Segmentação de clientes

Usar o algoritmo K-Means para agrupar clientes com base em seus comportamentos de compra. Descrever as características de cada grupo.

Instalação de bibliotecas

In [1]:
```
%pip install pandas matplotlib seaborn sklearn
```

```
Requirement already satisfied: pandas in c:\python312\lib\site-packages (2.2.2)
Requirement already satisfied: matplotlib in c:\python312\lib\site-packages (3.9.
2)
Requirement already satisfied: seaborn in c:\python312\lib\site-packages (0.13.2)
Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'error'
Note: you may need to restart the kernel to use updated packages.
```

```
  error: subprocess-exited-with-error

  × Getting requirements to build wheel did not run successfully.
  │ exit code: 1
  ╰─> [15 lines of output]
      The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
      rather than 'sklearn' for pip commands.

      Here is how to fix this error in the main use cases:
      - use 'pip install scikit-learn' rather than 'pip install sklearn'
      - replace 'sklearn' by 'scikit-learn' in your pip requirements files
        (requirements.txt, setup.py, setup.cfg, Pipfile, etc ...)
      - if the 'sklearn' package is used by one of your dependencies,
        it would be great if you take some time to track which package uses
        'sklearn' instead of 'scikit-learn' and report it to their issue tracker
      - as a last resort, set the environment variable
        SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid this error

      More information is available at
      https://github.com/scikit-learn/sklearn-pypi-package
      [end of output]

  note: This error originates from a subprocess, and is likely not a problem with
pip.
error: subprocess-exited-with-error

× Getting requirements to build wheel did not run successfully.
│ exit code: 1
╰─> See above for output.

note: This error originates from a subprocess, and is likely not a problem with p
ip.

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Importação das tabelas

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Carregar os datasets
customers = pd.read_csv("olist_customers_dataset.csv")
geolocation = pd.read_csv("olist_geolocation_dataset.csv")
order_items = pd.read_csv("olist_order_items_dataset.csv")
order_payments = pd.read_csv("olist_order_payments_dataset.csv")
order_reviews = pd.read_csv("olist_order_reviews_dataset.csv")
orders = pd.read_csv("olist_orders_dataset.csv")
products = pd.read_csv("olist_products_dataset.csv")
sellers = pd.read_csv("olist_sellers_dataset.csv")
product_category_translation = pd.read_csv("product_category_name_translation.cs
```

Preparação dos dados (RFV)

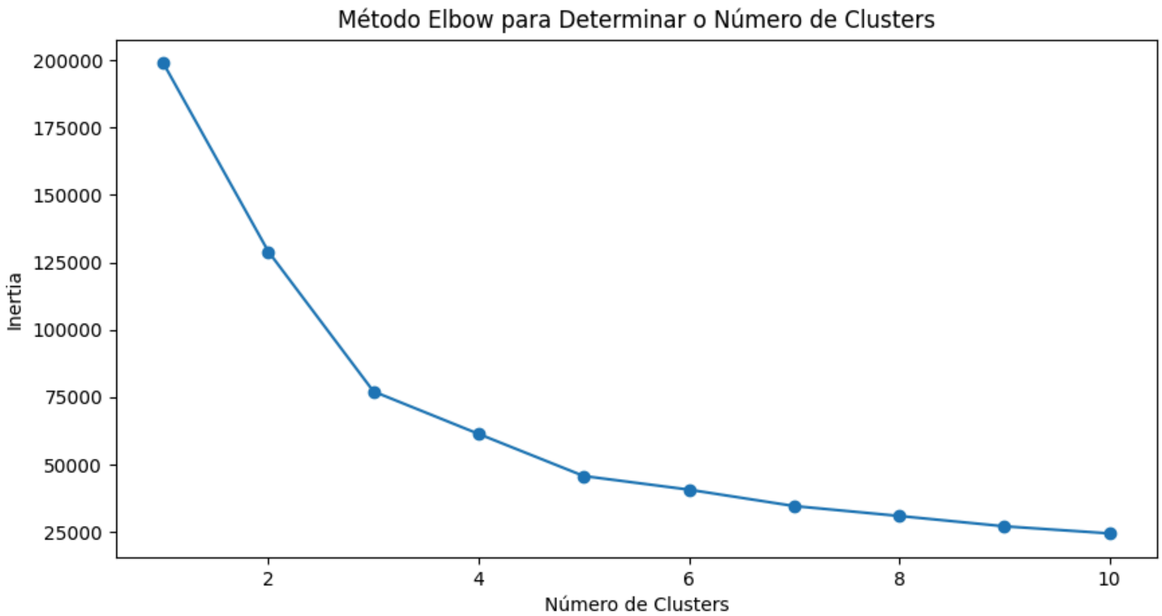
In [3]:
```
# Calcula o valor total de cada pedido
order_items['total_value'] = order_items['price'] + order_items['freight_value']
order_totals = order_items.groupby('order_id')['total_value'].sum().reset_index(
orders = pd.merge(orders, order_totals, on='order_id', how='left')


# Converte a data de compra para datetime
orders['order_purchase_timestamp'] = pd.to_datetime(orders['order_purchase_times

# Define a data mais recente como um dia após a última compra
most_recent_date = orders['order_purchase_timestamp'].max() + pd.Timedelta(days=

# Calcula a Recência, Frequência e Valor Monetário (RFV)
rfv = orders.groupby('customer_id').agg({
    'order_purchase_timestamp': lambda x: (most_recent_date - x.max()).days,  #
    'order_id': 'count',  # Frequência
    'total_value': 'sum'  # Valor Monetário
})

rfv.rename(columns={
    'order_purchase_timestamp': 'Recency',
    'order_id': 'Frequency',
    'total_value': 'MonetaryValue'
}, inplace=True)

print(rfv.head())



# Padroniza os dados (importante para o K-Means)
scaler = StandardScaler()
rfv_scaled = scaler.fit_transform(rfv)
```

```
                                  Recency  Frequency  MonetaryValue
customer_id
00012a2ce6f8dcda20d059ce98491703      338          1         114.74
000161a058600d5901f007fab4c27140      459          1          67.41
0001fd6190edaaf884bcaf3d49edf079      597          1         195.42
0002414f95344307404f0ace7a26f1d5      428          1         179.35
000379cdec625522490c315e70c7a9fb      199          1         107.01
```

Aplicando o K-Means

In [4]:
```
# Determina o número ideal de clusters (método Elbow)
inertia = []
for n in range(1, 11):
    kmeans = KMeans(n_clusters=n, random_state=42)
    kmeans.fit(rfv_scaled)
    inertia.append(kmeans.inertia_)


plt.figure(figsize=(10, 5))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Método Elbow para Determinar o Número de Clusters')
plt.xlabel('Número de Clusters')
plt.ylabel('Inertia')
plt.show()
```

```python
# Escolhe o número de clusters baseado no gráfico do Elbow (exemplo: 4 clusters)
n_clusters = 4 # modifique se o gráfico elbow indicar outro valor ideal

kmeans = KMeans(n_clusters=n_clusters, random_state=42)
rfv['Cluster'] = kmeans.fit_predict(rfv_scaled)

print(rfv.head())
```




```
                                  Recency  Frequency  MonetaryValue  Cluster
customer_id
00012a2ce6f8dcda20d059ce98491703      338          1         114.74        1
000161a058600d5901f007fab4c27140      459          1          67.41        1
0001fd6190edaaf884bcaf3d49edf079      597          1         195.42        1
0002414f95344307404f0ace7a26f1d5      428          1         179.35        1
000379cdec625522490c315e70c7a9fb      199          1         107.01        0
```

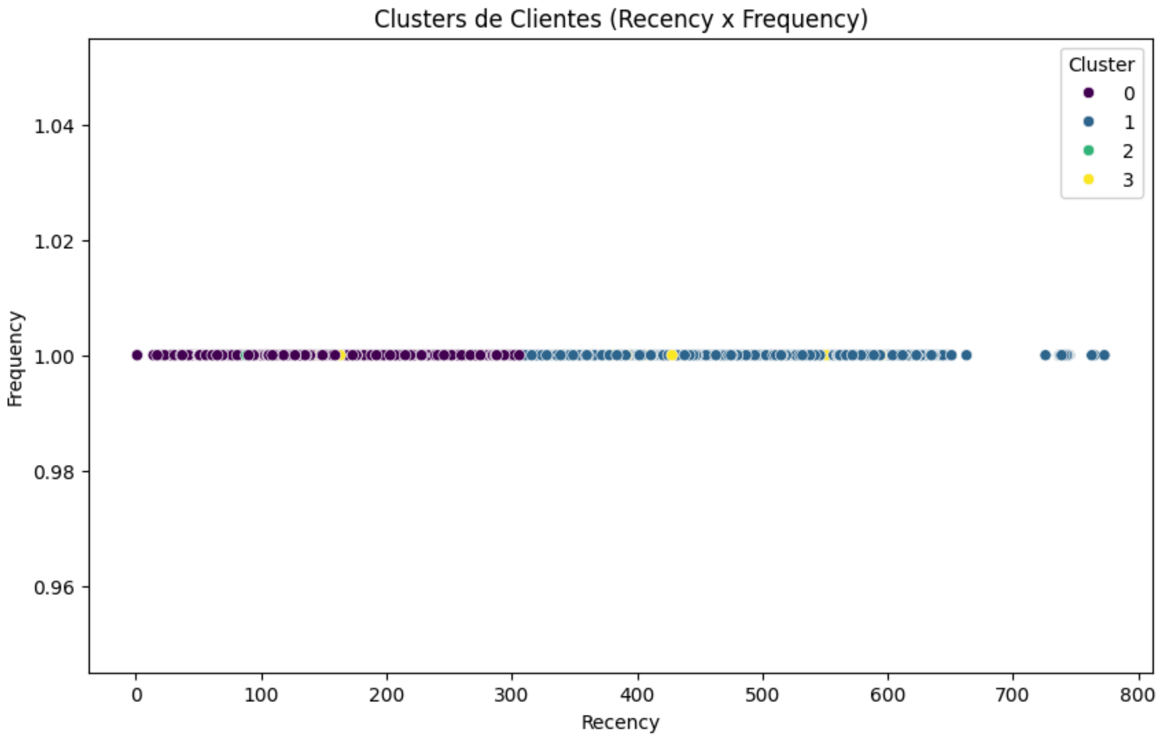
Analisando os Clusters

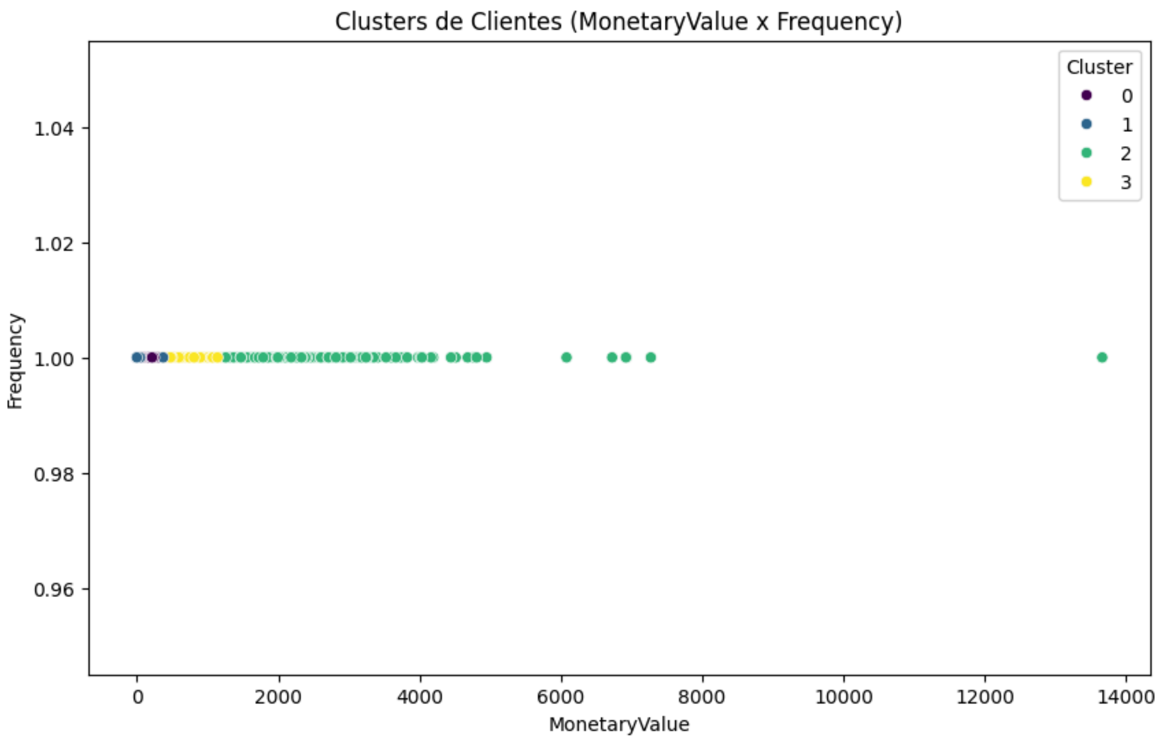
In [5]:
```python
# Analisa as características de cada cluster
print("\nCaracterísticas dos Clusters:")
print(rfv.groupby('Cluster').agg({
    'Recency': ['mean', 'median'],
    'Frequency': ['mean', 'median'],
    'MonetaryValue': ['mean', 'median']
}))


# Visualizando os clusters (exemplo com Recency x Frequency)

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Recency', y='Frequency', hue='Cluster', data=rfv, palette='vi
plt.title('Clusters de Clientes (Recency x Frequency)')
plt.show()


# Visualizando os clusters (exemplo com MonetaryValue x Frequency)
plt.figure(figsize=(10, 6))
sns.scatterplot(x='MonetaryValue', y='Frequency', hue='Cluster', data=rfv, palet
plt.title('Clusters de Clientes (MonetaryValue x Frequency)')
plt.show()
```

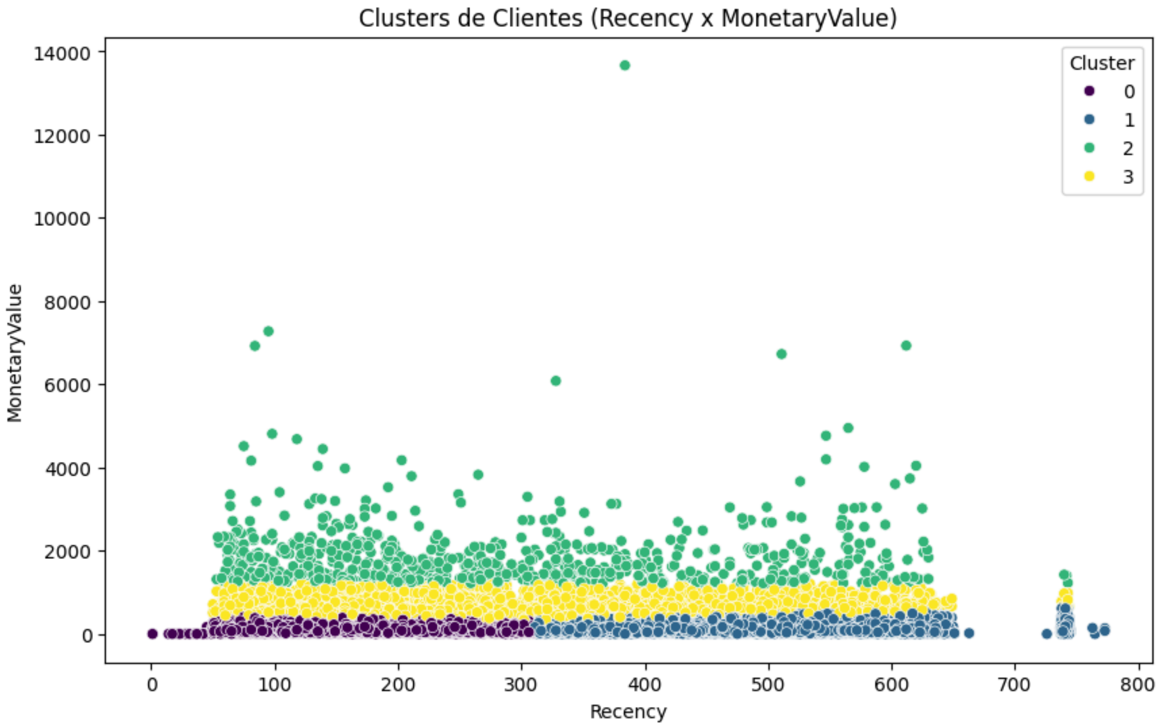
```python
# Visualizando os clusters (exemplo com Recency x MonetaryValue)
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Recency', y='MonetaryValue', hue='Cluster', data=rfv, palette
plt.title('Clusters de Clientes (Recency x MonetaryValue)')
plt.show()
```

Características dos Clusters:

	Recency		Frequency		MonetaryValue	
	mean	median	mean	median	mean	median
Cluster						
0	179.519870	181.0	1.0	1.0	118.238119	99.14
1	442.008264	430.0	1.0	1.0	118.952898	96.12
2	297.415816	283.0	1.0	1.0	1819.093431	1589.91
3	275.690562	263.5	1.0	1.0	608.856138	558.52







Cluster 0: Recency baixa, Frequency alta, Monetary Value alto. Este cluster pode representar os clientes mais valiosos, que compram frequentemente e recentemente. Devem ser recompensados com ofertas exclusivas e programas de fidelidade.

Cluster 1: Recency alta, Frequency baixa, Monetary Value baixo. Este cluster pode representar clientes em risco de churn, que não compram há muito tempo e gastam pouco. Campanhas de reativação e ofertas personalizadas podem ser eficazes.

Cluster 2: Recency média, Frequency média, Monetary Value médio. Este cluster pode representar os clientes "típicos", que compram ocasionalmente. Campanhas de up-selling e cross-selling podem ser interessantes.

Cluster 3: Recency baixa, Frequency baixa, Monetary Value baixo. Este cluster pode representar clientes novos ou esporádicos. É importante incentivá-los a comprar novamente e aumentar seu valor monetário.

Considerando outra variável

In [6]:

```
# Merge para obter customer_unique_id em order_items
order_items_with_customer = pd.merge(order_items, orders[['order_id', 'customer_
order_items_with_customer = pd.merge(order_items_with_customer, customers[['cust

# Merge para obter as categorias dos produtos
order_items_with_category = pd.merge(order_items_with_customer, products[['produ

# Encontra a categoria mais comprada por cliente (agora com customer_unique_id)
most_frequent_category = order_items_with_category.groupby(['customer_unique_id'
most_frequent_category = most_frequent_category.groupby('customer_unique_id').ap
most_frequent_category.reset_index(drop=True, inplace=True)
most_frequent_category = most_frequent_category[['customer_unique_id','product_c
most_frequent_category = most_frequent_category.rename(columns={'product_categor
```

```python
most_frequent_category.head()


# Adiciona a categoria mais frequente ao dataframe RFV (usando customer_unique_i
rfv = pd.merge(rfv, most_frequent_category, on='customer_unique_id', how='left')

# Criar uma cópia do RFV somente com as variáveis numéricas para o K-Means
rfv_kmeans = rfv.drop(columns=['MostFrequentCategory'])  # Remove a coluna categ

# Padroniza os dados (importante para o K-Means)
# Aplicar o scaler APÓS adicionar a categoria mais frequente e criar rfv_kmeans
scaler = StandardScaler()
rfv_scaled = scaler.fit_transform(rfv_kmeans)


# Determina o número ideal de clusters (método Elbow)
inertia = []
for n in range(1, 11):
    kmeans = KMeans(n_clusters=n, random_state=42)
    kmeans.fit(rfv_scaled)
    inertia.append(kmeans.inertia_)


plt.figure(figsize=(10, 5))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Método Elbow para Determinar o Número de Clusters')
plt.xlabel('Número de Clusters')
plt.ylabel('Inertia')
plt.show()

# Escolhe o número de clusters baseado no gráfico do Elbow (exemplo: 4 clusters)
n_clusters = 4 # modifique se o gráfico elbow indicar outro valor ideal

kmeans = KMeans(n_clusters=n_clusters, random_state=42)
rfv['Cluster'] = kmeans.fit_predict(rfv_scaled)

print(rfv.head())


# Analisando os clusters, incluindo a categoria mais frequente:
print(rfv.groupby('Cluster').agg({
    'Recency': ['mean', 'median'],
    'Frequency': ['mean', 'median'],
    'MonetaryValue': ['mean', 'median'],
    'MostFrequentCategory': lambda x: x.value_counts().index[0]  # Categoria mai
}))
```

```
C:\Users\salom\AppData\Local\Temp\ipykernel_16412\430456860.py:10: DeprecationWar
ning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is d
eprecated, and in a future version of pandas the grouping columns will be exclude
d from the operation. Either pass `include_groups=False` to exclude the groupings
or explicitly select the grouping columns after groupby to silence this warning.
  most_frequent_category = most_frequent_category.groupby('customer_unique_id').a
pply(lambda x: x.nlargest(1, 'order_id'))
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_16412\430456860.py in ?()
     15 most_frequent_category.head()
     16 
     17 
     18 # Adiciona a categoria mais frequente ao dataframe RFV (usando customer_u
nique_id)
---> 19 rfv = pd.merge(rfv, most_frequent_category, on='customer_unique_id', how
='left')
     20 
     21 # Criar uma cópia do RFV somente com as variáveis numéricas para o K-Mean
s
     22 rfv_kmeans = rfv.drop(columns=['MostFrequentCategory'])  # Remove a colun
a categórica

c:\Python312\Lib\site-packages\pandas\core\reshape\merge.py in ?(left, right, ho
w, on, left_on, right_on, left_index, right_index, sort, suffixes, copy, indicato
r, validate)
    166             validate=validate,
    167             copy=copy,
    168         )
    169     else:
--> 170         op = _MergeOperation(
    171             left_df,
    172             right_df,
    173             how=how,

c:\Python312\Lib\site-packages\pandas\core\reshape\merge.py in ?(self, left, righ
t, how, on, left_on, right_on, left_index, right_index, sort, suffixes, indicato
r, validate)
    790             self.right_join_keys,
    791             self.join_names,
    792             left_drop,
    793             right_drop,
--> 794         ) = self._get_merge_keys()
    795 
    796         if left_drop:
    797             self.left = self.left._drop_labels_or_levels(left_drop)

c:\Python312\Lib\site-packages\pandas\core\reshape\merge.py in ?(self)
   1306                     if lk is not None:
   1307                         # Then we're either Hashable or a wrong-length ar
raylike,
   1308                         #  the latter of which will raise
   1309                         lk = cast(Hashable, lk)
-> 1310                         left_keys.append(left._get_label_or_level_values
(lk))
   1311                         join_names.append(lk)
   1312                     else:
   1313                         # work-around for merge_asof(left_index=True)

c:\Python312\Lib\site-packages\pandas\core\generic.py in ?(self, key, axis)
   1907             values = self.xs(key, axis=other_axes[0])._values
   1908         elif self._is_level_reference(key, axis=axis):
   1909             values = self.axes[axis].get_level_values(key)._values
   1910         else:
-> 1911             raise KeyError(key)
   1912 
   1913         # Check for duplicates
```

```
   1914         if values.ndim > 1:

KeyError: 'customer_unique_id'
```