Descrição do Projeto (99Freelas)

Requisitos: Estruturar um estudo estatístico usando Python (em Jupyter Notebook), com dados de diagnósticos aplicados entre 2022 e 2024.

Objetivos:

- Identificar padrões entre os participantes com melhor desempenho em cada nível de maturidade (são 4 níveis), com base em variáveis como tempo de registro, porte, faturamento, número de pessoas envolvidas, etc.
- Indicar o momento ideal para transição entre níveis, com base em evidências estatísticas.
- Calcular e analisar o gap (nota máxima esperada menos nota observada), considerando o ano e o nível.
- Apresentar/sugerir as metodologias estatísticas/matemáticas mais adequadas para esse tipo de análise.

Observações: Gerei dados aleatórios para preservar as informações do cliente.

```
!pip install lifelines
```

```
Requirement already satisfied: lifelines in /usr/local/lib/python3.12/dist-packages (0.30.0)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.12/dist-packages (from lifelines) (2.0.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from lifelines) (1.16.1)
Requirement already satisfied: pandas>=2.1 in /usr/local/lib/python3.12/dist-packages (from lifelines) (2.2.2)
Requirement already satisfied: matplotlib>=3.0 in /usr/local/lib/python3.12/dist-packages (from lifelines) (3.10.0)
Requirement already satisfied: autograd>=1.5 in /usr/local/lib/python3.12/dist-packages (from lifelines) (1.8.0)
Requirement already satisfied: autograd-gamma>=0.3 in /usr/local/lib/python3.12/dist-packages (from lifelines) (0.5.0)
Requirement already satisfied: formulaic>=0.2.2 in /usr/local/lib/python3.12/dist-packages (from lifelines) (1.2.0)
Requirement already satisfied: interface-meta>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from formulaic>=0.2.2->lifelines) (1.3.0)
Requirement already satisfied: narwhals>=1.17 in /usr/local/lib/python3.12/dist-packages (from formulaic>=0.2.2->lifelines) (2.4.0)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.12/dist-packages (from formulaic>=0.2.2->lifelines) (4.15.0)
Requirement already satisfied: wrapt>=1.0 in /usr/local/lib/python3.12/dist-packages (from formulaic>=0.2.2->lifelines) (1.17.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (4.59.2)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.0->lifelines) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.1->lifelines) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.1->lifelines) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib>=3.0->lifelines) (1.17.0)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
from lifelines import KaplanMeierFitter
```

```
# Read the locally generated CSV file
df = pd.read_csv("dados_GAP_Ano_Nivel.csv")
```

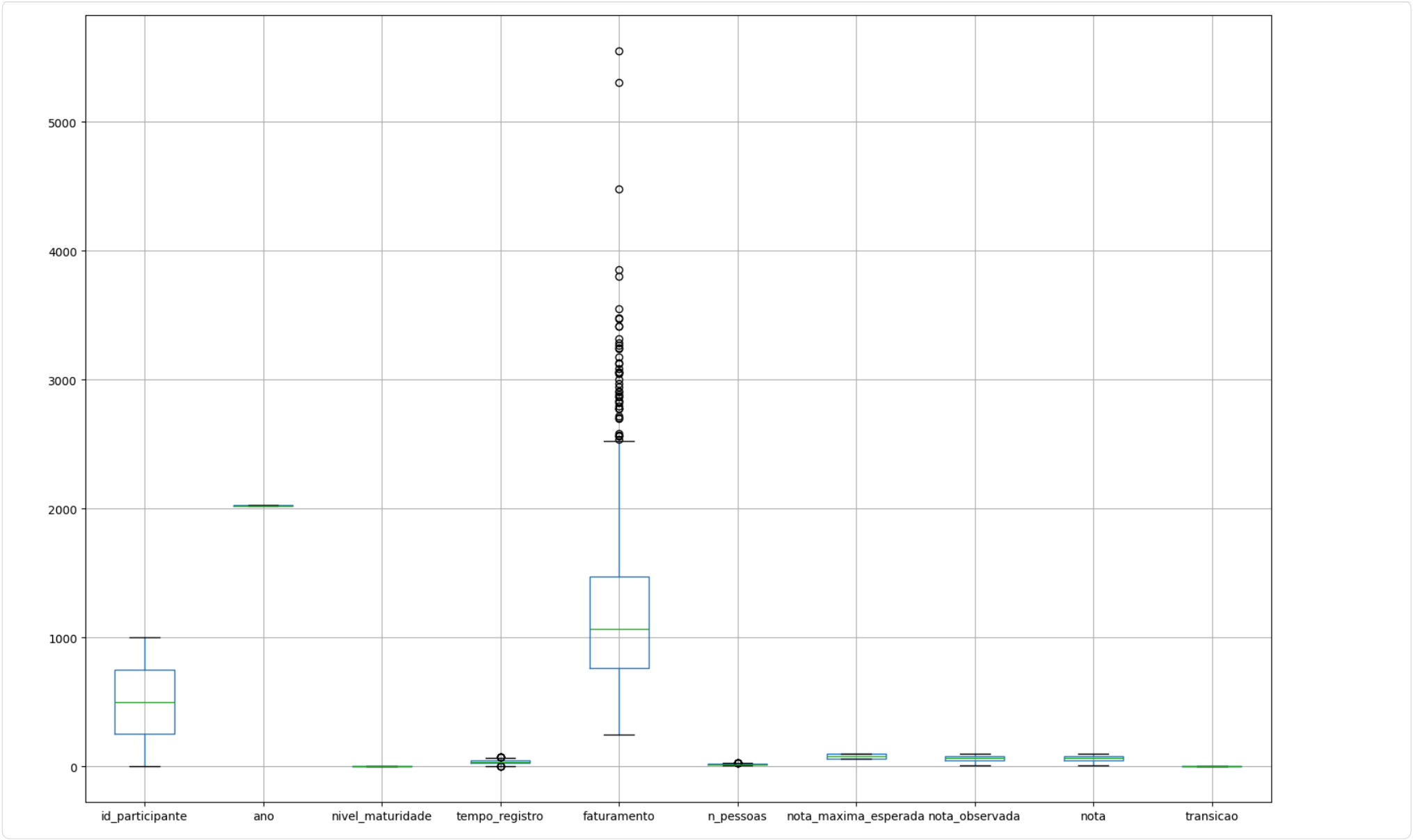
Exploração dos Dados

```
# Visualizar as primeiras linhas
df.head()

# Informações gerais
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id_participante       1000 non-null   int64
 1   ano                   1000 non-null   int64
 2   nivel_maturidade      1000 non-null   int64
 3   tempo_registro        1000 non-null   int64
 4   porte                 1000 non-null   object
 5   faturamento           1000 non-null   float64
 6   n_pessoas             1000 non-null   int64
 7   nota_maxima_esperada  1000 non-null   int64
 8   nota_observada        1000 non-null   float64
 9   nota                  1000 non-null   float64
 10  transicao             1000 non-null   int64
dtypes: float64(3), int64(7), object(1)
memory usage: 86.1+ KB
```

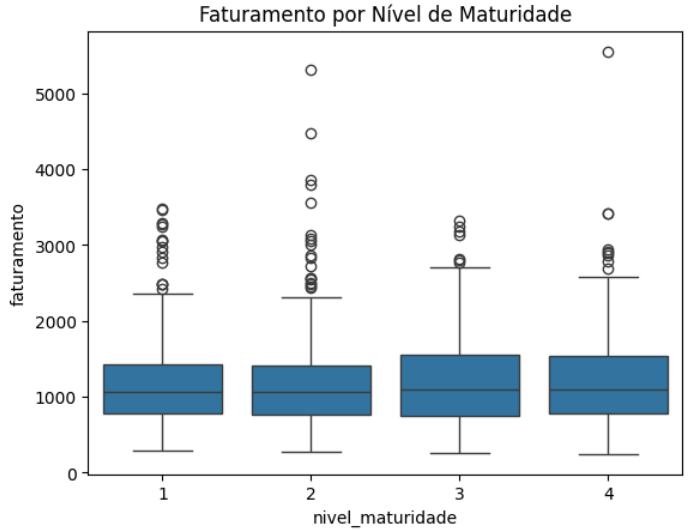
	id_participante	ano	nivel_maturidade	tempo_registro	faturamento	n_pessoas	nota_maxima_esperada	nota_observada	nota	transicao
count	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	500.500000	2022.969000	2.324000	35.65500	1219.517820	15.988000	82.050000	62.998500	62.998500	0.435000
std	288.819436	0.778875	1.019835	11.75228	657.111406	3.905523	14.918782	20.333511	20.333511	0.496005
min	1.000000	2022.000000	1.000000	0.00000	243.760000	6.000000	60.000000	7.500000	7.500000	0.000000
25%	250.750000	2022.000000	1.000000	28.00000	763.180000	13.000000	60.000000	48.700000	48.700000	0.000000
50%	500.500000	2023.000000	2.000000	36.00000	1065.970000	16.000000	80.000000	65.250000	65.250000	0.000000
75%	750.250000	2024.000000	3.000000	44.00000	1469.950000	18.000000	95.000000	78.300000	78.300000	1.000000
max	1000.000000	2024.000000	4.000000	74.00000	5549.960000	29.000000	100.000000	100.000000	100.000000	1.000000

```
# boxplot = df.boxplot(column=['population','housing_median_age','total_rooms','total_bedrooms','population','households','median_income'],figsize=(15, 10))
boxplot = df.boxplot(figsize=(15, 10))
plt.tight_layout()
plt.show()
```



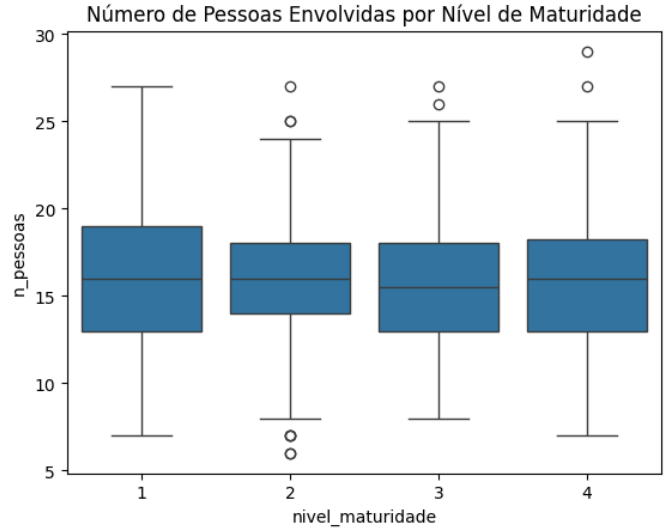
Boxplot faturamento vs. nível e outros

```
sns.boxplot(x='nivel_maturidade', y='faturamento', data=df)
plt.title('Faturamento por Nível de Maturidade')
plt.show()
```



Comece a programar ou gere código com IA.

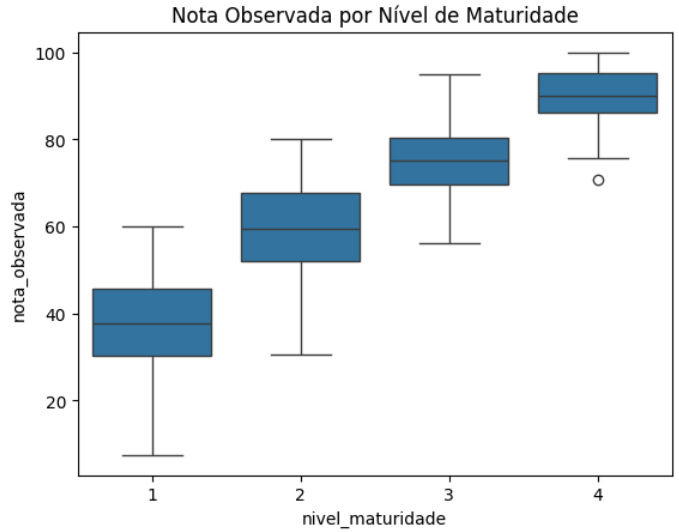
```
sns.boxplot(x='nivel_maturidade', y='n_pessoas', data=df)
plt.title('Número de Pessoas Envolvidas por Nível de Maturidade')
plt.show()
```



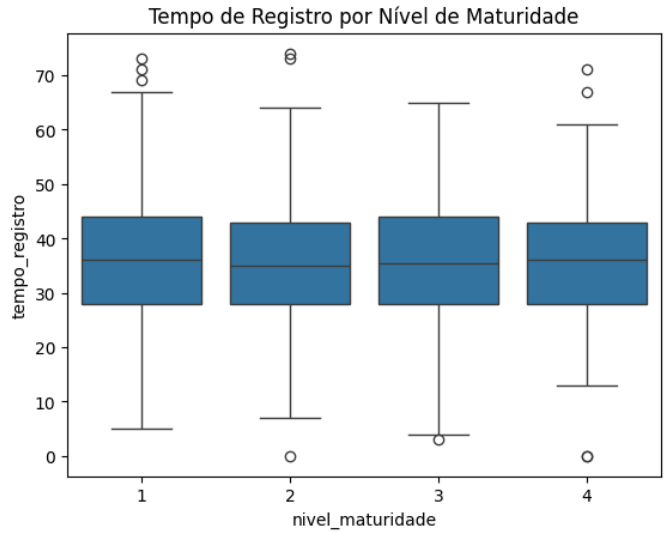
Comece a programar ou gere código com IA.

```
sns.boxplot(x='nivel_maturidade', y='nota_observada', data=df)
plt.title('Nota Observada por Nível de Maturidade')
```

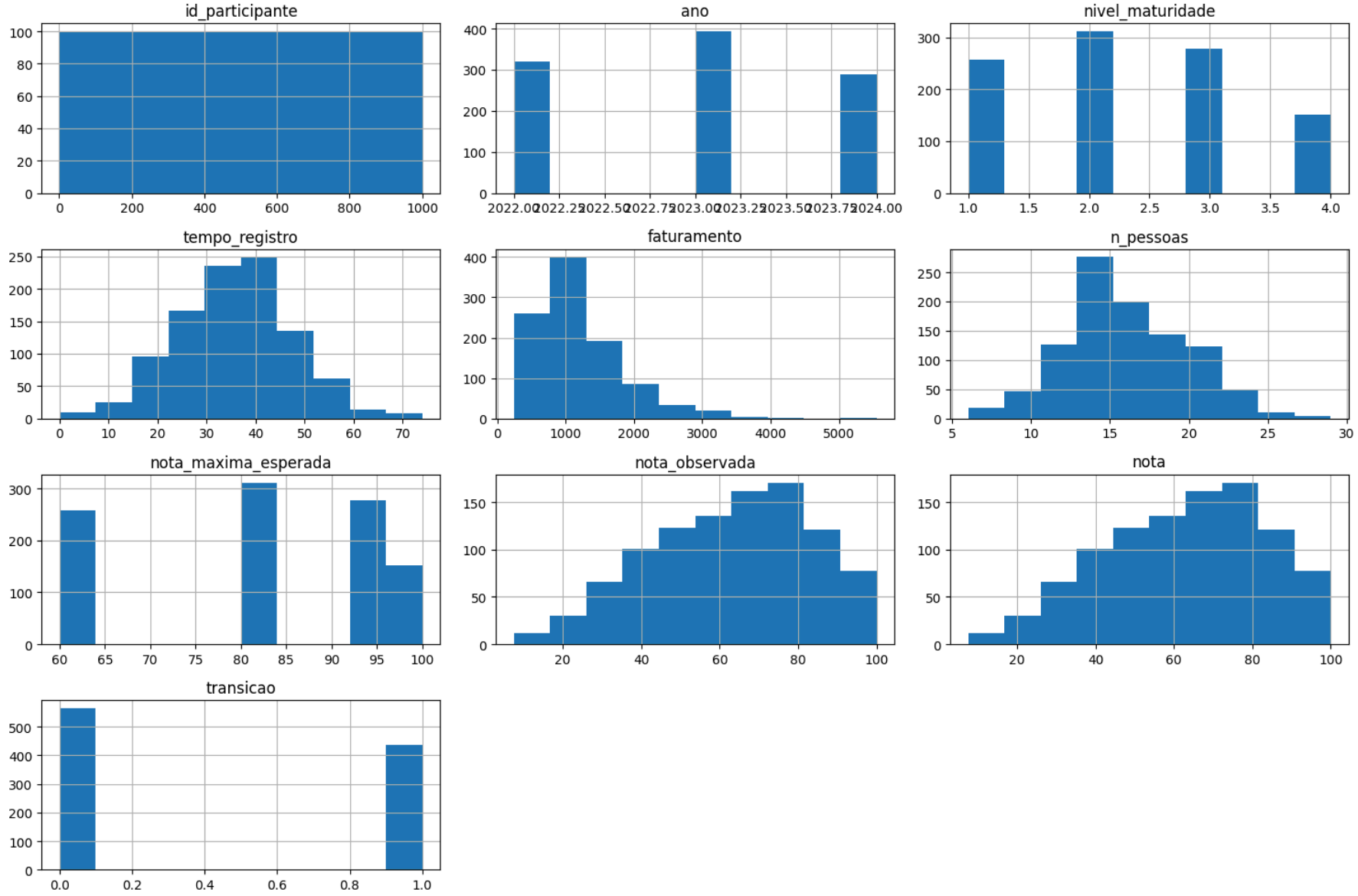
```
plt.show()
```




```
sns.boxplot(x='nivel_maturidade', y='tempo_registro', data=df)
plt.title('Tempo de Registro por Nível de Maturidade')
plt.show()
```

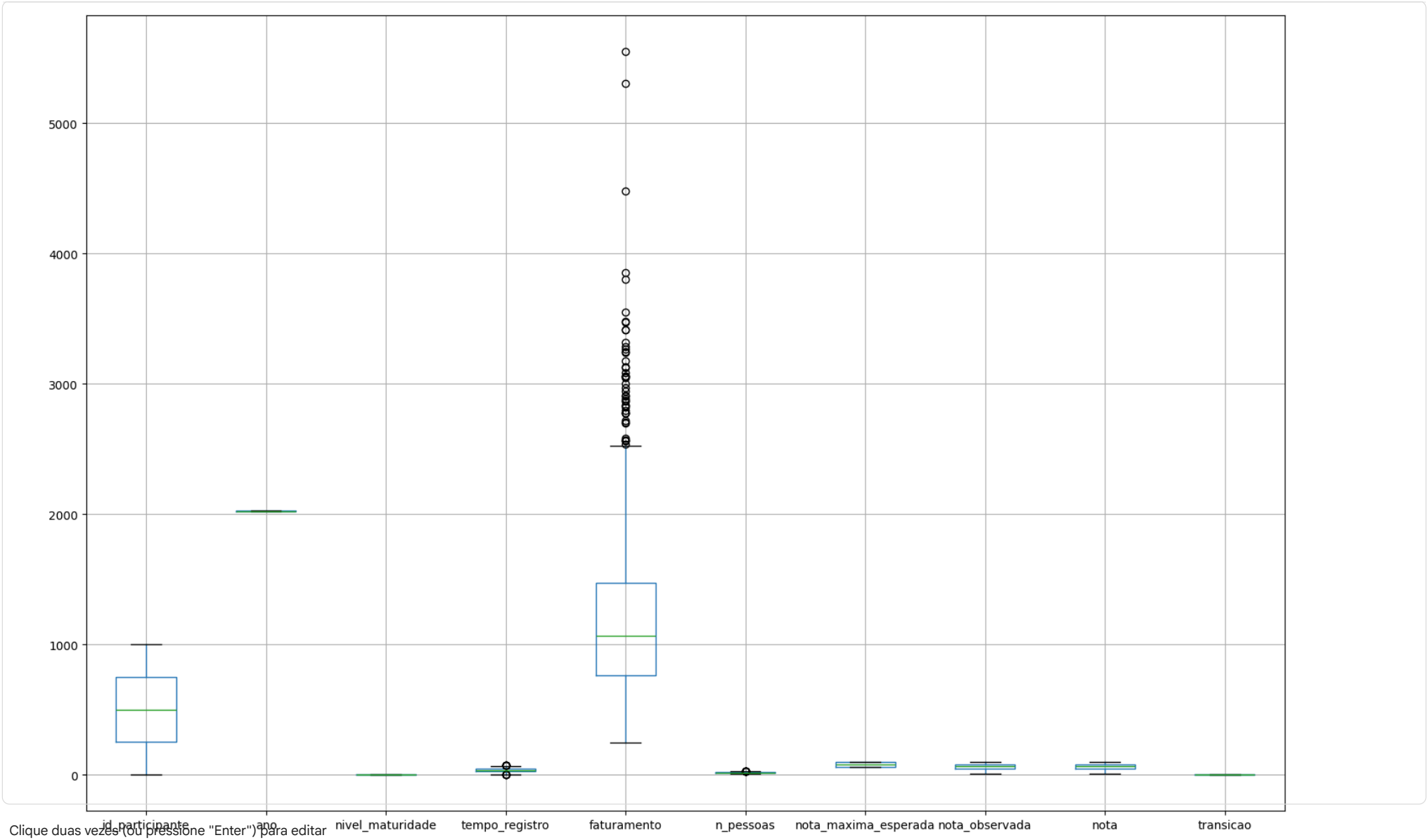



```
df.hist(figsize=(15, 10))
plt.tight_layout()
plt.show()
```



Clique duas vezes (ou pressione "Enter") para editar

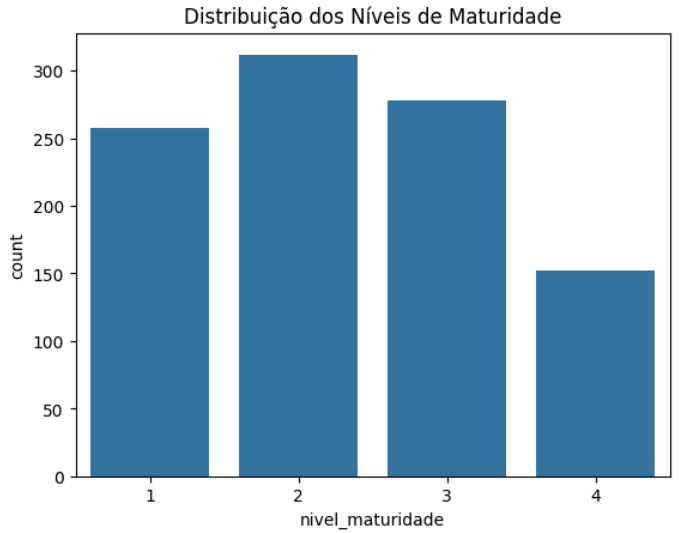
```
boxplot  = df.boxplot(figsize=(15, 10))
plt.tight_layout()
plt.show()
```

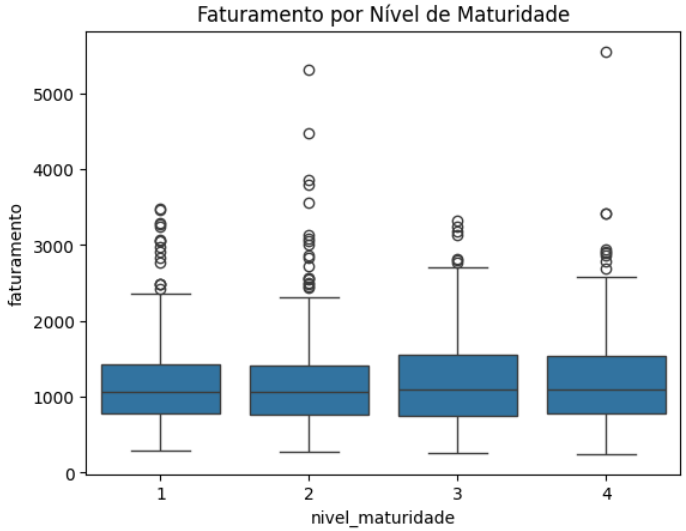


Clique duas vezes (ou pressione "Enter") para editar

```
# Distribuição dos níveis de maturidade
sns.countplot(x='nivel_maturidade', data=df)
plt.title('Distribuição dos Níveis de Maturidade')
plt.show()

# Relação entre desempenho e outras variáveis, exemplo: faturamento
sns.boxplot(x='nivel_maturidade', y='faturamento', data=df)
plt.title('Faturamento por Nível de Maturidade')
```

```
plt.show()
```



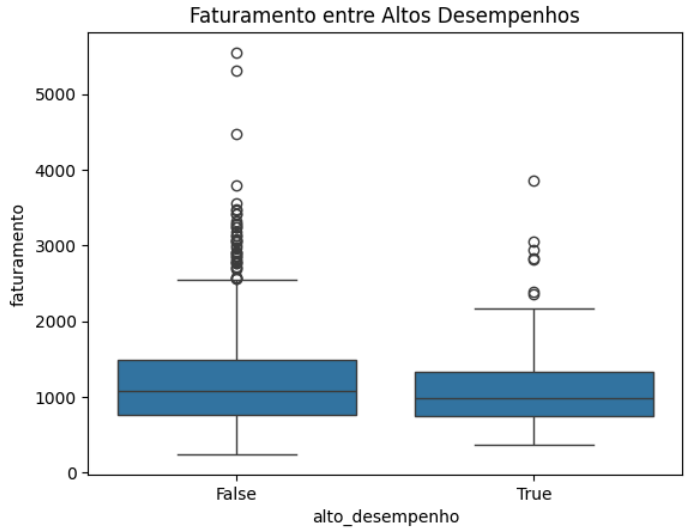


Definição do que é alto desempenho.

Identificação de Padrões de Alto Desempenho "alto desempenho" (ex: top 10% das notas em cada nível).

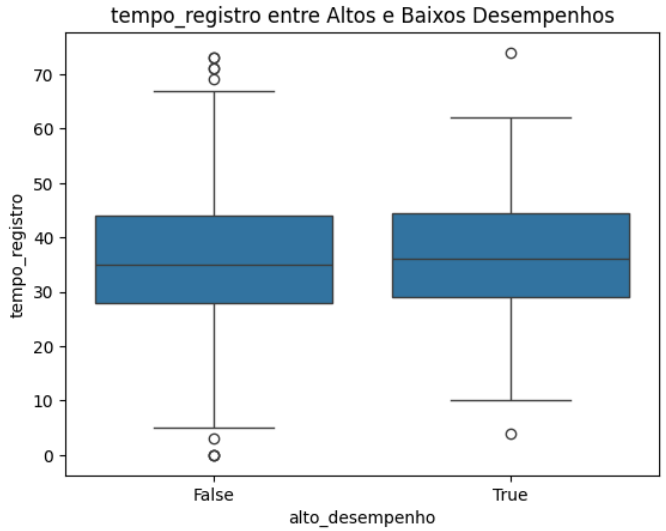
```
# Exemplo: identificar top 10% em cada nível
df['alto_desempenho'] = df.groupby('nivel_maturidade')['nota'].transform(
    lambda x: x >= x.quantile(0.9))

# Analisar variáveis por alto desempenho
sns.boxplot(x='alto_desempenho', y='faturamento', data=df)
```

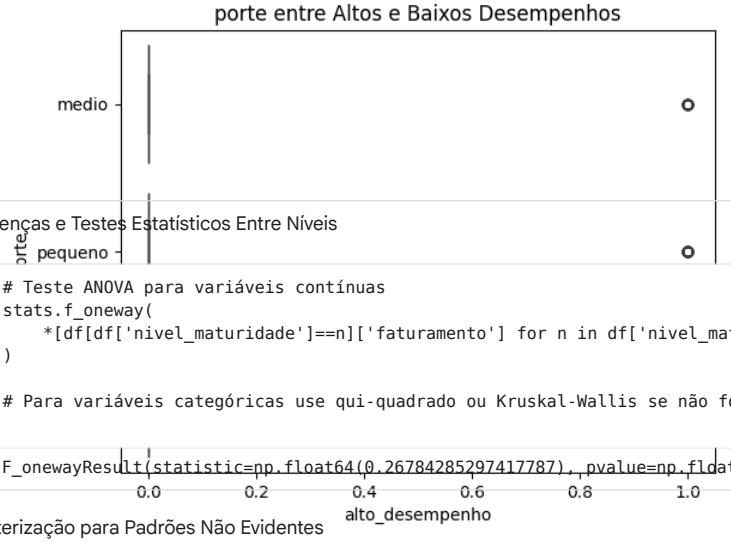
```
plt.title('Faturamento entre Altos Desempenhos')
plt.show()

# Outras variáveis: porte, tempo_registro, pessoas, etc.
```



Para múltiplas comparações (exemplo: tempo de registro, número de pessoas):

```
variaveis = ['tempo_registro', 'porte', 'faturamento', 'n_pessoas']
for var in variaveis:
    sns.boxplot(x='alto_desempenho', y=var, data=df)
    plt.title(f'{var} entre Altos e Baixos Desempenhos')
    plt.show()
```





Diferenças e Testes Estatísticos Entre Níveis

```
# Teste ANOVA para variáveis contínuas
stats.f_oneway(
    *[df[df['nivel_maturidade']==n]['faturamento'] for n in df['nivel_maturidade'].unique()]
)

# Para variáveis categóricas use qui-quadrado ou Kruskal-Wallis se não for paramétrico
```

```
F_onewayResult(statistic=np.float64(0.26784285297417787), pvalue=np.float64(0.8486039433763914))
```

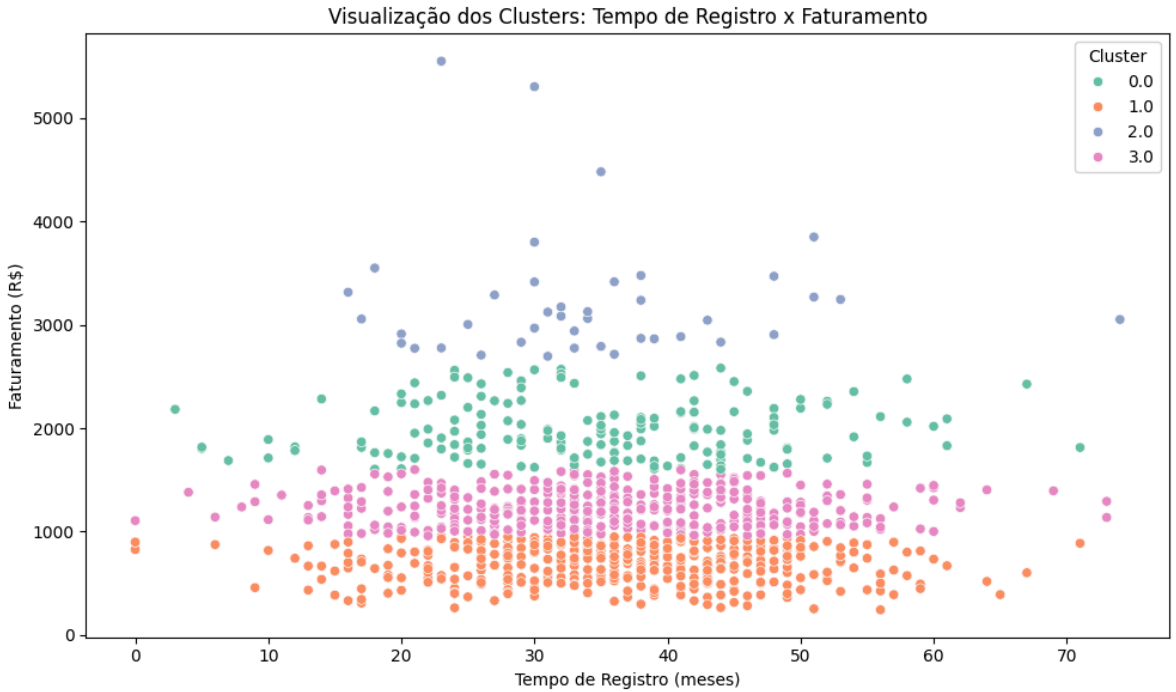
Clusterização para Padrões Não Evidentes


```
# Selecionar apenas variáveis numéricas e remover nulos
#X = df[variaveis].dropna()
#kmeans = KMeans(n_clusters=4)
#clusters = kmeans.fit_predict(X)
#df.loc[X.index, 'cluster'] = clusters

#sns.pairplot(df, hue='cluster', vars=variaveis)
#plt.show()

#=========================================================
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
```

```
# Exemplo: supondo que 'porte' é uma coluna do seu DataFrame
le = LabelEncoder()
df['porte_encoded'] = le.fit_transform(df['porte'])

# Agora use apenas variáveis numéricas na clusterização
variaveis_corrigidas = ['tempo_registro', 'porte_encoded', 'faturamento', 'n_pessoas']
X_corrigido = df[variaveis_corrigidas].dropna()

kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(X_corrigido)

# Adicione os clusters ao dataframe
df.loc[X_corrigido.index, 'cluster'] = clusters
```

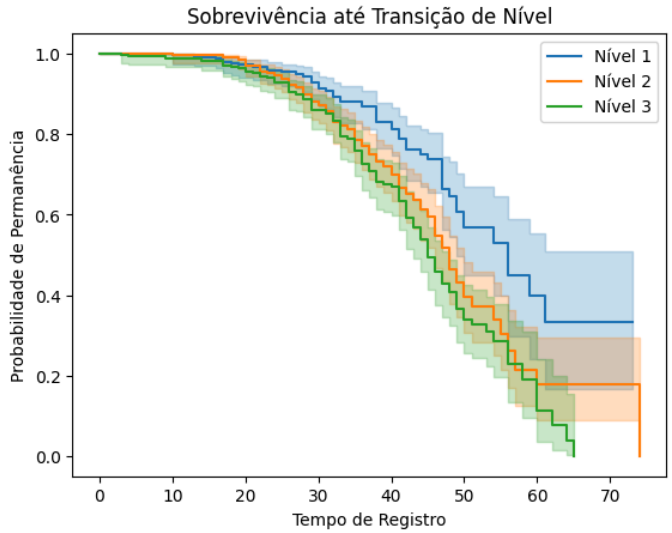
Apresentação visual dos clusters

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))

# Apenas para os dados usados na clusterização, garantindo integridade dos índices
dados_cluster = df.loc[X_corrigido.index].copy()

sns.scatterplot(
    x='tempo_registro',
    y='faturamento',
    hue='cluster',
    palette='Set2',
    data=dados_cluster,
    legend='full'
)

plt.title('Visualização dos Clusters: Tempo de Registro x Faturamento')
plt.xlabel('Tempo de Registro (meses)')
plt.ylabel('Faturamento (R$)')
plt.legend(title='Cluster')
plt.tight_layout()
plt.show()
```



Previsão de Momento Ideal para Transição de Nível Modelos de sobrevivência, como Kaplan-Meier, para ver em que momento (por tempo de registro ou outro critério) ocorrem as transições de nível.

```
# Crie uma variável de 'sobrevivência': tempo até transição.
kmf = KaplanMeierFitter()

for nivel in range(1, 4):  # niveis 1 a 3
    grupo = df[df['nivel_maturidade'] == nivel]
    T = grupo['tempo_registro']
    E = grupo['transicao']  # 1 se houve transição, 0 caso contrário
    kmf.fit(T, event_observed=E, label=f'Nível {nivel}')
    kmf.plot()

plt.title('Sobrevivência até Transição de Nível')
plt.xlabel('Tempo de Registro')
plt.ylabel('Probabilidade de Permanência')
plt.show()
```



Observação: a lógica depende de haver uma variável booleana indicando se houve transição.

Análise de GAP

```
# Calcular o gap
df['gap'] = df['nota_maxima_esperada'] - df['nota_observada']

# Analisar por ano e nível
gap_agg = df.groupby(['ano', 'nivel_maturidade'])['gap'].describe()
print(gap_agg)

sns.boxplot(x='ano', y='gap', hue='nivel_maturidade', data=df)
plt.title('Gap por Ano e Nível')
plt.show()
```