NOTEBOOK DESTINADO A REGISTRAR AS TAREFAS DA DISCIPLINA DE AEDI - 1º/2025

PROFESSOR: JOÃO GABRIEL DE MORAES SOUZA

ALUNO: GUSTAVO PARREIRA LIMA CUNHA

PROVA

QUESTÃO 1

1. Análise Descritiva dos Dados (20%)

In [1]:
```
import pandas as pd

# Especificação do caminho do arquivo csv

path = '/content/drive/MyDrive/AEDI/kc_house_data.csv'

dados = pd.read_csv(path)

# teste de exibição do df

print(dados)
```

```
                id             date     price  bedrooms  bathrooms  \
0       7129300520  20141013T000000  221900.0         3       1.00
1       6414100192  20141209T000000  538000.0         3       2.25
2       5631500400  20150225T000000  180000.0         2       1.00
3       2487200875  20141209T000000  604000.0         4       3.00
4       1954400510  20150218T000000  510000.0         3       2.00
...            ...              ...       ...       ...        ...
21608    263000018  20140521T000000  360000.0         3       2.50
21609   6600060120  20150223T000000  400000.0         4       2.50
21610   1523300141  20140623T000000  402101.0         2       0.75
21611    291310100  20150116T000000  400000.0         3       2.50
21612   1523300157  20141015T000000  325000.0         2       0.75

       sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
0             1180      5650     1.0           0     0  ...      7
1             2570      7242     2.0           0     0  ...      7
2              770     10000     1.0           0     0  ...      6
3             1960      5000     1.0           0     0  ...      7
4             1680      8080     1.0           0     0  ...      8
...            ...       ...     ...         ...   ...  ...    ...
21608         1530      1131     3.0           0     0  ...      8
21609         2310      5813     2.0           0     0  ...      8
21610         1020      1350     2.0           0     0  ...      7
21611         1600      2388     2.0           0     0  ...      8
21612         1020      1076     2.0           0     0  ...      7

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
0            1180              0      1955             0    98178  47.5112
1            2170            400      1951          1991    98125  47.7210
2             770              0      1933             0    98028  47.7379
3            1050            910      1965             0    98136  47.5208
4            1680              0      1987             0    98074  47.6168
...           ...            ...       ...           ...      ...      ...
21608        1530              0      2009             0    98103  47.6993
21609        2310              0      2014             0    98146  47.5107
21610        1020              0      2009             0    98144  47.5944
21611        1600              0      2004             0    98027  47.5345
21612        1020              0      2008             0    98144  47.5941

          long  sqft_living15  sqft_lot15
0     -122.257           1340        5650
1     -122.319           1690        7639
2     -122.233           2720        8062
3     -122.393           1360        5000
4     -122.045           1800        7503
...        ...            ...         ...
21608 -122.346           1530        1509
21609 -122.362           1830        7200
21610 -122.299           1020        2007
21611 -122.069           1410        1287
21612 -122.299           1020        1357

[21613 rows x 21 columns]
```

In [2]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Informações básicas do DataFrame
print(dados.info())
```

```
# Verificar valores ausentes
print("\nValores nulos por coluna:")
print(dados.isnull().sum())

# DATAFRAME SEM REGISTROS AUSENTES!

# Estatísticas descritivas
print("\nEstatísticas descritivas:")
print(dados.describe().T)

# Mediana
print("\nMedianas:")
print(dados.median(numeric_only=True))

# seleção das variáveis numéricas e relevantes

numeric_cols = [col for col in dados.select_dtypes(include='number').columns
                if col not in ['id', 'zipcode', 'sqft_lot', 'lat', 'waterfront',


print(numeric_cols)

print("\nDistribuição das variáveis numéricas")
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols[:9]):
    plt.subplot(3, 3, i+1)
    sns.histplot(dados[col], kde=True, bins=30)
    plt.title(f'Distribuição de {col}')
plt.tight_layout()
plt.show()

# Matriz de correlação
print("\nCorrelação entre variáveis numéricas:")
corr = dados[numeric_cols].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Mapa de Calor - Correlação')
plt.show()

# gráficos de dispersão para todas as variáveis em numeric_cols
target = 'price'

print(f"\nDispersão entre as variáveis numéricas relevantes e {target}:")

num_plots = len(numeric_cols)
num_cols_per_row = 3
num_rows = (num_plots + num_cols_per_row - 1) // num_cols_per_row

plt.figure(figsize=(num_cols_per_row * 5, num_rows * 4))

features_for_scatter = [col for col in numeric_cols if col != target]

for i, feature in enumerate(features_for_scatter):
    plt.subplot(num_rows, num_cols_per_row, i + 1)
    sns.scatterplot(data=dados, x=feature, y=target, alpha=0.5)
    plt.title(f'{feature} x {target}')
    plt.xlabel(feature)
    plt.ylabel(target)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
None

Valores nulos por coluna:
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64

Estatísticas descritivas:
                 count          mean           std           min  \
id             21613.0  4.580302e+09  2.876566e+09  1.000102e+06  
price          21613.0  5.400881e+05  3.671272e+05  7.500000e+04  
bedrooms       21613.0  3.370842e+00  9.300618e-01  0.000000e+00  
bathrooms      21613.0  2.114757e+00  7.701632e-01  0.000000e+00  
```

sqft_living	21613.0	2.079900e+03	9.184409e+02	2.900000e+02
sqft_lot	21613.0	1.510697e+04	4.142051e+04	5.200000e+02
floors	21613.0	1.494309e+00	5.399889e-01	1.000000e+00
waterfront	21613.0	7.541757e-03	8.651720e-02	0.000000e+00
view	21613.0	2.343034e-01	7.663176e-01	0.000000e+00
condition	21613.0	3.409430e+00	6.507430e-01	1.000000e+00
grade	21613.0	7.656873e+00	1.175459e+00	1.000000e+00
sqft_above	21613.0	1.788391e+03	8.280910e+02	2.900000e+02
sqft_basement	21613.0	2.915090e+02	4.425750e+02	0.000000e+00
yr_built	21613.0	1.971005e+03	2.937341e+01	1.900000e+03
yr_renovated	21613.0	8.440226e+01	4.016792e+02	0.000000e+00
zipcode	21613.0	9.807794e+04	5.350503e+01	9.800100e+04
lat	21613.0	4.756005e+01	1.385637e-01	4.715590e+01
long	21613.0	-1.222139e+02	1.408283e-01	-1.225190e+02
sqft_living15	21613.0	1.986552e+03	6.853913e+02	3.990000e+02
sqft_lot15	21613.0	1.276846e+04	2.730418e+04	6.510000e+02

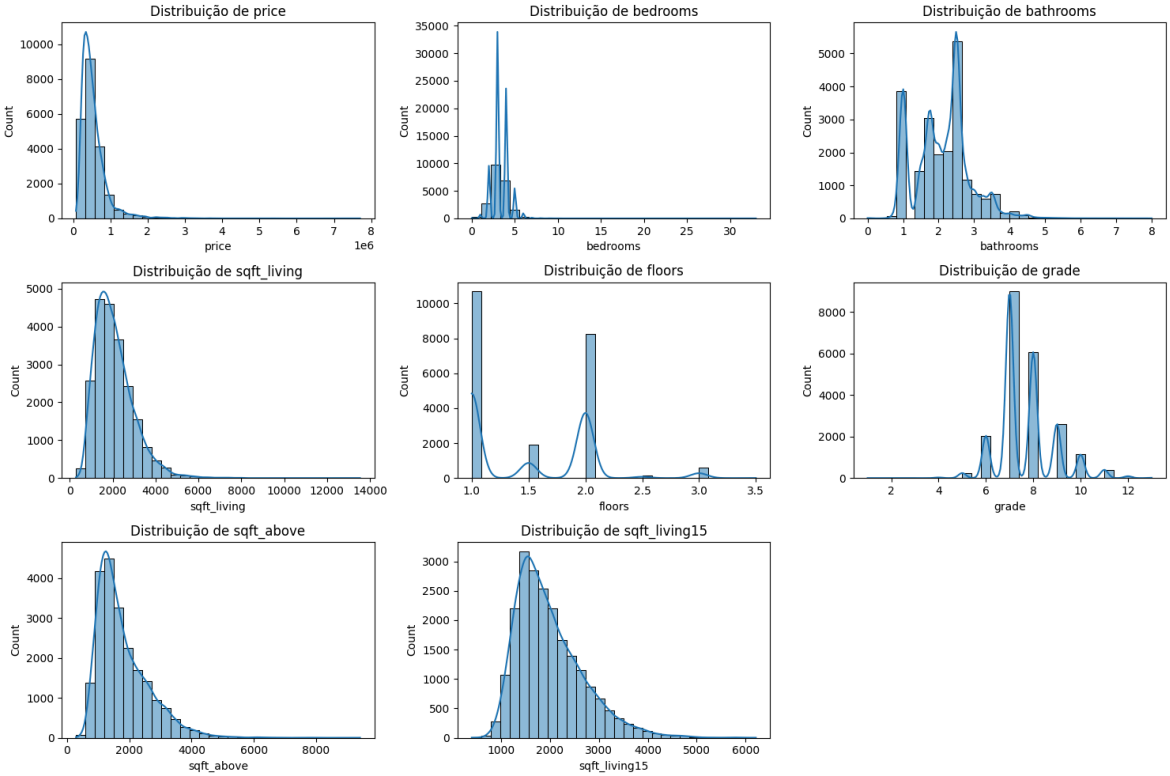
	25%	50%	75%	max
id	2.123049e+09	3.904930e+09	7.308900e+09	9.900000e+09
price	3.219500e+05	4.500000e+05	6.450000e+05	7.700000e+06
bedrooms	3.000000e+00	3.000000e+00	4.000000e+00	3.300000e+01
bathrooms	1.750000e+00	2.250000e+00	2.500000e+00	8.000000e+00
sqft_living	1.427000e+03	1.910000e+03	2.550000e+03	1.354000e+04
sqft_lot	5.040000e+03	7.618000e+03	1.068800e+04	1.651359e+06
floors	1.000000e+00	1.500000e+00	2.000000e+00	3.500000e+00
waterfront	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
view	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
condition	3.000000e+00	3.000000e+00	4.000000e+00	5.000000e+00
grade	7.000000e+00	7.000000e+00	8.000000e+00	1.300000e+01
sqft_above	1.190000e+03	1.560000e+03	2.210000e+03	9.410000e+03
sqft_basement	0.000000e+00	0.000000e+00	5.600000e+02	4.820000e+03
yr_built	1.951000e+03	1.975000e+03	1.997000e+03	2.015000e+03
yr_renovated	0.000000e+00	0.000000e+00	0.000000e+00	2.015000e+03
zipcode	9.803300e+04	9.806500e+04	9.811800e+04	9.819900e+04
lat	4.747100e+01	4.757180e+01	4.767800e+01	4.777760e+01
long	-1.223280e+02	-1.222300e+02	-1.221250e+02	-1.213150e+02
sqft_living15	1.490000e+03	1.840000e+03	2.360000e+03	6.210000e+03
sqft_lot15	5.100000e+03	7.620000e+03	1.008300e+04	8.712000e+05

Medianas:

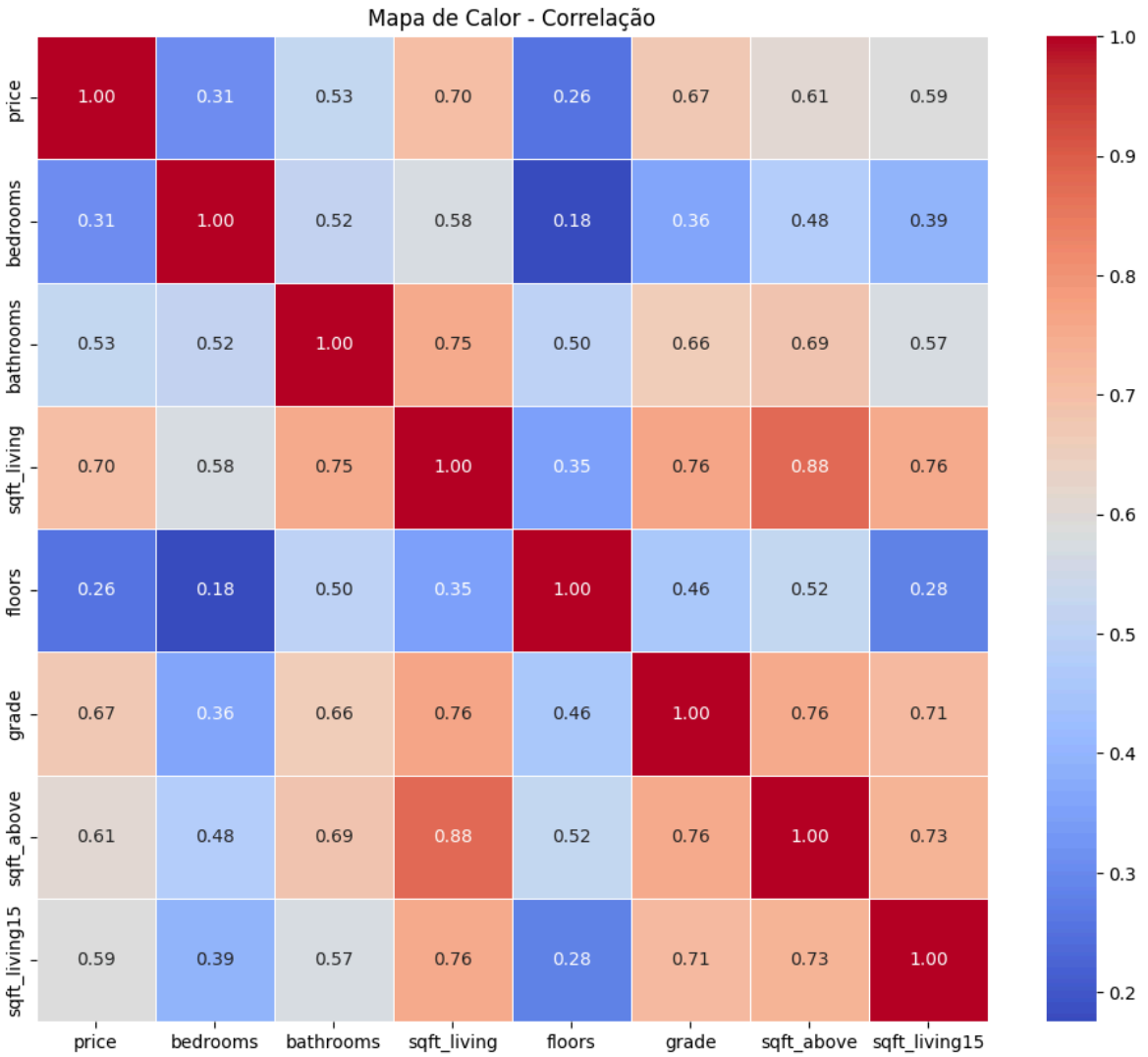
id	3.904930e+09
price	4.500000e+05
bedrooms	3.000000e+00
bathrooms	2.250000e+00
sqft_living	1.910000e+03
sqft_lot	7.618000e+03
floors	1.500000e+00
waterfront	0.000000e+00
view	0.000000e+00
condition	3.000000e+00
grade	7.000000e+00
sqft_above	1.560000e+03
sqft_basement	0.000000e+00
yr_built	1.975000e+03
yr_renovated	0.000000e+00
zipcode	9.806500e+04
lat	4.757180e+01
long	-1.222300e+02
sqft_living15	1.840000e+03
sqft_lot15	7.620000e+03

```
dtype: float64
['price', 'bedrooms', 'bathrooms', 'sqft_living', 'floors', 'grade', 'sqft_abov
e', 'sqft_living15']
```

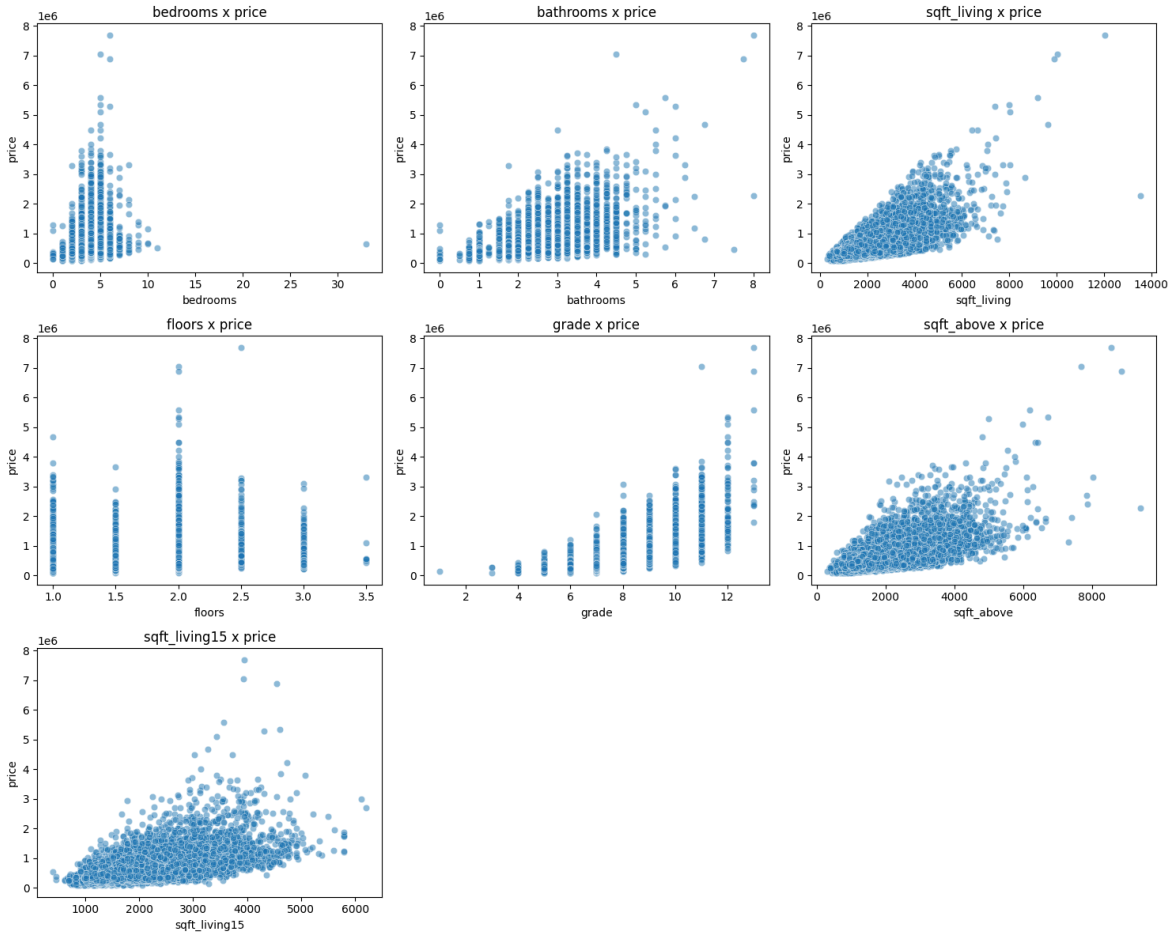
Distribuição das variáveis numéricas



Correlação entre variáveis numéricas:



Dispersão entre as variáveis numéricas relevantes e price:



Após análise da distribuição das variáveis VS o preço das residências foi possível extrair algumas variáveis e trabalhar somente com as supostamente relevantes no array numeric_cals. Em seguida, o mapa de calor da matriz de correlação mostrou que de fato as variáves escolhidas são relevantes para a definição do preço de venda, pois possuem alto valor de correlação (>0.5). Portanto, o modelo só levará em conta as variáveis estatisticamente relevantes.

2. Construção do Modelo de Regressão Linear (30%)

In [3]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Seleção de variáveis com boa correlação
X = dados[['bedrooms','bathrooms', 'sqft_living', 'grade', 'floors', 'sqft_livin

y = dados['price']

# Divisão em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Criação e treino do modelo
modelo = LinearRegression()
modelo.fit(X_train, y_train)
```

Out[3]:
```
▾ LinearRegression
LinearRegression()
```

In [4]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import pandas as pd


# Fazer previsões com o conjunto de teste
y_pred = modelo.predict(X_test)

# Avaliação do modelo
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5

# Extrair os coeficientes e o intercepto do modelo
intercepto = modelo.intercept_
coeficientes = modelo.coef_

# Imprimir o intercepto (coeficiente linear)
print("\n--- Coeficientes da Regressão Linear ---")
print(f"Intercepto (b0): {intercepto:,.2f}")
print("O intercepto é o valor esperado de 'price' quando todas as outras variáve

# Coeficientes (b1, b2, etc.) com seus respectivos nomes
print("\nCoeficientes (b1, b2, ...):")
for i, col in enumerate(X.columns):

    print(f"  - {col} (b{i+1}): {coeficientes[i]:,.2f}")


# Métricas de avaliação do modelo
print("\nMétricas de Avaliação do Modelo ---")
print(f"R² (Coeficiente de Determinação): {r2:.4f}")
print(f"MAE (Erro Médio Absoluto): {mae:,.2f}")
print(f"MSE (Erro Quadrático Médio): {mse:,.2f}")
print(f"RMSE (Raiz do Erro Quadrático Médio): {rmse:,.2f}")
```

```
--- Coeficientes da Regressão Linear ---
Intercepto (b0): -466,326.67
O intercepto é o valor esperado de 'price' quando todas as outras variáveis são z
ero. Não Possui significado real.

Coeficientes (b1, b2, ...):
  - bedrooms (b1): -39,387.44
  - bathrooms (b2): -9,127.98
  - sqft_living (b3): 216.08
  - grade (b4): 97,999.17
  - floors (b5): -38,322.40
  - sqft_living15 (b6): 7.71

Métricas de Avaliação do Modelo ---
R² (Coeficiente de Determinação): 0.5507
MAE (Erro Médio Absoluto): 165,364.00
MSE (Erro Quadrático Médio): 67,929,195,508.51
RMSE (Raiz do Erro Quadrático Médio): 260,632.30
```

3. Interpretação dos Resultados (10%)

O coeficiente negativo de bedrooms, bathrooms e de floors pode ser um indício de multicolinearidade entre as variáveis.

O modelo de regressão linear obteve os seguintes coeficientes:

Intercepto (b_0 = -466.326,67): Esse valor representa o preço estimado de uma casa caso todas as outras variáveis fossem zero. Não tem significado real ness contexto.

Coeficientes das Variáveis: bedrooms (b_1 = -39.387,44) Cada quarto a maisrepresenta uma redução de U$ 39.387,44 no preço, mantendo as outras variáveis constantes. É um resultado não esperado, pois teoricamente mais quartos representariam mais valor agregado das casas. Pode indicar ruído nos dados ou algum erro no modelo.

bathrooms (b_2 = -9.127,98) Banheiros adicionais também impactam negativamente o preço, o que é contraintuitivo. Também será melhor explorado posteriormente. Entretanto, pode indicar também correlação entre muitos banheiros e outras variáveis não explicadas (por exemplo, imóveis antigos ou mal localizados possuem muitos banheiros).

sqft_living (b_3 = 216,08) Cada unidade adicional de área útil (em pés quadrados) aumenta o preço em U$ 216,08, o que faz sentido, já que maior área construída agregar maior valor.

grade (b_4 = 97.999,17) A nota da construção tem peso alto: cada ponto a mais está associado a um aumento de quase U$ 98 mil no preço, mostrando que a qualidade percebida da construção é um dos fatores mais relevantes.

floors (b_5 = -38.322,40) Essa variável merece atenção, pois mais andares estão relacionados a uma queda no preço médio. Isso pode indicar preferência por casas térreas ou algum ruído ou presença de multicolinearidade.

sqft_living15 (b_6 = 7,71) Refere-se à média da área útil das 15 casas vizinhas. Tem efeito positivo, mas discreto: U$ 7,71 por pé quadrado. Reforça a ideia de que o valor do imóvel também é influenciado pela vizinhança.

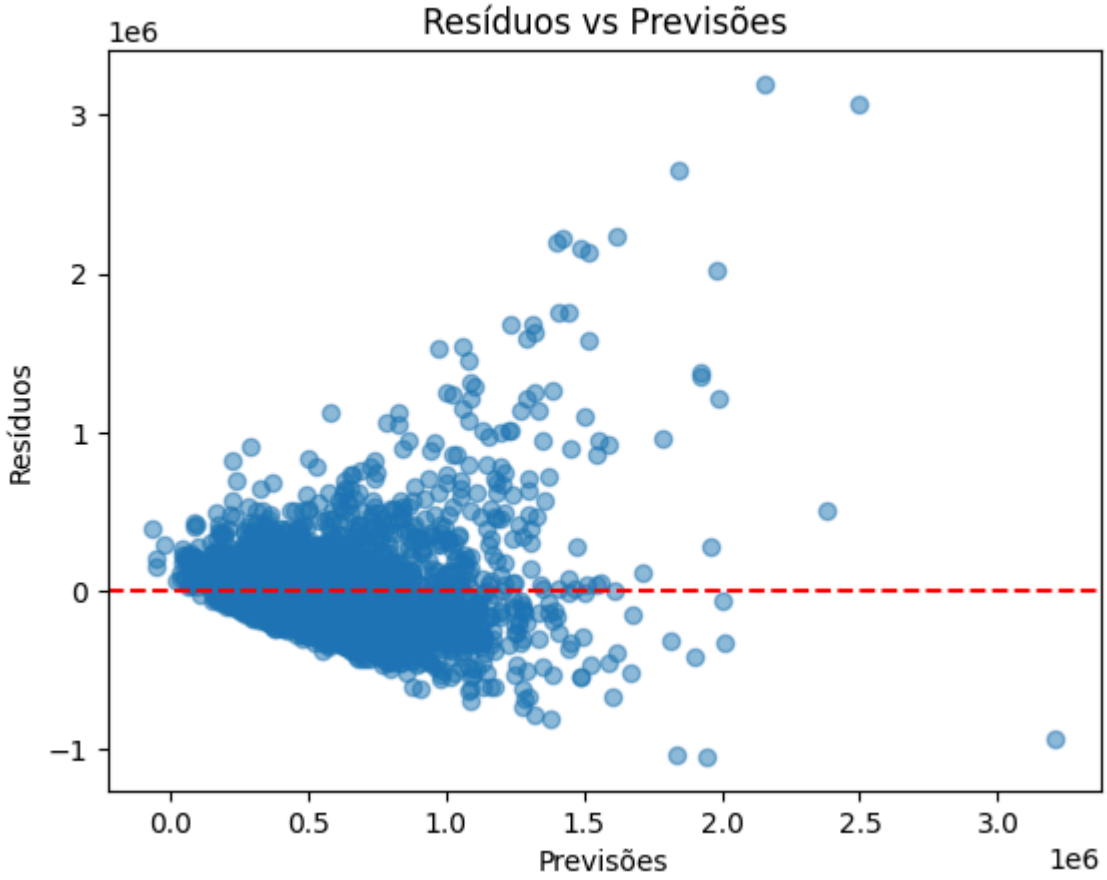
2. Avaliação da Qualidade do Modelo R^2 = 0,5507 O modelo consegue explicar 55,07% da variação nos preços com as variáveis fornecidas. É um valor razoável para dados de mercado imobiliário, que costumam ter ruídos e variáveis não observadas.

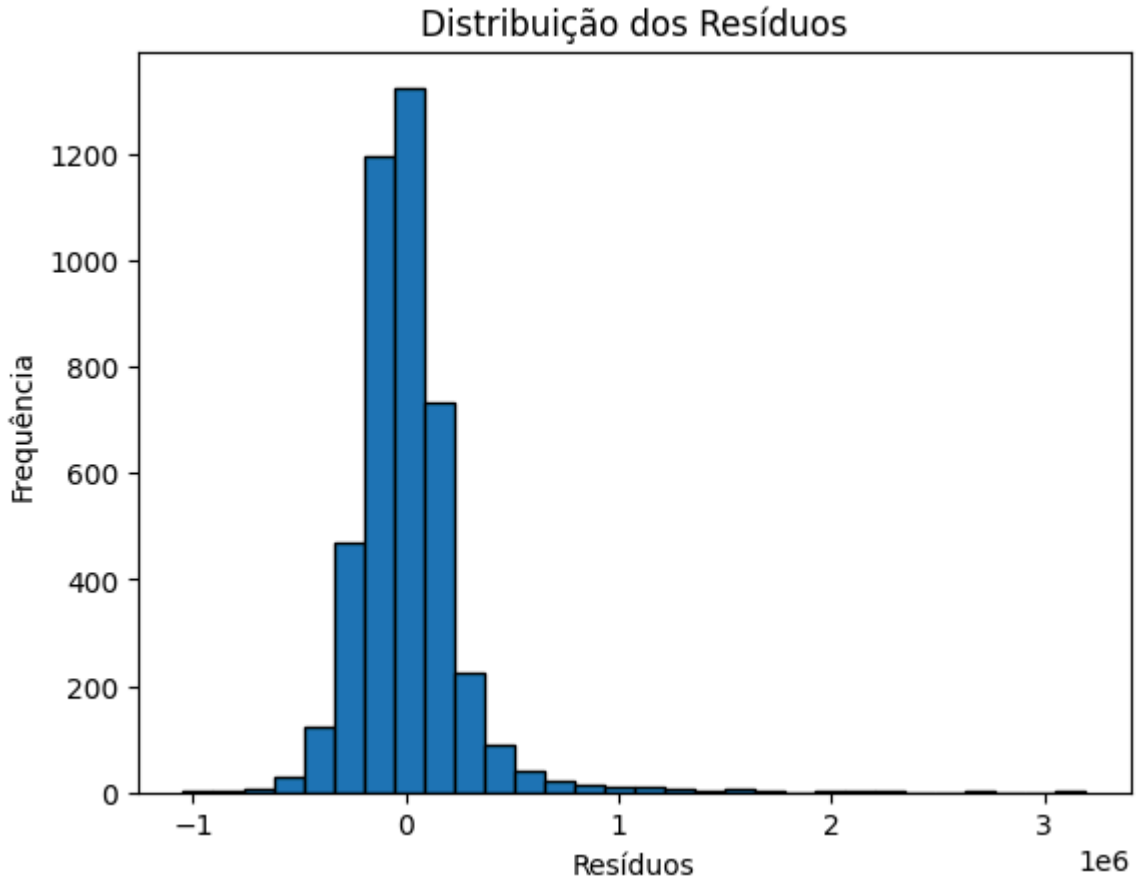
MAE : US 165.364,00 O erro médio absoluto indica que, em média, o modelo erra o preço estimado em cerca de U$ 165 mil. É um valor alto para o erro, o que leva a uma necessidade de melhorá-lo.

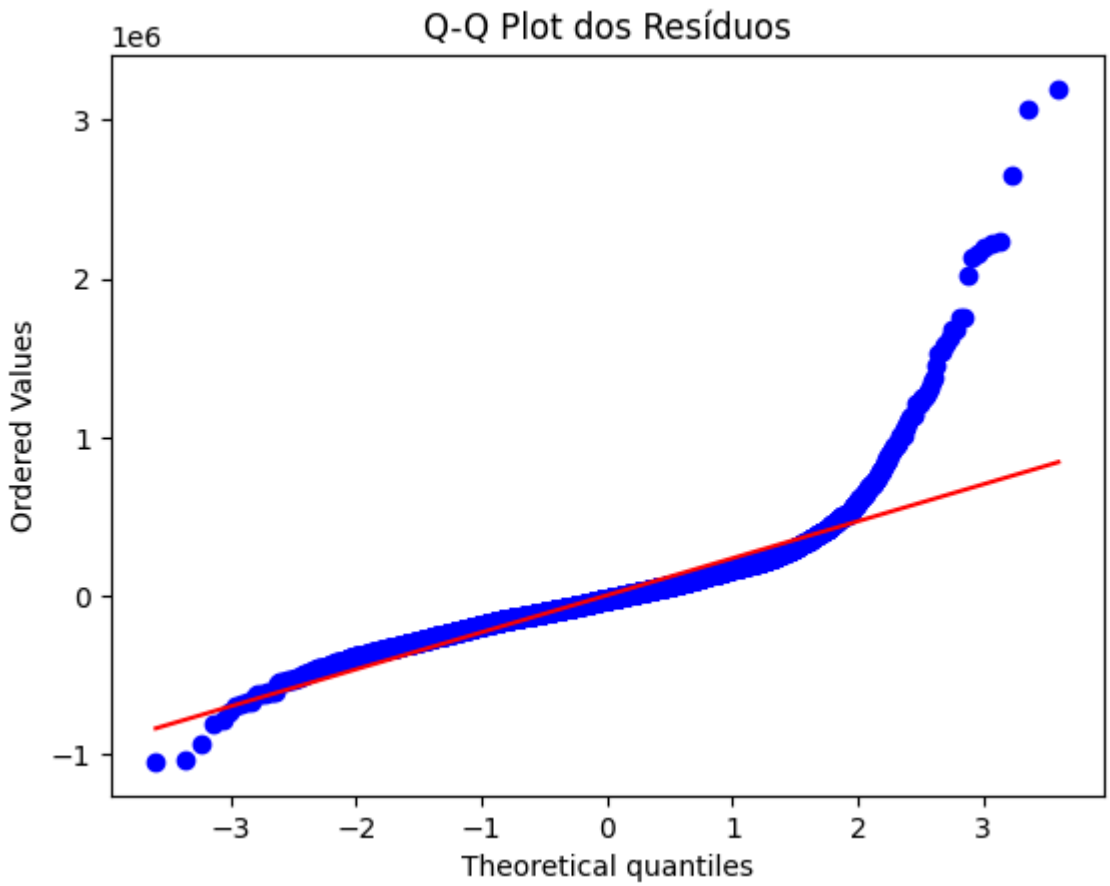
RMSE : US 260.632,30 A raiz do erro quadrático médio mostra a dispersão dos erros. Erros grandes têm mais peso aqui, e o valor ainda é consideravelmente alto.

In [5]:
```
# VERIFICAÇÃO DOS PRESSUPOSTOS DA REGRESSÃO LINEAR

import numpy as np
from scipy.stats import shapiro
```

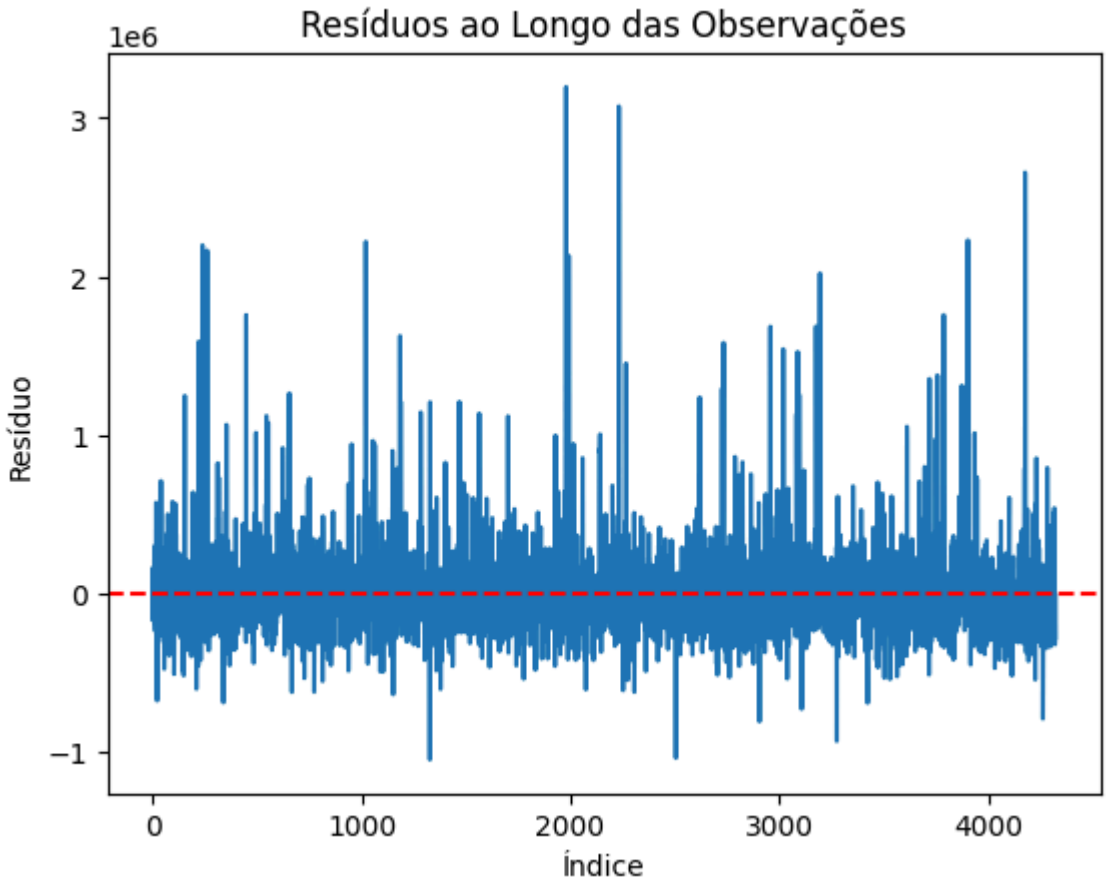
```
import matplotlib.pyplot as plt
import scipy.stats as stats


# Previsões com o conjunto de teste
y_pred = modelo.predict(X_test)

# Cálculo dos resíduos
residuos = y_test - y_pred

# Análises visuais

plt.scatter(y_pred, residuos, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Previsões')
plt.ylabel('Resíduos')
plt.title('Resíduos vs Previsões')
plt.show()
print()

plt.hist(residuos, bins=30, edgecolor='black')
plt.xlabel('Resíduos')
plt.ylabel('Frequência')
plt.title('Distribuição dos Resíduos')
plt.show()
print()

stats.probplot(residuos, dist="norm", plot=plt)
plt.title('Q-Q Plot dos Resíduos')
plt.show()
print()

# Teste de Shapiro-Wilk
stat, p_valor = shapiro(residuos)
print()
print(f'Estatística do teste: {stat:.4f}')
print()
print(f'Valor-p: {p_valor:.4f}')
print()
if p_valor > 0.05:
    print("resíduos seguem uma distribuição normal (não rejeita H0).")
else:
    print("resíduos não seguem uma distribuição normal (rejeita H0).")

print()
plt.plot(residuos.values)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Índice')
plt.ylabel('Resíduo')
plt.title('Resíduos ao Longo das Observações')
plt.show()


# H0 (hipótese nula): Os resíduos seguem distribuição normal.

# p_valor > 0.05: não rejeita H0 : normalidade

# p_valor <= 0.05: rejeita H0 : resíduos não seguem distribuição normal.
```


Resíduos vs Previsões
1e6
3
2
1
0
−1
Resíduos
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Previsões
1e6


Distribuição dos Resíduos
1200
1000
800
600
400
200
0
Frequência
−1
0
1
2
3
Resíduos
1e6




```
Estatística do teste: 0.8014

Valor-p: 0.0000

resíduos não seguem uma distribuição normal (rejeita H0).
```



Resíduos não distribuídos conforme curva normal, apresentam assimetria e heterocedasticidade. Foi observado que o modelo não atende aos pressupostos da regressão linear.

4. Ajustes no Modelo (30%)

A primeira tentativa para ajuste de modelo será ajustar a variável alvo ('price') aplicando ln('price') e avaliar a multicolinearidade ebtre as variáveis. Sm seguida, treinar o modelo novamente, avaliando seus resultados.

In [6]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm

# Aplicar log na variável target
y_log = np.log(dados['price'])

# Seleção das variáveis independentes
X = dados[['bedrooms','bathrooms', 'sqft_living', 'grade', 'floors', 'sqft_livin
y = y_log

# Cálculo do VIF
X_const = sm.add_constant(X)
vif_data = pd.DataFrame()
vif_data["Variável"] = X_const.columns
```

```
vif_data["VIF"] = [variance_inflation_factor(X_const.values, i) for i in range(X

print("VIF das variáveis:")
print(vif_data)

# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Treinamento do modelo
modelo_log = LinearRegression()
modelo_log.fit(X_train, y_train)

#  Prever em escala log e reverter para escala original
y_pred_log = modelo_log.predict(X_test)
y_pred_original = np.exp(y_pred_log)

#  Avaliação do desempenho em escala original
y_test_original = np.exp(y_test)

mae = mean_absolute_error(y_test_original, y_pred_original)
mse = mean_squared_error(y_test_original, y_pred_original)
rmse = np.sqrt(mse)
r2 = r2_score(y_test_original, y_pred_original)

print("\nDesempenho do modelo ajustado")
print(f"MAE: R$ {mae:,.2f}")
print(f"MSE: {mse:,.2f}")
print(f"RMSE: R$ {rmse:,.2f}")
print(f"R²: {r2:.4f}")
```

```
VIF das variáveis:
        Variável        VIF
0          const  78.153029
1       bedrooms   1.599750
2      bathrooms   2.864624
3    sqft_living   4.666741
4          grade   3.093656
5         floors   1.441553
6  sqft_living15   2.637705

Desempenho do modelo ajustado
MAE: R$ 156,979.13
MSE: 82,574,154,508.67
RMSE: R$ 287,357.19
R²: 0.4538
```

O cálculo do VIF não evidenciou presença de multicolinearidade e a transformação da variável dependente não gerou resultados positivos. Será implementada uma engenharia de features para tentar obter features mais significativas a partir das variáveis já presentes no dataset, bem como manter a variável dependente sem aplicação de log.

As seguintes novas variáveis serão criadas:

bathrooms_per_bedroom sqft_per_floor age recent_renovation (booleano 0 ou 1)

Substituir zipcode pela média de preços por região. Para isso, criou-se uma nova variável chamada zipcode_avg_price, que representa o preço médio dos imóveis por zipcode.

In [7]:
```
# retomando variável independente sem transformação logarítimica

y = dados['price']
```

In [8]:
```
import numpy as np
import pandas as pd

# Cópia de segurança do dataframe
dados_novo = dados.copy()

# Remover linhas com bedrooms = 0 para evitar divisão por zero
dados_novo = dados_novo[dados_novo['bedrooms'] > 0]
dados_novo = dados_novo[dados_novo['floors'] > 0]

# Novas variáveis
dados_novo['bathrooms_per_bedroom'] = dados_novo['bathrooms'] / dados_novo['bedr
dados_novo['sqft_per_floor'] = dados_novo['sqft_living'] / dados_novo['floors']
dados_novo['age'] = 2025 - dados_novo['yr_built']
dados_novo['recent_renovation'] = (dados_novo['yr_renovated'] > 0).astype(int)

# Média de preços por região (zipcode)
zipcode_avg = dados_novo.groupby('zipcode')['price'].mean().to_dict()
dados_novo['zipcode_avg_price'] = dados_novo['zipcode'].map(zipcode_avg)

# Verificando se tudo foi criado
print(dados_novo[['bathrooms_per_bedroom', 'sqft_per_floor', 'age', 'recent_reno

print()

print(dados_novo.columns)
```

```
   bathrooms_per_bedroom  sqft_per_floor  age  recent_renovation  \
0               0.333333          1180.0   70                  0
1               0.750000          1285.0   74                  1
2               0.500000           770.0   92                  0
3               0.750000          1960.0   60                  0
4               0.666667          1680.0   38                  0

   zipcode_avg_price
0      310612.755725
1      469899.427873
2      462480.035336
3      551688.673004
4      685605.775510

Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15', 'bathrooms_per_bedroom',
       'sqft_per_floor', 'age', 'recent_renovation', 'zipcode_avg_price'],
      dtype='object')
```

In [9]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

# Seleção de novas variáveis
```

```
X = dados_novo[['bathrooms_per_bedroom', 'sqft_living', 'grade', 'floors',
                'sqft_living15', 'sqft_per_floor', 'age',
                'recent_renovation', 'zipcode_avg_price']]

# Adiciona constante para o cálculo do VIF
X_const = sm.add_constant(X)

# Calcula o VIF para cada variável
vif_data = pd.DataFrame()
vif_data["Variável"] = X_const.columns
vif_data["VIF"] = [variance_inflation_factor(X_const.values, i) for i in range(X

#  resultados
print(vif_data)
```

```
                Variável         VIF
0                  const  150.195275
1  bathrooms_per_bedroom    1.412635
2            sqft_living    9.840958
3                  grade    3.363469
4                 floors    6.188284
5          sqft_living15    2.732057
6         sqft_per_floor    8.040727
7                    age    1.841006
8      recent_renovation    1.102221
9      zipcode_avg_price    1.297665
```

O cálculo do VIF permite retirar algumas variáveis redundantes, como:

sqft_per_floor 8.040727

floors 6.188284

sqft_living 9.840958

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Seleção de variáveis com boa correlação

X = dados_novo[['bathrooms_per_bedroom', 'grade',
                'sqft_living15', 'age',
                'recent_renovation', 'zipcode_avg_price']]


y = dados_novo['price']

# Divisão em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Criação e treino do modelo
modelo = LinearRegression()
modelo.fit(X_train, y_train)
```

Out[10]:
```
▾ LinearRegression
LinearRegression()
```

REAVALIAÇÃO DO MODELO AJUSTADO:

In [11]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

#  Previsões
y_pred = modelo.predict(X_test)

#  Métricas
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

#  Impressão dos Resultados
print("Avaliação do Modelo Linear:")
print(f"MAE : R$ {mae:,.2f}")
print(f"MSE : {mse:,.2f}")
print(f"RMSE: R$ {rmse:,.2f}")
print(f"R²  : {r2:.4f}")
```

```
Avaliação do Modelo Linear:
MAE : R$ 123,575.57
MSE : 43,774,658,876.88
RMSE: R$ 209,223.94
R²  : 0.6779
```

A MANUTENÇÃO DA VARIÁVEL DEPENDENTE SEM TRANSFORMAÇÃO LOGARÍTIMICA ALIADA À ENGENHARIA DE FEATURES PERMITIU OBTER UM MODELO QUE EXPLICA 67,8% DA VARIAÇÃO DOS PREÇOS E QUE POSSUI MENOR ERRO ASSOCIADO, COM MENOR ERRO ASSOCIADO (MAE)

5. Tomada de Decisão (10%)

- ESSE RESULTADO PODE SER APLICADO EM UM CONTEXTO REAL DE MERCADO IMOBILIÁRIO PARA PREVER O PREÇO DE VENDA DE UMA CASA, MAS NÃO COMO ÚNICA FERRAMENTA, MAS COMO FORMA COMPLEMENTAR DE SUPORTE PARA DETERMINAÇÃO DO VALOR DA CASA. HÁ OUTRAS VARIÁVEIS QUE INFLUENCIAM FORTEMENTE NO VALOR QUE NÃO FORAM CONSIDERADAS NO MODELO, COMO SE A CASA É OU NÃO MOBILIADA, SE O VENDEDOR ESTÁ OU NÃO COM PRESSA DE VENDÊ-LA, ETC.
- PODE SER UTILIZADO PARA PREVISÃO DO PREÇO DE VENDA, ANTES OU DEPOIS DE CONSTRUÍDA A CASA. NOS CASOS EM QUE A RESIDÊNCIA NÃO ESTEJA CONSTRUÍDA, O MODELO TAMBÉM É CAPAZ DE DAR SUPORTE A PREVISÕES DE LUCRO DA CONSTRUTORA.
- ESSE TIPO DE INFORMAÇÃO PODE SER USADO COMO BASE PARA ESTRATÉGIAS COMO POSICIONAMENTO DE MERCADO DA EMPRESA. POR EXEMPLO, ANALISAR A REGIÃO PARA DECISÃO ESTRATÉGICA DE CONSTRUÇÃO DE IMÓVEIS, ANALISAR VIABILIDADE DE COMPRA/REFORMA/REVENDA, ENTRE OUTROS.

QUESTÃO 2

a) Análise Descritiva dos Dados (10%)

In [12]:
```
import pandas as pd

# Especificação do caminho do arquivo csv

path = '/content/drive/MyDrive/AEDI/hotel_bookings.csv'

dados = pd.read_csv(path)

# Estrutura da base
dados.info()

# Estatísticas para variáveis numéricas
dados.describe()

# Valores ausentes
dados.isnull().sum().sort_values(ascending=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119390 non-null  object
 1   is_canceled                     119390 non-null  int64
 2   lead_time                       119390 non-null  int64
 3   arrival_date_year               119390 non-null  int64
 4   arrival_date_month              119390 non-null  object
 5   arrival_date_week_number        119390 non-null  int64
 6   arrival_date_day_of_month       119390 non-null  int64
 7   stays_in_weekend_nights         119390 non-null  int64
 8   stays_in_week_nights            119390 non-null  int64
 9   adults                          119390 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64
 12  meal                            119390 non-null  object
 13  country                         118902 non-null  object
 14  market_segment                  119390 non-null  object
 15  distribution_channel            119390 non-null  object
 16  is_repeated_guest               119390 non-null  int64
 17  previous_cancellations          119390 non-null  int64
 18  previous_bookings_not_canceled  119390 non-null  int64
 19  reserved_room_type              119390 non-null  object
 20  assigned_room_type              119390 non-null  object
 21  booking_changes                 119390 non-null  int64
 22  deposit_type                    119390 non-null  object
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64
 26  customer_type                   119390 non-null  object
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64
 29  total_of_special_requests       119390 non-null  int64
 30  reservation_status              119390 non-null  object
 31  reservation_status_date         119390 non-null  object
dtypes: float64(4), int64(16), object(12)
memory usage: 29.1+ MB
```

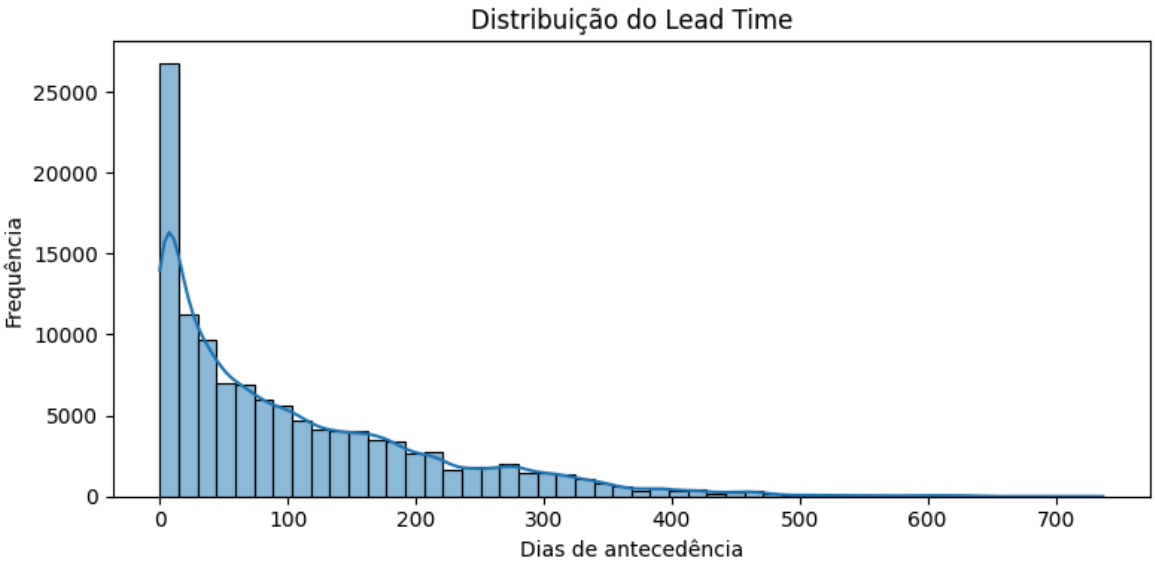
Out[12]:

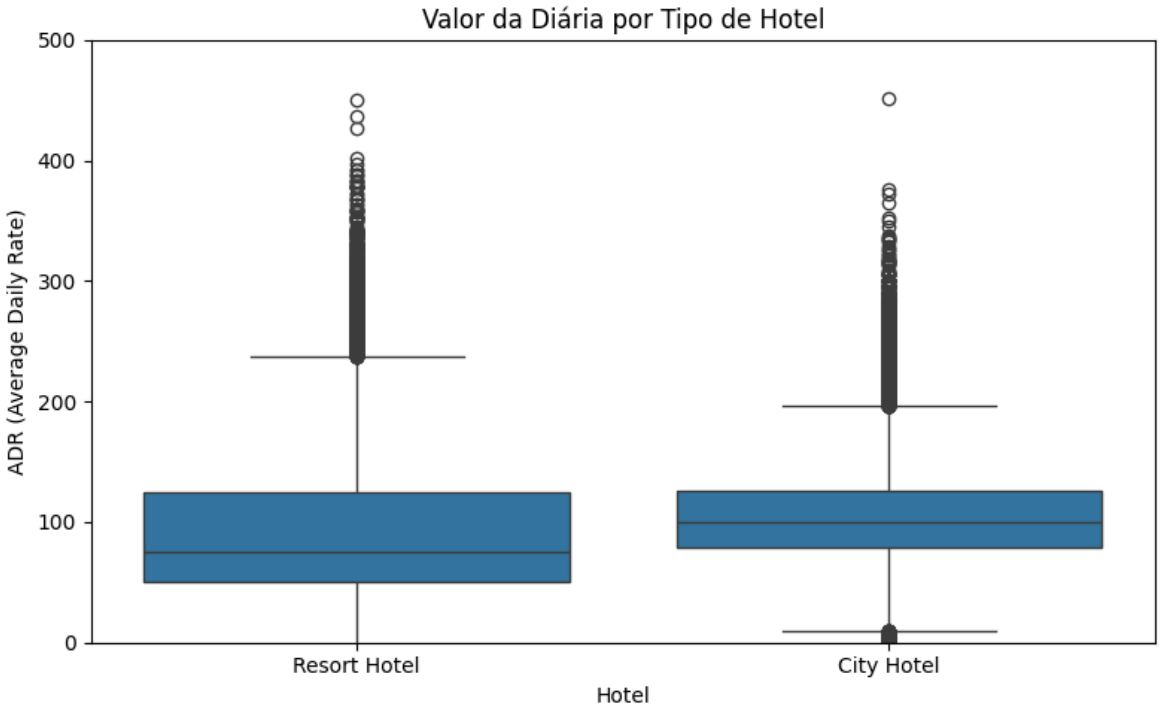
	0
company	112593
agent	16340
country	488
children	4
arrival_date_month	0
arrival_date_week_number	0
hotel	0
is_canceled	0
stays_in_weekend_nights	0
arrival_date_day_of_month	0
adults	0
stays_in_week_nights	0
babies	0
meal	0
lead_time	0
arrival_date_year	0
distribution_channel	0
market_segment	0
previous_bookings_not_canceled	0
is_repeated_guest	0
reserved_room_type	0
assigned_room_type	0
booking_changes	0
previous_cancellations	0
deposit_type	0
days_in_waiting_list	0
customer_type	0
adr	0
required_car_parking_spaces	0
total_of_special_requests	0
reservation_status	0
reservation_status_date	0

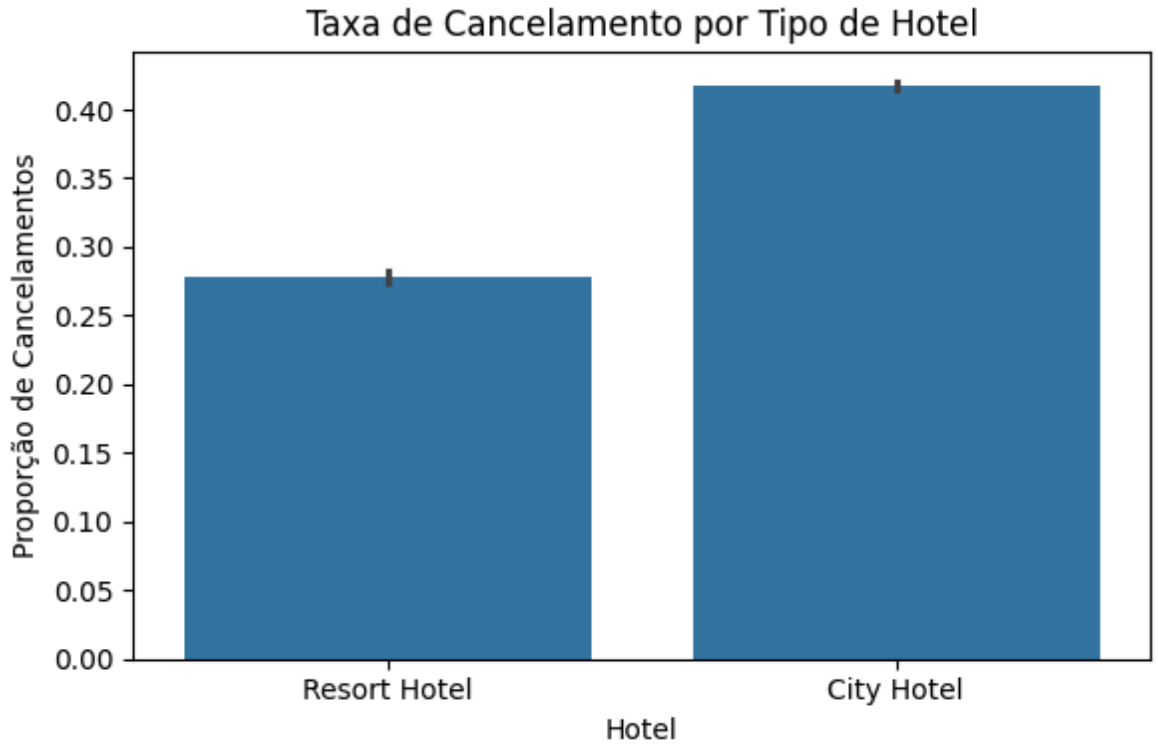
dtype: int64

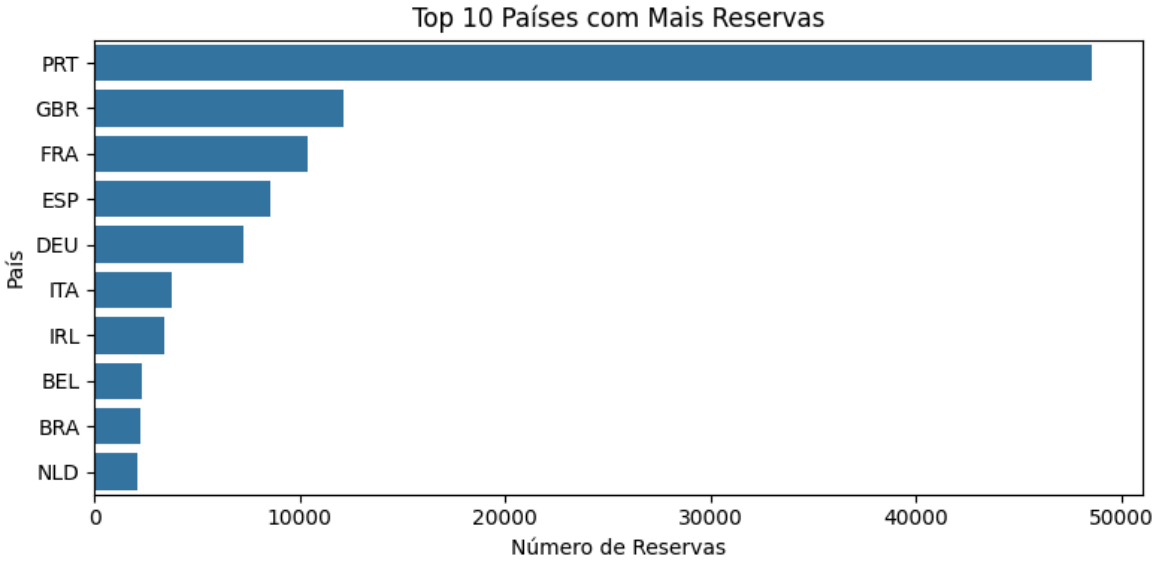
In [13]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
sns.countplot(data=dados, x='hotel')
plt.title('Reservas por Tipo de Hotel')
plt.xlabel('Hotel')
plt.ylabel('Quantidade')
plt.tight_layout()
plt.show()
print()

plt.figure(figsize=(8,4))
sns.histplot(data=dados, x='lead_time', bins=50, kde=True)
plt.title('Distribuição do Lead Time')
plt.xlabel('Dias de antecedência')
plt.ylabel('Frequência')
plt.tight_layout()
plt.show()
print()

plt.figure(figsize=(8,5))
sns.boxplot(data=dados, x='hotel', y='adr')
plt.title('Valor da Diária por Tipo de Hotel')
plt.ylabel('ADR (Average Daily Rate)')
plt.xlabel('Hotel')
plt.ylim(0, 500)  # Limite para facilitar visualização
plt.tight_layout()
plt.show()

print()

plt.figure(figsize=(6,4))
sns.barplot(data=dados, x='hotel', y='is_canceled')
plt.title('Taxa de Cancelamento por Tipo de Hotel')
plt.ylabel('Proporção de Cancelamentos')
plt.xlabel('Hotel')
plt.tight_layout()
plt.show()

print()

top_paises = dados['country'].value_counts().head(10)
plt.figure(figsize=(8,4))
sns.barplot(x=top_paises.values, y=top_paises.index)
plt.title('Top 10 Países com Mais Reservas')
plt.xlabel('Número de Reservas')
plt.ylabel('País')
plt.tight_layout()
plt.show()


dados['customer_type'].value_counts()
dados.groupby('customer_type')['is_canceled'].mean().sort_values(ascending=False
```


Reservas por Tipo de Hotel
Quantidade
80000
70000
60000
50000
40000
30000
20000
10000
0
Resort Hotel
City Hotel
Hotel


Distribuição do Lead Time
Frequência
25000
20000
15000
10000
5000
0
0
100
200
300
400
500
600
700
Dias de antecedência


Valor da Diária por Tipo de Hotel
ADR (Average Daily Rate)
500
400
300
200
100
0
Resort Hotel
City Hotel
Hotel


Taxa de Cancelamento por Tipo de Hotel
Proporção de Cancelamentos
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Resort Hotel
City Hotel
Hotel



Out[13]:

customer_type	is_canceled
Transient	0.407463
Contract	0.309617
Transient-Party	0.254299
Group	0.102253

dtype: float64

b) Modelo de Regressão Logística (60%)

DEFINIÇÃO DE VARIÁVEIS CATEGÓRICAS E NUMÉRICAS E CÁLCULO DO VIF PARA IDENTIFICAÇÃO DE MULTICOLINEARIDADE.

In [14]:
```
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

numericas = [
    'lead_time',
    'adults',
    'previous_cancellations',
    'previous_bookings_not_canceled',
    'adr'
]

categoricas = [
    'arrival_date_month',
    'country',
    'deposit_type',
    'customer_type'
]

# One-Hot Encoding
```

```
dados_modelo = pd.get_dummies(dados[numericas + categoricas], drop_first=True)

# Converte booleanos para inteiros
dados_modelo = dados_modelo.astype(int)

# Garante que todos são numéricos
dados_modelo = dados_modelo.apply(pd.to_numeric, errors='coerce')
dados_modelo = dados_modelo.replace([np.inf, -np.inf], np.nan)
dados_modelo = dados_modelo.dropna()

# Amostragem de 30% dos dados para cálculo do VIF
amostra = dados_modelo.sample(frac=0.1, random_state=42)

# Cálculo do VIF
X = add_constant(amostra)
vif = pd.DataFrame()
vif['Variável'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif.sort_values(by='VIF', ascending=False)

# Filtrar apenas as variáveis com VIF > 10 (desconsiderando a constante)
vif_altas = vif[(vif['Variável'] != 'const') & (vif['VIF'] > 10)]

# Exibir o resultado
vif_altas.sort_values(by='VIF', ascending=False)
```

```
/usr/local/lib/python3.11/dist-packages/statsmodels/regression/linear_model.py:17
82: RuntimeWarning: invalid value encountered in scalar divide
  return 1 - self.ssr/self.centered_tss
```

Out[14]:

	Variável	VIF
151	country_PRT	47.544563
75	country_GBR	19.032362
72	country_FRA	16.629404
67	country_ESP	13.439630
59	country_DEU	11.531607

Considerando os valores obtidos para VIF na variável country, optar-se-á por excluir a variável country, pois além de provavelmente apresentar multicolinearidade, gera muitas colunas a partir da aplicação do one_Hot_Encoding.

In [15]:

```
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Variáveis selecionadas (sem 'country')
numericas = [
    'lead_time',
    'adults',
    'previous_cancellations',
    'previous_bookings_not_canceled',
    'adr'
]
```

```
categoricas = [
    'arrival_date_month',
    'deposit_type',
    'customer_type'
]

# One-Hot Encoding (sem country)
dados_modelo = pd.get_dummies(dados[numericas + categoricas], drop_first=True)

# Converte booleanos para inteiros
dados_modelo = dados_modelo.astype(int)

# Garante que todos são numéricos e remove valores inválidos
dados_modelo = dados_modelo.apply(pd.to_numeric, errors='coerce')
dados_modelo = dados_modelo.replace([np.inf, -np.inf], np.nan)
dados_modelo = dados_modelo.dropna()
```

MODELAGEM PRESSUPOSTOS:

1. Na Regressão Logística Binária a variável dependente é binária. No caso da análise de cancelamento, a variável dependente é dados_modelo['is_canceled'], que assume valores 0 ou 1, ou seja, binária.

2. Para Regressão Logística Binária, o fator da variável dependente deve representar o desejo do analista.

O "desejo do analista" significa que o valor 1 deve representar o desfecho de interesse, ou seja, aquilo que o analista quer prever ou entender melhor. No caso de churn: A variável 'is_canceled' já está assim: is_canceled = 1 → cliente cancelou reserva is_canceled = 0 → cliente não cancelou

Então, ao fazer a regressão logística, o modelo vai estimar a probabilidade de is_canceled = 1, ou seja, a probabilidade do cliente cancelar.

3. Apenas as variáveis importantes deve ser alocadas no modelo. Será realizado.

4. Não deverá possuir multicolinearidade. Já testado.

5. As varáveis independentes são lineares em relação ao log das probabilidades no Logit. Será analisado a seguir.

6. Regressão Logística requer um número grande de observações. Considerando que se tem 119390 observações válidas, considera-se atendido esse pressuposto.

Serão então testados os pressupostos 3 e 5.

APLICAÇÃO DA REGRESSÃO LOGÍSTICA

In [16]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

```
# Alvo e preditores
X = dados_modelo
y = dados['is_canceled'].loc[X.index]

# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Treinar modelo
modelo = LogisticRegression(max_iter=1000)
modelo.fit(X_train, y_train)

# Prever
y_pred = modelo.predict(X_test)

# Avaliação
print("Acurácia       :", accuracy_score(y_test, y_pred))
print("Precisão       :", precision_score(y_test, y_pred))
print("Recall         :", recall_score(y_test, y_pred))
print("F1-score       :", f1_score(y_test, y_pred))
print("\nMatriz de Confusão:\n", confusion_matrix(y_test, y_pred))
print("\nRelatório de Classificação:\n")
print(classification_report(y_test, y_pred))

# Análise dos coeficientes
coeficientes = pd.Series(modelo.coef_[0], index=X.columns)
coef_ordenados = coeficientes.sort_values(key=np.abs, ascending=False)

print("\n Variáveis mais influentes no cancelamento (análise do módulo do coefic
print(coef_ordenados.head(10))
```

```
Acurácia       : 0.7714213920763883
Precisão       : 0.9375
Recall         : 0.41382412474698255
F1-score       : 0.5741925417381807

Matriz de Confusão:
 [[22110   368]
 [ 7819  5520]]

Relatório de Classificação:

              precision    recall  f1-score   support

           0       0.74      0.98      0.84     22478
           1       0.94      0.41      0.57     13339

    accuracy                           0.77     35817
   macro avg       0.84      0.70      0.71     35817
weighted avg       0.81      0.77      0.74     35817


 Variáveis mais influentes no cancelamento (análise do módulo do coeficiente):

deposit_type_Non Refund           5.353013
previous_cancellations            2.826253
customer_type_Transient           1.036378
previous_bookings_not_canceled   -0.820465
arrival_date_month_September     -0.438211
customer_type_Group              -0.395801
arrival_date_month_July          -0.328345
arrival_date_month_August        -0.278725
customer_type_Transient-Party     0.277023
arrival_date_month_June          -0.241295
dtype: float64
```

O modelo só chama classifica como "cancelado" se tiver muita certeza, o que aumenta a precisão mas reduz o recall.

Isso pode indicar base desbalanceada, fato eventualmente representativo de mais reservas não canceladas do que canceladas.

c) Análise das features

Variáveis mais influentes no cancelamento (análise do módulo do coeficiente):

1. deposit_type_Non Refund 5.353013
2. previous_cancellations 2.826253
3. customer_type_Transient 1.036378
4. previous_bookings_not_canceled -0.820465
5. arrival_date_month_September -0.438211
6. customer_type_Group -0.395801
7. arrival_date_month_July -0.328345
8. arrival_date_month_August -0.278725
9. customer_type_Transient-Party 0.277023
10. arrival_date_month_June -0.241295lise das Features (20%)

- Variáveis que aumentam a chance de cancelamento: O modelo de regressão logística identificou algumas variáveis com coeficientes positivos, ou seja, quanto mais presentes nos dados, maior a probabilidade de cancelamento da reserva:

deposit_type_Non Refund: Clientes que optam por depósitos não reembolsáveis apresentaram uma probabilidade significativamente maior de cancelamento. Embora isso pareça contraditório, pode estar relacionado a reservas feitas impulsivamente, clientes menos comprometidos ou falhas na política de cobrança.

previous_cancellations: O histórico de cancelamentos é um forte preditor. Clientes que já cancelaram reservas anteriormente têm maior intenção de cancelar novamente, sugerindo um padrão de comportamento.

customer_type_Transient: Clientes classificados como transitórios — geralmente viajantes independentes sem contrato ou vínculo corporativo — tendem a cancelar com maior frequência, possivelmente pela flexibilidade.

customer_type_Transient-Party: Grupos de clientes transitórios também apresentaram maior risco de cancelamento, talvez por serem mais suscetíveis a mudanças de planos em conjunto.

- Variáveis que reduzem a chance de cancelamento Outras variáveis apresentaram coeficientes negativos, o que significa que sua presença está associada a uma menor probabilidade de cancelamento:

previous_bookings_not_canceled: Clientes com um histórico de reservas concluídas (sem cancelamentos) têm uma chance significativamente menor de cancelar novamente. Isso mostra que o comportamento anterior confiável é um bom indicativo de estabilidade, evidenciando mais uma vez que o padrão de comportamento é um fator que tem alto impacto em comportamento futuro.

arrival_date_month_September, July, August, June: Reservas com chegada nos meses de alta temporada ou férias escolares (como julho e agosto) tendem a ser mais estáveis. A

demanda maior e o planejamento antecipado nesses períodos podem reduzir o índice de desistências.

customer_type_Group: Clientes identificados como parte de grupos organizados ou corporativos apresentam menor tendência ao cancelamento, o que pode refletir compromissos já assumidos, contratos ou maior rigidez no planejamento.

Conclusão:

- Comportamento passado (cancelamentos anteriores, histórico de comparecimento) é altamente preditivo.
- Tipo de cliente e forma de pagamento têm grande influência.
- Sazonalidade também afeta cancelamentos: reservas em meses turísticos tendem a ser mais estáveis.

d) Justificativa do Método (10%)

A regressão linear é um modelo de previsão ou explicação, que aprende a partir do conjunto de dados e, respeitados os pressupostos da regressão, traça uma reta que representa a relação entre variável explicativa e variável dependente, para que seja possível prever a variável dependente a partir de um valor da variável explicativa.

A regressão logística tem o objetivo de classificação binária, que tem como saída um valor entre 0 e 1, que indica a probabilidade de se rejeitar a hipótese nula, que é justamente o desejo do analista. Em outras palavras, a partir de veriáveis explicativas, a regressão fornece um resultado entre 0 e 1, que é então arredondado para 0 ou 1, que indica a classificação da target em uma classe ou em outra. Então, no caso do exercício, ela fornecerá o resultado se determinado cliente, dados vários valores para as variáveis explicativas, irá cancelar ou não a reserva.

Questão 3

a) Análise Descritiva dos Dados (10%)

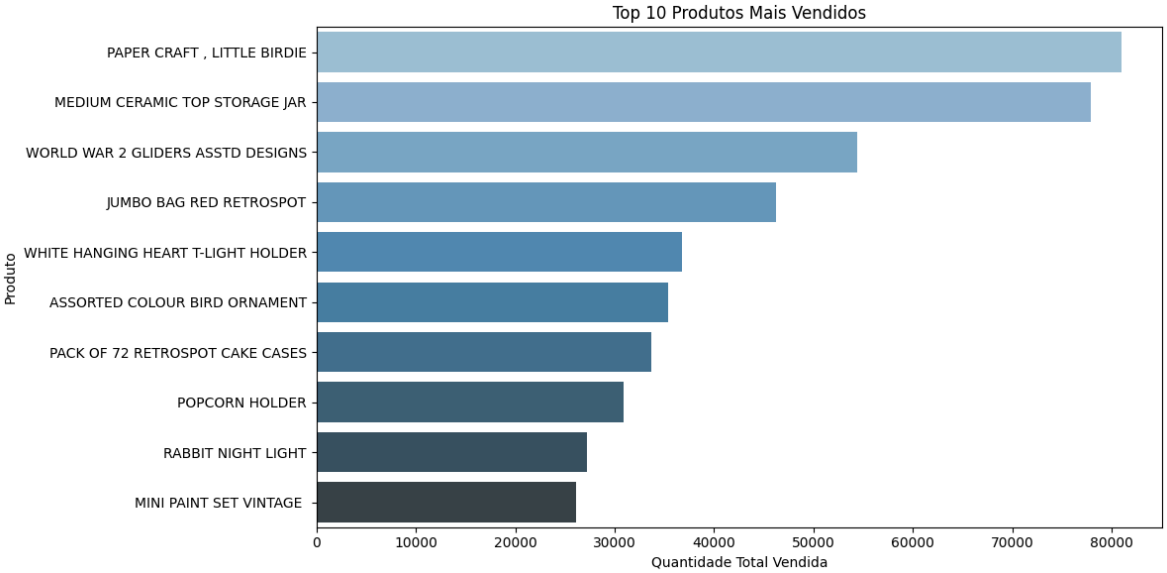
In [17]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Especificação do caminho do arquivo csv

path = '/content/drive/MyDrive/AEDI/Year2010-2011.csv'

dados = pd.read_csv(path, encoding='latin1')

# Estrutura da base
dados.info()

# Estatísticas para variáveis numéricas
```

```
dados.describe()

# Valores ausentes
dados.isnull().sum().sort_values(ascending=False)


# Converter InvoiceDate para datetime
dados['InvoiceDate'] = pd.to_datetime(dados['InvoiceDate'], errors='coerce')

# Remover linhas com customer ID e StockCode ausentes (mas pode ter description

dados_clean = dados.dropna(subset=['StockCode', 'Customer ID'])

# Remover valores negativos de quantidade e preço (devoluções e erros)
dados_clean = dados_clean[(dados_clean['Quantity'] > 0) & (dados_clean['Price']

# número de produtos únicos por código

print("\nNúmero de produtos únicos:", dados['StockCode'].nunique())

# VISUALIZAÇÕES INTERESSANTES

# top 10 mais vendidos

top_products = (
    dados_clean.groupby('Description')['Quantity']
    .sum()
    .sort_values(ascending=False)
    .head(10)
)

plt.figure(figsize=(12,6))
sns.barplot(x=top_products.values, y=top_products.index, palette='Blues_d')
plt.title('Top 10 Produtos Mais Vendidos')
plt.xlabel('Quantidade Total Vendida')
plt.ylabel('Produto')
plt.tight_layout()
plt.show()

# relação entre top 10 e preço médio

top10_names = top_products.index.tolist()

# Filtrar só os top 10
top_dados = dados_clean[dados_clean['Description'].isin(top10_names)]

# Calcular preço médio por produto
avg_prices = top_dados.groupby('Description')['Price'].mean()

plt.figure(figsize=(12,6))
sns.barplot(x=avg_prices.values, y=avg_prices.index, palette='Oranges_d')
plt.title('Preço Médio dos Top 10 Produtos Mais Vendidos')
plt.xlabel('Preço Médio (unidade)')
plt.ylabel('Produto')
plt.tight_layout()
plt.show()


# Criar coluna com valor total da linha (valor da compra individual)
dados_clean['TotalValue'] = dados_clean['Quantity'] * dados_clean['Price']
```

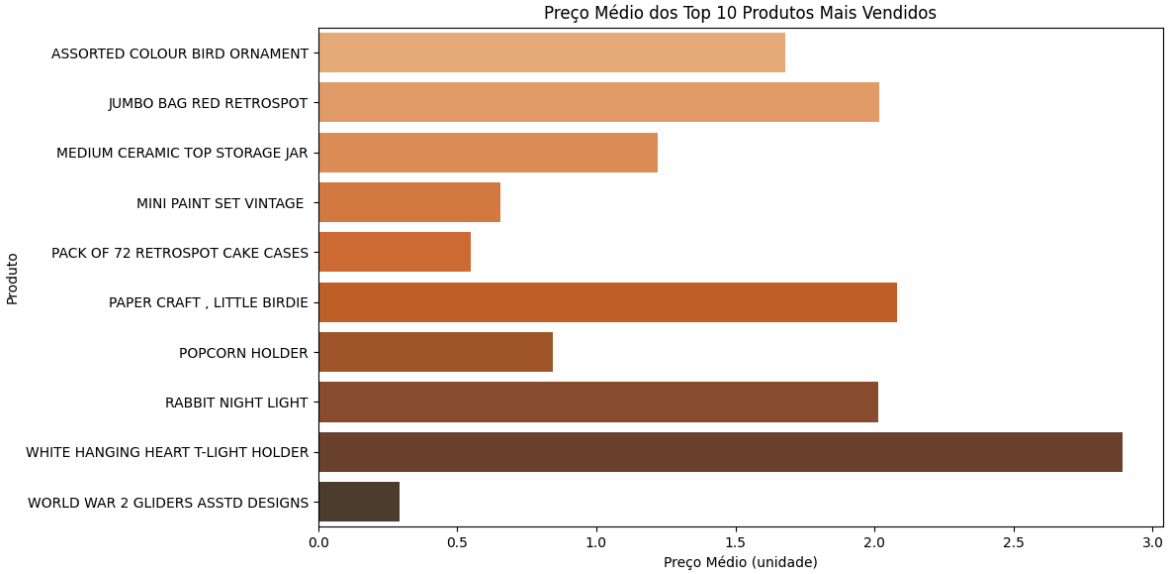
```
# Agrupar por Customer ID e somar o valor total comprado
top_customers_value = (
    dados_clean.groupby('Customer ID')['TotalValue']
    .sum()
    .sort_values(ascending=False)
    .head(10)
)



# Plotar
plt.figure(figsize=(10,5))
sns.barplot(x=top_customers_value.index.astype(int), y=top_customers_value.value
plt.title('Top 10 Compradores (por Valor Total Comprado)')
plt.xlabel('Customer ID')
plt.ylabel('Valor Total Comprado (£)')
plt.tight_layout()
plt.show()

top10_customers_ids = top_customers_value.index.tolist()

# Filtrar apenas vendas dos top 10 clientes (por valor total comprado)
dados_top_cust = dados[dados['Customer ID'].isin(top10_customers_ids)]

# Agrupar por StockCode e somar a quantidade comprada
stockcodes_top10 = (
    dados_top_cust.groupby('StockCode')['Quantity']
    .sum()
    .sort_values(ascending=False)
    .head(10)
)

# Plotar gráfico de barras
plt.figure(figsize=(10,5))
sns.barplot(x=stockcodes_top10.index, y=stockcodes_top10.values, palette='Set2')
plt.title('Top 10 Produtos (StockCodes) Mais Comprados pelos Top 10 Clientes')
plt.xlabel('StockCode')
plt.ylabel('Quantidade Total Comprada')
plt.tight_layout()
plt.show()


# quantidade de vendas por mês

dados_clean['YearMonth'] = dados_clean['InvoiceDate'].dt.to_period('M')
monthly_sales = dados_clean.groupby('YearMonth')['Quantity'].sum()

plt.figure(figsize=(14,6))
monthly_sales.plot(kind='bar', color='teal')
plt.title('Quantidade de Vendas por Mês')
plt.xlabel('Ano-Mês')
plt.ylabel('Quantidade Total Vendida')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541910 entries, 0 to 541909
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      541910 non-null  object
 1   StockCode    541910 non-null  object
 2   Description  540456 non-null  object
 3   Quantity     541910 non-null  int64
 4   InvoiceDate  541910 non-null  object
 5   Price        541910 non-null  float64
 6   Customer ID  406830 non-null  float64
 7   Country      541910 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB

Número de produtos únicos: 4070
```

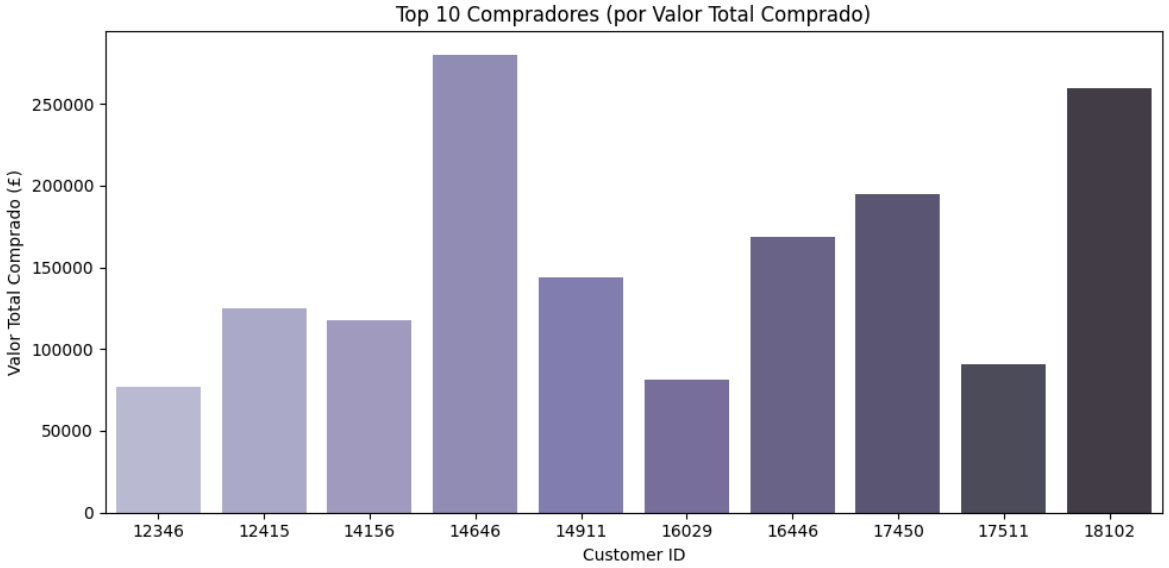
```
/tmp/ipython-input-17-3746571649.py:47: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=top_products.values, y=top_products.index, palette='Blues_d')
```



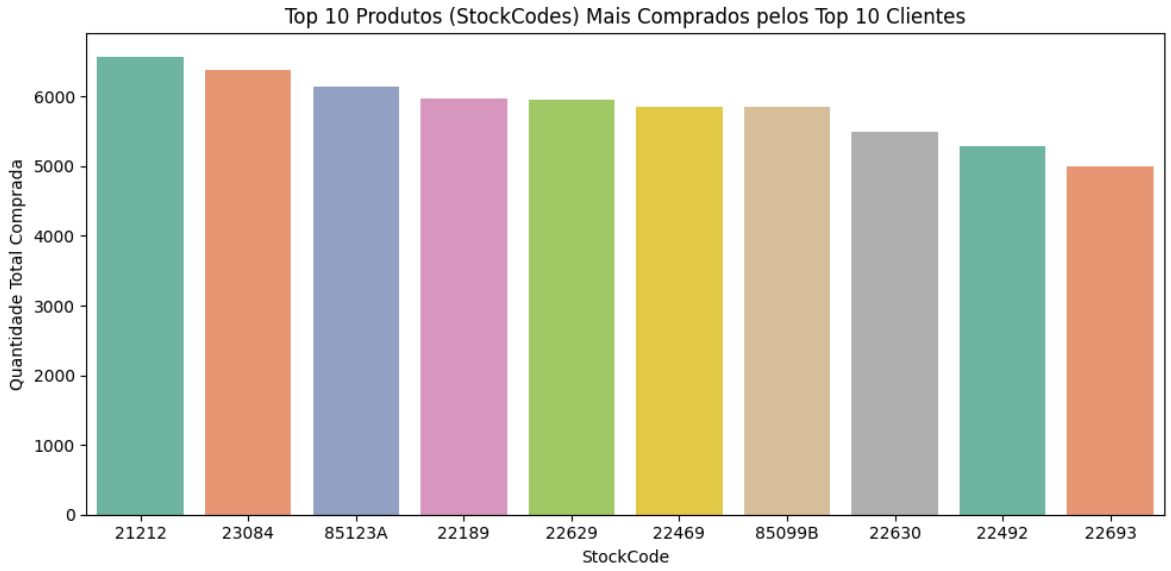

```
/tmp/ipython-input-17-3746571649.py:65: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=avg_prices.values, y=avg_prices.index, palette='Oranges_d')
```

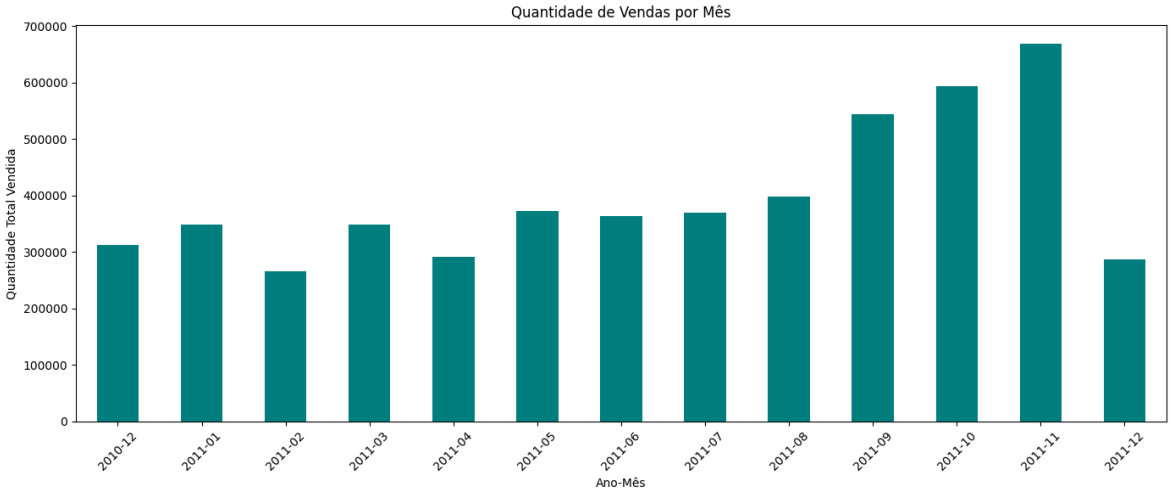



```
/tmp/ipython-input-17-3746571649.py:88: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=top_customers_value.index.astype(int), y=top_customers_value.valu
es, palette='Purples_d')
```




```
/tmp/ipython-input-17-3746571649.py:110: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=stockcodes_top10.index, y=stockcodes_top10.values, palette='Set
2')
```





b) Comparação entre Países (ANOVA) (40%)

Aqui será avaliado se há diferenças entre valores e quantidades nas compras a depender do país analisado.

In [18]:
```
import pandas as pd
from scipy.stats import f_oneway

# Filtrar apenas as colunas necessárias e remover outliers (opcional)
dados_anova = dados[(dados['Quantity'] > 0) & (dados['Price'] > 0)]

#quantidade de registros em dados_anova

print(dados_anova.info())

# Verificar países com mais de X registros para evitar viés de amostras muito pe
min_samples = 1000
countries_valid = dados_anova['Country'].value_counts()
valid_countries = countries_valid[countries_valid >= min_samples].index
dados_anova = dados_anova[dados_anova['Country'].isin(valid_countries)]

# Total de países antes e depois da filtragem
total_paises = dados['Country'].nunique()
paises_selecionados = len(valid_countries)
```

```
print(f'\nForam selecionados {paises_selecionados} países com pelo menos {min_sa


# Agrupar os valores de Quantity por país para o ANOVA
quantity_groups = [group['Quantity'].values for name, group in dados_anova.group
price_groups = [group['Price'].values for name, group in dados_anova.groupby('Co

# ANOVA para Quantity
f_stat_q, p_value_q = f_oneway(*quantity_groups)

# ANOVA para Price
f_stat_p, p_value_p = f_oneway(*price_groups)

# Resultados
print("ANOVA - Quantity por País")
print(f"F = {f_stat_q:.2f}, p-valor = {p_value_q:.4f}")
if p_value_q < 0.05:
    print("Diferenças estatisticamente significativas entre as médias de quantid
else:
    print("Não há evidência de diferenças significativas entre as médias de quan
print()
print("\nANOVA - Preço por País")
print(f"F = {f_stat_p:.2f}, p-valor = {p_value_p:.4f}")
if p_value_p < 0.05:
    print("Diferenças estatisticamente significativas entre as médias de preço p
else:
    print("Não há evidência de diferenças significativas entre as médias de preç
print()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 530105 entries, 0 to 541909
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      530105 non-null  object
 1   StockCode    530105 non-null  object
 2   Description  530105 non-null  object
 3   Quantity     530105 non-null  int64
 4   InvoiceDate  530105 non-null  datetime64[ns]
 5   Price        530105 non-null  float64
 6   Customer ID  397885 non-null  float64
 7   Country      530105 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 36.4+ MB
None

Foram selecionados 11 países com pelo menos 1000 registros, de um total de 38 paí
ses únicos no dataset.

ANOVA - Quantity por País
F = 75.91, p-valor = 0.0000
Diferenças estatisticamente significativas entre as médias de quantidade por paí
s.


ANOVA - Preço por País
F = 1.95, p-valor = 0.0346
Diferenças estatisticamente significativas entre as médias de preço por país.
```

ANOVA -

INTERPRETAÇÃO

Quantity por País F = 75.91: indica que há uma variação considerável entre as médias de quantidade entre diferentes países.

p-valor = 0.0000: indica que a probabilidade de essa diferença ter ocorrido ao acaso é praticamente nula.

Conclusão: há diferenças estatisticamente significativas nas quantidades compradas entre os países. Portanto, clientes de diferentes países compram quantidades diferentes de produtos.

Price por País F = 1.95: indica que existe uma pequena variação nas médias dos preços entre diferentes países, mas bem menor do que no caso de quantidade.

p-valor = 0.0346: como está abaixo de 0.05, também indica uma diferença estatisticamente significativa, embora mais discreta.

Conclusão: as médias dos preços de venda diferem a depender do país, mas as diferenças são menores do que as observadas na quantidade.

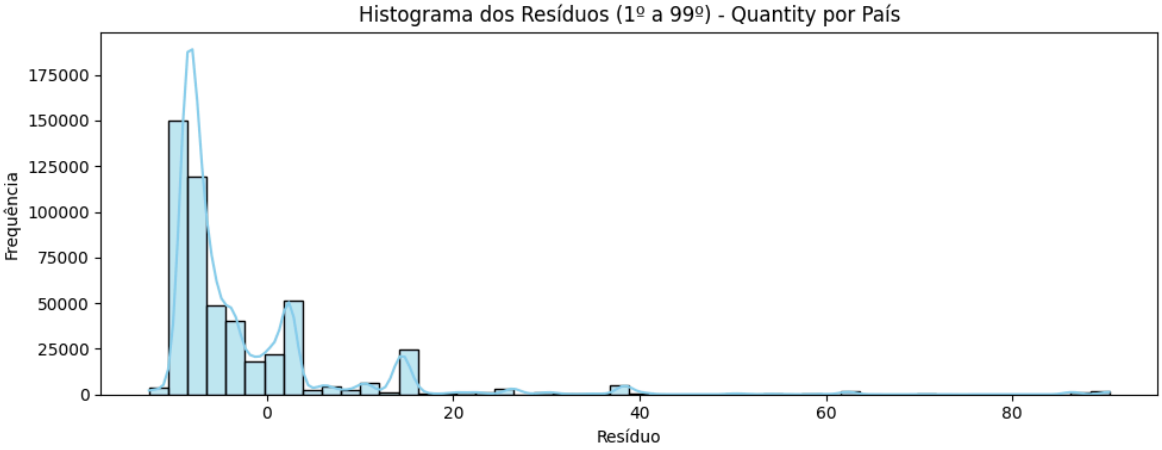
Dessa forma, o país afeta mais significativamente a quantidade de compra do que o preço médio dos produtos.

c) Ajustes no Modelo de ANOVA (40%)

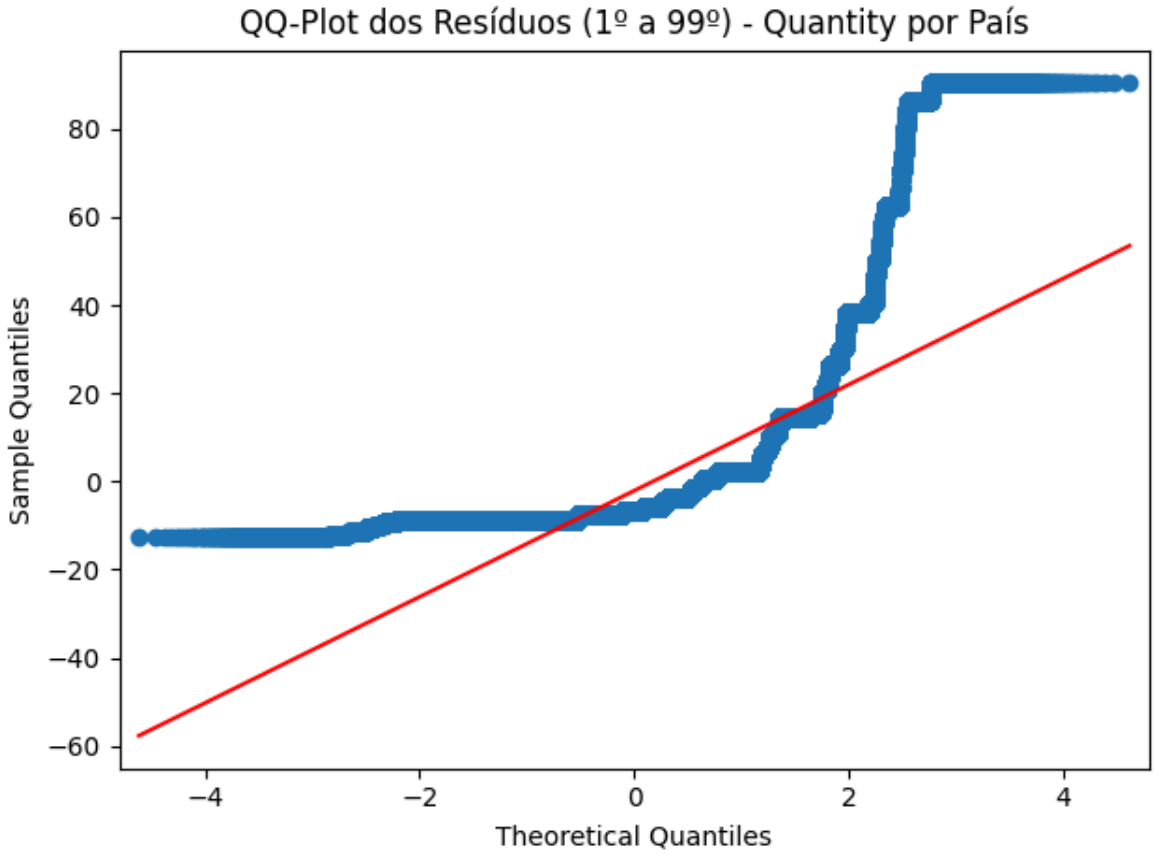
Serão agora verificados os pressupostos do ANOVA aplicado anteriormente.

In [19]:
```
# Verificação dos resíduos

import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import numpy as np
import scipy.stats as stats

# RESÍDUOS - QUANTITY por País


# Média por país
mean_quantity = dados_anova.groupby('Country')['Quantity'].transform('mean')
# Resíduos
residuos_quantity = dados_anova['Quantity'] - mean_quantity
# Filtragem (1º a 99º percentil)
q1_q, q99_q = np.percentile(residuos_quantity, [1, 99])
residuos_quantity_filtrados = residuos_quantity[(residuos_quantity >= q1_q) & (r

# Histograma
plt.figure(figsize=(10, 4))
sns.histplot(residuos_quantity_filtrados, bins=50, kde=True, color='skyblue', ed
plt.title('Histograma dos Resíduos (1º a 99º) - Quantity por País')
plt.xlabel('Resíduo')
```

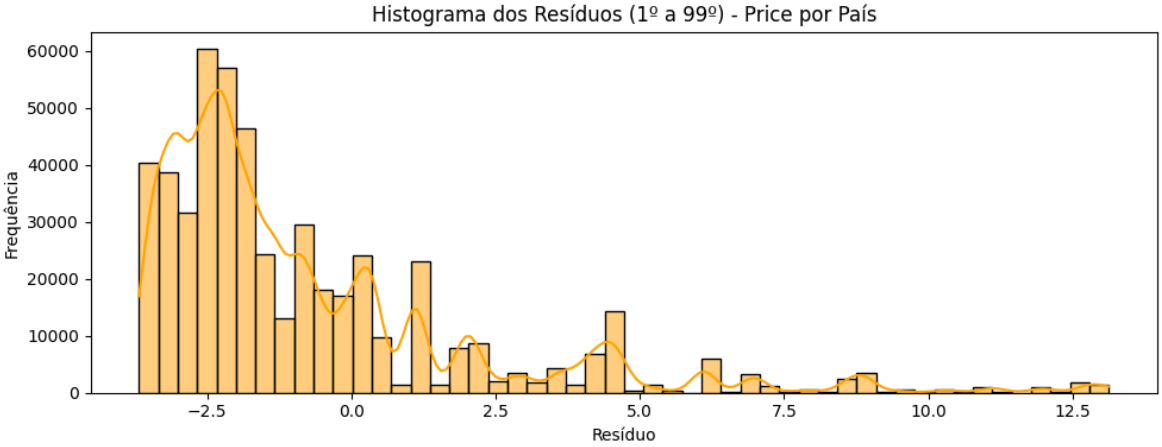
```
plt.ylabel('Frequência')
plt.tight_layout()
plt.show()

# QQ-Plot
plt.figure(figsize=(6, 6))
sm.qqplot(residuos_quantity_filtrados, line='s')
plt.title('QQ-Plot dos Resíduos (1º a 99º) - Quantity por País')
plt.tight_layout()
plt.show()

# Teste de normalidade
amostra_q = residuos_quantity_filtrados.sample(500, random_state=42) if len(resi
stat_q, p_q = stats.shapiro(amostra_q)
print(f"Shapiro Wilk (Quantity): estatística = {stat_q:.4f}, p valor = {p_q:.4f}
if p_q < 0.05:
    print("Os resíduos de Quantity NÃO seguem distribuição normal.")
else:
    print("Os resíduos de Quantity seguem distribuição normal.")

# RESÍDUOS - PRICE por País

# Média por país
mean_price = dados_anova.groupby('Country')['Price'].transform('mean')
# Resíduos
residuos_price = dados_anova['Price'] - mean_price
# Filtragem (1º a 99º percentil)
q1_p, q99_p = np.percentile(residuos_price, [1, 99])
residuos_price_filtrados = residuos_price[(residuos_price >= q1_p) & (residuos_p

# Histograma
plt.figure(figsize=(10, 4))
sns.histplot(residuos_price_filtrados, bins=50, kde=True, color='orange', edgeco
plt.title('Histograma dos Resíduos (1º a 99º) - Price por País')
plt.xlabel('Resíduo')
plt.ylabel('Frequência')
plt.tight_layout()
plt.show()

# QQ-Plot
plt.figure(figsize=(6, 6))
sm.qqplot(residuos_price_filtrados, line='s')
plt.title('QQ-Plot dos Resíduos (1º a 99º) - Price por País')
plt.tight_layout()
plt.show()

# Teste de normalidade
amostra_p = residuos_price_filtrados.sample(500, random_state=42) if len(residuo
stat_p, p_p = stats.shapiro(amostra_p)
print(f"Shapiro Wilk (Price): estatística = {stat_p:.4f}, p valor = {p_p:.4f}")
if p_p < 0.05:
    print("→ Os resíduos de Price NÃO seguem distribuição normal.")
else:
    print("Os resíduos de Price seguem distribuição normal.")
```



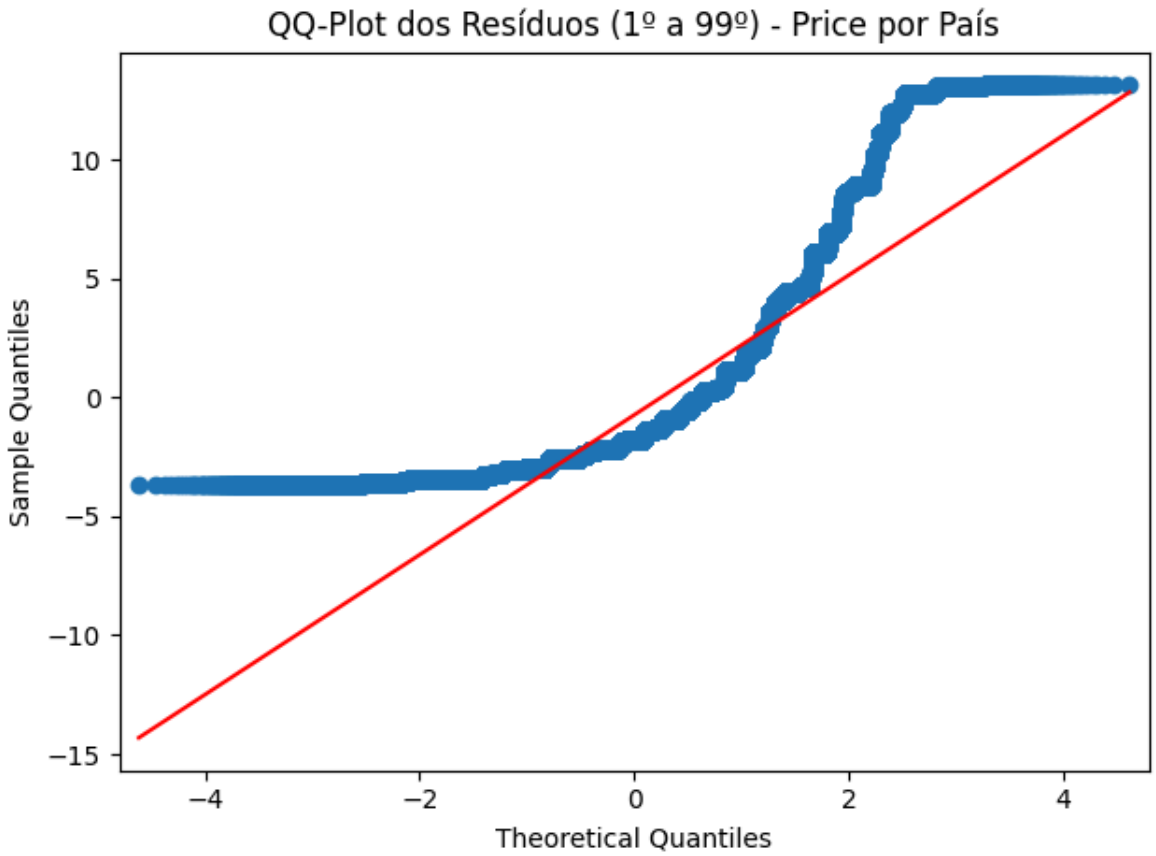

```
<Figure size 600x600 with 0 Axes>
```




```
Shapiro Wilk (Quantity): estatística = 0.5868, p valor = 0.0000
Os resíduos de Quantity NÃO seguem distribuição normal.
```




```
<Figure size 600x600 with 0 Axes>
```




```
Shapiro Wilk (Price): estatística = 0.7441, p valor = 0.0000
→ Os resíduos de Price NÃO seguem distribuição normal.
```

VISUALMENTE FOI POSSÍVEL OBSERVAR QUE OS RESÍDOUS NÃO SEGUEM UMA CURVA NORMAL. SERÁ AVALIADA EM SEGUIDA A HOMOCEDASTICIDADE, OU SEJA, A HOMOGENEIDADE DAS VARIÂNCIAS POR MEIO DO TESTE DE LEVENE.

In [20]:
```python
from scipy.stats import levene

# Teste de Homocedasticidade (Levene)

# Agrupar valores por país
quantity_groups = [g['Quantity'].values for _, g in dados_anova.groupby('Country
price_groups = [g['Price'].values for _, g in dados_anova.groupby('Country')]

# para imprimir o resultado do teste
def testar_levene(grupos, nome_variavel):
    stat, p = levene(*grupos)
    print(f"Teste de Levene - {nome_variavel} por País")
    print(f"Estatística = {stat:.2f}, p valor = {p:.4f}")
    if p < 0.05:
        print("As variâncias NÃO são homogêneas (não homocedasticidade).\n")
    else:
        print("As variâncias são homogêneas (homocedasticidade).\n")

# Aplicar o teste
testar_levene(quantity_groups, 'Quantity')
testar_levene(price_groups, 'Price')
```

```
Teste de Levene - Quantity por País
Estatística = 41.77, p valor = 0.0000
As variâncias NÃO são homogêneas (não homocedasticidade).

Teste de Levene - Price por País
Estatística = 2.01, p valor = 0.0283
As variâncias NÃO são homogêneas (não homocedasticidade).
```

COMO OS DADOS NÃO SÃO HOMOCEDÁSTICOS NEM POSSUEM RESÍDUOS QUE RSPEITAM UMA NORMAL, SERÁ APLICADA UMA TRANSFORMAÇÃO NOS DADOS PARA AJUSTÁ-LOS PARA A APLICAÇÃO DO ANOVA. CASO NÃO SEJA POSSÍVEL, SERÃO USADOS MODELOS MAIS ROBUSTOS, QUE NÃO REQUEREM HOMOCEDDASTICIDADE NEM NORMALIDADE DOS RESÍDUOS.

In [21]:
```python
import numpy as np

# transformação com box cox

from scipy.stats import boxcox

dados_anova['boxcox_Quantity'], _ = boxcox(dados_anova['Quantity'])
dados_anova['boxcox_Price'], _ = boxcox(dados_anova['Price'])
```

In [22]:
```python
# refazimento do teste de Levene com variável transformada

# Reagrupar os valores por país com os dados transformados
boxcox_quantity_groups = [g['boxcox_Quantity'].values for _, g in dados_anova.gr
boxcox_price_groups = [g['boxcox_Price'].values for _, g in dados_anova.groupby(

# teste de Levene
def testar_levene(grupos, nome_variavel):
    stat, p = levene(*grupos)
    print(f"Teste de Levene - {nome_variavel} (Box-Cox)")
    print(f"Estatística = {stat:.2f}, p valor = {p:.4f}")
    if p < 0.05:
        print("As variâncias AINDA NÃO são homogêneas\n")
    else:
        print("As variâncias SÃO homogêneas (homocedasticidade atendida)\n")

# Executar o teste com os dados transformados
testar_levene(boxcox_quantity_groups, 'Quantity')
testar_levene(boxcox_price_groups, 'Price')
```

```
Teste de Levene - Quantity (Box-Cox)
Estatística = 1569.86, p valor = 0.0000
As variâncias AINDA NÃO são homogêneas

Teste de Levene - Price (Box-Cox)
Estatística = 31.44, p valor = 0.0000
As variâncias AINDA NÃO são homogêneas
```

Após aplicação do logarítimo na variável quantity, observou-se a permanencia da heterocedasticidade, portanto, será aplicado o modelo de Kruskal-Walli.

O teste de Kruskal-Wallis é uma alternativa não paramétrica ao ANOVA e não exige normalidade nem homogeneidade de variâncias, sendo apropriado para o caso em

questão. Esse teste avalia a mediana do conjunto de dados, e portanto não é sensível a outliers como o ANOVA, que analisa a média.

In [23]:
```
from scipy.stats import kruskal

# Agrupar os valores por país
quantity_groups = [g['Quantity'].values for _, g in dados_anova.groupby('Country
price_groups = [g['Price'].values for _, g in dados_anova.groupby('Country')]

# Kruskal-Wallis em Quantity
stat_kw_q, p_kw_q = kruskal(*quantity_groups)
print("Kruskal-Wallis - Quantity por País")
print(f"Estatística = {stat_kw_q:.2f}, p-valor = {p_kw_q:.4f}")
if p_kw_q < 0.05:
    print("Diferenças significativas nas medianas de Quantity entre os países.\n
else:
    print("Não há diferenças significativas nas medianas de Quantity entre os pa

# Kruskal-Wallis - Price
stat_kw_p, p_kw_p = kruskal(*price_groups)
print("Kruskal-Wallis - Price por País")
print(f"Estatística = {stat_kw_p:.2f}, p-valor = {p_kw_p:.4f}")
if p_kw_p < 0.05:
    print("Diferenças significativas nas medianas de Price entre os países.")
else:
    print("Não há diferenças significativas nas medianas de Price entre os paíse
```

```
Kruskal-Wallis - Quantity por País
Estatística = 24491.95, p-valor = 0.0000
Diferenças significativas nas medianas de Quantity entre os países.

Kruskal-Wallis - Price por País
Estatística = 560.61, p-valor = 0.0000
Diferenças significativas nas medianas de Price entre os países.
```

d) Interpretação e Tomada de Decisão (10%)

- Quantity por País Estatística = 24.491,95 | p-valor = 0.000

Isso indica que as quantidades vendidas diferem entre os países, ou seja, países diferentes tendem a ter padrões de compra diferentes (comprar mais ou menos).

- Price por País Estatística = 560,61 | p-valor = 0.000

As medianas de preço também variam entre os países, indicando que países diferentes possuem "disposições" diferentes para o gasto com as compras.

DECISÕES ESTRATÉGICAS

1- Para países com Quantity alta, para se manter essa variável alta é possível gerar ações de fidelização de clientes, como também programas de pontos, descontos progressivos ou benefícios exclusivos para clientes recorrentes, para evitar que o volume de compras reduza com o tempo.

2- Para países com Quantity baixa, podem ser dados alguns incentivos à compra, como frete grátis, cupons de primeira compra ou kits promocionais, ou aumentar a demanda com maiores investimentos em marketing digital ou local.

3- Para países com Price baixo, pode ser interessante agregar maior valor aos produtos mais comprados, como ofertas combinadas ou garantias estendidas, e fazer estudos sobre o poder de compra local, para analisar o aumento do valor dos produtos sem provocar queda proporcional do número de compras.

Questão 4

a) Discussãao sobre o problema (10%)

Fazer uma boa gestão das reclamações em ambiente de varejo é crucial para sustentar a lucratividade do negócio. Primeiro porque é necessário manter saudável a base ativa de clientes, ou seja, quem já efetuou uma compra precisa ter uma boa experiência para comprar mais vezes. Por outro lado, aquele que não comprou, mas ouviu relatos de experiências ruins pode ser desencorajado a comprar. Portanto, a importância estratégica está relacionada diretamente à imagem da empresa com relação à experiência do usuário ao fazer uma compra. É necessário ainda monitorar e atender com qualidade e de forma eficiente todos os canais de atendimento disponibilizados com o cliente (loja, app, sac), pois as insatisfações podem ser apresentadas por qualquer um dos meios disponíveis e, caso não atendidos, podem se tornar questões comerciais ou até judiciais, gerando alto custo para a empresa.

Nesse sentido, e considerando que é mais barato reter um cliente do que conquistar um novo, são bem vindas quaisquer ações que usem dados dos clientes para preverem reclamações ou insatisfações de forma proativa. Isso além de melhorar o relacionamento com o cliente promove de forma positiva a reputação da empresa. A antecipação da reclamação por parte da empresa mostra ao cliente potencialmente insatisfeito que a empresa está atenta e preocupada com sua experiência. O efeito disso é um cliente mais disposto a flexibilizar sua expectativa de resolução com a empresa, que já antecipou sua reclamação. A empresa com isso ganha mais tempo e um cliente menos reativo, que tende a aceitar melhor o desenrolar da questão. Além disso, após a resolução, o cliente tende a dar um feedback positivo da empresa a terceiros, relatando a forma surpreendente e antecipada com que foi abordado pela empresa.

Essa estratégia permite ainda direcionar a mão de obra disponível da empresa para os casos mais críticos, para os quais o sistema emitiu alerta ou identificou como potencial reclamação. Além disso, permite dar uma resolução mais personalizada e alinhada ao perfil do cliente, considerando seu histórico de compra, relacionamento com a empresa, etc. Essas ações também ajudam a reduzir o churn e evitar que o cliente abandone a marca.

b) Análise Descritiva dos Dados (15%)

In [24]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Especificação do caminho do arquivo csv

path = '/content/drive/MyDrive/AEDI/marketing_campaign.csv'

dados = pd.read_csv(path, sep='\t')

# Estrutura da base
dados.info()
dados.columns
dados.head(3)

# Estatísticas para variáveis numéricas
dados.describe()
print(dados)

# Valores ausentes
dados.isnull().sum().sort_values(ascending=False)

# separação por tipo de dado

# Identificando colunas numéricas
num = dados.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Identificando colunas categóricas (objetos e datas, por exemplo)
cat = dados.select_dtypes(include=['object']).columns.tolist()

# Exibindo os resultados
print("Colunas numéricas:", num)
print("Colunas categóricas:", cat)
```

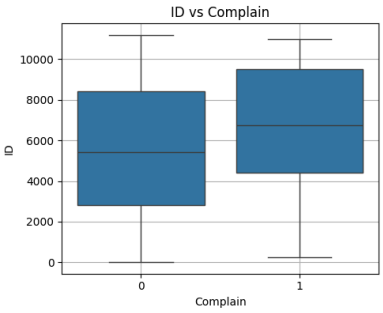
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
 20  AcceptedCmp3         2240 non-null   int64  
 21  AcceptedCmp4         2240 non-null   int64  
 22  AcceptedCmp5         2240 non-null   int64  
 23  AcceptedCmp1         2240 non-null   int64  
 24  AcceptedCmp2         2240 non-null   int64  
 25  Complain             2240 non-null   int64  
 26  Z_CostContact        2240 non-null   int64  
 27  Z_Revenue            2240 non-null   int64  
 28  Response             2240 non-null   int64  
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
         ID  Year_Birth   Education Marital_Status   Income  Kidhome  \
0      5524        1957  Graduation         Single  58138.0        0   
1      2174        1954  Graduation         Single  46344.0        1   
2      4141        1965  Graduation       Together  71613.0        0   
3      6182        1984  Graduation       Together  26646.0        1   
4      5324        1981         PhD        Married  58293.0        1   
...     ...         ...         ...            ...      ...      ...   
2235  10870        1967  Graduation        Married  61223.0        0   
2236   4001        1946         PhD       Together  64014.0        2   
2237   7270        1981  Graduation       Divorced  56981.0        0   
2238   8235        1956      Master       Together  69245.0        0   
2239   9405        1954         PhD        Married  52869.0        1   

      Teenhome Dt_Customer  Recency  MntWines  ...  NumWebVisitsMonth  \
0            0  04-09-2012       58       635  ...                  7   
1            1  08-03-2014       38        11  ...                  5   
2            0  21-08-2013       26       426  ...                  4   
3            0  10-02-2014       26        11  ...                  6   
4            0  19-01-2014       94       173  ...                  5   
...        ...         ...      ...       ...  ...                ...   
2235         1  13-06-2013       46       709  ...                  5   
2236         1  10-06-2014       56       406  ...                  7   
2237         0  25-01-2014       91       908  ...                  6   
2238         1  24-01-2014        8       428  ...                  3   
```

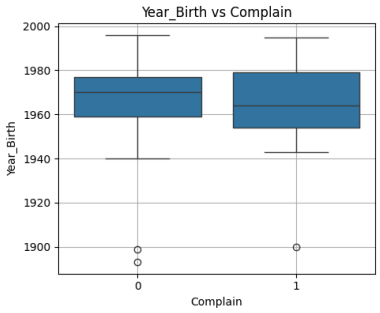
```
2239         1  15-10-2012       40        84  ...                 7

      AcceptedCmp3  AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  \
0                0             0             0             0             0
1                0             0             0             0             0
2                0             0             0             0             0
3                0             0             0             0             0
4                0             0             0             0             0
...            ...           ...           ...           ...           ...
2235             0             0             0             0             0
2236             0             0             0             1             0
2237             0             1             0             0             0
2238             0             0             0             0             0
2239             0             0             0             0             0

      Complain  Z_CostContact  Z_Revenue  Response
0            0              3         11         1
1            0              3         11         0
2            0              3         11         0
3            0              3         11         0
4            0              3         11         0
...        ...            ...        ...       ...
2235         0              3         11         0
2236         0              3         11         0
2237         0              3         11         0
2238         0              3         11         0
2239         0              3         11         1

[2240 rows x 29 columns]
Colunas numéricas: ['ID', 'Year_Birth', 'Income', 'Kidhome', 'Teenhome', 'Recenc
y', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProdu
cts', 'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchase
s', 'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3', 'AcceptedCmp4', 'Ac
ceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Reve
nue', 'Response']
Colunas categóricas: ['Education', 'Marital_Status', 'Dt_Customer']
```

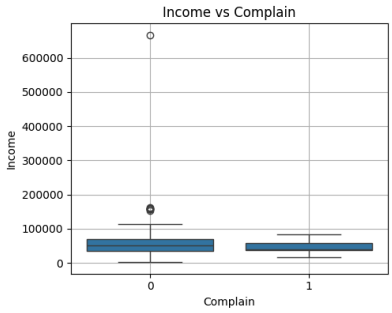
In [25]:
```
# Converter data para datetime
dados['Dt_Customer'] = pd.to_datetime(dados['Dt_Customer'], errors='coerce')
```

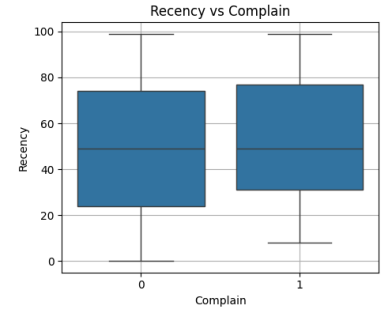
In [26]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import math

# Lista das variáveis desejadas
variaveis = [
    'ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income',
    'Kidhome', 'Teenhome', 'Dt_Customer', 'Recency', 'MntWines',
    'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
    'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
    'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth'
]

# Separar variáveis numéricas e categóricas
numericas = ['ID', 'Year_Birth', 'Income', 'Recency',
             'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts',
             'MntSweetProducts', 'MntGoldProds', 'NumDealsPurchases',
             'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases',
             'NumWebVisitsMonth']
categoricas = ['Education', 'Marital_Status']
```

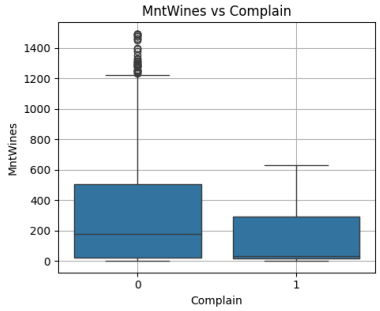
```
# Plot das variáveis numéricas em relação a Complain
cols = 3
rows = math.ceil(len(numericas) / cols)
plt.figure(figsize=(cols * 5, rows * 4))

for i, var in enumerate(numericas):
    plt.subplot(rows, cols, i + 1)
    sns.boxplot(x='Complain', y=var, data=dados)
    plt.title(f'{var} vs Complain')
    plt.grid(True)

plt.tight_layout()
plt.show()

# Plot das variáveis categóricas em relação a Complain
for var in categoricas:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=var, hue='Complain', data=dados)
    plt.title(f'{var} vs Complain')
    plt.xticks(rotation=30)
    plt.tight_layout()
    plt.show()
```

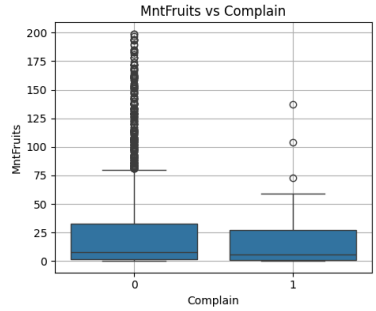

ID vs Complain
ID
Complain

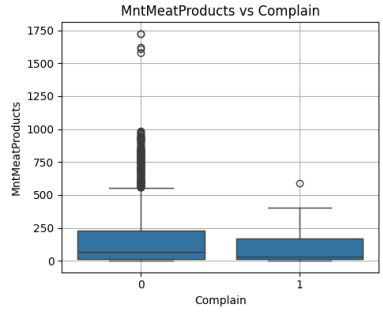

Year_Birth vs Complain
Year_Birth
Complain

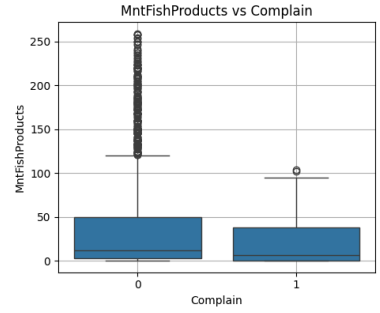

Income vs Complain
Income
Complain

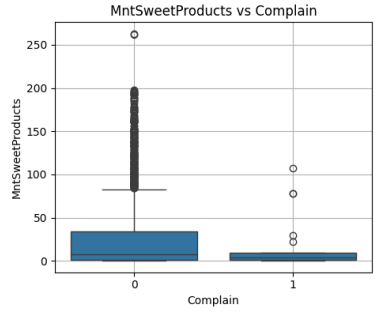

Recency vs Complain
Recency
Complain

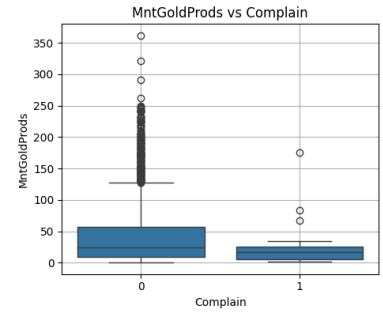

MntWines vs Complain
MntWines
Complain


MntFruits vs Complain
MntFruits
Complain

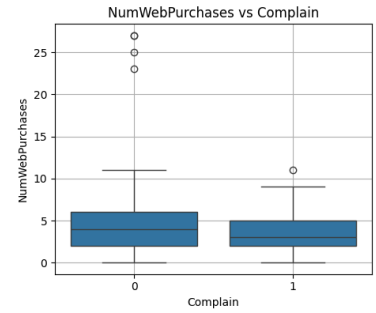

MntMeatProducts vs Complain
MntMeatProducts
Complain

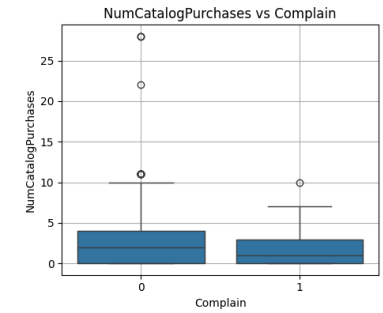

MntFishProducts vs Complain
MntFishProducts
Complain


MntSweetProducts vs Complain
MntSweetProducts
Complain

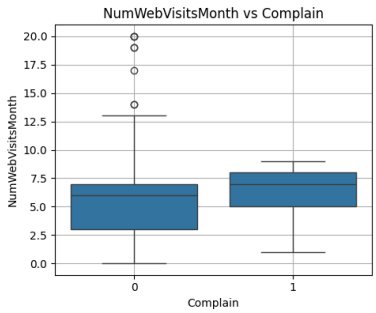

MntGoldProds vs Complain
MntGoldProds
Complain

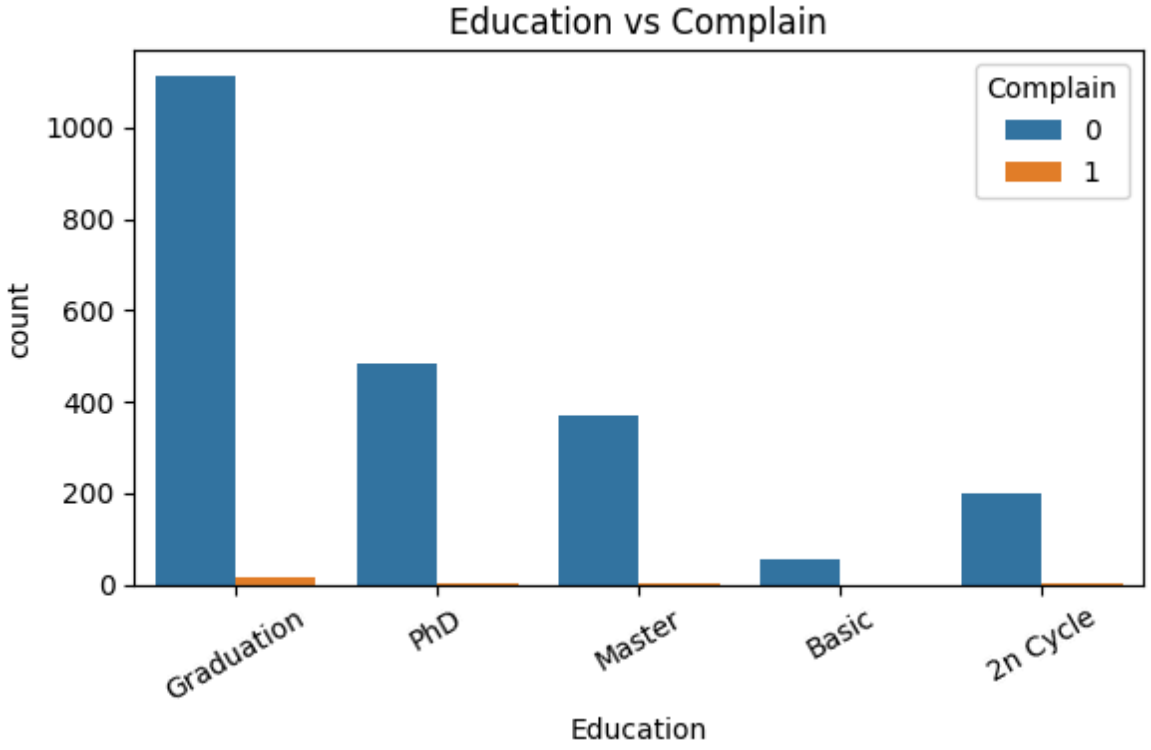

NumDealsPurchases vs Complain
NumDealsPurchases
Complain

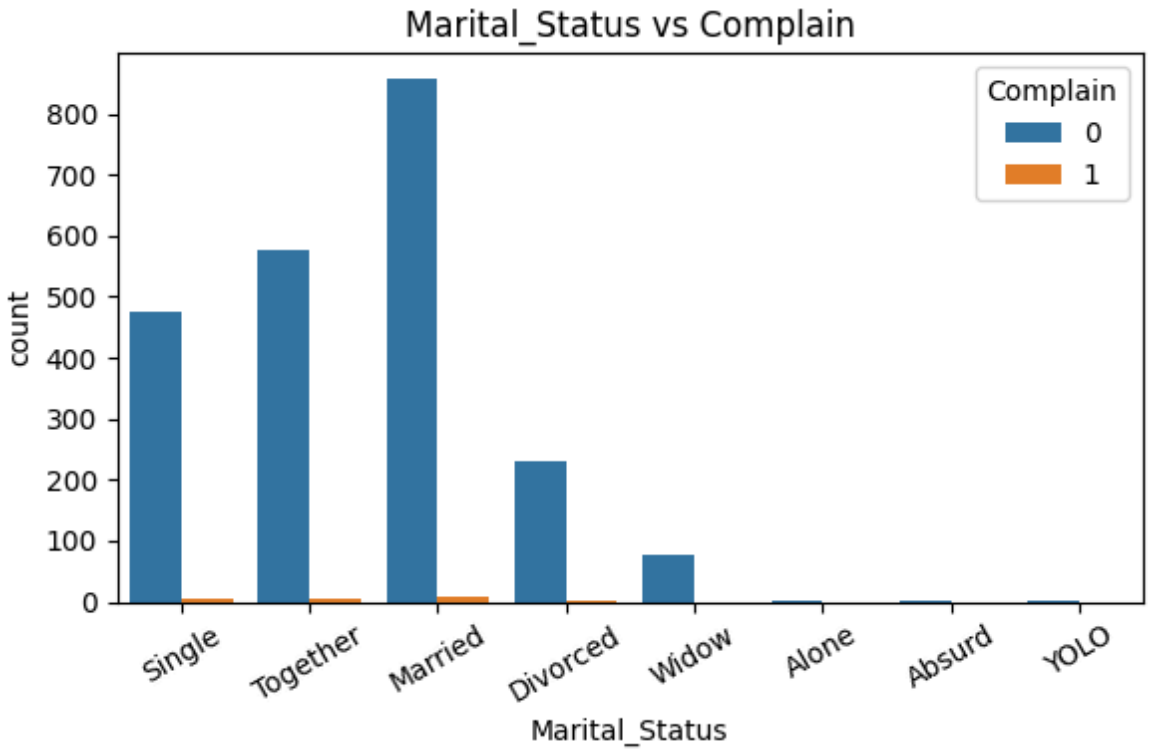

NumWebPurchases vs Complain
NumWebPurchases
Complain


NumCatalogPurchases vs Complain
NumCatalogPurchases
Complain


NumStorePurchases vs Complain
NumStorePurchases
Complain

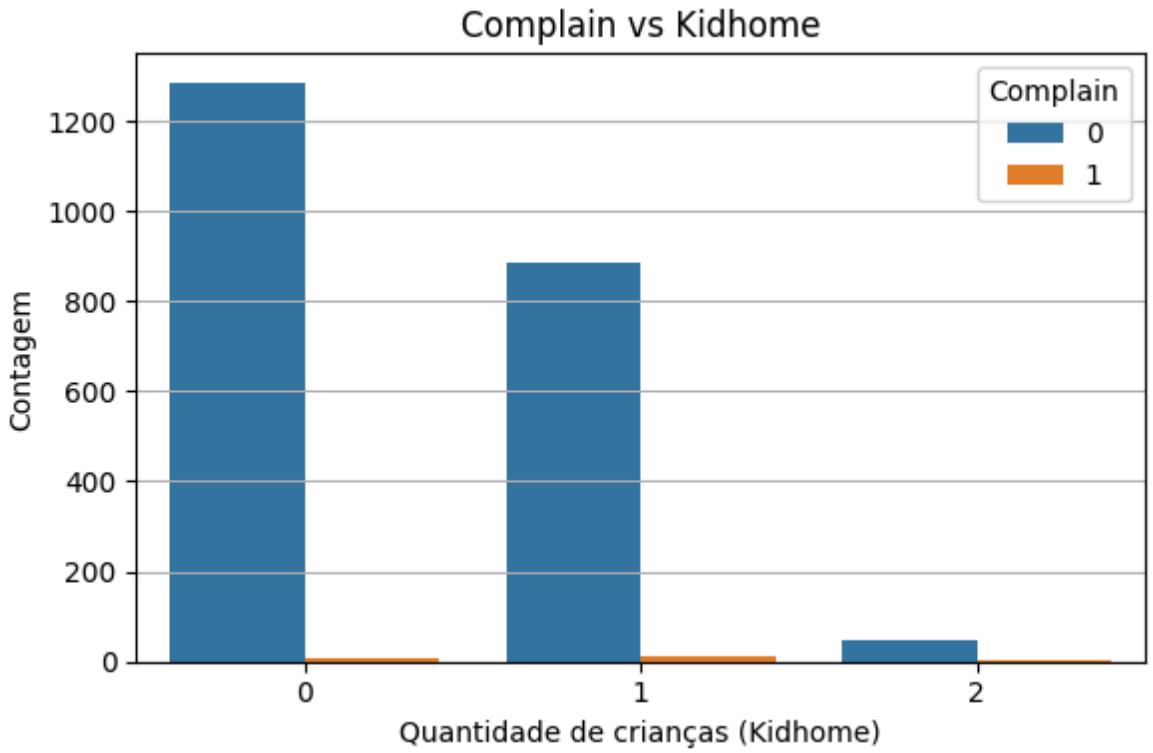

NumWebVisitsMonth vs Complain
NumWebVisitsMonth
Complain

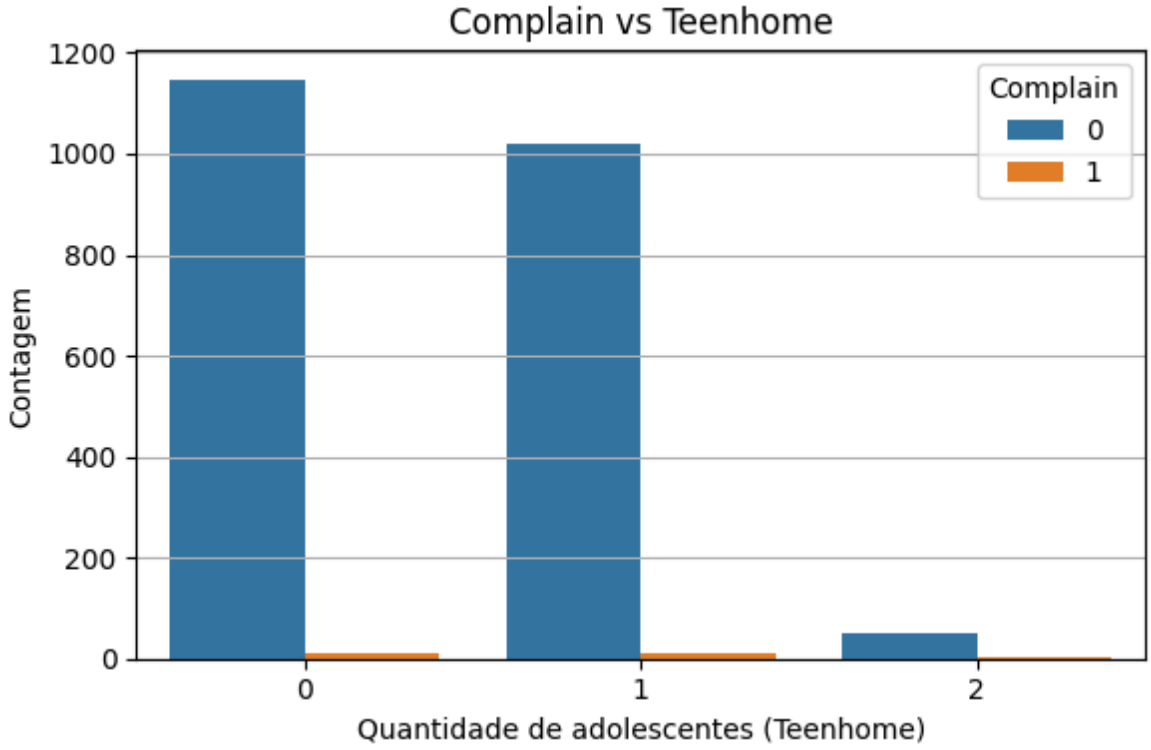


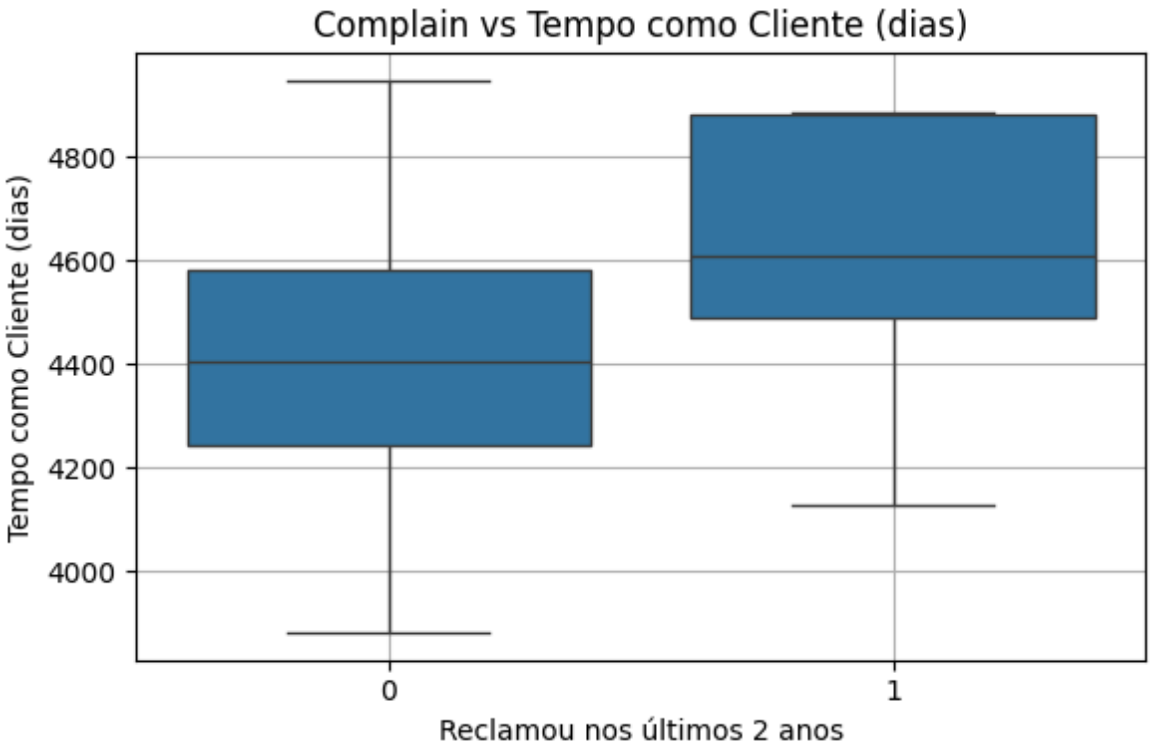


In [27]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import math
import pandas as pd

# Converter Dt_Customer para datetime
dados['Dt_Customer'] = pd.to_datetime(dados['Dt_Customer'], errors='coerce')

# Criar nova coluna com tempo como cliente em anos
dados['Customer_For_Years'] = ((pd.to_datetime('today') - dados['Dt_Customer']).

# Gráfico de barras para Kidhome
plt.figure(figsize=(6, 4))
```

```
sns.countplot(x='Kidhome', hue='Complain', data=dados)
plt.title('Complain vs Kidhome')
plt.xlabel('Quantidade de crianças (Kidhome)')
plt.ylabel('Contagem')
plt.grid(axis='y')
plt.tight_layout()
plt.show()

# Gráfico de barras para Teenhome
plt.figure(figsize=(6, 4))
sns.countplot(x='Teenhome', hue='Complain', data=dados)
plt.title('Complain vs Teenhome')
plt.xlabel('Quantidade de adolescentes (Teenhome)')
plt.ylabel('Contagem')
plt.grid(axis='y')
plt.tight_layout()
plt.show()

#  nova coluna com tempo de cliente em dias
dados['Customer_For_Days'] = (pd.to_datetime('today') - dados['Dt_Customer']).dt

# Boxplot relacionando tempo como cliente e reclamações
plt.figure(figsize=(6, 4))
sns.boxplot(x='Complain', y='Customer_For_Days', data=dados)
plt.title('Complain vs Tempo como Cliente (dias)')
plt.xlabel('Reclamou nos últimos 2 anos')
plt.ylabel('Tempo como Cliente (dias)')
plt.grid(True)
plt.tight_layout()
plt.show()
```







c) Definição e Seleção dos Modelos (30%)

In [35]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc


# Remover linhas com qualquer valor ausente
dados = dados.dropna()

# Garantir que Dt_Customer esteja como datetime
```

```
dados['Dt_Customer'] = pd.to_datetime(dados['Dt_Customer'], errors='coerce')

# Codificação das variáveis categóricas
dados_encoded = pd.get_dummies(dados, columns=['Education', 'Marital_Status'], d

# Criação da variável-alvo
y = dados_encoded['Complain']

# Seleção de features (remover colunas não preditivas)
X = dados_encoded.drop(columns=['Complain', 'ID', 'Dt_Customer'])

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Escalonamento para modelos que precisam (Regressão Logística, XGBoost)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [36]:
```
from sklearn.linear_model import LogisticRegression

# Modelo de regressão logística
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_scaled, y_train)
y_pred_log = log_model.predict(X_test_scaled)
y_proba_log = log_model.predict_proba(X_test_scaled)[:, 1]
```

b) Justificativa A Regressão Logística é um modelo simples e interpretável, ideal como ponto de partida para problemas de classificação binária como este (prever se o cliente reclamou ou não). Ela permite entender a relação entre variáveis independentes e a probabilidade de ocorrência de uma classe, e geralmente tem bom desempenho em bases estruturadas e balanceadas.

In [37]:
```
print("Regressão Logística")
print("Acurácia:", accuracy_score(y_test, y_pred_log))
print("Precisão:", precision_score(y_test, y_pred_log))
print("Recall:", recall_score(y_test, y_pred_log))
print("F1-score:", f1_score(y_test, y_pred_log))
print("AUC:", roc_auc_score(y_test, y_proba_log))
```

```
Regressão Logística
Acurácia: 0.9834254143646409
Precisão: 0.0
Recall: 0.0
F1-score: 0.0
AUC: 0.7902621722846441
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no p
redicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [38]:
```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier(random_state=42)
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)
y_proba_tree = tree_model.predict_proba(X_test)[:, 1]
```

b) Justificativa A Árvore de Decisão é um modelo intuitivo que funciona bem em conjuntos de dados com variáveis mistas (numéricas e categóricas). Ela é útil para explicabilidade e análise de regras, além de não exigir normalização dos dados. É adequada para identificar padrões complexos em bases tabulares como esta.

In [48]:
```
print("Árvore de Decisão")
print("Acurácia:", accuracy_score(y_test, y_pred_tree))
print("Precisão:", precision_score(y_test, y_pred_tree))
print("Recall:", recall_score(y_test, y_pred_tree))
print("F1-score:", f1_score(y_test, y_pred_tree))
print("AUC:", roc_auc_score(y_test, y_proba_tree))
```

```
Árvore de Decisão
Acurácia: 0.988950276243094
Precisão: 1.0
Recall: 0.3333333333333333
F1-score: 0.5
AUC: 0.6666666666666666
```

In [49]:
```
# Garantir que todas as colunas são numéricas e válidas
dados_encoded = pd.get_dummies(dados.drop(columns=['Dt_Customer'], errors='ignor
dados_encoded = dados_encoded.dropna()

# Separar variável-alvo e preditores
y = dados_encoded['Complain']
X = dados_encoded.drop(columns=['Complain', 'ID'], errors='ignore')

# Garantir tipos corretos
X = X.astype(float)

# Train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Re-treinar o modelo com DataFrame corretamente alinhado
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

Out[49]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

b) Justificativa O Random Forest combina várias árvores de decisão para reduzir overfitting e melhorar a precisão e robustez do modelo. É indicado quando há múltiplas variáveis com interações complexas, como neste caso com dados demográficos, comportamentais e financeiros.

In [50]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

# Geração das previsões
y_pred_rf = rf_model.predict(X_test)
y_proba_rf = rf_model.predict_proba(X_test)[:, 1]

# Exibição das métricas
print("Random Forest")
```

```
print("Acurácia:", accuracy_score(y_test, y_pred_rf))
print("Precisão:", precision_score(y_test, y_pred_rf, zero_division=0))
print("Recall:", recall_score(y_test, y_pred_rf, zero_division=0))
print("F1-score:", f1_score(y_test, y_pred_rf, zero_division=0))
print("AUC:", roc_auc_score(y_test, y_proba_rf))
```

```
Random Forest
Acurácia: 0.988950276243094
Precisão: 1.0
Recall: 0.3333333333333333
F1-score: 0.5
AUC: 0.9307116104868914
```

In [51]:
```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)
y_proba_xgb = xgb_model.predict_proba(X_test)[:, 1]
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning: [2
1:57:22] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

b) Justificativa O XGBoost é um dos algoritmos mais poderosos para tarefas de classificação em dados tabulares. Ele é eficaz para detectar padrões sutis, trata bem dados desbalanceados, e geralmente oferece melhor desempenho preditivo, mesmo com menos necessidade de ajustes manuais.

In [52]:
```
print("XGBoost")
print("Acurácia:", accuracy_score(y_test, y_pred_xgb))
print("Precisão:", precision_score(y_test, y_pred_xgb))
print("Recall:", recall_score(y_test, y_pred_xgb))
print("F1-score:", f1_score(y_test, y_pred_xgb))
print("AUC:", roc_auc_score(y_test, y_proba_xgb))
```

```
XGBoost
Acurácia: 0.988950276243094
Precisão: 1.0
Recall: 0.3333333333333333
F1-score: 0.5
AUC: 0.7172284644194756
```

Explicação básica sobre algumas métricas de avaliação dos modelos:

1. Acurácia: A proporção de previsões corretas em relação ao total de previsões. Quando é útil: Quando as classes estão balanceadas.

2. Precisão: Entre todos os clientes que o modelo previu como “reclamantes”, quantos realmente reclamaram.

Quando é útil: Quando o custo de um falso positivo é alto, ou seja, evitar classificar erroneamente um cliente como reclamante quando ele não é.

3. Recall: Entre todos os clientes que realmente reclamaram, quantos foram identificados pelo modelo.

Quando é útil: Quando o mais importante é não perder casos positivos, como aqui — identificar todos os reclamantes possíveis para agir antes da insatisfação.

4. F1-score: Um equilíbrio entre precisão e recall.

Quando é útil: Quando você precisa de um equilíbrio entre identificar corretamente os positivos e evitar alarmes falsos — especialmente em bases desbalanceadas.

5. AUC: A capacidade do modelo de distinguir entre classes (reclamante vs não reclamante). Varia de 0 a 1, quanto mais próximo de 1, melhor.

Quando é útil: Quando queremos saber o quão bem o modelo separa os casos positivos dos negativos, independentemente do limiar de decisão.

Apesar da acurácia semelhante entre os modelos testados, essa métrica não é suficiente para avaliar o desempenho, dado o desbalanceamento natural da variável Complain. As diferenças relevantes aparecem em métricas como AUC, recall e F1-score.

XGBoost e Random Forest tiveram resultados idênticos nas métricas básicas, mas o Random Forest apresentou AUC significativamente maior (0.93 x 0.71), o que indica maior capacidade de distinguir entre clientes reclamantes e não reclamantes.

A Árvore de Decisão também identificou os reclamantes, mas teve AUC inferior (0.66), sugerindo menor capacidade de generalização.

A Regressão Logística teve o pior desempenho. Não identificou nenhum cliente reclamante (recall = 0), e apesar do AUC razoável (0.79), falhou completamente na detecção de casos positivos.

Modelo que obteve melhores resultados para o problema proposto: Random Forest.

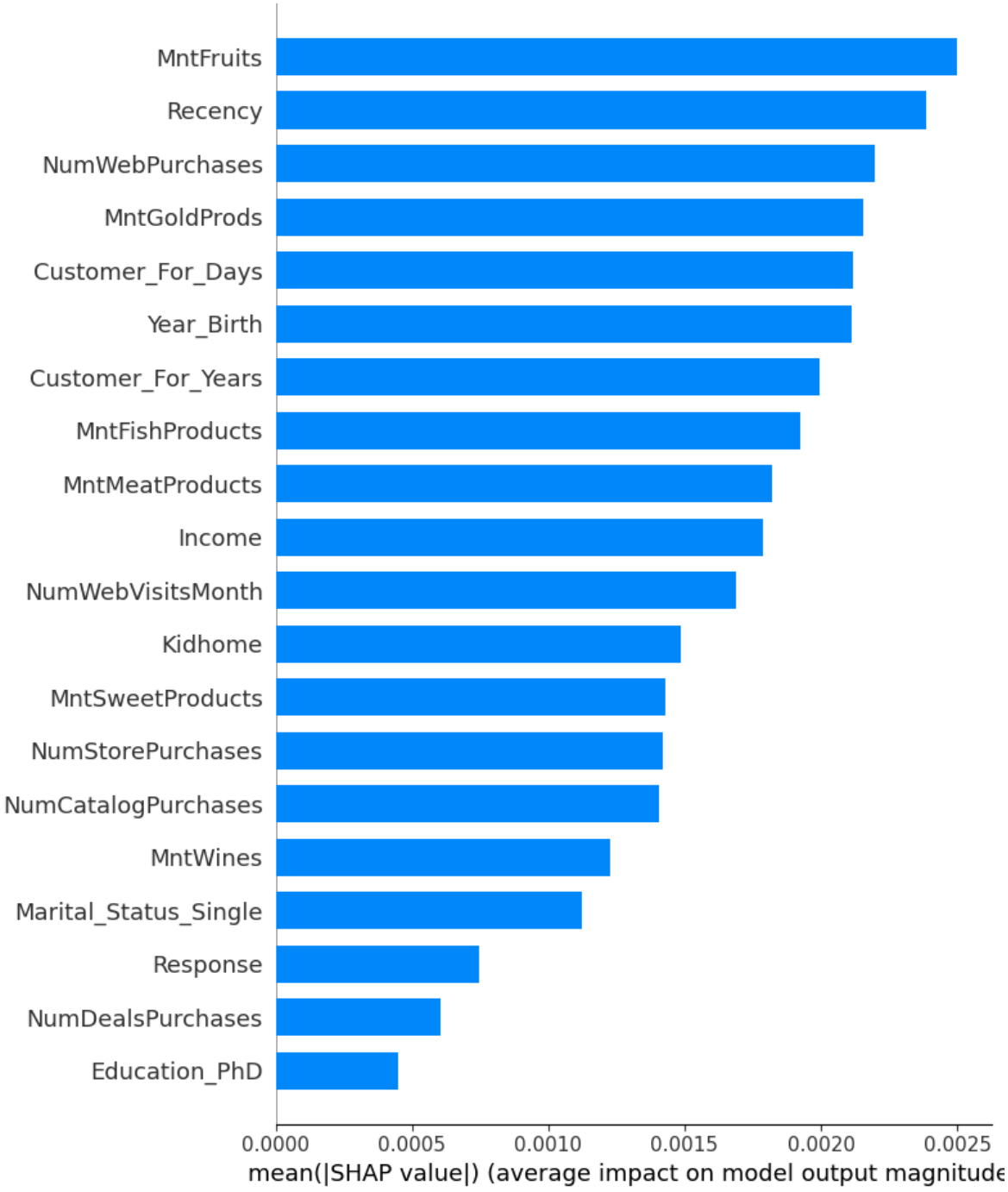
O objetivo é antecipar reclamações para agir antes que o cliente manifeste insatisfação. Isso exige um modelo que consiga identificar corretamente os casos positivos, mesmo que sejam minoria. Nesse sentido, o Random Forest se destacou porque:

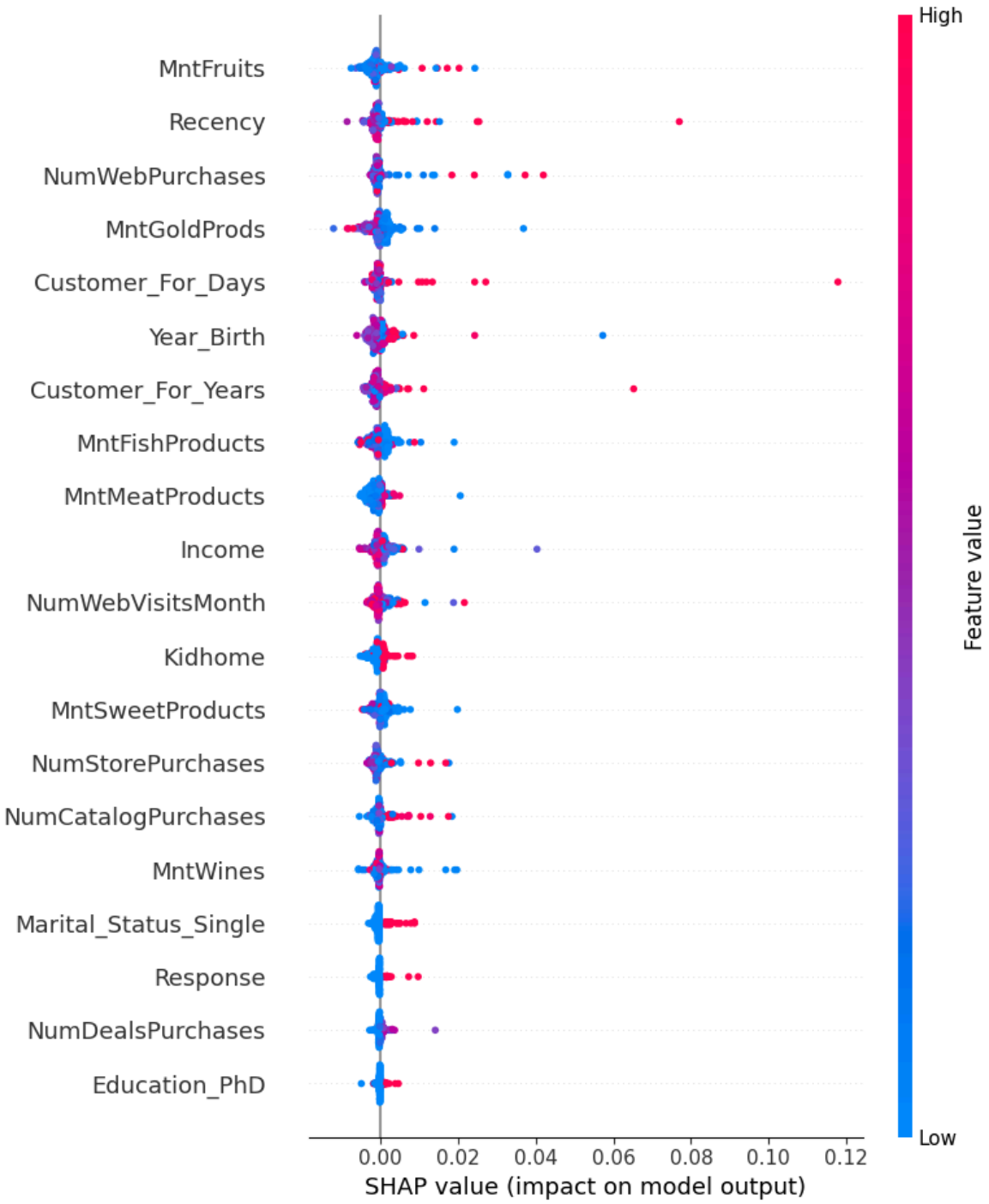
1. Manteve boa acurácia sem perder recall
2. Alcançou o melhor AUC, com boa separação entre classes
3. É resistente a dados desbalanceados (como no caso dessa base)

d) Explicabilidade das Variáveis : SHAP value (15%)

In [53]:
```
import shap

# Garantir que os dados estão em formato de DataFrame com colunas nomeadas
X_test_df = pd.DataFrame(X_test, columns=X.columns)

# Criar o explainer com modelo já treinado
explainer = shap.TreeExplainer(rf_model)
```

```
# Calcular os SHAP values
shap_values = explainer.shap_values(X_test_df)

# Extrair SHAP values da classe 1 (reclamou)
shap_values_class1 = shap_values[:, :, 1]

# Gerar gráficos
shap.summary_plot(shap_values_class1, X_test_df, plot_type='bar')
shap.summary_plot(shap_values_class1, X_test_df)
```





De acordo com o SHAP Summary Plot acima, é possível concluir que:

1- A variável que tem maior efeito sobre o ato de reclamar é o alto gasto do cliente com frutas, ou seja, clientes com alto volume de compra de frutas tendem a reclamar mais quando insatisfeitos.

2- Em segundo lugar, clientes com mais tempo de relacionamento também tendem a reclamar mais, ou seja, usuários antigos são mais propensos a reclamar.

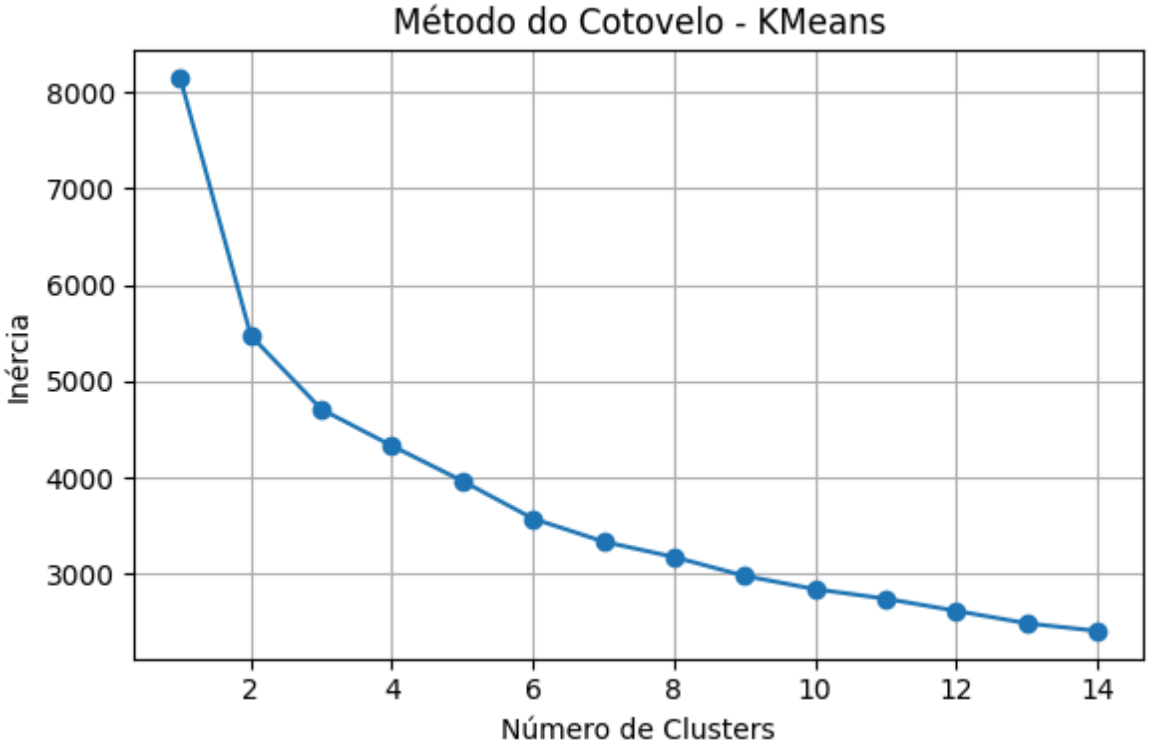
3- Clientes que gastam menos com peixe tendem a reclamar mais e clientes que gastam mais com carne tendem a reclamar mais. Ou seja, o padrão de consumo de proteínas influencia a tendência de reclamação do cliente.

4- Clientes mais novos reclamam mais

5- Clientes com padrão de vida maior (maior Income) tendem a reclamar menos

e) Análise não Supervisionada com K-Means e DBSCAN (25%)

In [54]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Seleção de variáveis para clusterização
cluster_vars = ['Income', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishPr
                'MntSweetProducts', 'Kidhome', 'Teenhome', 'Customer_For_Years']

# Normalização dos dados
X_cluster = dados[cluster_vars].dropna()
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)

# Teste do método do cotovelo para definir número de clusters
inertia = []
for k in range(1, 15):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot
plt.figure(figsize=(6,4))
plt.plot(range(1, 15), inertia, marker='o')
plt.title('Método do Cotovelo - KMeans')
plt.xlabel('Número de Clusters')
plt.ylabel('Inércia')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Com base na análise do gráfico acima, pelo método do cotovelo, determinou-se a quantidade de cluster como sendo 3, pois nesse valor, é possível perceber que a diferença de inércia já é menor (menor inclinação da curva) com relação ao n = 2 e não há grande diferença de inclinação para n = 4. Considerando o cenário real, de necessidade de criação de perfis de clientes, opta-se por manter também um baixo número de cluster para que cada cluster possa representar um perfil diferente.

In [55]:
```
# Definição do número de clusters com base no cotovelo
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
dados['Cluster_KMeans'] = kmeans.fit_predict(X_scaled)

# Visualização com PCA - redução de dimensionalidade
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
dados['PCA1'] = X_pca[:, 0]
dados['PCA2'] = X_pca[:, 1]

# Visualização dos clusters
plt.figure(figsize=(6,5))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_KMeans', data=dados, palette='S
plt.title('Clusters de Clientes - KMeans + PCA')
plt.grid(True)
plt.tight_layout()
plt.show()

# Perfil médio dos clusters
perfil_clusters = dados.groupby('Cluster_KMeans')[cluster_vars].mean()
print(perfil_clusters)
```

Cluster_KMeans	Income	MntWines	MntFruits	MntMeatProducts
0	76785.473282	588.709924	71.099237	444.732824
1	34578.829787	55.800532	6.401596	28.747340
2	57063.955056	389.063670	15.955056	110.157303

Cluster_KMeans	MntFishProducts	MntSweetProducts	Kidhome	Teenhome
0	101.820611	70.717557	0.061069	0.217557
1	8.816489	5.587766	0.960106	0.372340
2	20.970037	14.872659	0.116105	0.992509

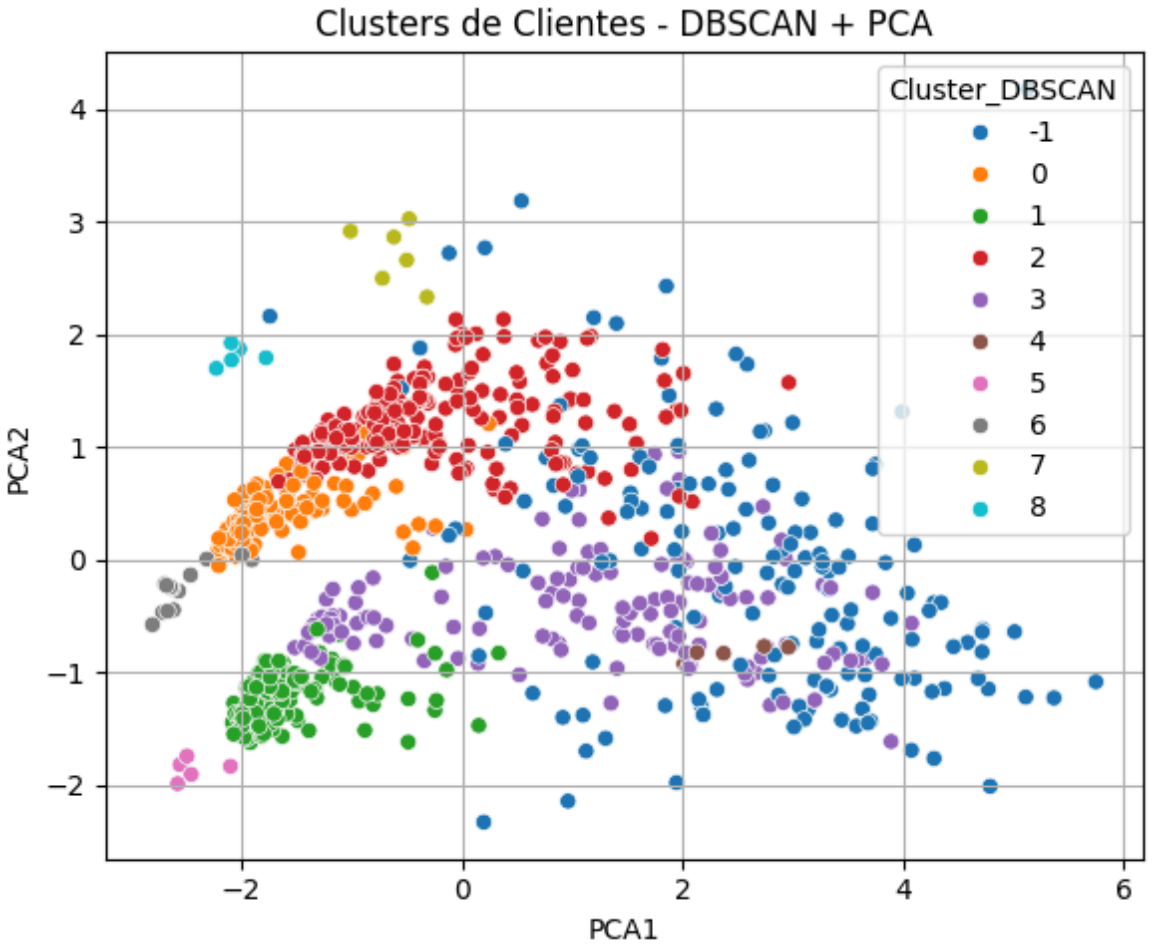
Cluster_KMeans	Customer_For_Years
0	12.144695
1	12.020266
2	12.068951

A quantidade de cluster pode parecer elevada para a variável customer for years, por exemplo, pois varia-se muito pouco o tempo médio para cada cluster. Porém, ao se analisar a diferença de valores para a variável MntFruits (com alta relevância no cenário de complain), é possível perceber que há variação significativa entre cada cluster, o que justifica o valor de n = 3. Os perfis encontrados podem ser interpretados com base na variável de maior relevância de acordo com o valor SHAP, ou seja, clientes com alto, médio e baixo volume de compra de frutas.

In [56]:
```
from sklearn.cluster import DBSCAN

# DBSCAN usa dados escalados
```

```
dbscan = DBSCAN(eps=1.5, min_samples=5)
dados['Cluster_DBSCAN'] = dbscan.fit_predict(X_scaled)

# -1 representa outliers detectados
outliers = dados[dados['Cluster_DBSCAN'] == -1]
print("Quantidade de outliers detectados:", outliers.shape[0])

# Visualização dos clusters e outliers
plt.figure(figsize=(6,5))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_DBSCAN', data=dados, palette='t
plt.title('Clusters de Clientes - DBSCAN + PCA')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Quantidade de outliers detectados: 173
```



In [57]:
```
# Comparação entre métodos
print("Distribuição de clientes nos clusters KMeans:")
print(dados['Cluster_KMeans'].value_counts())

print("\nDistribuição de clientes nos clusters DBSCAN:")
print(dados['Cluster_DBSCAN'].value_counts())

# Comparação com a variável alvo (reclamações)
print("\nTaxa de reclamações por cluster (KMeans):")
print(dados.groupby('Cluster_KMeans')['Complain'].mean())

print("\nTaxa de reclamações por cluster (DBSCAN):")
print(dados.groupby('Cluster_DBSCAN')['Complain'].mean())
```

```
Distribuição de clientes nos clusters KMeans:
Cluster_KMeans
1    376
2    267
0    262
Name: count, dtype: int64

Distribuição de clientes nos clusters DBSCAN:
Cluster_DBSCAN
 2    211
 1    189
-1    173
 3    156
 0    137
 6     16
 7      8
 4      5
 5      5
 8      5
Name: count, dtype: int64

Taxa de reclamações por cluster (KMeans):
Cluster_KMeans
0    0.003817
1    0.015957
2    0.007491
Name: Complain, dtype: float64

Taxa de reclamações por cluster (DBSCAN):
Cluster_DBSCAN
-1    0.011561
 0    0.014599
 1    0.010582
 2    0.000000
 3    0.012821
 4    0.000000
 5    0.000000
 6    0.000000
 7    0.000000
 8    0.200000
Name: Complain, dtype: float64
```

O K-Means permitiu identificar 3 perfis distintos de clientes com base no padrão de consumo de frutas.

O DBSCAN destacou perfis fora do padrão, úteis para identificar casos atípicos ou recorrentes de insatisfação.

Ambos os métodos complementam a análise supervisionada, oferecendo uma visão estratégica para personalização e priorização do atendimento.

Tomada de Decisão Estratégica (10%)

Ações Estratégicas para Prevenir Reclamações

- Priorizar o acompanhamento do Cluster 1 do K-Means, que apresentou a maior taxa de reclamações (1,59%), podendo incluir campanhas de retenção e análise

qualitativa das reclamações.

- Investigar o Cluster 8 do DBSCAN, com taxa de reclamação de 20%, muito acima dos demais. Trata-se de um grupo pequeno e atípico, mas com forte sinal de insatisfação. Deve ser tratado como caso crítico, com atuação individualizada.

- Clientes com menor gasto em frutas e peixe estão mais propensos a reclamar, conforme indicado pelos SHAP values. Campanhas de incentivo ao consumo desses itens ou avaliação da percepção de valor desses produtos podem ajudar a reverter esse padrão.

- Clientes antigos (Customer_For_Years) também aparecem com maior impacto nas previsões de reclamação. Estratégias de valorização de clientes de longa data, como programas de fidelidade exclusivos ou contato proativo, são recomendadas.

- Renda mais baixa está associada a maior chance de reclamação, de acordo com o SHAP. Isso sugere que esse público pode ser mais sensível a falhas ou ao custo-benefício. Investir em comunicação clara e garantir entrega de valor percebido é fundamental nesse segmento.