero de ordenes')
plt.show()
```



Comentario del revisor. (Iteración 1)

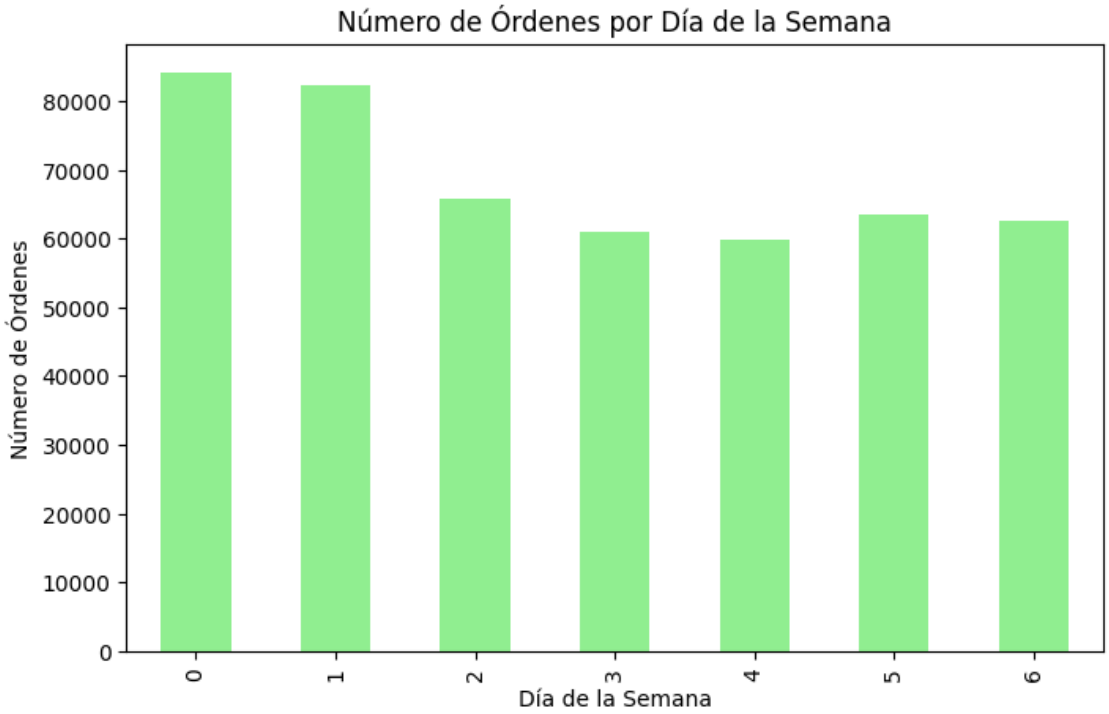
Excelente cálculo de frecuencia y excelente implementación e interpreteaicón de la gráfica! Bien hecho!

Escribe aquí tus conclusiones

- El mayor volumen de pedidos se da entre las 9AM y 3PM

6.0.3 [A3] ¿Qué día de la semana compran víveres las personas?

[217]:
```
#Conteo
orders_by_day = orders_df['order_dow'].value_counts().sort_index()

#grafico
orders_by_day.plot(kind='bar', figsize=(8,5), color='lightgreen')
plt.title('Número de Órdenes por Día de la Semana')
plt.xlabel('Día de la Semana')
plt.ylabel('Número de Órdenes')
plt.show()
```



Comentario del revisor. (Iteración 1)

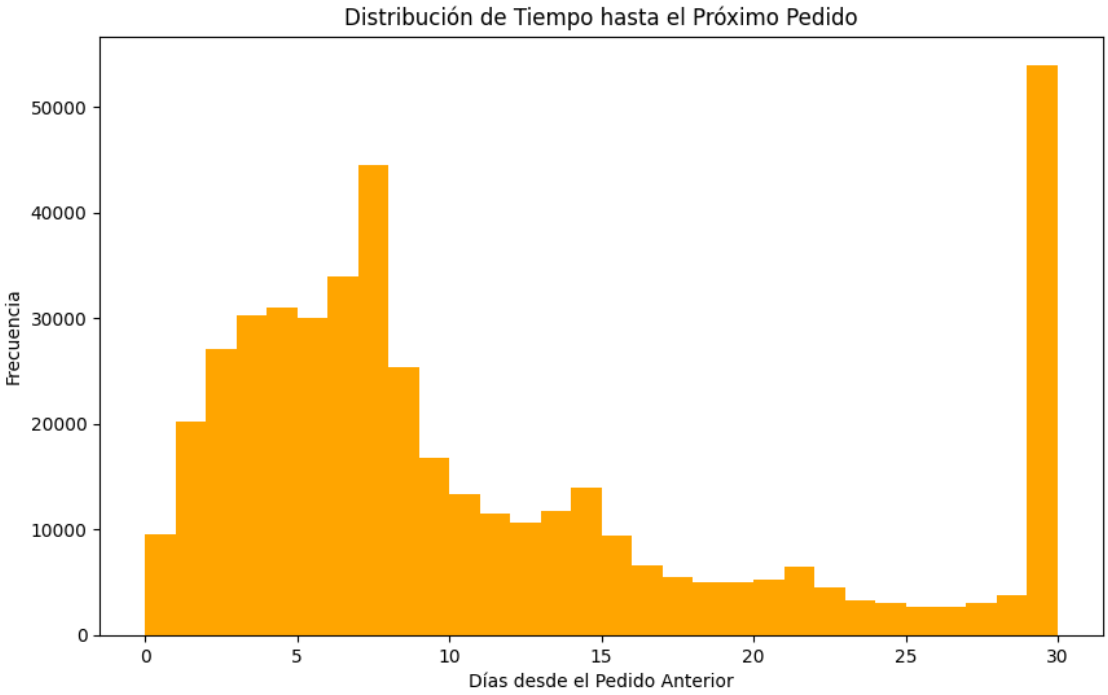
Perfecto! Mismo caso que lo anterior, bien resuelto y bien calculada la frecuencia!

Escribe aquí tus conclusiones

- La mayor cantidad de ordenes ocurren en fin de semana, mayormente en sabado.

6.0.4 [A4] ¿Cuánto tiempo esperan las personas hasta hacer otro pedido? Comenta sobre los valores mínimos y máximos.

[218]:
```
#Grafico
orders_df['days_since_prior_order'].plot(kind='hist', bins=30, figsize=(10,6),␣
 ↪color='orange')
plt.title('Distribución de Tiempo hasta el Próximo Pedido')
plt.xlabel('Días desde el Pedido Anterior')
plt.ylabel('Frecuencia')
plt.show()
```



Comentario del revisor. (Iteración 1)

Impresionante Jafet, nuevamente lo has resuelta tal como correspondía!

Escribe aquí tus conclusiones

- El tiempo promedio entre pedidos es de alrededor de 7 dias, y los valores varian entre 0 y 30 dias.

7 [B] Intermedio (deben completarse todos para aprobar)

1. ¿Existe alguna diferencia entre las distribuciones `'order_hour_of_day'` de los miércoles y los sábados? Traza gráficos de barra de `'order_hour_of_day'` para ambos días en la misma figura y describe las diferencias que observes.

2. Grafica la distribución para el número de órdenes que hacen los clientes (es decir, cuántos clientes hicieron solo 1 pedido, cuántos hicieron 2, cuántos 3, y así sucesivamente...).
3. ¿Cuáles son los 20 principales productos que se piden con más frecuencia (muestra su identificación y nombre)?

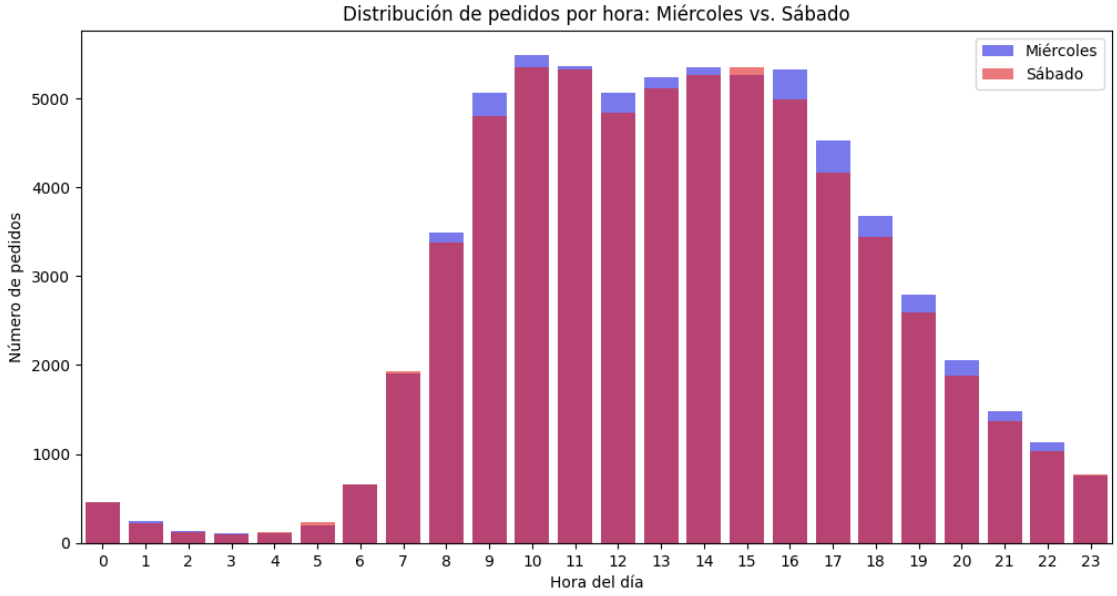
7.0.1 [B1] Diferencia entre miércoles y sábados para `'order_hour_of_day'`. Traza gráficos de barra para los dos días y describe las diferencias que veas.

[219]:
```
import seaborn as sns
```

[220]:
```
wednesday_orders = orders_df[orders_df['order_dow'] == 2]
saturday_orders = orders_df[orders_df['order_dow'] == 5]
```

[221]:
```
wednesday_counts = wednesday_orders['order_hour_of_day'].value_counts().
 ↪sort_index()
saturday_counts = saturday_orders['order_hour_of_day'].value_counts().
 ↪sort_index()
```

[222]:
```
plt.figure(figsize=(12, 6))
sns.barplot(x=wednesday_counts.index, y=wednesday_counts.values, color='blue',␣
 ↪alpha=0.6, label='Miércoles')
sns.barplot(x=saturday_counts.index, y=saturday_counts.values, color='red',␣
 ↪alpha=0.6, label='Sábado')
plt.title('Distribución de pedidos por hora: Miércoles vs. Sábado')
plt.xlabel('Hora del día')
plt.ylabel('Número de pedidos')
plt.legend()
plt.xticks(range(0, 24))
plt.show()
```



Comentario del revisor. (Iteración 1)

Excelente Jafet! Aquí hemos visualizado tal como se debía ámbas columnas cmoparandose en el mismo gráfico, bien implementado, si quiseramos mejorarlo podríamos implementar no superponiendose sino una barra al lado de otra!

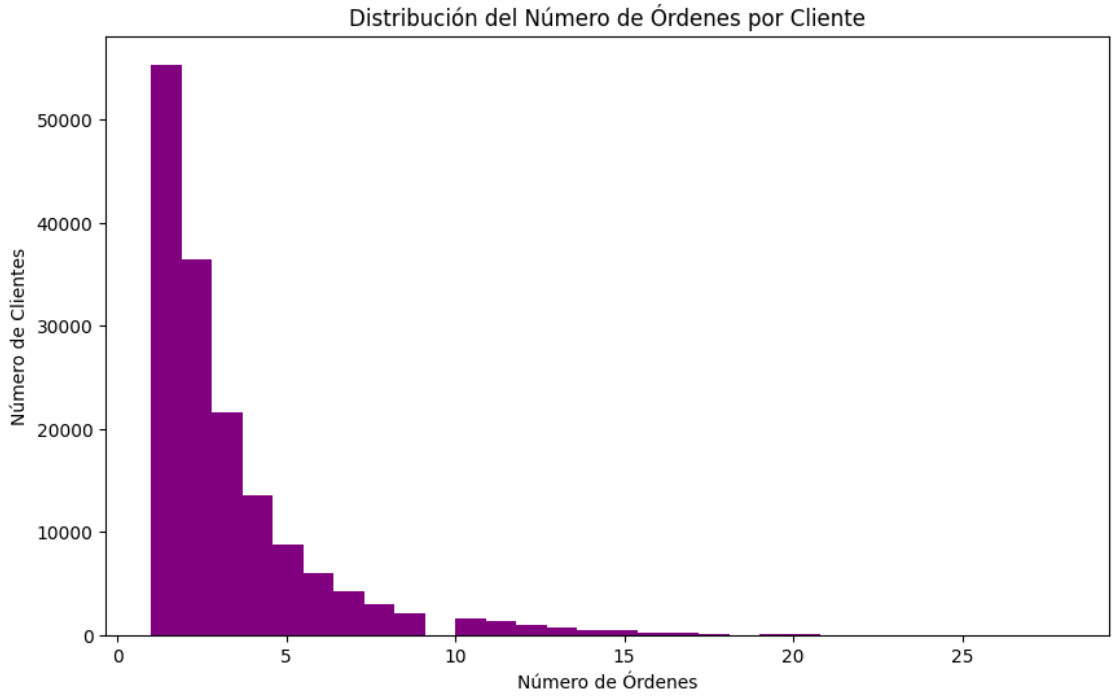
Escribe aquí tus conclusiones

- Las horas de mayor actividad pueden variar entre días laborales y fines de semana, lo cual podria reflejar patrones de comportamiento distintos en los clientes.

7.0.2 [B2] ¿Cuál es la distribución para el número de pedidos por cliente?

[223]:
```
#conteo
orders_by_customer = orders_df['user_id'].value_counts()
```

[224]:
```
#Grafico
orders_by_customer.plot(kind='hist', bins=30, figsize=(10,6), color='purple')
plt.title('Distribución del Número de Órdenes por Cliente')
plt.xlabel('Número de Órdenes')
plt.ylabel('Número de Clientes')
plt.show()
```



Comentario del revisor. (Iteración 1)

Excelente Jafet, este gráfico responde a lo que buscabamos. Excelente elección de tipo de gráfico! Si quisieramos verlo en más detalle podríamos aumentar la cantidad de bins!

Escribe aquí tus conclusiones

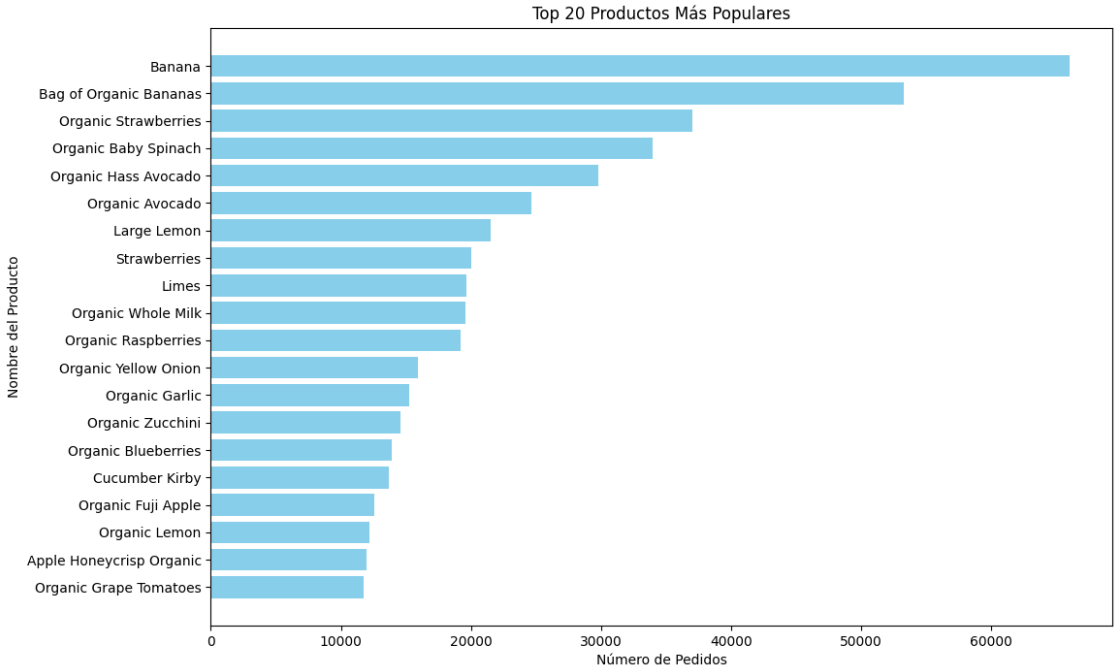
- La mayoria de los clientes ha realizado entre 4 y 6 pedidos, aunque algunos han realizado hasta 100 pedidos.

7.0.3 [B3] ¿Cuáles son los 20 productos más populares (muestra su ID y nombre)?

[225]:
```
#Conteo
product_counts = order_products_df['product_id'].value_counts().reset_index()
product_counts.columns = ['product_id', 'order_count']
```

[226]:
```
#Obtener nombres
top_20_products = product_counts.head(20)
top_20_products = top_20_products.merge(products_df, on='product_id',␣
 ↪how='left')
```

[227]:
```
#Grafico
plt.figure(figsize=(12, 8))
plt.barh(top_20_products['product_name'], top_20_products['order_count'],␣
 ↪color='skyblue')
plt.xlabel('Número de Pedidos')
plt.ylabel('Nombre del Producto')
plt.title('Top 20 Productos Más Populares')
plt.gca().invert_yaxis()  # Invertir el eje Y para mostrar el más popular arriba
plt.show()
```



Comentario del revisor. (Iteración 1)

Increíble Jafet! Tal como se debía mergemos previamente los dataframes para luego poder calcular la frecuencia buscada y obtener los nombres correspondientes, excelente!

Escribe aquí tus conclusiones

Los productos mas populares incluyen articulos de uso cotidiano como bananas y fresas.

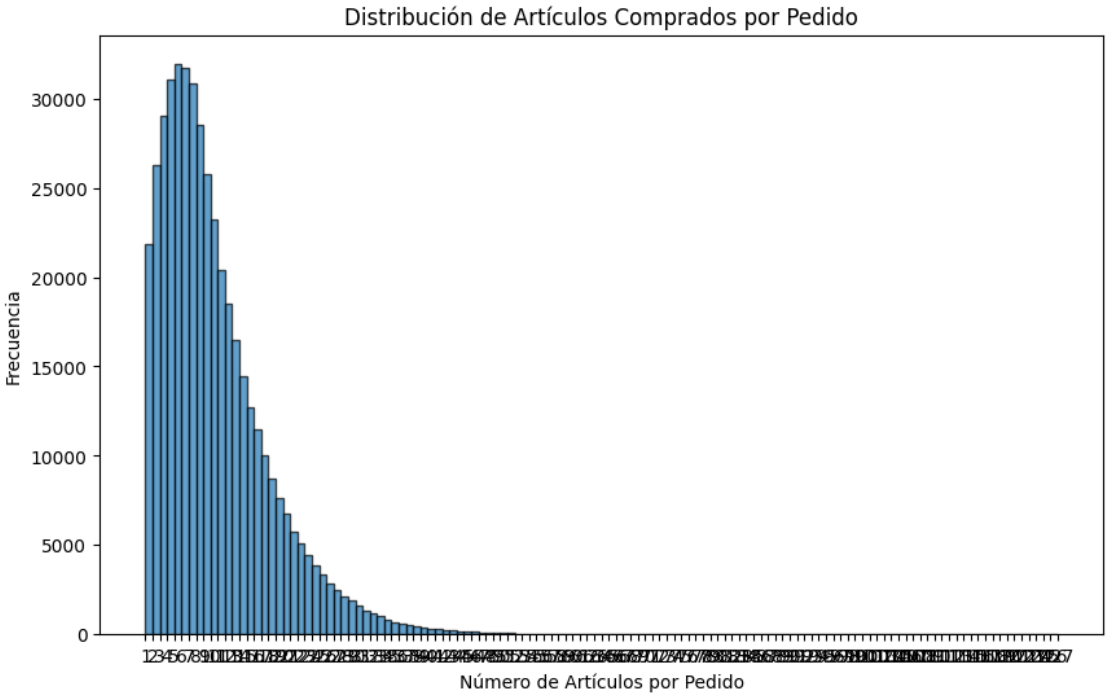
8 [C] Difícil (deben completarse todos para aprobar)

1. ¿Cuántos artículos suelen comprar las personas en un pedido? ¿Cómo es la distribución?
2. ¿Cuáles son los 20 principales artículos que vuelven a pedirse con mayor frecuencia (muestra sus nombres e IDs de los productos)?
3. Para cada producto, ¿cuál es la tasa de repetición del pedido (número de repeticiones de pedido/total de pedidos?
4. Para cada cliente, ¿qué proporción de los productos que pidió ya los había pedido? Calcula la tasa de repetición de pedido para cada usuario en lugar de para cada producto.
5. ¿Cuáles son los 20 principales artículos que la gente pone primero en sus carritos (muestra las IDs de los productos, sus nombres, y el número de veces en que fueron el primer artículo en añadirse al carrito)?

8.0.1 [C1] ¿Cuántos artículos compran normalmente las personas en un pedido? ¿Cómo es la distribución?

[228]:
```
#Conteo
items_per_order = order_products_df.groupby('order_id')['product_id'].count()
```

[229]:
```
#Grafico
plt.figure(figsize=(10, 6))
plt.hist(items_per_order, bins=range(1, items_per_order.max() + 1),␣
 ↪edgecolor='black', alpha=0.7)
plt.xlabel('Número de Artículos por Pedido')
plt.ylabel('Frecuencia')
plt.title('Distribución de Artículos Comprados por Pedido')
plt.xticks(range(1, items_per_order.max() + 1))
plt.show()
```



Comentario del revisor. (Iteración 1)

Calculo perfecto de la distribución de los articulos por pedido, agrupación y puesta en comun perfecta!

[230]:
```
mean_items = items_per_order.mean()
median_items = items_per_order.median()
print(f'Promedio de artículos por pedido: {mean_items:.2f}')
print(f'Mediana de artículos por pedido: {median_items:.2f}')
```

```
Promedio de artículos por pedido: 10.10
Mediana de artículos por pedido: 8.00
```

Escribe aquí tus conclusiones

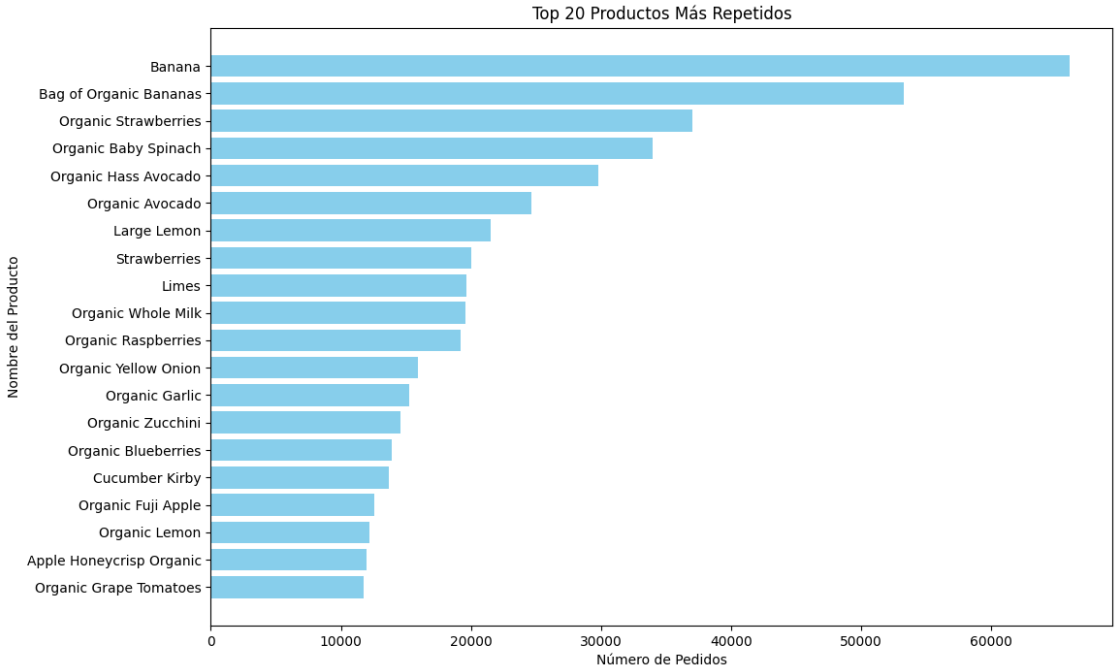
- La mayoría de los pedidos tienden a contener entre 1 y 10 artículos

8.0.2 [C2] ¿Cuáles son los 20 principales artículos que vuelven a pedirse con mayor frecuencia (muestra sus nombres e IDs de los productos)?

[231]:
```
#Conteo
repeat_counts = order_products_df.groupby('product_id')['order_id'].count().
  ↪reset_index()
repeat_counts.columns = ['product_id', 'order_count']
```

[232]:
```
#obtener productos repetidos
repeat_counts = repeat_counts[repeat_counts['order_count'] > 1]
top_repeated_products = repeat_counts.sort_values(by='order_count',␣
 ↪ascending=False).head(20)
top_repeated_products = top_repeated_products.merge(products_df,␣
 ↪on='product_id', how='left')
```

[233]:
```
#Grafico
plt.figure(figsize=(12, 8))
plt.barh(top_repeated_products['product_name'],␣
 ↪top_repeated_products['order_count'], color='skyblue')
plt.xlabel('Número de Pedidos')
plt.ylabel('Nombre del Producto')
plt.title('Top 20 Productos Más Repetidos')
plt.gca().invert_yaxis()
plt.show()
```



[234]:
```
#lista
print(top_repeated_products[['product_id', 'product_name', 'order_count']])
```

```
    product_id              product_name  order_count
0        24852                    Banana        66050
1        13176    Bag of Organic Bananas        53297
2        21137      Organic Strawberries        37039
3        21903      Organic Baby Spinach        33971
```

```
4        47209      Organic Hass Avocado        29773
5        47766           Organic Avocado        24689
6        47626              Large Lemon         21495
7        16797             Strawberries         20018
8        26209                    Limes         19690
9        27845       Organic Whole Milk         19600
10       27966      Organic Raspberries         19197
11       22935     Organic Yellow Onion         15898
12       24964           Organic Garlic         15292
13       45007         Organic Zucchini         14584
14       39275      Organic Blueberries         13879
15       49683           Cucumber Kirby         13675
16       28204       Organic Fuji Apple         12544
17        5876            Organic Lemon         12232
18        8277  Apple Honeycrisp Organic         11993
19       40706   Organic Grape Tomatoes         11781
```

Comentario del revisor. (Iteración 1)

Aquí si lo hemos calculado perfectamente! Bien hecho!

Escribe aquí tus conclusiones

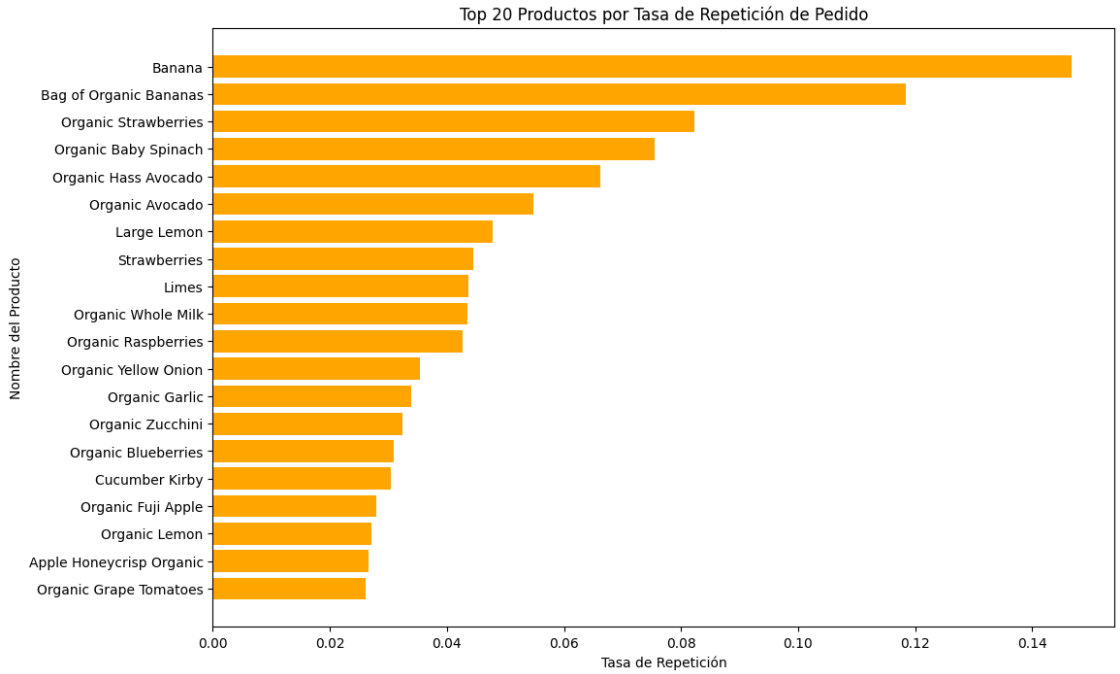
- Los productos mas repetidos son las frutas y productos de origen organico

8.0.3 [C3] Para cada producto, ¿cuál es la proporción de las veces que se pide y que se vuelve a pedir?

[235]:
```
#Calcular repeticion de predidos
total_orders = order_products_df['order_id'].nunique()
repeat_counts['repeat_rate'] = repeat_counts['order_count'] / total_orders
```

[236]:
```
#Tasa de repeticion
top_repeated_products = repeat_counts.sort_values(by='repeat_rate',␣
 ↪ascending=False).head(20)
top_repeated_products = top_repeated_products.merge(products_df,␣
 ↪on='product_id', how='left')
```

[237]:
```
#Grafico
plt.figure(figsize=(12, 8))
plt.barh(top_repeated_products['product_name'],␣
 ↪top_repeated_products['repeat_rate'], color='orange')
plt.xlabel('Tasa de Repetición')
plt.ylabel('Nombre del Producto')
plt.title('Top 20 Productos por Tasa de Repetición de Pedido')
plt.gca().invert_yaxis()
plt.show()

print(top_repeated_products[['product_id', 'product_name', 'repeat_rate']])
```



	product_id	product_name	repeat_rate
0	24852	Banana	0.146763
1	13176	Bag of Organic Bananas	0.118426
2	21137	Organic Strawberries	0.082300
3	21903	Organic Baby Spinach	0.075483
4	47209	Organic Hass Avocado	0.066155
5	47766	Organic Avocado	0.054859
6	47626	Large Lemon	0.047762
7	16797	Strawberries	0.044480
8	26209	Limes	0.043751
9	27845	Organic Whole Milk	0.043551
10	27966	Organic Raspberries	0.042656
11	22935	Organic Yellow Onion	0.035325
12	24964	Organic Garlic	0.033979
13	45007	Organic Zucchini	0.032406
14	39275	Organic Blueberries	0.030839
15	49683	Cucumber Kirby	0.030386
16	28204	Organic Fuji Apple	0.027873
17	5876	Organic Lemon	0.027179
18	8277	Apple Honeycrisp Organic	0.026648
19	40706	Organic Grape Tomatoes	0.026177

Comentario del revisor. (Iteración 1)

Una forma muy interesante de resolverlo Jafet, solo tengamos en cuenta que si de base nos pide la proporcion por producto que se ha repetido deberíamos hacer el cálculo inicial con product_id,

una forma sencillo de varlo es sobre el dataframe ya unificado agrupar por product_id y calcular el promedio de reordered.

`df=pd.DataFrame(df_merged.groupby("product_id")["reordered"].mean()).reset_index()`

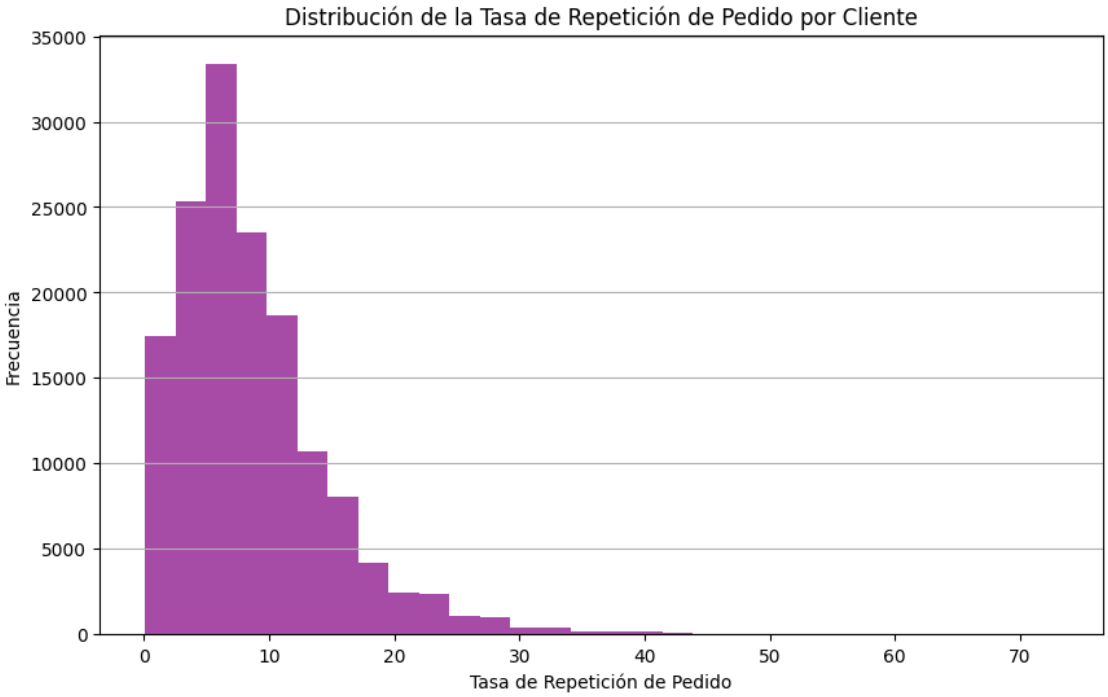
Escribe aquí tus conclusiones

- La tabla muestra que los productos con mayor tasa de repeticion son principalmente frutas y vegetales orgánicos, siendo la banana el más popular con un 14.68%. Esto indica una fuerte preferencia por productos frescos, lo que sugiere que los minoristas deben enfocarse en promocionar estos artículos para fomentar la recompra.

8.0.4 [C4] Para cada cliente, ¿qué proporción de sus productos ya los había pedido?

[240]:
```
merged_df = order_products_df.merge(orders_df[['order_id', 'user_id']],␣
 ↪on='order_id', how='left')

total_orders = merged_df.groupby('user_id')['order_id'].nunique().reset_index()
total_orders.columns = ['user_id', 'total_orders']

distinct_products = merged_df.groupby('user_id')['product_id'].nunique().
 ↪reset_index()
distinct_products.columns = ['user_id', 'distinct_products']

customer_summary = total_orders.merge(distinct_products, on='user_id',␣
 ↪how='left')
customer_summary['repeat_rate'] = customer_summary['distinct_products'] /␣
 ↪customer_summary['total_orders']
```

[244]:
```
plt.figure(figsize=(10, 6))
plt.hist(customer_summary['repeat_rate'], bins=30, color='purple', alpha=0.7)
plt.xlabel('Tasa de Repetición de Pedido')
plt.ylabel('Frecuencia')
plt.title('Distribución de la Tasa de Repetición de Pedido por Cliente')
plt.grid(axis='y')
plt.show()

print(customer_summary.head())
```



	user_id	total_orders	distinct_products	repeat_rate
0	2	2	25	12.5
1	4	1	2	2.0
2	5	1	12	12.0
3	6	1	4	4.0
4	7	2	13	6.5

Comentario del revisor. (Iteración 1)

Aquí una situación similar a lo anterior, el enfoque es interesante pero la forma en que queremos resolverlo es más compleja de lo que necesitamos, si queremos saber la tasa de repetición por usuario solo debemos agrupar directamente por usuario y calcular el promedio de reordered

```
df_merged_user = df_merged_user.groupby('user_id')['reordered'].mean()
```

Escribe aquí tus conclusiones

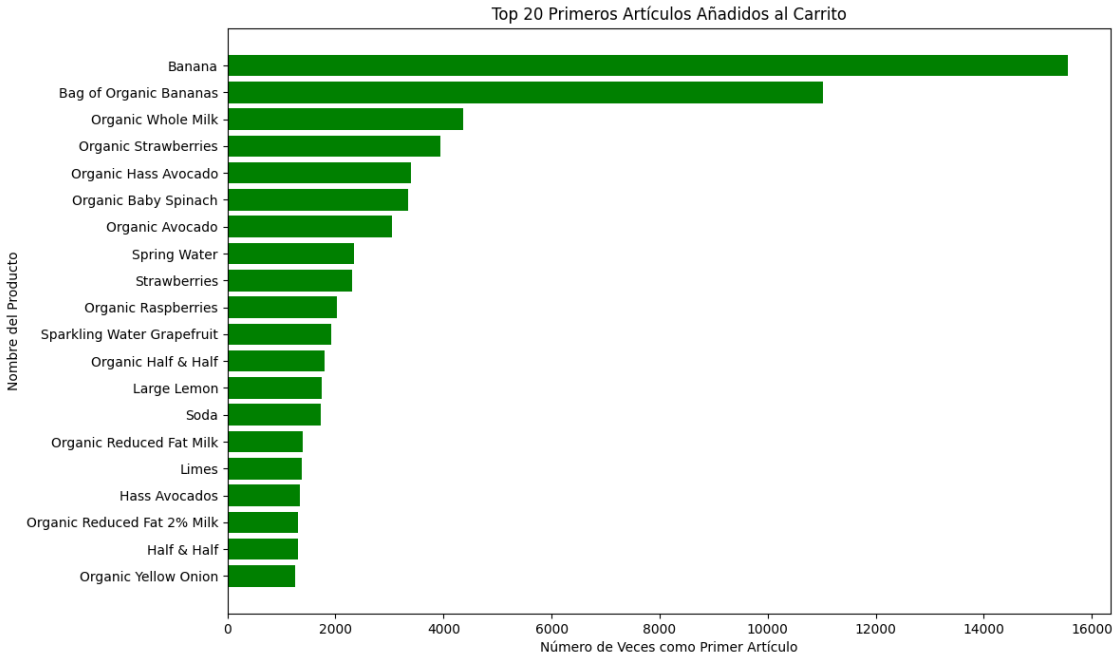
-La tabla muestra que los clientes tienen una variabilidad significativa en la cantidad de pedidos y productos distintos adquiridos. Aunque algunos clientes como el usuario 2 han realizado mas pedidos y han comprado una mayor diversidad de productos, la tasa de repetición varía, lo que sugiere diferencias en el comportamiento de compra.

8.0.5 [C5] ¿Cuáles son los 20 principales artículos que las personas ponen primero en sus carritos?

[174]:
```
first_item = order_products_df.loc[order_products_df.
↪groupby('order_id')['add_to_cart_order'].idxmin()]
```

[175]:
```
first_item_counts = first_item['product_id'].value_counts().reset_index()
first_item_counts.columns = ['product_id', 'first_item_count']
top_first_items = first_item_counts.head(20).merge(products_df,␣
↪on='product_id', how='left')
```

[243]:
```
plt.figure(figsize=(12, 8))
plt.barh(top_first_items['product_name'], top_first_items['first_item_count'],␣
↪color='green')
plt.xlabel('Número de Veces como Primer Artículo')
plt.ylabel('Nombre del Producto')
plt.title('Top 20 Primeros Artículos Añadidos al Carrito')
plt.gca().invert_yaxis()
plt.show()
```



Comentario del revisor. (Iteración 1)

Para este caso si un cálculo excelente Jafet, bien hecho!

Escribe aquí tus conclusiones

- Las bananas son los productos preferidos por los clientes, posterior mente la leche y los vegetales organicos.

8.0.6 Conclusion general del proyecto:

- Los analisis realizados sobre los productos que se vuelven a pedir con mayor frecuencia revelan que ciertos articulos son significativamente mas populares entre los clientes, lo que indica una posible lealtad del consumidor hacia estos productos. La representacion grafica muestra claramente que los productos de uso cotidiano, como frutas y articulos esenciales, dominan la lista. Esto sugiere que los clientes tienden a repetir compras de productos que consideran necesarios o de alta calidad, lo que puede ofrecer a los minoristas la oportunidad de implementar estrategias de marketing enfocadas en estos articulos para maximizar la retencion de clientes.

[ ]:

[ ]:

[ ]: