

✓ Data-Driven Customer Insights with KMeans Clustering

Sobre a base de dados e o projeto:

Este projeto tem como objetivo aplicar o algoritmo de KMeans Clustering para segmentação de clientes com base em seus comportamentos e padrões de compra. Utilizando bibliotecas populares do Python, como Pandas, Scikit-learn, e Matplotlib, o projeto demonstra como preparar os dados, escolher o número ideal de clusters usando o Elbow Method e interpretar os resultados para gerar insights valiosos para negócios.

Inspiração: <https://www.youtube.com/watch?v=afPJeQuVeuY>

Data-set: <https://archive.ics.uci.edu/dataset/502/online+retail+ii>

Online Retail Mining Whitepaper: <https://link.springer.com/article/10.1057/dbm.2012.17>

O dataset utilizado no projeto contém dados sobre clientes, incluindo variáveis como idade, renda anual e pontuação de gastos em uma loja. Ele é amplamente usado em análises de segmentação para identificar perfis distintos de consumidores. A partir dessas variáveis, o KMeans Clustering pode agrupar os clientes em clusters com base em semelhanças, permitindo identificar padrões de comportamento. Esse tipo de análise é útil em estratégias de marketing personalizadas, ajudando empresas a compreender melhor seus clientes e otimizar ofertas de produtos e serviços conforme os perfis identificados.

✓ 1. Importando as bibliotecas

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

✓ 2. Baixando o data-set:



```
# Previamente carregado no meu colab
```

```
df = pd.read_excel('/content/online_retail_II.xlsx') # Formato de excel é um pouco mais p
```

```
df.head(10)
```



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
5	489434	22064	PINK DOUGHNUT TRINKET POT	24	2009-12-01 07:45:00	1.65	13085.0	United Kingdom
6	489434	21871	SAVE THE PLANET MUG	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
7	489434	21523	FANCY FONT HOME SWEET HOME DOORMAT	10	2009-12-01 07:45:00	5.95	13085.0	United Kingdom
8	489435	22350	CAT BOWL	12	2009-12-01 07:46:00	2.55	13085.0	United Kingdom
9	489435	22349	DOG BOWL , CHASING BALL DESIGN	12	2009-12-01 07:46:00	3.75	13085.0	United Kingdom



3. Explorando os dados

```
# Informações gerais sobre o data-set:
df.info()
```

Vemos que as colunas 'Description' e 'Customer ID' apresentam valores ausentes quando c
Podemos perceber também que a coluna 'InvoiceDate' já está em formato de tempo, o que é




```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525461 entries, 0 to 525460
```



```
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Invoice      525461 non-null    object
1   StockCode    525461 non-null    object
2   Description   522533 non-null    object
3   Quantity     525461 non-null    int64
4   InvoiceDate   525461 non-null    datetime64[ns]
5   Price        525461 non-null    float64
6   Customer ID  417534 non-null    float64
7   Country      525461 non-null    object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 32.1+ MB
```

```
df.describe()

# Algo estranho de notar é o valor de quantity (min) sendo negativo, o mesmo também ocorr
# Outra coisa a se perceber é que por padrão aqui estão apenas os valores numéricos e não
```



	Quantity	InvoiceDate	Price	Customer ID
count	525461.000000	525461	525461.000000	417534.000000
mean	10.337667	2010-06-28 11:37:36.845017856	4.688834	15360.645478
min	-9600.000000	2009-12-01 07:45:00	-53594.360000	12346.000000
25%	1.000000	2010-03-21 12:20:00	1.250000	13983.000000
50%	3.000000	2010-07-06 09:51:00	2.100000	15311.000000
75%	10.000000	2010-10-15 12:45:00	4.210000	16799.000000
max	19152.000000	2010-12-09 20:01:00	25111.090000	18287.000000
std	107.424110	NaN	146.126914	1680.811316



```
# Vendo os objetos:

df.describe(include = 'O')

# Esse parece ok, os valores não me geram nenhuma estranheza.
```



	Invoice	StockCode	Description	Country
count	525461	525461	522533	525461
unique	28816	4632	4681	40
top	537434	85123A	WHITE HANGING HEART T-LIGHT HOLDER	United Kingdom
freq	675	3516	3549	485852



```
# Vamos dar uma olhada nos valores faltantes ['CustomerID']:
```

```
df[df["Customer ID"].isna()].head(10)
```

```
# Vemos que '85123a mixed', '21733 mixed' e 'short' apresentam valores negativos com preço
# Esses parecem dados válidos de serem eliminados!
```



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
263	489464	21733	85123a mixed	-96	2009-12-01 10:52:00	0.00	NaN	United Kingdom
283	489463	71477	short	-240	2009-12-01 10:52:00	0.00	NaN	United Kingdom
284	489467	85123A	21733 mixed	-192	2009-12-01 10:53:00	0.00	NaN	United Kingdom
470	489521	21646	NaN	-50	2009-12-01 11:44:00	0.00	NaN	United Kingdom
577	489525	85226C	BLUE PULL BACK RACING CAR	1	2009-12-01 11:49:00	0.55	NaN	United Kingdom
578	489525	85227	SET/6 3D KIT CARDS FOR KIDS	1	2009-12-01 11:49:00	0.85	NaN	United Kingdom
1055	489548	22271	FELTCRAFT DOLL ROSIE	1	2009-12-01 12:32:00	2.95	NaN	United Kingdom
1056	489548	22254	FELT TOADSTOOL LARGE	12	2009-12-01 12:32:00	1.25	NaN	United Kingdom
1057	489548	22273	FELTCRAFT DOLL MOLLY	3	2009-12-01 12:32:00	2.95	NaN	United Kingdom
1058	489548	22195	LARGE HEART MEASURING SPOONS	1	2009-12-01 12:32:00	1.65	NaN	United Kingdom

```
# Na realidade, tendo em vista que estamos clusterizando os clientes, e é impossível repetir
# Iremos fazer isso em breve!
```

```
# Vamos dar uma olhada especificamente nos valores negativos para compreender eles melhor
```

```
df[df['Quantity'] < 0].head(10)
```

```
# Podemos reparar que grande parte desses valores possui um 'C' a frente da sequência em
# Na descrição dos dados somos informados que C é para produtos Cancelados!
```



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Count
178	C489449	22087	PAPER BUNTING WHITE LACE	-12	2009-12-01 10:33:00	2.95	16321.0	Austra
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	2009-12-01 10:33:00	1.65	16321.0	Austra
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	2009-12-01 10:33:00	4.25	16321.0	Austra
181	C489449	21896	POTTING SHED TWINE	-6	2009-12-01 10:33:00	2.10	16321.0	Austra
182	C489449	22083	PAPER CHAIN KIT RETRO SPOT	-12	2009-12-01 10:33:00	2.95	16321.0	Austra
183	C489449	21871	SAVE THE PLANET MUG	-12	2009-12-01 10:33:00	1.25	16321.0	Austra
184	C489449	84946	ANTIQUE SILVER TEA GLASS ETCHED	-12	2009-12-01 10:33:00	1.25	16321.0	Austra
185	C489449	84970S	HANGING HEART ZINC T-LIGHT HOLDER	-24	2009-12-01 10:33:00	0.85	16321.0	Austra
186	C489449	22090	PAPER BUNTING RETRO SPOTS	-12	2009-12-01 10:33:00	2.95	16321.0	Austra
196	C489459	90200A	PURPLE SWEETHEART BRACELET	-3	2009-12-01 10:44:00	4.25	17592.0	Unit Kingdc



Vamos olhar esses dados de Cancelados usando Regex:

```
df['Invoice'] = df['Invoice'].astype('str')
df[df['Invoice'].str.match("^\d{6}$") == False]
# Pegando as colunas onde não são exatos 6 dígitos, como diz a expressão!
```



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Cou
178	C489449	22087	PAPER BUNTING WHITE LACE	-12	2009-12-01 10:33:00	2.95	16321.0	Aus
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	2009-12-01 10:33:00	1.65	16321.0	Aus
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	2009-12-01 10:33:00	4.25	16321.0	Aus
181	C489449	21896	POTTING SHED TWINE	-6	2009-12-01 10:33:00	2.10	16321.0	Aus
182	C489449	22083	PAPER CHAIN KIT RETRO SPOT	-12	2009-12-01 10:33:00	2.95	16321.0	Aus
...
524695	C538123	22956	36 FOIL HEART CAKE CASES	-2	2010-12-09 15:41:00	2.10	12605.0	Gerr
524696	C538124	M	Manual	-4	2010-12-09 15:43:00	0.50	15329.0	Ui King
524697	C538124	22699	ROSES REGENCY TEACUP AND SAUCER	-1	2010-12-09 15:43:00	2.95	15329.0	Ui King
524698	C538124	22423	REGENCY CAKESTAND 3 TIER	-1	2010-12-09 15:43:00	12.75	15329.0	Ui King
525282	C538164	35004B	SET OF 3 BLACK FLYING DUCKS	-1	2010-12-09 17:32:00	1.95	14031.0	Ui King

10209 rows × 8 columns



Será que o C é a única letra possível de aparecer na frente do 'Invoice'?

```
df['Invoice'].str.replace('[0-9]', '', regex = True).unique()
```

```
# Vemos que temos também um A, que é inesperado, pois não está no index dos dados!
```

```
array(['', 'C', 'A'], dtype=object)
```

```
# Apenas os valores iniciados com A
```

```
df[df['Invoice'].str.startswith('A')]
```

```
# Vemos que eles correspondem a categoria 'Adjust bad debt'
```

```
# Também são possíveis valores para excluir
```



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	
179403	A506401	B	Adjust bad debt	1	2010-04-29 13:36:00	-53594.36	NaN	
276274	A516228	B	Adjust bad debt	1	2010-07-19 11:24:00	-44031.79	NaN	
403472	A528059	B	Adjust bad debt	1	2010-10-20 12:04:00	-38925.87	NaN	



```
# Inspeccionando a coluna 'StockCode':
```

```
# Na descrição dos dados, vemos que são valores compostos por A-5 dígitos designada para
```

```
df['StockCode'] = df['StockCode'].astype('str')
```

```
df[df['StockCode'].str.match('^\d{5}$') == False]
```

```
# Voltamos uma lista de valores que não correspondem a isso, mas eles não parecem ter nada
```



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	United Kingdom

525461 rows × 8 columns



```
df[(df['StockCode'].str.match('^\d{5}$') == False) & (df['StockCode'].str.match('^\d{5}

# Vemos que temos muitos valores que não seguem esse padrão
```




	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Co
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	l Kin
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	l Kin
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	l Kin
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	l Kin
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	l Kin
...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	l Kin
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	l Kin
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	l Kin
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	l Kin
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	l Kin

525461 rows × 8 columns



```
# Retornando os valores unicos:

df[(df['StockCode'].str.match('^\d{5}$') == False) & (df['StockCode'].str.match('^\d{5}

array(['85048', '79323P', '79323W', ..., '22935', '22933', '21120'],
      dtype=object)

# Vamos olhar especificamente e de forma manual cada um desses valores:

df[df['StockCode'].str.contains('^DOT')]
```

No caso de DOT por exemplo, não é útil usar esses dados pois não temos 'CustomerID'



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Cou
2379	489597	DOT	DOTCOM POSTAGE	1	2009-12-01 14:28:00	647.19	NaN	Ui King
2539	489600	DOT	DOTCOM POSTAGE	1	2009-12-01 14:43:00	55.96	NaN	Ui King
2551	489601	DOT	DOTCOM POSTAGE	1	2009-12-01 14:44:00	68.39	NaN	Ui King
2571	489602	DOT	DOTCOM POSTAGE	1	2009-12-01 14:45:00	59.35	NaN	Ui King
2619	489603	DOT	DOTCOM POSTAGE	1	2009-12-01 14:46:00	42.39	NaN	Ui King
...
524272	538071	DOT	DOTCOM POSTAGE	1	2010-12-09 14:09:00	885.94	NaN	Ui King
524887	538148	DOT	DOTCOM POSTAGE	1	2010-12-09 16:26:00	547.32	NaN	Ui King
525000	538149	DOT	DOTCOM POSTAGE	1	2010-12-09 16:27:00	620.68	NaN	Ui King
525126	538153	DOT	DOTCOM POSTAGE	1	2010-12-09 16:31:00	822.94	NaN	Ui King
525147	538154	DOT	DOTCOM POSTAGE	1	2010-12-09 16:35:00	85.79	NaN	Ui King

736 rows × 8 columns



Resumindo os dados de todos eles em formato de tabela, temos:

Code	Description	Acti
DCGS	Looks valid, some quantities are negative though and customer ID is null	Exclude from
D	Looks valid, represents discount values	Exclude from
DOT	Looks valid, represents postage charges	Exclude from
M or m	Looks valid, represents manual transactions	Exclude from
C2	Carriage transaction - not sure what this means	Exclude from
C3	Not sure, only 1 transaction	Exclude
BANK CHARGES or B	Bank charges	Exclude from
S	Samples sent to customer	Exclude from
TESTXXX	Testing data, not valid	Exclude from
gift_XXX	Purchases with gift cards, might be interesting for another analysis, but no customer data	Exclude
PADS	Looks like a legit stock code for padding	Include
SP1002	Looks like a special request item, only 2 transactions, 3 look legit, 1 has 0 pricing	Exclude for n

Code	Description	Acti
AMAZONFEE	Looks like fees for Amazon shipping or something	Exclude for n
ADJUSTX	Looks like manual account adjustments by admins	Exclude for n

Resumidamente, vamos usar apenas 'PADS', que parece válida, o resto será excluído!

✓ 4. Limpando os dados:

```
# Vamos criar uma cópia para poder realizar a limpeza dos dados:

cleaned_df = df.copy()

# Limpando a coluna 'Invoice':

cleaned_df['Invoice'] = cleaned_df['Invoice'].astype('str')

# Criando uma máscara, que é basicamente uma variável para filtrar dados:

mask = (
    cleaned_df['Invoice'].str.match('^\\d{6}$') == True
)

cleaned_df = cleaned_df[mask]

cleaned_df

# Com isso, excluimos todos os 'Invoices' que não seguem o padrão que comentamos antes.
```



	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Co
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	l Kin
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	l Kin
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	l Kin
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	l Kin
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	l Kin
...	
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	l Kin
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	l Kin
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	l Kin
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	l Kin
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	l Kin

515252 rows × 8 columns



```
# Fazendo a mesma coisa para a coluna 'StockCode':

cleaned_df['StockCode'] = cleaned_df['StockCode'].astype('str')

# Criando uma máscara, que é basicamente uma variável para filtrar dados:

mask = (
    (cleaned_df['StockCode'].str.match('^\\d{5}$') == True) # Apenas 5 dígitos (padrão)
    | (cleaned_df['StockCode'].str.match('^\\d{5}[a-zA-Z]$') == True) # ou 5 dígitos segui
    | (cleaned_df['StockCode'].str.match('^PADS$') == True) # ou a PADS que verificamos s
```

```
# Agora o padrão é o que vimos para StockCode!  
)  
  
cleaned_df = cleaned_df[mask]  
  
cleaned_df  
  
# Com isso, excluimos todos os 'StockCodes' que não seguem o padrão que comentamos antes.
```



<ipython-input-19-119511e3d4fa>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/using_indexers.html
cleaned_df['StockCode'] = cleaned_df['StockCode'].astype('str')

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	United Kingdom

511918 rows × 8 columns



Vamos ver se com isso o preço negativo foi corrigido junto:

cleaned_df.describe()

Vemos que não foi o caso, então precisamos arrumar isso também!
Price agora está 0 e quantitv apresenta valor negativo ainda



	Quantity	InvoiceDate	Price	Customer ID
count	511918.000000	511918	511918.000000	405555.000000
mean	11.012285	2010-06-28 20:18:55.216929536	3.385997	15373.275965
min	-9600.000000	2009-12-01 07:45:00	0.000000	12346.000000
25%	1.000000	2010-03-21 14:24:00	1.250000	14004.000000
50%	3.000000	2010-07-06 14:25:00	2.100000	15326.000000
75%	11.000000	2010-10-15 15:06:00	4.210000	16814.000000
max	19152.000000	2010-12-09 20:01:00	1157.150000	18287.000000
std	104.292136	NaN	5.069715	1677.247500



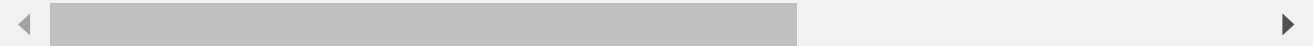
Vamos eliminar os valores nulos:

```
cleaned_df.dropna(subset = ['Customer ID'], inplace = True)
```



<ipython-input-21-0f3a3f958082>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/using_indexing.html
cleaned_df.dropna(subset = ['Customer ID'], inplace = True)



```
cleaned_df.describe()
```

Com isso, é possível perceber que eliminamos o problema dos valores negativos!



	Quantity	InvoiceDate	Price	Customer ID
count	405555.000000	405555	405555.000000	405555.000000
mean	13.624024	2010-07-01 12:37:55.533823744	2.985737	15373.275965
min	1.000000	2009-12-01 07:45:00	0.000000	12346.000000
25%	2.000000	2010-03-26 14:16:00	1.250000	14004.000000
50%	5.000000	2010-07-11 10:28:00	1.950000	15326.000000
75%	12.000000	2010-10-14 17:23:00	3.750000	16814.000000
max	19152.000000	2010-12-09 20:01:00	295.000000	18287.000000
std	97.075664	NaN	4.287946	1677.247500



Conferindo os preços iguais a 0:

```
cleaned_df[cleaned_df['Price'] == 0]
```

```
# Não sabemos exatamente o que são esses preços 0, se são por exemplo itens grátis, mas d
```




	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID
4674	489825	22076	6 RIBBONS EMPIRE	12	2009-12-02 13:34:00	0.0	16126.0
6781	489998	48185	DOOR MAT FAIRY CAKE	2	2009-12-03 11:19:00	0.0	15658.0
18738	490961	22065	CHRISTMAS PUDDING TRINKET POT	1	2009-12-08 15:25:00	0.0	14108.0
18739	490961	22142	CHRISTMAS CRAFT WHITE FAIRY	12	2009-12-08 15:25:00	0.0	14108.0
32916	492079	85042	ANTIQUE LILY FAIRY LIGHTS	8	2009-12-15 13:49:00	0.0	15070.0
40101	492760	21143	ANTIQUE GLASS HEART DECORATION	12	2009-12-18 14:22:00	0.0	18071.0
47126	493761	79320	FLAMINGO LIGHTS	24	2010-01-06 14:54:00	0.0	14258.0
48342	493899	22355	CHARLOTTE BAG , SUKI DESIGN	10	2010-01-08 10:43:00	0.0	12417.0
57619	494607	21533	RETRO SPOT LARGE MILK JUG	12	2010-01-15 12:43:00	0.0	16858.0
111348	500073	21662	VINTAGE GLASS COFFEE CADDY	1	2010-03-04 11:44:00	0.0	13047.0
149201	503585	22459	CAST IRON HOOK GARDEN TROWEL	8	2010-04-01 17:13:00	0.0	13047.0
149202	503585	22458	CAST IRON HOOK GARDEN FORK	8	2010-04-01 17:13:00	0.0	13047.0
166143	505083	22376	AIRLINE BAG VINTAGE JET SET WHITE	1	2010-04-20 09:56:00	0.0	12623.0
232526	511902	21765	HANGING METAL BIRD BATH	1	2010-06-11 11:12:00	0.0	12748.0
240455	512609	20914	SET/5 RED SPOTTY LID	2	2010-06-17 10:40:00	0.0	14045.0

			GLASS BOWLS		10:12:00		
248583	513416	22423	REGENCY CAKESTAND 3 TIER	5	2010-06-24 12:34:00	0.0	13089.0
276858	516304	22690	DOORMAT HOME SWEET HOME BLUE	6	2010-07-19 13:13:00	0.0	14025.0
296375	518231	22472	TV DINNER TRAY DOLLY GIRL	9	2010-08-05 15:28:00	0.0	12471.0
327801	521375	22202	MILK PAN PINK RETROSPOT	3	2010-09-05 11:58:00	0.0	12647.0
358820	524181	46000M	POLYESTER FILLER PAD 45x45cm	648	2010-09-27 16:59:00	0.0	17450.0
364333	524701	22218	CAKE STAND LACE WHITE	2	2010-09-30 12:19:00	0.0	17667.0
392008	527084	22630	DOLLY GIRL LUNCH BOX	64	2010-10-14 15:33:00	0.0	14646.0
400047	527696	22121	NOEL WOODEN BLOCK LETTERS	1	2010-10-18 15:13:00	0.0	13554.0
439309	531361	21843	RED RETROSPOT CAKE STAND	2	2010-11-07 14:26:00	0.0	12820.0
453705	532470	22624	IVORY KITCHEN SCALES	2	2010-11-12 11:41:00	0.0	12647.0
471775	533822	22846	BREAD BIN DINER STYLE RED	1	2010-11-19 09:40:00	0.0	12647.0
471776	533822	22845	VINTAGE CREAM CAT FOOD CONTAINER	1	2010-11-19 09:40:00	0.0	12647.0
512240	537197	22841	ROUND CAKE TIN VINTAGE GREEN	1	2010-12-05 14:02:00	0.0	12647.0

```
len(cleaned_df[cleaned_df['Price'] == 0])
# Vendo o tamanho, podemos reparar que são apenas 28 valores, então realmente não é um pr
```

→ 28

```
cleaned_df = cleaned_df[cleaned_df['Price'] > 0]
```

```
cleaned_df.describe()
```

```
# Vemos que o valor ainda está 0 mesmo tendo sido apagado, vamos conferir melhor isso:
```

→

	Quantity	InvoiceDate	Price	Customer ID
count	405527.000000	405527	405527.000000	405527.000000
mean	13.622846	2010-07-01 12:41:15.852705280	2.985943	15373.365389
min	1.000000	2009-12-01 07:45:00	0.001000	12346.000000
25%	2.000000	2010-03-26 14:16:00	1.250000	14004.000000
50%	5.000000	2010-07-11 10:28:00	1.950000	15326.000000
75%	12.000000	2010-10-14 17:23:00	3.750000	16814.000000
max	19152.000000	2010-12-09 20:01:00	295.000000	18287.000000
std	97.073841	NaN	4.288022	1677.211001

```
cleaned_df['Price'].min()
```

```
# Vemos que há valores bem baixos próximos a 0, como 0,001.
# Nesse caso, vamos deixar da forma que está
```

→ 0.001

```
# Agora que terminamos a parte de limpar, quanto dados 'perdemos'?
```

```
(len(cleaned_df)/ len(df))*100
```

```
# Sobrou cerca de 77% do total de dados que tínhamos ao iniciar o processo de cleaning!
```

→ 77.17547068193453

✓ 5. Feature engineering:

```
# Criando uma coluna para total de transações:
```

```
cleaned_df['SalesLineTotal'] = cleaned_df['Quantity']*cleaned_df['Price']
cleaned_df
```

Vemos agora uma coluna de total!



```
<ipython-input-29-493afa16332b>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/10min.html#copy-on-write>

```
cleaned_df['SalesLineTotal'] = cleaned_df['Quantity']*cleaned_df['Price']
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Col
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	l Kin
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	l Kin
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	l Kin
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	l Kin
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	l Kin
...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	l Kin
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	l Kin
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	l Kin
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	l Kin
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	l Kin

405527 rows × 9 columns



```
# Vamos agregar os nossos dados por Cliente, ou seja, pelo 'CustomerID':
# Esse data-frame vai ter algumas novas colunas:
# MonetaryValue, Frequency
```

```
agg_df = cleaned_df.groupby(by = 'Customer ID', as_index = False) \
    .agg(
        MonetaryValue = ('SalesLineTotal', 'sum'),
        Frequency = ('Invoice', 'nunique'),
        LastInvoiceData = ('InvoiceDate', 'max')
    )
```

```
agg_df.head(10)
```

	Customer ID	MonetaryValue	Frequency	LastInvoiceData
0	12346.0	163.41	2	2010-06-28 13:53:00
1	12347.0	1323.32	2	2010-12-07 14:57:00
2	12348.0	221.16	1	2010-09-27 14:59:00
3	12349.0	2221.14	2	2010-10-28 08:23:00
4	12351.0	300.93	1	2010-11-29 15:23:00
5	12352.0	343.80	2	2010-11-29 10:07:00
6	12353.0	317.76	1	2010-10-27 12:44:00
7	12355.0	488.21	1	2010-05-21 11:59:00
8	12356.0	3126.25	3	2010-11-24 12:24:00
9	12357.0	11229.99	1	2010-11-16 10:05:00

Próximas etapas:

[Gerar código com agg_df](#)



[Ver gráficos recomendados](#)

[New interactive sheet](#)

A coluna Recency representa o número de dias desde a última compra feita por cada cliente até a data mais recente de faturamento no dataset.

Esse conceito é utilizado na análise RFM (Recency, Frequency, Monetary Value) para entender o comportamento dos clientes em termos de:

Recency (Recência): Quando foi a última compra do cliente? Frequency (Frequência): Com que frequência o cliente faz compras? Monetary Value (Valor Monetário): Quanto o cliente gasta em média?

```
# Criando a coluna 'Recency':
```

```
max_invoice_date = agg_df['LastInvoiceData'].max()
max_invoice_date
```

```
agg_df['Recency'] = (max_invoice_date - agg_df['LastInvoiceData']).dt.days # em dias
```

```
agg_df.head(10)
```



	Customer ID	MonetaryValue	Frequency	LastInvoiceData	Recency
0	12346.0	163.41	2	2010-06-28 13:53:00	164
1	12347.0	1323.32	2	2010-12-07 14:57:00	2
2	12348.0	221.16	1	2010-09-27 14:59:00	73
3	12349.0	2221.14	2	2010-10-28 08:23:00	42
4	12351.0	300.93	1	2010-11-29 15:23:00	10
5	12352.0	343.80	2	2010-11-29 10:07:00	10
6	12353.0	317.76	1	2010-10-27 12:44:00	43
7	12355.0	488.21	1	2010-05-21 11:59:00	202
8	12356.0	3126.25	3	2010-11-24 12:24:00	15
9	12357.0	11229.99	1	2010-11-16 10:05:00	23



Próximas etapas:

[Gerar código com agg_df](#)



[Ver gráficos recomendados](#)

[New interactive sheet](#)

Sabendo que outliers podem ser um problema para o nosso modelo, vamos tentar lidar com

Plotando todos as variáveis com histogramas:

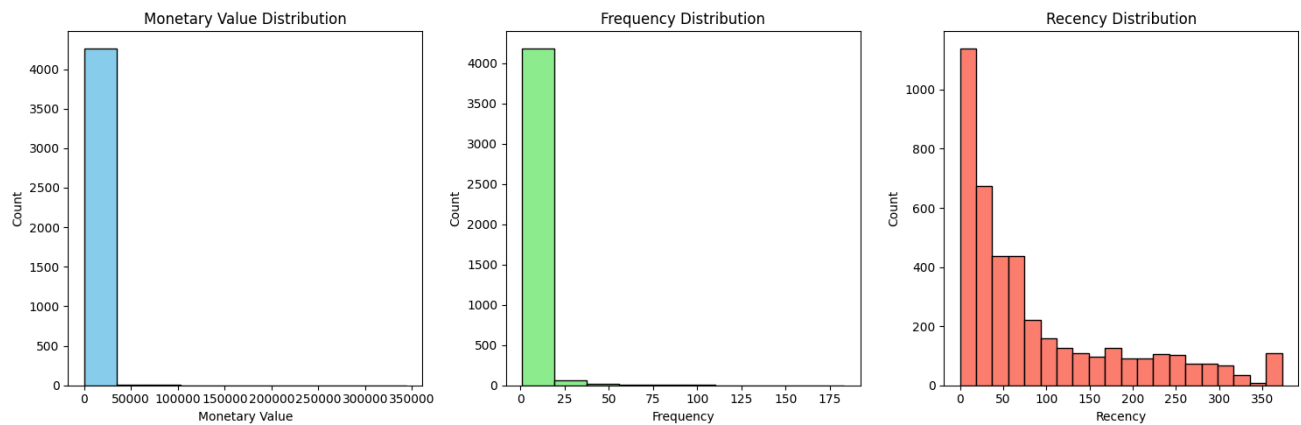
```
plt.figure(figsize=(15, 5))
```

```
plt.subplot(1, 3, 1)
plt.hist(agg_df['MonetaryValue'], bins=10, color='skyblue', edgecolor='black')
plt.title('Monetary Value Distribution')
plt.xlabel('Monetary Value')
plt.ylabel('Count')
```

```
plt.subplot(1, 3, 2)
plt.hist(agg_df['Frequency'], bins=10, color='lightgreen', edgecolor='black')
plt.title('Frequency Distribution')
plt.xlabel('Frequency')
plt.ylabel('Count')
```

```
plt.subplot(1, 3, 3)
plt.hist(agg_df['Recency'], bins=20, color='salmon', edgecolor='black')
plt.title('Recency Distribution')
plt.xlabel('Recency')
plt.ylabel('Count')
```

```
plt.tight_layout()
plt.show()
```



Nos podemos ver que nos dois primeiros casos, temos alguns poucos outliers, grande parte dos dados está nas primeiras volunas. Por outro lado, a coluna de 'Recency' parece estar dentro dos padrões que esperamos.

Usando um boxplot para investigar melhor os outliers:

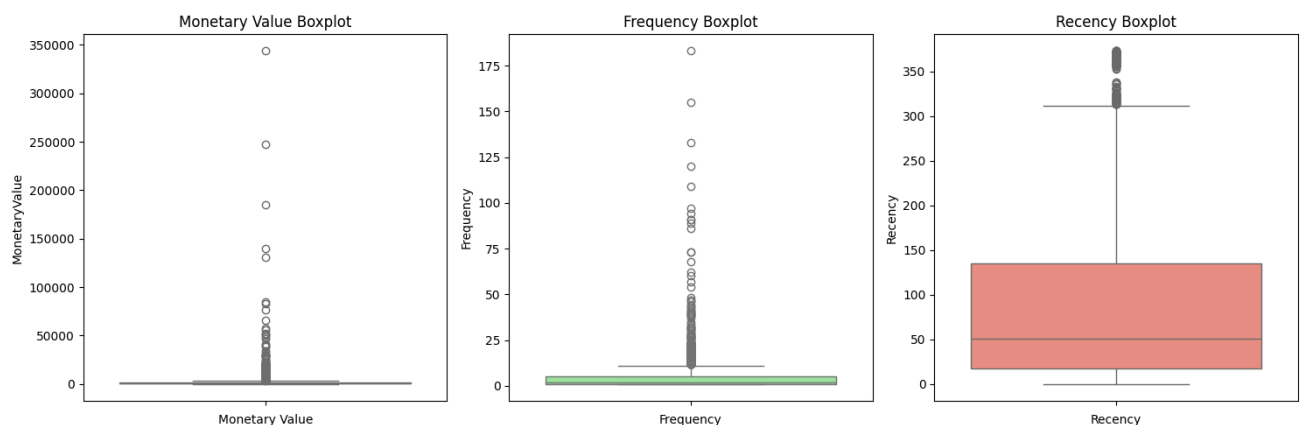
```
plt.figure(figsize=(15, 5))
```

```
plt.subplot(1, 3, 1)
sns.boxplot(data=agg_df['MonetaryValue'], color='skyblue')
plt.title('Monetary Value Boxplot')
plt.xlabel('Monetary Value')
```

```
plt.subplot(1, 3, 2)
sns.boxplot(data=agg_df['Frequency'], color='lightgreen')
plt.title('Frequency Boxplot')
plt.xlabel('Frequency')
```

```
plt.subplot(1, 3, 3)
sns.boxplot(data=agg_df['Recency'], color='salmon')
plt.title('Recency Boxplot')
plt.xlabel('Recency')
```

```
plt.tight_layout()
plt.show()
```



Confirmando o que vimos anteriormente, temos grande parte dos valores em uma faixa única para as duas primeiras variáveis e alguns outliers muito fora disso, já a 3 temos uma quantidade menor de outliers, o que condiz com a sua distribuição.

Como os outliers nesse caso são extremamente importantes, pois representam clientes que gastaram muito e frequentemente, nós não podemos simplesmente remover eles, eles são os mais valiosos! Por isso vamos separar os dois e conduzir duas análises separadas.

```
# Vamos separar os alcances dos quartis:
# Monetary Value:
M_Q1 = agg_df['MonetaryValue'].quantile(0.25) # 1 quartil corresponde a 25% dos dados.
M_Q1
```

⇒ 307.1

```
M_Q3 = agg_df['MonetaryValue'].quantile(0.75) # 3 quartil corresponde a 75% dos dados.
M_Q3
```

⇒ 1702.98

```
# Alcance interquartilico:
```

```
M_IQR = M_Q3 - M_Q1
M_IQR
```

⇒ 1395.88

```
# Pegando todos os dados desse intervalo:
```

```
#top_range
monetary_outliers_df = agg_df[(agg_df['MonetaryValue'] > (M_Q3 + 1.5* M_IQR)) | (agg_df['
#botton_range - Não acho que vá ter pois a maioria dos dados é abaixo mais vamos adiciona
```

```
monetary_outliers_df.describe()
```




	Customer ID	MonetaryValue	Frequency	LastInvoiceData	Recency
count	421.000000	421.000000	421.000000	421	421.000000
mean	15107.242280	12150.487568	17.225653	2010-11-09 16:56:38.764845568	29.847981
min	12357.000000	3796.930000	1.000000	2009-12-10 18:03:00	0.000000
25%	13615.000000	4590.530000	8.000000	2010-11-08 15:42:00	3.000000
50%	15005.000000	6191.320000	12.000000	2010-11-26 12:44:00	13.000000
75%	16700.000000	10164.490000	18.000000	2010-12-06 11:06:00	31.000000
max	18260.000000	343764.350000	183.000000	2010-12-09 19:32:00	364.000000
std	1731.684418	25632.405012	19.758275	NaN	51.563698



Fazendo isso para FrequencyValues:

```
F_Q1 = agg_df['Frequency'].quantile(0.25)
```

```
F_Q3 = agg_df['Frequency'].quantile(0.75)
```

```
F_IQR = F_Q3 - F_Q1
```

```
frequency_outliers_df = agg_df[(agg_df['Frequency'] > (F_Q3 + 1.5 * F_IQR)) | (ag
```

```
frequency_outliers_df.describe()
```





	Customer ID	MonetaryValue	Frequency	LastInvoiceData	Recency
count	279.000000	279.000000	279.000000	279	279.000000
mean	15352.655914	14309.816724	23.810036	2010-11-23 11:06:20.645161216	16.089606
min	12437.000000	1088.440000	12.000000	2010-05-12 16:51:00	0.000000
25%	13800.000000	4321.670000	13.000000	2010-11-20 13:14:30	2.000000
50%	15465.000000	6590.060000	17.000000	2010-12-02 10:46:00	7.000000
75%	16828.500000	11692.405000	23.000000	2010-12-07 11:08:30	19.000000
max	18260.000000	343764.350000	183.000000	2010-12-09 19:32:00	211.000000
std	1748.429987	31069.985754	21.932937	NaN	26.589117



m novo df para os dados que não são outliers:

```
s_df = agg_df[(~agg_df.index.isin(monetary_outliers_df.index)) & (~agg_df.index.i
s_df.describe()
```

embora ainda tenha uma certa variabilidade, já está bem melhor do que antes!



	Customer ID	MonetaryValue	Frequency	LastInvoiceData	Recency
count	3811.000000	3811.000000	3811.000000	3811	3811.000000
mean	15375.875098	884.438897	2.864602	2010-09-03 11:37:33.077932288	97.067699
min	12346.000000	1.550000	1.000000	2009-12-01 10:49:00	0.000000
25%	13912.500000	277.925000	1.000000	2010-07-08 16:27:00	22.000000
50%	15387.000000	587.690000	2.000000	2010-10-12 16:25:00	58.000000
75%	16851.000000	1269.070000	4.000000	2010-11-17 13:14:00	154.000000
max	18287.000000	3788.210000	11.000000	2010-12-09 20:01:00	373.000000
std	1692.963969	816.814742	2.244160	NaN	98.089381



```
# Plotando o box plot para esses dados (sem outliers):
```

```
plt.figure(figsize=(15, 5))

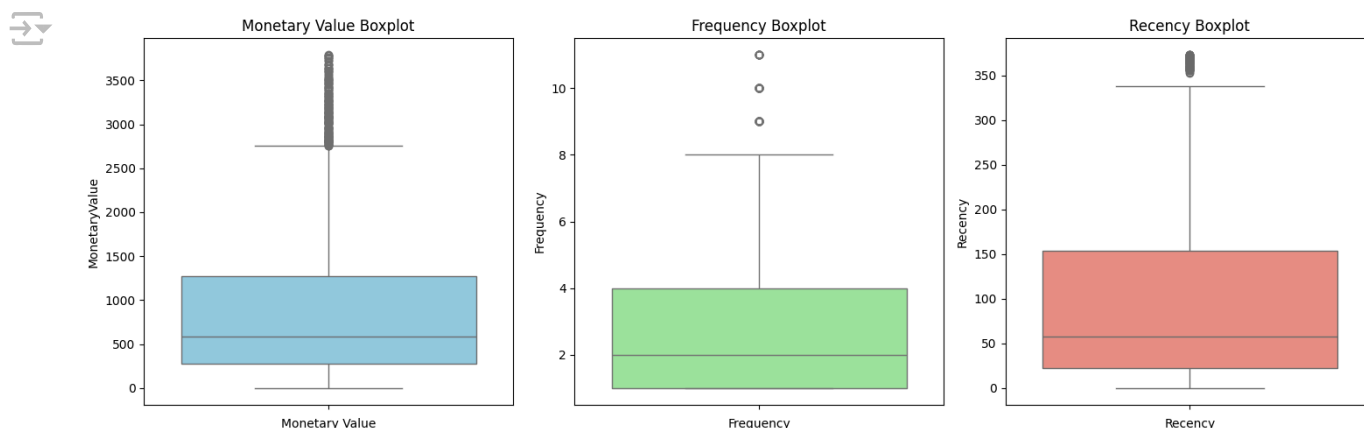
plt.subplot(1, 3, 1)
sns.boxplot(data=non_outliers_df['MonetaryValue'], color='skyblue')
plt.title('Monetary Value Boxplot')
plt.xlabel('Monetary Value')

plt.subplot(1, 3, 2)
sns.boxplot(data=non_outliers_df['Frequency'], color='lightgreen')
plt.title('Frequency Boxplot')
plt.xlabel('Frequency')

plt.subplot(1, 3, 3)
sns.boxplot(data=non_outliers_df['Recency'], color='salmon')
plt.title('Recency Boxplot')
plt.xlabel('Recency')

plt.tight_layout()
plt.show()
```

```
# Vemos que ainda há alguns outliers, mas está muito mais tolerável do que antes!
```



```
# Plotando um gráfico em 3 dimensões desses dados:
```

```
fig = plt.figure(figsize=(8, 8))

ax = fig.add_subplot(projection="3d")

scatter = ax.scatter(non_outliers_df["MonetaryValue"], non_outliers_df["Frequency"], non_

ax.set_xlabel('Monetary Value')
ax.set_ylabel('Frequency')
ax.set_zlabel('Recency')

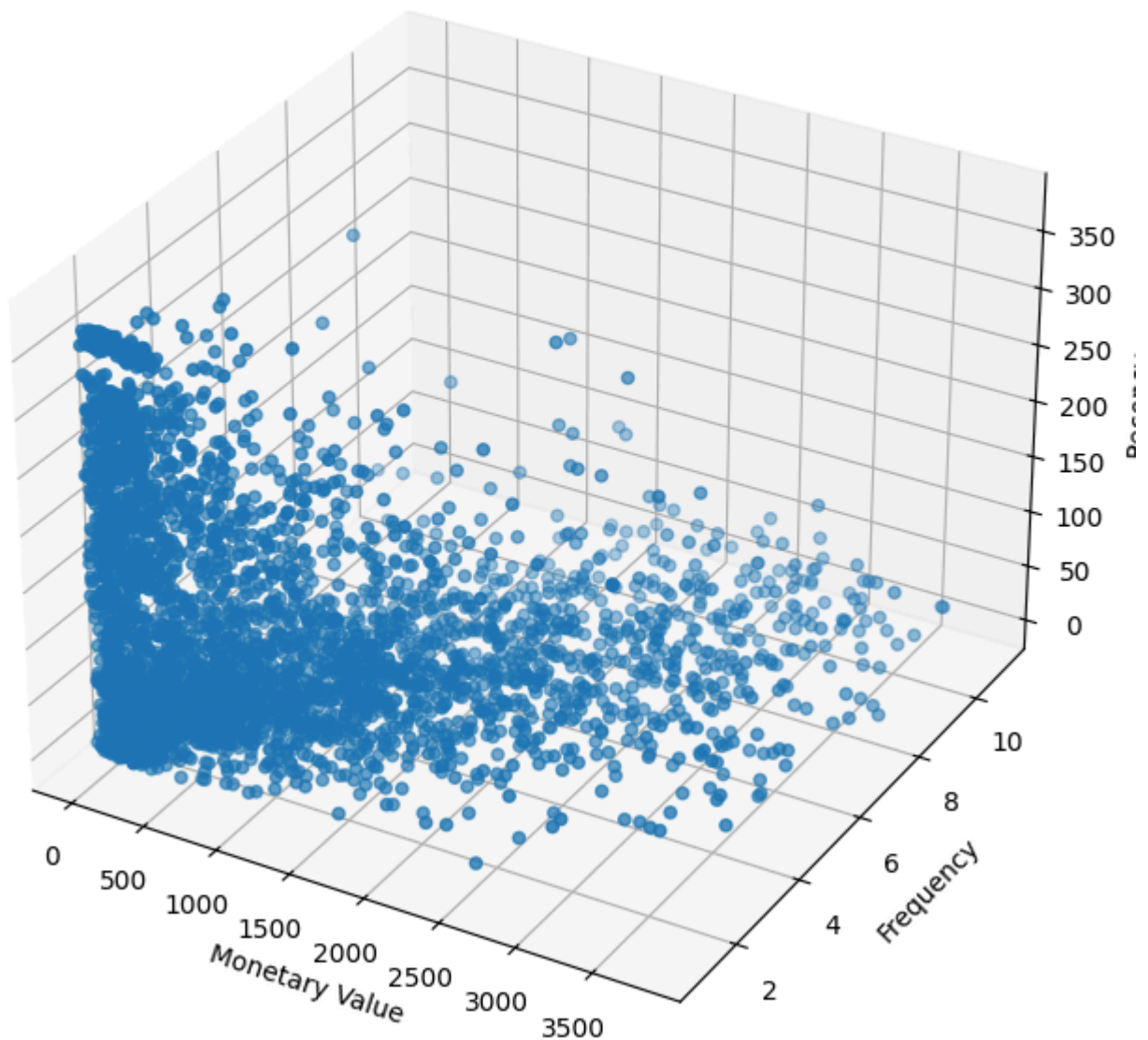
ax.set_title('3D Scatter Plot of Customer Data')

plt.show()
```

```
# Essa visualização é muito importante para ter uma noção de como esses gráficos serão cl
```



3D Scatter Plot of Customer Data



Repare que o matplotlib já coloca em uma escala, mas para o K-means, a gente precisa de

6. Criando escalas para os dados

✓ Vamos usar o Standard Scaling

Escalação padrão (Standard Scaling) transforma as características (features) dos seus dados para terem uma média de 0 e um desvio padrão de 1, garantindo que cada característica contribua igualmente para a análise.

A fórmula utilizada para o escalonamento padrão é:

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean

σ = Standard Deviation

Onde:

- **z** é o valor padronizado,
- **x** é o valor original,
- **μ** é a média da característica,
- **σ** é o desvio padrão da característica.

```
scaler = StandardScaler()
```

```
scaled_data = scaler.fit_transform(non_outliers_df[['MonetaryValue', 'Frequency']],
scaled_data
```

```
# Agora vemos que todos os nossos dados estão de fato, na mesma escala
```

```
array([[ -0.88284831, -0.38531824,  0.68244986],
       [ 0.53737852, -0.38531824, -0.96932179],
       [-0.81213757, -0.83097766, -0.24539718],
       ...,
       [-0.62132263, -0.83097766,  2.01814175],
       [ 0.44323221, -0.38531824,  0.14205543],
       [ 1.72800052,  0.50600061, -0.81637997]])
```

```
# Convertendo o array em data frame novamente:
```

```
scaled_data_df = pd.DataFrame(scaled_data, index = non_outliers_df.index, columns = ('Mon
scaled_data_df
```



	MonetaryValue	Frequency	Recency	
0	-0.882848	-0.385318	0.682450	
1	0.537379	-0.385318	-0.969322	
2	-0.812138	-0.830978	-0.245397	
3	1.636695	-0.385318	-0.561477	
4	-0.714465	-0.830978	-0.887753	
...	
4280	-0.297131	1.397319	-0.816380	
4281	-0.578859	-0.830978	-0.316770	
4282	-0.621323	-0.830978	2.018142	
4283	0.443232	-0.385318	0.142055	
4284	1.728001	0.506001	-0.816380	

3811 rows × 3 columns

Próximas etapas:

[código](#) [scaled_data_df](#)[Ver graficos recomendados](#)[New interactive sheet](#)

Plotando novamente os dados, mas em escala:

```
fig = plt.figure(figsize=(8, 8))
```

```
ax = fig.add_subplot(projection="3d")
```

```
scatter = ax.scatter(scaled_data_df["MonetaryValue"], scaled_data_df["Frequency"], scaled
```

```
ax.set_xlabel('Monetary Value')
```

```
ax.set_ylabel('Frequency')
```

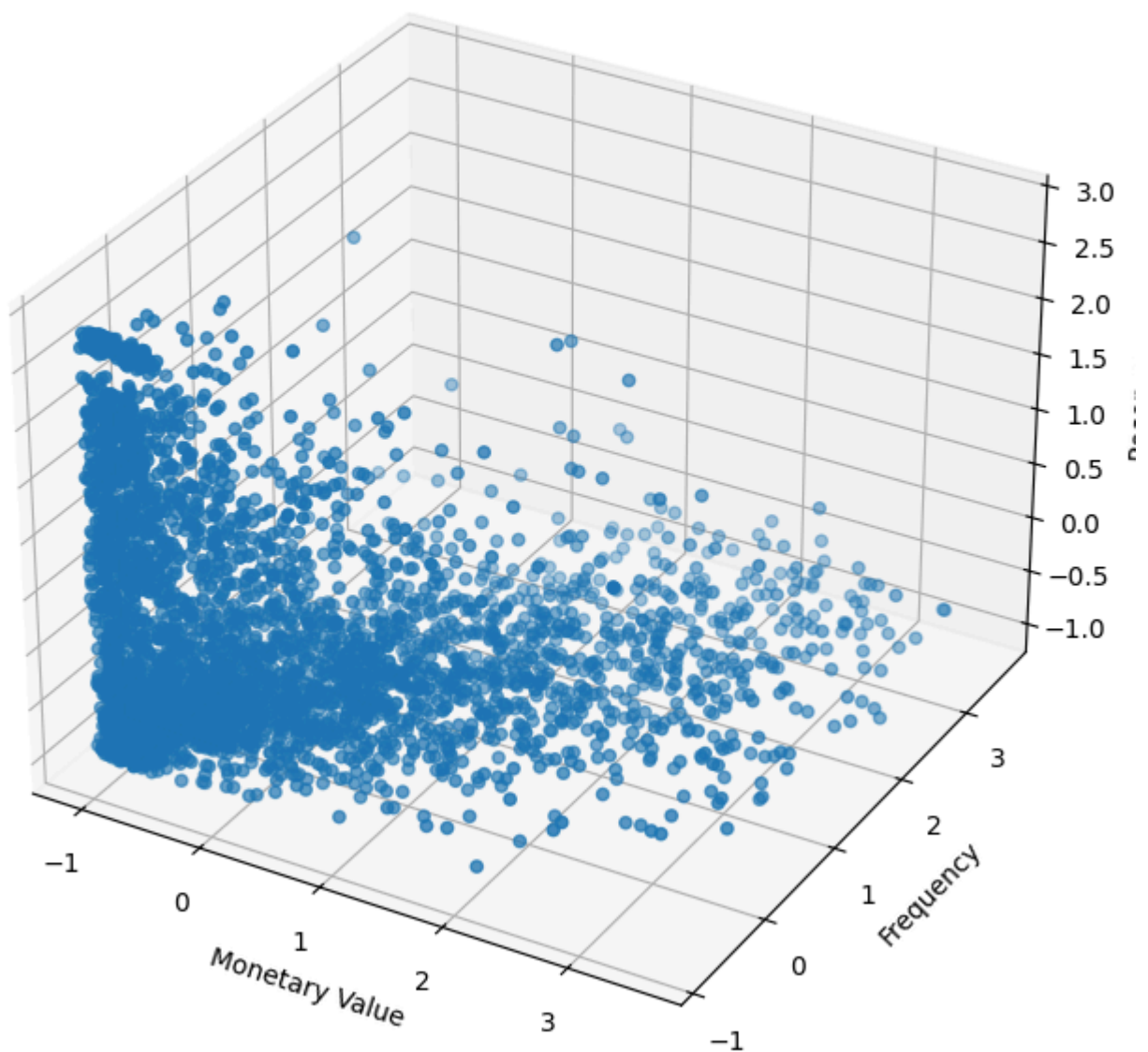
```
ax.set_zlabel('Recency')
```

```
ax.set_title('3D Scatter Plot of Customer Data')
```

```
plt.show()
```



3D Scatter Plot of Customer Data



✓ 7. K-means clustering:

Vamos testar varios valores de K e ver com base na inercia, qual seria o melhor

```
max_k = 12
```

```
inertia = []
```

```
k_values = range(2, max_k + 1)
```

```
for k in k_values:
```

```
    kmeans = KMeans(n_clusters = k, random_state = 42, max_iter = 1000)
```

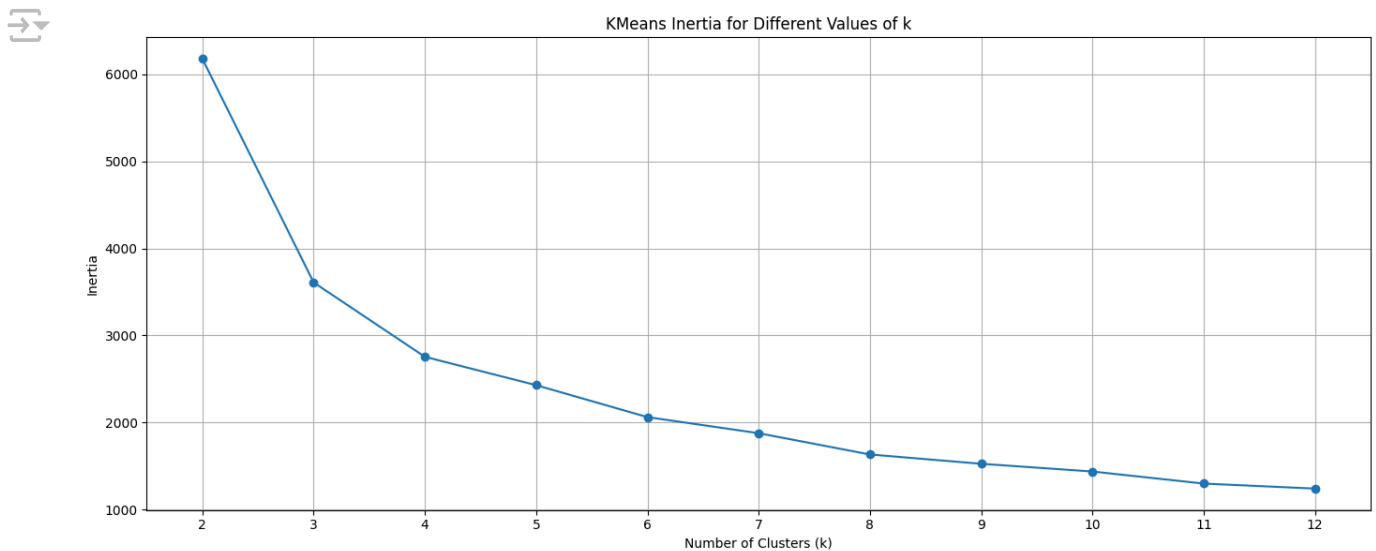
```
    kmeans.fit_predict(scaled_data_df)
```

```
    inertia.append(kmeans.inertia_)
```

```
plt.figure(figsize=(14,6))

plt.plot(k_values, inertia, marker = 'o')
plt.title('KMeans Inertia for Different Values of k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_values)
plt.grid(True)

plt.tight_layout()
plt.show()
```



O gráfico apresentado mostra a inércia do KMeans para diferentes valores de k (número de clusters) e é utilizado para determinar o número ideal de clusters por meio do método do "cotovelo". A inércia representa a soma das distâncias quadradas entre cada ponto de dados e o centróide mais próximo. Observa-se que, à medida que k aumenta, a inércia diminui, pois os clusters se tornam mais específicos e próximos dos dados. No entanto, essa redução na inércia começa a desacelerar significativamente em torno de $k = 4$, indicando o ponto de cotovelo. Esse ponto é geralmente considerado o número ideal de clusters, pois balanceia a compactação dos clusters com a simplicidade do modelo, evitando a sobresegmentação dos dados. Portanto, com base no gráfico, o número ideal de clusters parece ser entre 4 e 5.

Para decididr entre 4 e 5, nos usamos o Silhouette Score:

A pontuação de silhueta é calculada usando a seguinte fórmula:

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Onde: $s(i)$ é a pontuação de silhueta para uma única amostra i ; $a(i)$ é a distância média entre i e todos os outros pontos no mesmo cluster; $b(i)$ é a distância média mínima entre i e todos os pontos no cluster mais próximo ao qual i não pertence. A pontuação de silhueta varia entre $[-1, 1]$, sendo que valores mais altos indicam clusters mais distintos e bem definidos.

```
max_k = 12

inertia = []
silhoutte_scores = []
k_values = range(2, max_k + 1)

for k in k_values:

    kmeans = KMeans(n_clusters=k, random_state=42, max_iter=1000)

    cluster_labels = kmeans.fit_predict(scaled_data_df)

    sil_score = silhouette_score(scaled_data_df, cluster_labels)

    silhoutte_scores.append(sil_score)

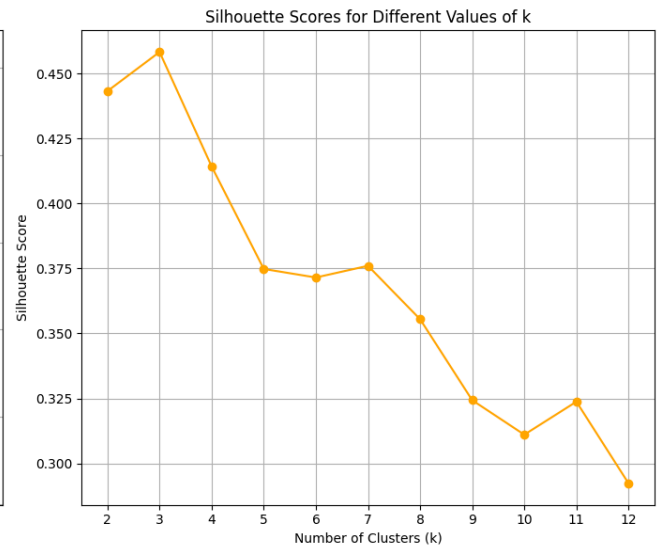
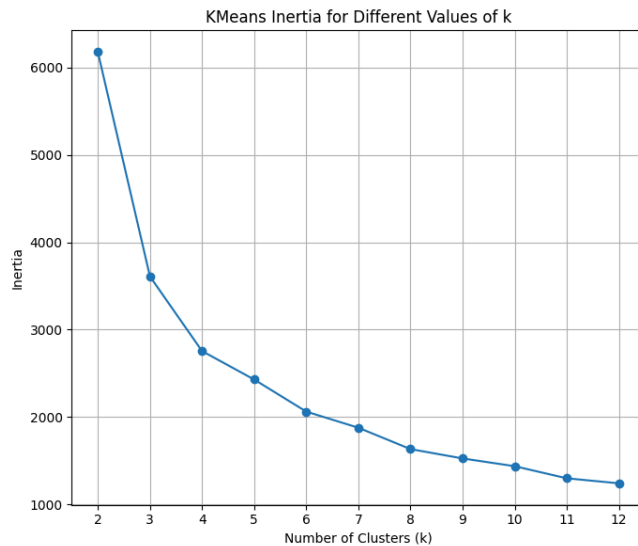
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(k_values, inertia, marker='o')
plt.title('KMeans Inertia for Different Values of k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_values)
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(k_values, silhoutte_scores, marker='o', color='orange')
plt.title('Silhouette Scores for Different Values of k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.xticks(k_values)
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```



Um score alto score é bom, é um baixo é ruim, onde 1 é perfeito.

Os gráficos comparam a inércia e o Silhouette Score para diferentes valores de clusters (k). A inércia diminui conforme k aumenta, sugerindo k = 4 como ponto de cotovelo. Já o Silhouette Score alcança o pico em k = 3, indicando clusters bem separados. Portanto, ambos os gráficos sugerem que entre 3 e 4 clusters seria uma escolha adequada para um bom agrupamento, mas como o primeiro gráfico indica 4, vamos seguir com K = 4!

✓ 8. 4-means clustering:

Agora podemos criar nosso modelo:

```
kmeans = KMeans(n_clusters = 4, random_state = 42, max_iter = 1000)
```

```
cluster_labels = kmeans.fit_predict(scaled_data_df)
```

```
cluster_labels
```

```
array([8, 5, 3, ..., 7, 5, 4], dtype=int32)
```

Adicionando as clusters labels ao nosso conjunto de dados original:

```
non_outliers_df['Cluster'] = cluster_labels
```

```
non_outliers_df
```



```
<ipython-input-61-21a8a4f2cad0>:3: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/using_hierarchical_index.html#using-hierarchical-index
non_outliers_df['Cluster'] = cluester_labels

	Customer ID	MonetaryValue	Frequency	LastInvoiceData	Recency	Cluster
0	12346.0	163.41	2	2010-06-28 13:53:00	164	0
1	12347.0	1323.32	2	2010-12-07 14:57:00	2	1
2	12348.0	221.16	1	2010-09-27 14:59:00	73	3
3	12349.0	2221.14	2	2010-10-28 08:23:00	42	1
4	12351.0	300.93	1	2010-11-29 15:23:00	10	3
...
4280	18283.0	641.77	6	2010-11-22 15:30:00	17	1
4281	18284.0	411.68	1	2010-10-04 11:33:00	66	3
4282	18285.0	377.00	1	2010-02-17 10:24:00	295	0
4283	18286.0	1246.43	2	2010-08-20 11:57:00	111	1
4284	18287.0	2295.71	4	2010-11-22 11:51:00	17	1

3811 rows × 6 columns



Próximas etapas:

código

non_outliers_df

Ver gráficos recomendados

New interactive sheet

Vemos que agora temos uma coluna so para os cluster aos quais os dados foram classificados!

9. Vizualizando os dados e tirando conclusões sobre eles:

Plotando o gráfico separando por cores:

```
cluster_colors = {0: '#1f77b4', # Blue  
                  1: '#ff7f0e', # Orange  
                  2: '#2ca02c', # Green
```

```
3: '#d62728'} # Red
```

```
colors = non_outliers_df['Cluster'].map(cluster_colors)

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(projection='3d')

scatter = ax.scatter(non_outliers_df['MonetaryValue'],
                     non_outliers_df['Frequency'],
                     non_outliers_df['Recency'],
                     c=colors, # Use mapped solid colors
                     marker='o')

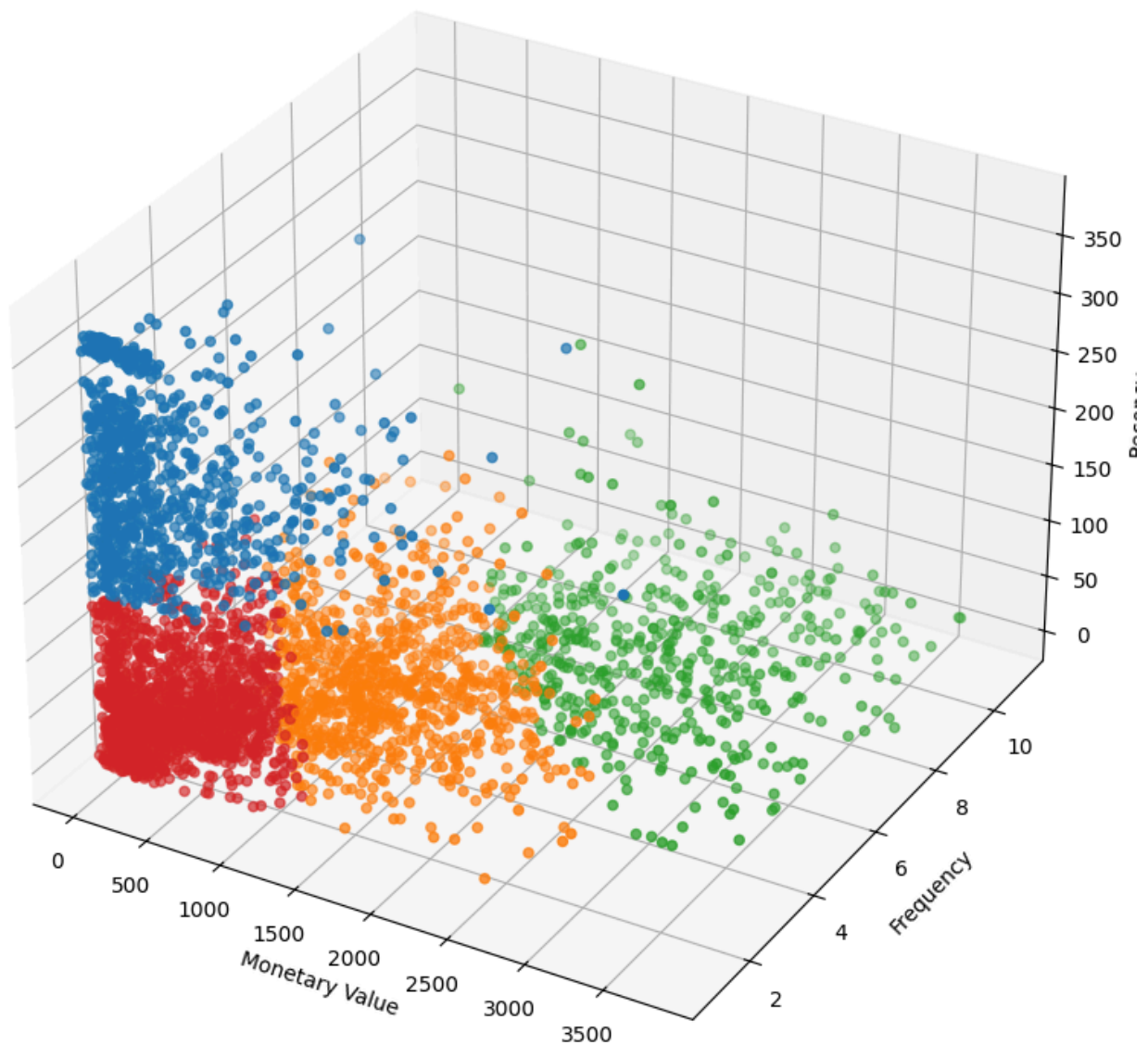
ax.set_xlabel('Monetary Value')
ax.set_ylabel('Frequency')
ax.set_zlabel('Recency')

ax.set_title('3D Scatter Plot of Customer Data by Cluster')

plt.show()
```



3D Scatter Plot of Customer Data by Cluster



Violin Plots para cada variável

```
plt.figure(figsize=(12, 18))
```

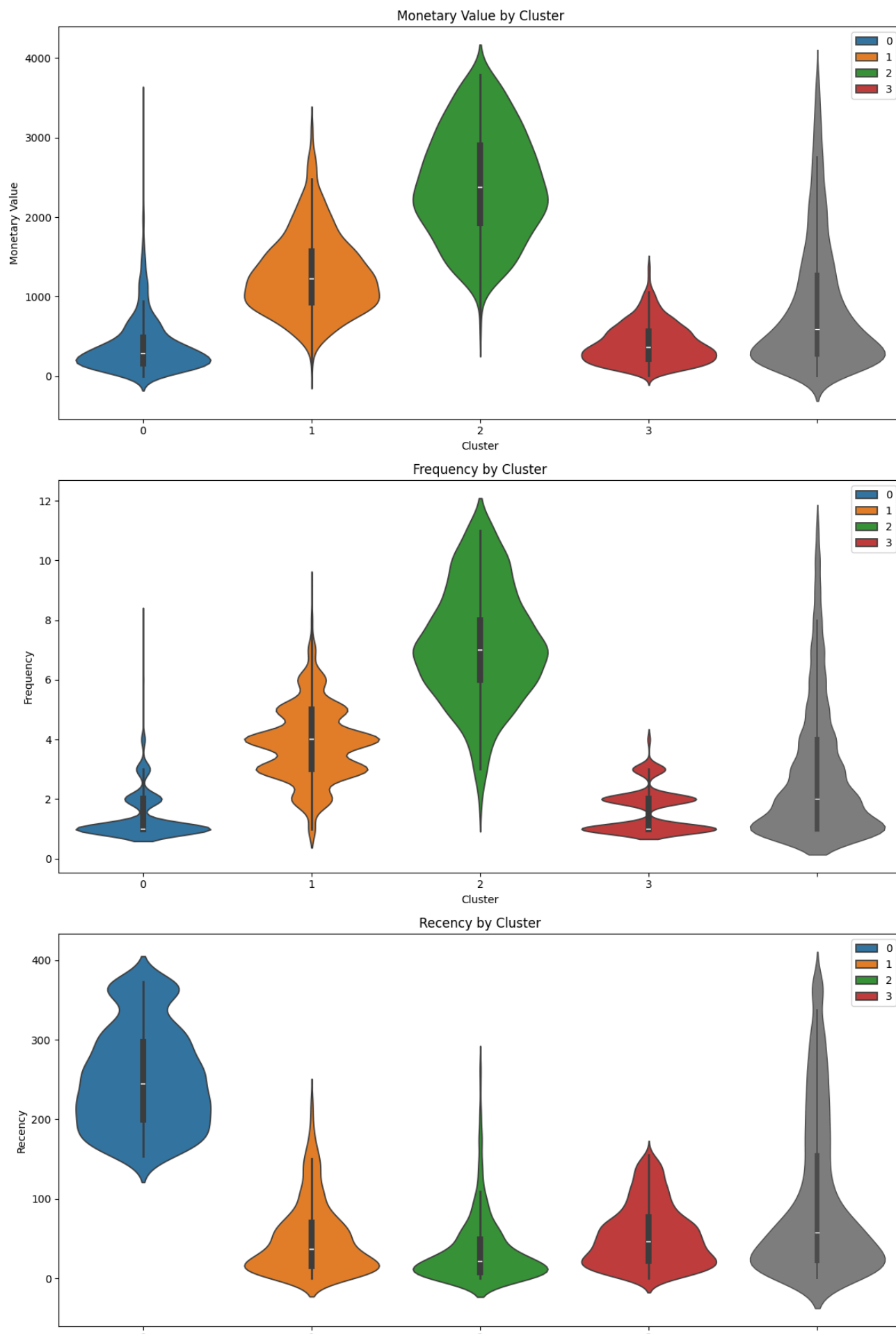
```
plt.subplot(3, 1, 1)
sns.violinplot(x=non_outliers_df['Cluster'], y=non_outliers_df['MonetaryValue'], palette=
sns.violinplot(y=non_outliers_df['MonetaryValue'], color='gray', linewidth=1.0)
plt.title('Monetary Value by Cluster')
plt.ylabel('Monetary Value')
```

```
plt.subplot(3, 1, 2)
sns.violinplot(x=non_outliers_df['Cluster'], y=non_outliers_df['Frequency'], palette=clus
sns.violinplot(y=non_outliers_df['Frequency'], color='gray', linewidth=1.0)
plt.title('Frequency by Cluster')
plt.ylabel('Frequency')
```

```
plt.subplot(3, 1, 3)
sns.violinplot(x=non_outliers_df['Cluster'], y=non_outliers_df['Recency'], palette=cluste
```

```
sns.violinplot(y=non_outliers_df['Recency'], color='gray', linewidth=1.0)
plt.title('Recency by Cluster')
plt.ylabel('Recency')

plt.tight_layout()
plt.show()
```



Os gráficos de violino mostram a distribuição dos valores monetários, frequência e recência para cada cluster identificado. O Cluster 2 se destaca com os maiores valores monetários e de

frequência, indicando clientes de alto valor e engajamento. O Cluster 3 apresenta os menores valores, indicando clientes menos ativos. O Cluster 0 possui a maior recência, sugerindo clientes menos recentes. Cada cluster tem um perfil distinto que pode guiar estratégias de marketing personalizadas.

✓ Análise de Clusters e Estratégias de Ação

Cluster 0 (Azul): "Retenção"

Justificativa:

Esse cluster representa clientes de **alto valor** que compram regularmente, embora não muito recentemente. O foco deve ser em mantê-los engajados para garantir sua lealdade e o nível de gastos.

Ação:

- Implementar **programas de fidelidade**.
 - Oferecer **ofertas personalizadas**.
 - Realizar **comunicação frequente** para garantir que continuem ativos.
-

Cluster 1 (Laranja): "Reativação"

Justificativa:

Esse grupo inclui clientes de **menor valor** que compram com **pouca frequência** e não fizeram **compras recentes**. O foco deve ser em trazê-los de volta ao ciclo de compras.

Ação:

- Utilizar **campanhas de marketing direcionadas**.
 - Oferecer **descontos especiais**.
 - Enviar **lembretes** para incentivá-los a voltar a comprar.
-

Cluster 2 (Verde): "Nutrição"

Justificativa:

Esse cluster representa clientes **menos ativos** e de **menor valor**, mas que fizeram **compras recentes**. Eles podem ser **novos clientes** ou precisar de mais atenção para aumentar o engajamento e os gastos.

Ação:

- Focar no **desenvolvimento do relacionamento** com esses clientes.
 - Oferecer um **excelente atendimento ao cliente**.
 - Criar **incentivos** para aumentar a frequência de compras.
-

Cluster 3 (Vermelho): "Recompensa"

Justificativa:

Esse cluster inclui clientes de **alto valor** que compram com **muita frequência** e continuam **ativos**. Eles são os clientes mais leais, e o foco deve

10. Lidando com os outliers de maneira separada:

```
monetary_outliers_df
frequency_outliers_df
```

	Customer ID	MonetaryValue	Frequency	LastInvoiceData	Recency
65	12437.0	6834.99	20	2010-11-09 14:46:00	30
84	12471.0	17721.45	44	2010-11-30 14:35:00	9
85	12472.0	10426.48	13	2010-12-05 14:19:00	4
92	12482.0	21941.72	27	2010-05-12 16:51:00	211
115	12523.0	2330.38	12	2010-11-30 12:31:00	9
...
4236	18225.0	7545.14	15	2010-12-09 15:46:00	0
4237	18226.0	6650.83	15	2010-11-26 15:51:00	13
4241	18231.0	4791.80	23	2010-10-29 14:17:00	41
4250	18245.0	3757.92	13	2010-11-25 16:52:00	14
4262	18260.0	7318.91	17	2010-11-30 12:25:00	9

279 rows × 5 columns

Próximas etapas:

código frequency_outliers_df

Ver gráficos recomendados

New interactive sheet

```
overlap_indices = monetary_outliers_df.index.intersection(frequency_outliers_df.i

monetary_only_outliers = monetary_outliers_df.drop(overlap_indices)
frequency_only_outliers = frequency_outliers_df.drop(overlap_indices)
monetary_and_frequency_outliers = monetary_outliers_df.loc[overlap_indices]

monetary_only_outliers['Cluster'] = -1
frequency_only_outliers['Cluster'] = -2
monetary_and_frequency_outliers['Cluster'] = -3

outlier_clusters_df = pd.concat([monetary_only_outliers, frequency_only_outliers,

outlier_clusters_df
```



	Customer ID	MonetaryValue	Frequency	LastInvoiceData	Recency	Cluster
9	12357.0	11229.99	1	2010-11-16 10:05:00	23	-1
25	12380.0	4782.84	4	2010-08-31 14:54:00	100	-1
42	12409.0	12346.62	4	2010-10-15 10:24:00	55	-1
48	12415.0	19468.84	4	2010-11-29 15:07:00	10	-1
61	12431.0	4145.52	11	2010-12-01 10:03:00	8	-1
...
4235	18223.0	7409.21	12	2010-11-17 12:20:00	22	-3
4236	18225.0	7545.14	15	2010-12-09 15:46:00	0	-3
4237	18226.0	6650.83	15	2010-11-26 15:51:00	13	-3
4241	18231.0	4791.80	23	2010-10-29 14:17:00	41	-3
4262	18260.0	7318.91	17	2010-11-30 12:25:00	9	-3

474 rows × 6 columns



Próximas etapas:

`código outlier_clusters_df`



Ver gráficos recomendados

New interactive sheet

11. Vizualizando os dados e tirando conclusões sobre eles (outliers):

```
cluster_colors = {-1: '#9467bd',
                  -2: '#8c564b',
                  -3: '#e377c2'}

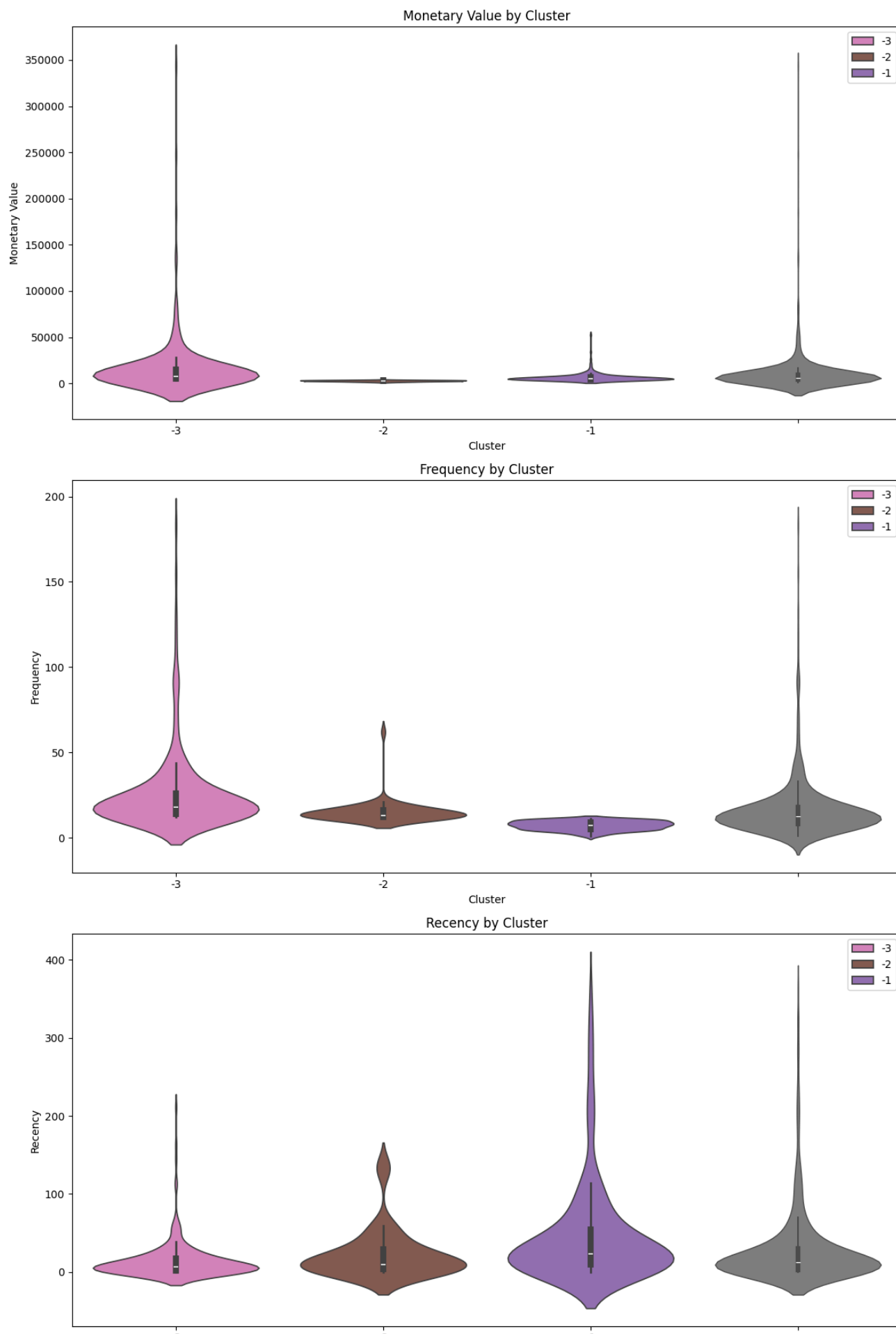
plt.figure(figsize=(12, 18))

plt.subplot(3, 1, 1)
sns.violinplot(x=outlier_clusters_df['Cluster'], y=outlier_clusters_df['MonetaryValue'],
sns.violinplot(y=outlier_clusters_df['MonetaryValue'], color='gray', linewidth=1.0)
plt.title('Monetary Value by Cluster')
plt.ylabel('Monetary Value')
```

```
plt.subplot(3, 1, 2)
sns.violinplot(x=outlier_clusters_df['Cluster'], y=outlier_clusters_df['Frequency'], palette='palegray', linewidth=1.0)
plt.title('Frequency by Cluster')
plt.ylabel('Frequency')

plt.subplot(3, 1, 3)
sns.violinplot(x=outlier_clusters_df['Cluster'], y=outlier_clusters_df['Recency'], palette='palegray', linewidth=1.0)
plt.title('Recency by Cluster')
plt.ylabel('Recency')

plt.tight_layout()
plt.show()
```



✓ Análise dos Clusters de Outliers e Estratégias de Ação