LH_CD_ISAAC_JEFFERSON_SOUSA_SABOIA

April 7, 2025

1 Desafio Cientista de Dados

Você está trabalhando atualmente junto a um cliente no processo de criação de uma plataforma de aluguéis temporários na cidade de Nova York. Para o desenvolvimento de sua estratégia de precificação, pediu para que você fizesse uma análise exploratória dos dados de seu maior concorrente, assim como um teste de validação de um modelo preditivo.

O objetivo é desenvolver um modelo de previsão de preços a partir do dataset oferecido, e avaliar tal modelo utilizando as métricas de avaliação que mais fazem sentido para o problema.

Vamos analisar os dados referentes à cidade de Nova York e responder às possíveis dúvidas que possam surgir.

2 Passo 1: Importação, análise primitiva e limpeza dos dados

2.1 Importação dos dados

Importação das bibliotecas que serão necessárias, em primeira mão, para o carregamento do dataset, para a análise e preparação dos dados para a próxima etapa. Esta etapa é importante pois essas bibliotecas oferecem funções prontas e otimizadas para diversas tarefas, permitindo a facilidade e agilidade na análise.

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import random
import datetime
```

Carregamento do *dataset* e transformação em *dataframe* para facilitar a manipulação, análise e visualização dos dados.

[2]:
```
# Carrega o dataset e o transforma em um dataframe.
df = pd.read_csv('teste_indicium_precificacao.csv')
df.head()
```

[2]:

	id	nome	host_id	\
0	2595	Skylit Midtown Castle	2845	
1	3647	THE VILLAGE OF HARLEM…NEW YORK !	4632	
2	3831	Cozy Entire Floor of Brownstone	4869	

```
3  5022  Entire Apt: Spacious Studio/Loft by central park     7192
4  5099         Large Cozy 1 BR Apartment In Midtown East     7322

     host_name bairro_group        bairro  latitude  longitude  \
0     Jennifer    Manhattan       Midtown  40.75362  -73.98377
1    Elisabeth    Manhattan        Harlem  40.80902  -73.94190
2  LisaRoxanne     Brooklyn  Clinton Hill  40.68514  -73.95976
3        Laura    Manhattan   East Harlem  40.79851  -73.94399
4        Chris    Manhattan   Murray Hill  40.74767  -73.97500

         room_type  price  minimo_noites  numero_de_reviews ultima_review  \
0  Entire home/apt    225              1                 45    2019-05-21
1     Private room    150              3                  0           NaN
2  Entire home/apt     89              1                270    2019-07-05
3  Entire home/apt     80             10                  9    2018-11-19
4  Entire home/apt    200              3                 74    2019-06-22

   reviews_por_mes  calculado_host_listings_count  disponibilidade_365
0             0.38                              2                  355
1              NaN                              1                  365
2             4.64                              1                  194
3             0.10                              1                    0
4             0.59                              1                  129
```

2.2 Análise primitiva dos dados

O objetivo desta etapa é proporcionar uma visão geral inicial e compreender a estrutura dos dados.

Dicionário dos dados

`A base de dados contém 16 colunas.`

- `id` - Atua como uma chave exclusiva para cada anúncio nos dados do aplicativo
- `nome` - Representa o nome do anúncio
- `host_id` - Representa o id do usuário que hospedou o anúncio
- `host_name` – Contém o nome do usuário que hospedou o anúncio
- `bairro_group` - Contém o nome do bairro onde o anúncio está localizado
- `bairro` - Contém o nome da área onde o anúncio está localizado
- `latitude` - Contém a latitude do local
- `longitude` - Contém a longitude do local
- `room_type` – Contém o tipo de espaço de cada anúncio
- `price` - Contém o preço por noite em dólares listado pelo anfitrião
- `minimo_noites` - Contém o número mínimo de noites que o usuário deve reservar
- `numero_de_reviews` - Contém o número de comentários dados a cada listagem
- `ultima_review` - Contém a data da última revisão dada à listagem
- `reviews_por_mes` - Contém o número de avaliações fornecidas por mês
- `calculado_host_listings_count` - Contém a quantidade de listagem por host
- `disponibilidade_365` - Contém o número de dias em que o anúncio está disponível para reserva

[3]:
```
# Imprime a quantidade de dados nulos existentes em cada coluna do dataframe.
print(df.isnull().sum())
```

```
id                                  0
nome                               16
host_id                             0
host_name                          21
bairro_group                        0
bairro                              0
latitude                            0
longitude                           0
room_type                           0
price                               0
minimo_noites                       0
numero_de_reviews                   0
ultima_review                   10052
reviews_por_mes                 10052
calculado_host_listings_count       0
disponibilidade_365                 0
dtype: int64
```

[4]:
```
print(((df.isnull().sum() / df.shape[0] * 100).round(2)).
  ↪sort_values(ascending=False).astype(str) + "%")
```

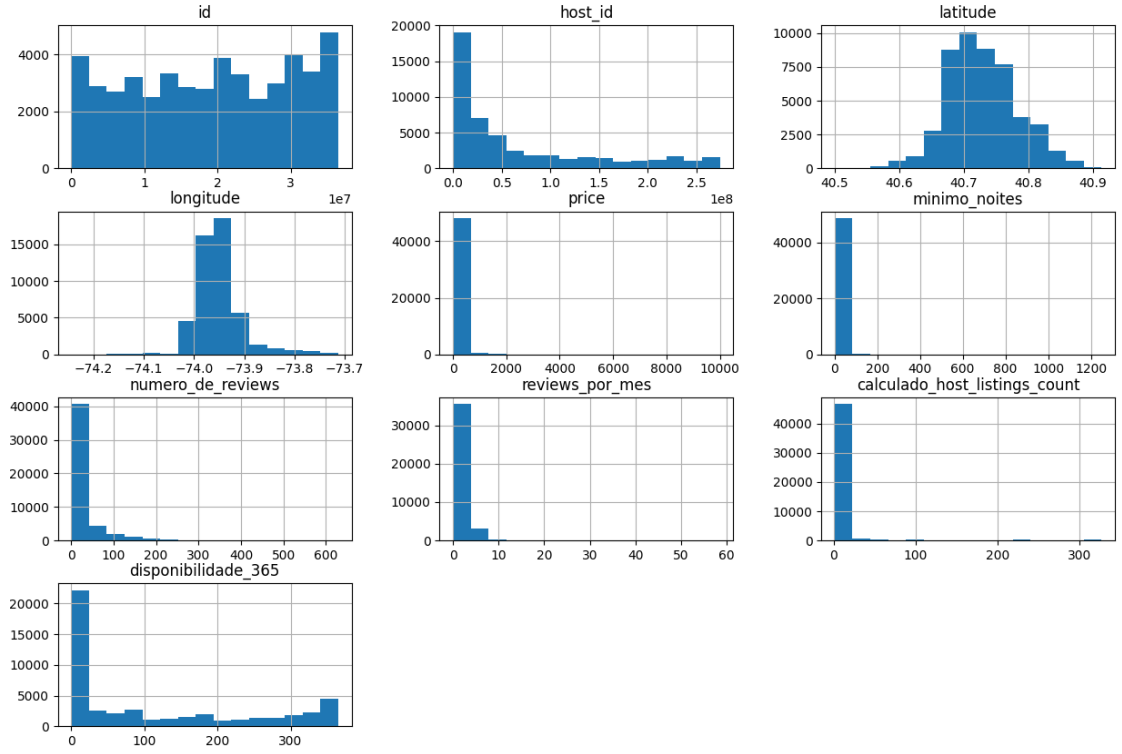
```
ultima_review                   20.56%
reviews_por_mes                 20.56%
host_name                        0.04%
nome                             0.03%
bairro_group                      0.0%
bairro                            0.0%
id                                0.0%
host_id                           0.0%
longitude                         0.0%
latitude                          0.0%
room_type                         0.0%
price                             0.0%
numero_de_reviews                 0.0%
minimo_noites                     0.0%
calculado_host_listings_count     0.0%
disponibilidade_365               0.0%
dtype: object
```

[5]:
```
# Imprime os tipos de dados de cada coluna para confirmar se estão no formato␣
  ↪adequado.
print(df.dtypes)
```

```
id                                int64
nome                             object
host_id                           int64
```

```
host_name                          object
bairro_group                       object
bairro                             object
latitude                          float64
longitude                         float64
room_type                          object
price                               int64
minimo_noites                       int64
numero_de_reviews                   int64
ultima_review                      object
reviews_por_mes                   float64
calculado_host_listings_count       int64
disponibilidade_365                 int64
dtype: object
```

[6]:
```
df.hist(bins=15, figsize=(15,10));
```



[7]:
```
df[['price', 'minimo_noites', 'numero_de_reviews', 'reviews_por_mes',␣
  ↪'calculado_host_listings_count', 'disponibilidade_365']].describe()
```

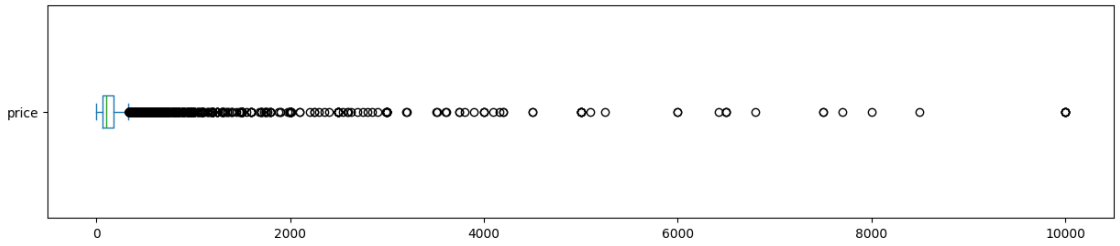
[7]:

	price	minimo_noites	numero_de_reviews	reviews_por_mes	\
count	48894.000000	48894.000000	48894.000000	38842.000000	
mean	152.720763	7.030085	23.274758	1.373251	

```
std      240.156625      20.510741          44.550991         1.680453
min        0.000000       1.000000           0.000000         0.010000
25%       69.000000       1.000000           1.000000         0.190000
50%      106.000000       3.000000           5.000000         0.720000
75%      175.000000       5.000000          24.000000         2.020000
max    10000.000000    1250.000000         629.000000        58.500000

       calculado_host_listings_count  disponibilidade_365
count                   48894.000000         48894.000000
mean                        7.144005           112.776169
std                        32.952855           131.618692
min                         1.000000             0.000000
25%                         1.000000             0.000000
50%                         1.000000            45.000000
75%                         2.000000           227.000000
max                       327.000000           365.000000
```

Com base nas informações acima, podemos inferir que, nas colunas **price** e **minimo_noites**, há *outliers*. - 75% dos preços (`price`) estão abaixo de 175, mas há registros com valores tão altos quanto 10.000. - O valor mínimo de noites (`minimum_nights`) ultrapassa o limite real de 365 dias.

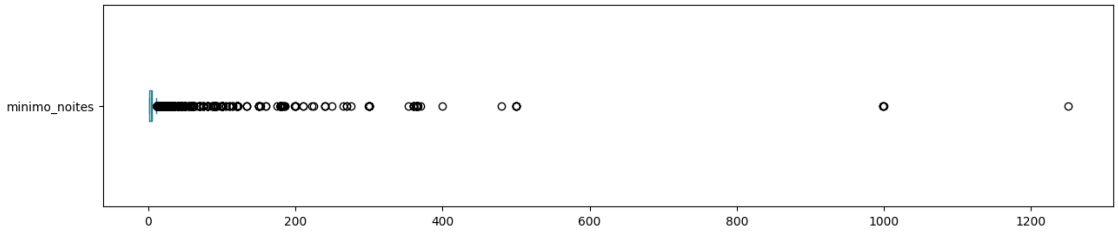
[8]:
```
df.price.plot(kind='box', vert=False, figsize=(15, 3),)
plt.show()

# ver quantidade de valores acima de 240 para price
print("\nprice: valores acima de 240")
print("{} entradas".format(len(df[df.price > 240])))
print("{:.4f}%".format((len(df[df.price > 240]) / df.shape[0])*100))
```




```
price: valores acima de 240
6523 entradas
13.3411%
```

[9]:
```
df.minimo_noites.plot(kind='box', vert=False, figsize=(15, 3),)
plt.show()

# ver quantidade de valores acima de 30 para price
```

```
print("\nminimo_noites: valores acima de 30")
print("{} entradas".format(len(df[df.minimo_noites > 30])))
print("{:.4f}%".format((len(df[df.minimo_noites > 30]) / df.shape[0])*100))
```

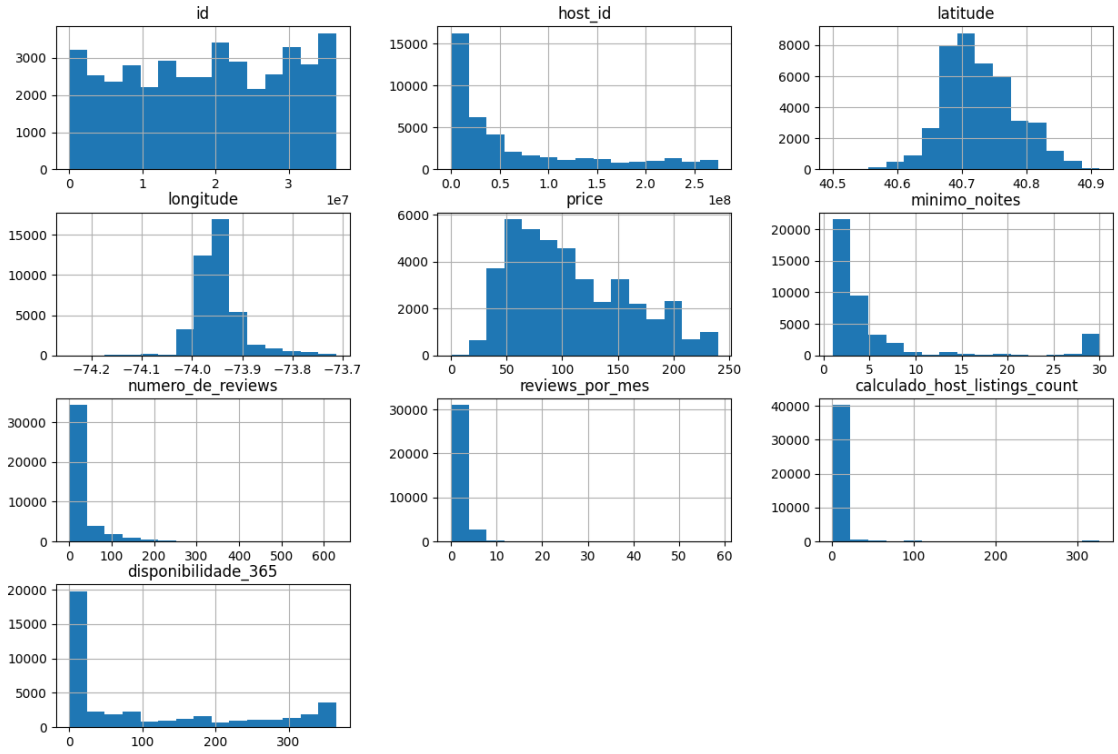



```
minimo_noites: valores acima de 30
747 entradas
1.5278%
```

2.3 Limpeza dos dados

A limpeza dos dados melhora a qualidade da análise, evitando distorções causadas por outliers e informações ausentes. Isso garante maior precisão nos insights e modelos preditivos.

[10]:
```
# remover os *outliers* em um novo DataFrame
df_clean = df.copy()
df_clean.drop(df_clean[df_clean.price > 240].index, axis=0, inplace=True)
df_clean.drop(df_clean[df_clean.minimo_noites > 30].index, axis=0, inplace=True)

# plotar o histograma para as variáveis numéricas
df_clean.hist(bins=15, figsize=(15,10));
```



[11]:
```
print(df_clean.isnull().sum())
```

```
id                                0
nome                             14
host_id                           0
host_name                        20
bairro_group                      0
bairro                            0
latitude                          0
longitude                         0
room_type                         0
price                             0
minimo_noites                     0
numero_de_reviews                 0
ultima_review                  7779
reviews_por_mes                7779
calculado_host_listings_count     0
disponibilidade_365               0
dtype: int64
```

[12]:
```
# Transforma os dados da coluna "ultima_review" para o formato de datetime.
df_clean['ultima_review'] = pd.to_datetime(df_clean['ultima_review'])
```

```
# Busca e imprime as datas de menor e maior valor, criando um parâmetro.
min_date_review = df_clean['ultima_review'].min()
max_date_review = df_clean['ultima_review'].max()

dt = 0
datas = []
while (dt < 7779):
    random_date = min_date_review + (max_date_review - min_date_review) *␣
 ↪random.random()
    datas.append(random_date)
    dt += 1

df_clean['ultima_review'] = df_clean['ultima_review'].fillna(pd.Series(datas,␣
 ↪index=df_clean[df_clean['ultima_review'].isnull()].index))

# Preenche os dados nulos da coluna "reviews_por_mes" com a mediana dos dados␣
 ↪existentes na coluna.
df_clean['reviews_por_mes'] = df_clean['reviews_por_mes'].transform(lambda x: x.
 ↪fillna(x.median()))
```

[13]:
```
print(df_clean.isnull().sum())
```

```
id                                0
nome                             14
host_id                           0
host_name                        20
bairro_group                      0
bairro                            0
latitude                          0
longitude                         0
room_type                         0
price                             0
minimo_noites                     0
numero_de_reviews                 0
ultima_review                     0
reviews_por_mes                   0
calculado_host_listings_count     0
disponibilidade_365               0
dtype: int64
```

[14]:
```
# Transforma os dados da coluna "ultima_review" para o formato de datetime.
df['ultima_review'] = pd.to_datetime(df['ultima_review'])

# Busca e imprime as datas de menor e maior valor, criando um parâmetro.
min_date_review = df['ultima_review'].min()
max_date_review = df['ultima_review'].max()
```

```
dt = 0
datas = []
while (dt < 10052):
    random_date = min_date_review + (max_date_review - min_date_review) *␣
 ↪random.random()
    datas.append(random_date)
    dt += 1

df['ultima_review'] = df['ultima_review'].fillna(pd.Series(datas,␣
 ↪index=df[df['ultima_review'].isnull()].index))

# Preenche os dados nulos da coluna "reviews_por_mes" com a mediana dos dados␣
 ↪existentes na coluna.
df['reviews_por_mes'] = df['reviews_por_mes'].transform(lambda x: x.fillna(x.
 ↪median()))
```

[15]:
```
from scipy.spatial import cKDTree

# Substitui o "host_name" nulo pelo "host_name" com as coordenadas mais␣
 ↪próximas.
df_treino = df_clean[df_clean['host_name'].notnull()]
tree = cKDTree(df_treino[['latitude', 'longitude']])

for index, row in df_clean[df_clean['host_name'].isnull()].iterrows():
    _, idx = tree.query([row['latitude'], row['longitude']])
    df_clean.at[index, 'host_name'] = df_treino.iloc[idx]['host_name']

# Substitui o "nome" nulo pelo "nome" com as coordenadas mais próximas.
    df_treino = df_clean[df_clean['nome'].notnull()]
tree = cKDTree(df_treino[['latitude', 'longitude']])

for index, row in df_clean[df_clean['nome'].isnull()].iterrows():
    _, idx = tree.query([row['latitude'], row['longitude']])
    df_clean.at[index, 'nome'] = df_treino.iloc[idx]['nome']
```

[16]:
```
from scipy.spatial import cKDTree

# Substitui o "host_name" nulo pelo "host_name" com as coordenadas mais␣
 ↪próximas.
df_treino = df[df['host_name'].notnull()]
tree = cKDTree(df_treino[['latitude', 'longitude']])

for index, row in df[df['host_name'].isnull()].iterrows():
    _, idx = tree.query([row['latitude'], row['longitude']])
    df.at[index, 'host_name'] = df_treino.iloc[idx]['host_name']

# Substitui o "nome" nulo pelo "nome" com as coordenadas mais próximas.
```

```
    df_treino = df[df['nome'].notnull()]
tree = cKDTree(df_treino[['latitude', 'longitude']])

for index, row in df[df['nome'].isnull()].iterrows():
    _, idx = tree.query([row['latitude'], row['longitude']])
    df.at[index, 'nome'] = df_treino.iloc[idx]['nome']
```

[17]:
```
df.head(5)
```

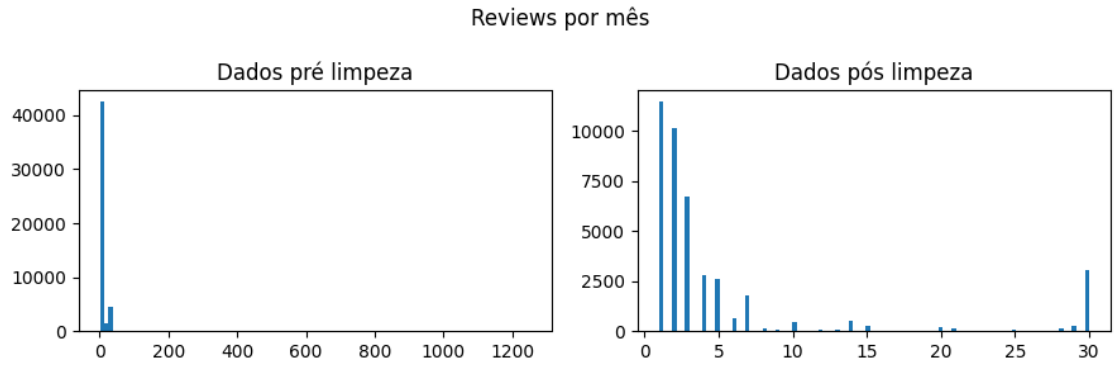
[17]:
```
     id                                              nome  host_id  \
0  2595                            Skylit Midtown Castle     2845
1  3647              THE VILLAGE OF HARLEM…NEW YORK !     4632
2  3831                  Cozy Entire Floor of Brownstone     4869
3  5022  Entire Apt: Spacious Studio/Loft by central park     7192
4  5099         Large Cozy 1 BR Apartment In Midtown East     7322

     host_name bairro_group        bairro  latitude  longitude  \
0     Jennifer    Manhattan       Midtown  40.75362  -73.98377
1    Elisabeth    Manhattan        Harlem  40.80902  -73.94190
2  LisaRoxanne     Brooklyn  Clinton Hill  40.68514  -73.95976
3        Laura    Manhattan   East Harlem  40.79851  -73.94399
4        Chris    Manhattan   Murray Hill  40.74767  -73.97500

         room_type  price  minimo_noites  numero_de_reviews  \
0  Entire home/apt    225              1                 45
1     Private room    150              3                  0
2  Entire home/apt     89              1                270
3  Entire home/apt     80             10                  9
4  Entire home/apt    200              3                 74

                 ultima_review  reviews_por_mes  \
0 2019-05-21 00:00:00.000000000             0.38
1 2016-08-17 12:12:58.674637760             0.72
2 2019-07-05 00:00:00.000000000             4.64
3 2018-11-19 00:00:00.000000000             0.10
4 2019-06-22 00:00:00.000000000             0.59

   calculado_host_listings_count  disponibilidade_365
0                              2                  355
1                              1                  365
2                              1                  194
3                              1                    0
4                              1                  129
```

[18]:
```
print(df_clean.isnull().sum())
```

```
id                                0
nome                              0
```

```
host_id                          0
host_name                        0
bairro_group                     0
bairro                           0
latitude                         0
longitude                        0
room_type                        0
price                            0
minimo_noites                    0
numero_de_reviews                0
ultima_review                    0
reviews_por_mes                  0
calculado_host_listings_count    0
disponibilidade_365              0
dtype: int64
```

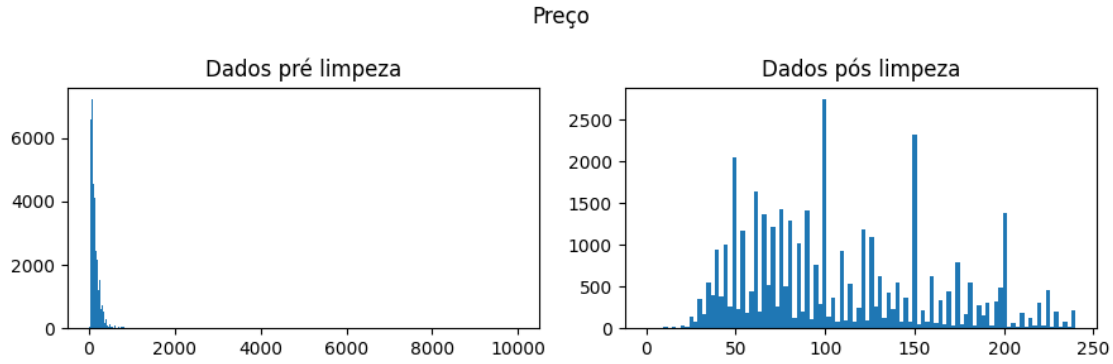
[19]:
```python
# Criação dos subplots
plt.figure(figsize=(9, 3))

# Primeiro histograma
plt.subplot(1, 2, 1)
plt.hist(df['minimo_noites'], bins=100)
plt.title('Dados pré limpeza')

# Segundo histograma
plt.subplot(1, 2, 2)
plt.hist(df_clean['minimo_noites'], bins=100)
plt.title('Dados pós limpeza')

#Título geral
plt.suptitle('Reviews por mês')

# Exibir os gráficos
plt.tight_layout()
plt.show()

print("\nDados pré limpeza:")
print("Mínimo de noites:", df['minimo_noites'].min())
print("Mínimo de noites:", df['minimo_noites'].max())

print("\nDados pós limpeza:")
print("Mínimo de noites:", df_clean['minimo_noites'].min())
print("Mínimo de noites:", df_clean['minimo_noites'].max())
```




```
Dados pré limpeza:
Mínimo de noites: 1
Mínimo de noites: 1250

Dados pós limpeza:
Mínimo de noites: 1
Mínimo de noites: 30
```

[20]:
```
# Criação dos subplots
plt.figure(figsize=(9, 3))

# Primeiro histograma
plt.subplot(1, 2, 1)
plt.hist(df['price'], bins=500)
plt.title('Dados pré limpeza')

# Segundo histograma
plt.subplot(1, 2, 2)
plt.hist(df_clean['price'], bins=100)
plt.title('Dados pós limpeza')

# Título geral
plt.suptitle('Preço')

# Exibir os gráficos
plt.tight_layout()
plt.show()

print("\nDados pré limpeza:")
print("Preço mínimo:", df['price'].min())
print("Preço máximo:", df['price'].max())

print("\nDados pós limpeza:")
```

```
print("Preço mínimo:", df_clean['price'].min())
print("Preço máximo:", df_clean['price'].max())
```




```
Dados pré limpeza:
Preço mínimo: 0
Preço máximo: 10000

Dados pós limpeza:
Preço mínimo: 0
Preço máximo: 240
```

---

3 Passo 2: Análise Exploratória dos Dados (EDA)

[21]:
```
df_clean.describe()
```

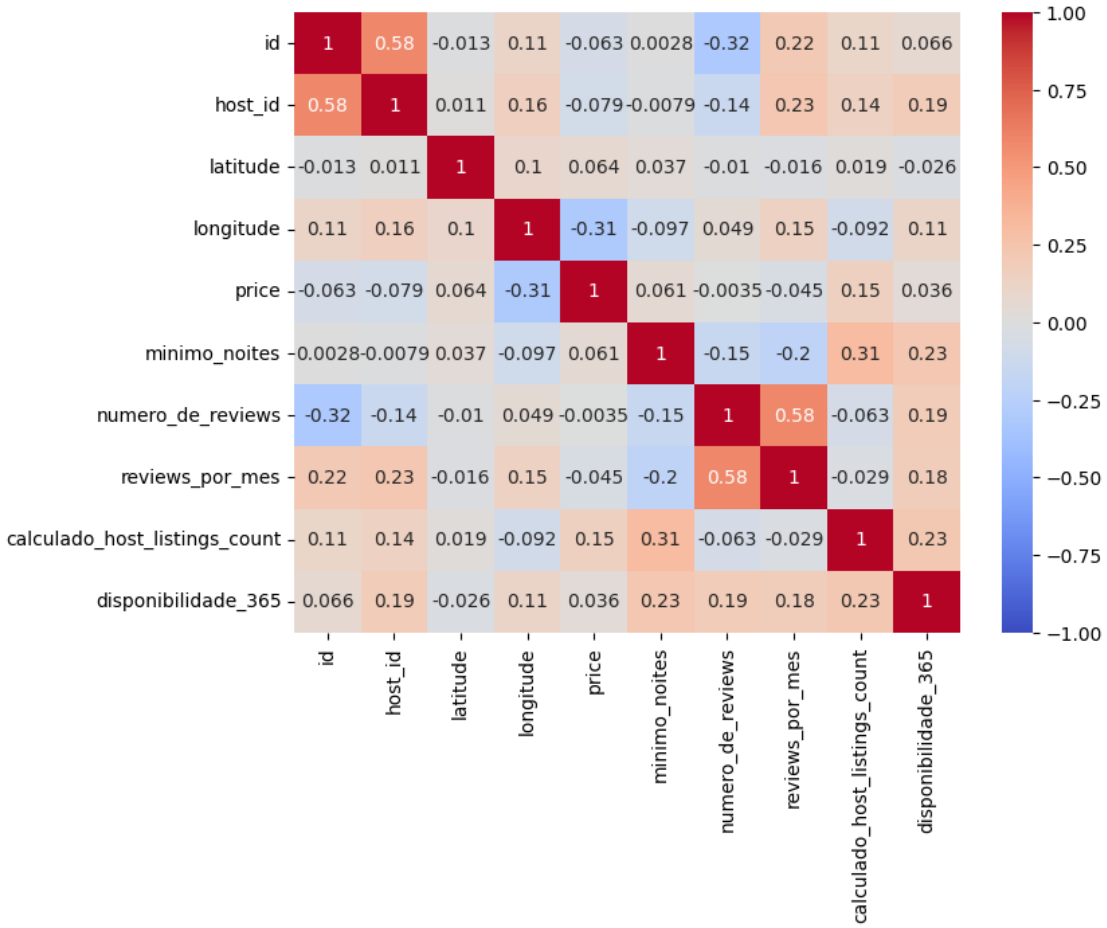
[21]:

	id	host_id	latitude	longitude	price	\
count	4.175100e+04	4.175100e+04	41751.000000	41751.000000	41751.000000	
mean	1.879927e+07	6.589371e+07	40.727867	-73.948771	106.895476	
min	2.595000e+03	2.571000e+03	40.499790	-74.244420	0.000000	
25%	9.447984e+06	7.853548e+06	40.688160	-73.979790	65.000000	
50%	1.938656e+07	3.044353e+07	40.720050	-73.952870	99.000000	
75%	2.858478e+07	1.042684e+08	40.763695	-73.931500	149.000000	
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	240.000000	
std	1.083360e+07	7.686833e+07	0.056342	0.046866	52.719169	

	minimo_noites	numero_de_reviews	ultima_review	\
count	41751.000000	41751.000000	41751	
mean	5.468276	24.661685	2018-02-14 17:31:28.859643648	
min	1.000000	0.000000	2011-03-28 00:00:00	
25%	1.000000	1.000000	2017-05-28 08:57:45.987716864	

```
50%          2.000000           6.000000            2019-02-17 00:00:00
75%          5.000000          26.000000            2019-06-20 00:00:00
max         30.000000         629.000000            2019-07-08 00:00:00
std          8.063413          46.168898                            NaN

       reviews_por_mes  calculado_host_listings_count  disponibilidade_365
count     41751.000000                   41751.000000         41751.000000
mean          1.266889                       5.569304           106.313214
min           0.010000                       1.000000             0.000000
25%           0.270000                       1.000000             0.000000
50%           0.730000                       1.000000            35.000000
75%           1.670000                       2.000000           206.000000
max          58.500000                     327.000000           365.000000
std           1.560914                      26.634834           129.312700
```

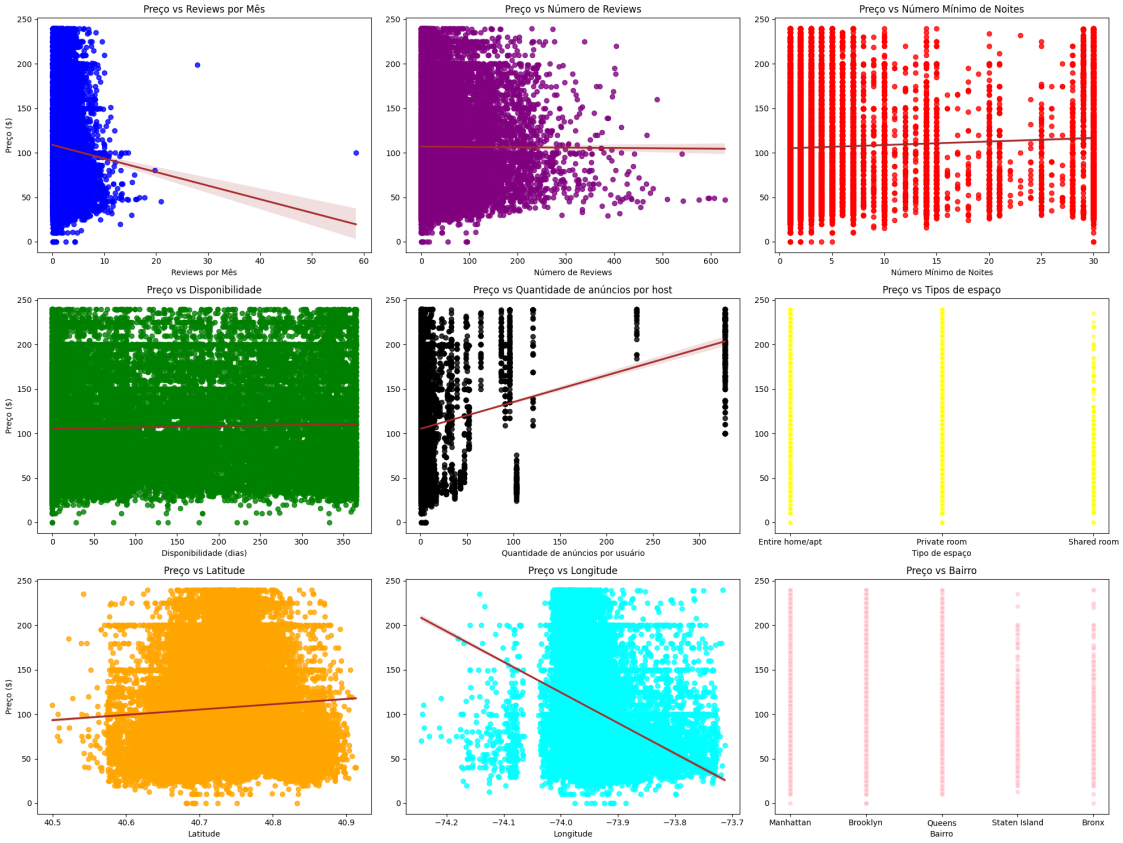
[22]:
```
#Adiciona apenas as colunas que contenham valores numéricos para a visualização
 ↪na matriz de correlação
df_numeric = df_clean.select_dtypes(include=[np.number])
plt.figure(figsize=(8, 6))
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.show()
```



[23]:
```
# Define o tamanho da figura e o número de subplots (3 linha, 3 colunas)
fig, axs = plt.subplots(3, 3, figsize=(20, 15))


#Gráficos:

# 1. Preço vs Reviews por Mês
sns.regplot(data=df_clean, x='reviews_por_mes', y='price', ax=axs[0, 0],␣
 ↪color='blue', line_kws={'color': 'brown'})
axs[0, 0].set_title('Preço vs Reviews por Mês')
axs[0, 0].set_xlabel('Reviews por Mês')
axs[0, 0].set_ylabel('Preço ($)')

# 2. Preço vs Número de Reviews
sns.regplot(data=df_clean, x='numero_de_reviews', y='price', ax=axs[0, 1],␣
 ↪color='purple', line_kws={'color': 'brown'})
axs[0, 1].set_title('Preço vs Número de Reviews')
```

```
axs[0, 1].set_xlabel('Número de Reviews')
axs[0, 1].set_ylabel('')

# 3. Gráfico: Preço vs Número Mínimo de Noites
sns.regplot(data=df_clean, x='minimo_noites', y='price', ax=axs[0, 2],␣
 ↪color='red', line_kws={'color': 'brown'})
axs[0, 2].set_title('Preço vs Número Mínimo de Noites')
axs[0, 2].set_xlabel('Número Mínimo de Noites')
axs[0, 2].set_ylabel('')

# 4. Gráfico: Preço vs Disponibilidade
sns.regplot(data=df_clean, x='disponibilidade_365', y='price', ax=axs[1, 0],␣
 ↪color='green', line_kws={'color': 'brown'})
axs[1, 0].set_title('Preço vs Disponibilidade')
axs[1, 0].set_xlabel('Disponibilidade (dias)')
axs[1, 0].set_ylabel('Preço ($)')

# 5. Preço vs Quantidade de anúncios por host
sns.regplot(data=df_clean, x='calculado_host_listings_count', y='price',␣
 ↪ax=axs[1, 1], color='black', line_kws={'color': 'brown'})
axs[1, 1].set_title('Preço vs Quantidade de anúncios por host')
axs[1, 1].set_xlabel('Quantidade de anúncios por usuário')
axs[1, 1].set_ylabel('')

# 6. Preço vs Tipos de quartos
sns.scatterplot(data=df_clean, x='room_type', y='price', ax=axs[1, 2],␣
 ↪color='yellow', alpha=0.6)
axs[1, 2].set_title('Preço vs Tipos de espaço')
axs[1, 2].set_xlabel('Tipo de espaço')
axs[1, 2].set_ylabel('')

# 7. Preço vs Latitude
sns.regplot(data=df_clean, x='latitude', y='price', ax=axs[2, 0],␣
 ↪color='orange', line_kws={'color': 'brown'})
axs[2, 0].set_title('Preço vs Latitude')
axs[2, 0].set_xlabel('Latitude')
axs[2, 0].set_ylabel('Preço ($)')

# 8. Preço vs Longitude
sns.regplot(data=df_clean, x='longitude', y='price', ax=axs[2, 1],␣
 ↪color='cyan', line_kws={'color': 'brown'})
axs[2, 1].set_title('Preço vs Longitude')
axs[2, 1].set_xlabel('Longitude')
axs[2, 1].set_ylabel('')

# 9. Preço vs Bairro
```

```
sns.scatterplot(data=df_clean, x='bairro_group', y='price', ax=axs[2, 2],␣
 ↪color='pink', alpha=0.6)
axs[2, 2].set_title('Preço vs Bairro')
axs[2, 2].set_xlabel('Bairro')
axs[2, 2].set_ylabel('')

# Ajusta o layout para evitar sobreposição
plt.tight_layout()
plt.show()
```



3.1 Conclusões a partir dos gráficos

1. **Preço vs Reviews por Mês & Número de Reviews:**
   - A maioria dos imóveis com preços mais baixos tende a ter mais reviews por mês e maior número de reviews no total. Imóveis mais caros têm menos reviews, sugerindo que imóveis acessíveis têm maior rotatividade de hóspedes.
2. **Preço vs Número Mínimo de Noites:**
   - A maior parte dos anúncios com preços baixos exige poucas noites mínimas. Há poucos anúncios caros com exigências de muitas noites, indicando que a maioria dos hóspedes prefere estadias mais curtas.
3. **Preço vs Disponibilidade:**

- Não há uma correlação clara entre disponibilidade e preço. Anúncios de todos os preços estão distribuídos ao longo de diferentes níveis de disponibilidade.

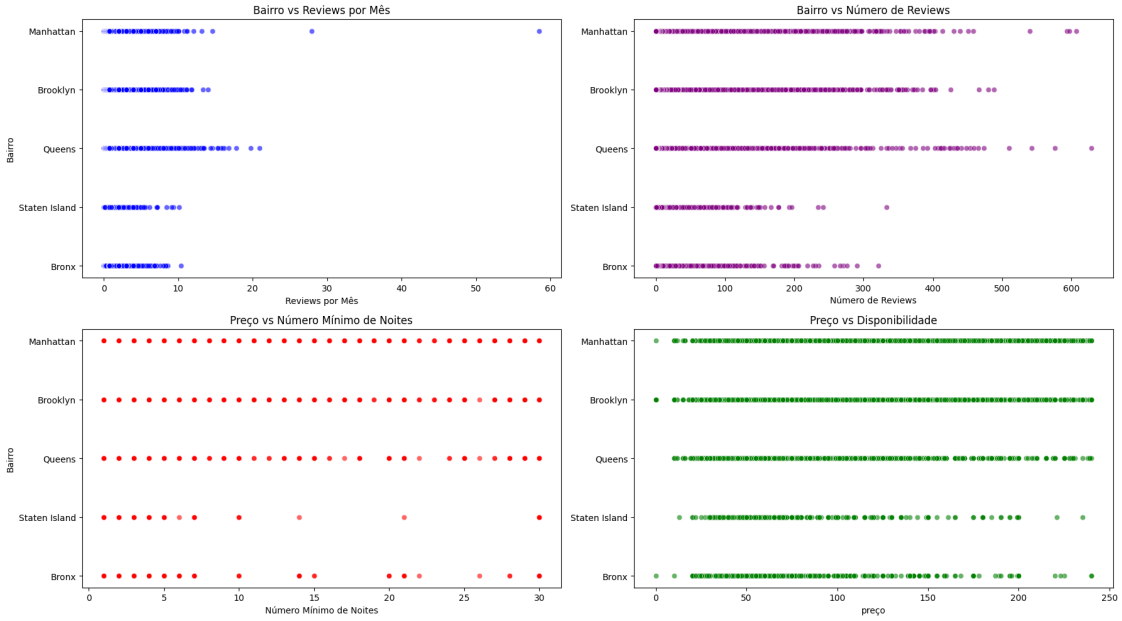
4. **Preço vs Quantidade de Anúncios por Host:**
   - Hosts com muitos anúncios tendem a ter preços mais baixos, sugerindo uma possível gestão profissional.
5. **Preço vs Tipos de Espaço:** O tipo de espaço influencia fortemente o preço:
   - Entire home/apt: Preços altos.
   - Private room: Preços medianos.
   - Shared room: Preços mais baixos.
6. **Preço vs Latitude & Longitude:**
   - A concentração de preços mais altos parece estar localizada em regiões específicas, possivelmente áreas centrais ou turísticas.
7. **Preço vs Bairro:**
   - Bairros como Manhattan apresentam preços mais altos em comparação com bairros como Staten Island e Bronx. Isso reflete a valorização imobiliária das diferentes regiões.

4 Pergunta 1:

Supondo que uma pessoa esteja pensando em investir em um apartamento para alugar na plataforma, onde seria mais indicada a compra?

[24]:
```
# Define o tamanho da figura e o número de subplots (3 linha, 3 colunas)
fig, axs = plt.subplots(2, 2, figsize=(18, 10))


#Gráficos:

# 1. Preço vs Reviews por Mês
sns.scatterplot(data=df_clean, x='reviews_por_mes', y='bairro_group', ax=axs[0,␣
 ↪0], color='blue', alpha=0.6)
axs[0, 0].set_title('Bairro vs Reviews por Mês')
axs[0, 0].set_xlabel('Reviews por Mês')
axs[0, 0].set_ylabel('Bairro')

# 2. Bairro vs Número de Reviews
sns.scatterplot(data=df_clean, x='numero_de_reviews', y='bairro_group',␣
 ↪ax=axs[0, 1], color='purple', alpha=0.6)
axs[0, 1].set_title('Bairro vs Número de Reviews')
axs[0, 1].set_xlabel('Número de Reviews')
axs[0, 1].set_ylabel('')

# 3. Gráfico: Bairro vs Número Mínimo de Noites
sns.scatterplot(data=df_clean, x='minimo_noites', y='bairro_group', ax=axs[1,␣
 ↪0], color='red', alpha=0.6)
axs[1, 0].set_title('Preço vs Número Mínimo de Noites')
axs[1, 0].set_xlabel('Número Mínimo de Noites')
axs[1, 0].set_ylabel('Bairro')
```

```
# 4. Gráfico: Bairro vs Preço
sns.scatterplot(data=df_clean, x='price', y='bairro_group', ax=axs[1, 1],␣
 ↪color='green', alpha=0.6)
axs[1, 1].set_title('Preço vs Disponibilidade')
axs[1, 1].set_xlabel('preço')
axs[1, 1].set_ylabel('')

# Ajusta o layout para evitar sobreposição
plt.tight_layout()
plt.show()
```



4.1 Resposta:

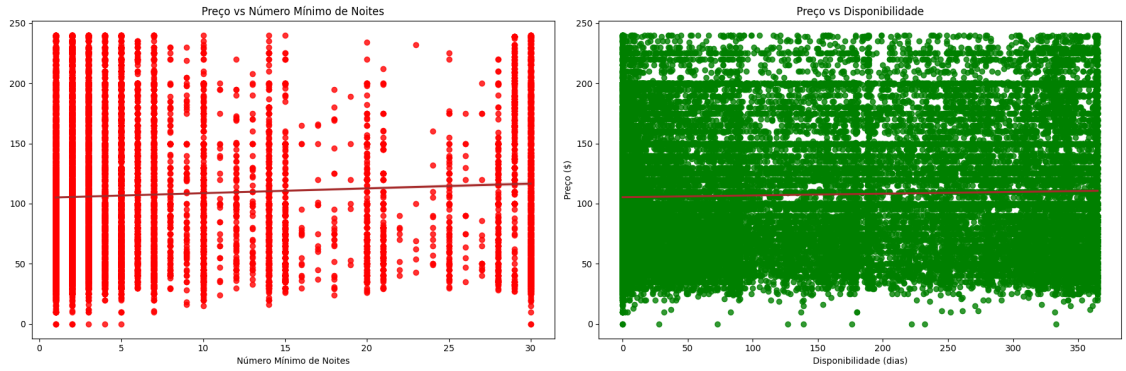
Por meio dos gráficos, é possível concluir que o Bairro Queens seria o mais apropriado a se comprar um imóvel para alugar, pois a rotatividade é maior (maior número de reviews por mês e total), além do número mínimo de noites ser menor e o preço do aluguel ser um valor médio.

5 Pergunta 2:

`O número mínimo de noites e a disponibilidade ao longo do ano interferem no preço?`

[25]:
```
# Define o tamanho da figura e o número de subplots (3 linha, 3 colunas)
fig, axs = plt.subplots(1, 2, figsize=(18, 6))
```

```
#Gráficos:

# 1. Gráfico: Preço vs Número Mínimo de Noites
sns.regplot(data=df_clean, x='minimo_noites', y='price', ax=axs[0],␣
 ↪color='red', line_kws={'color': 'brown'})
axs[0].set_title('Preço vs Número Mínimo de Noites')
axs[0].set_xlabel('Número Mínimo de Noites')
axs[0].set_ylabel('')

# 2. Gráfico: Preço vs Disponibilidade
sns.regplot(data=df_clean, x='disponibilidade_365', y='price', ax=axs[1],␣
 ↪color='green', line_kws={'color': 'brown'})
axs[1].set_title('Preço vs Disponibilidade')
axs[1].set_xlabel('Disponibilidade (dias)')
axs[1].set_ylabel('Preço ($)')

# Ajusta o layout para evitar sobreposição
plt.tight_layout()
plt.show()
```



5.1 Resposta:

É possível inferir, pelos gráficos, que quando o número mínimo de noites é menor, o preço tende a ser menor. Porém a relação entre disponibilidade e preço é bem distribuída, sugerindo que não há interferência significativa da disponibilidade no preço.

6 Pergunta 3:

Existe algum padrão no texto do nome do local para lugares de mais alto valor?

[26]:
```
# Criando faixas de preço
df_clean['faixa_preco'] = pd.cut(df_clean['price'], bins=[0, 69, 175,␣
 ↪df_clean['price'].max()], labels=['Baixo', 'Médio', 'Alto'])
```

[27]:
```
from collections import Counter
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

stop_words = set(stopwords.words('english'))

# Função para processar texto
def processar_texto(texto):
    tokens = word_tokenize(texto.lower().translate(str.maketrans('', '', string.
 ↪punctuation)))
    palavras_filtradas = [palavra for palavra in tokens if palavra not in␣
 ↪stop_words]
    return palavras_filtradas

# Inicializando variáveis
faixa_alto_palavras = []
faixa_medio_baixo_palavras = []

# Coletando palavras da faixa de preço "Alto"
locais_alto = df_clean[df_clean['faixa_preco'] == 'Alto']['nome'].tolist()
for local in locais_alto:
    faixa_alto_palavras.extend(processar_texto(local))

# Coletando palavras das faixas de preço "Médio" e "Baixo"
locais_outros = df_clean[df_clean['faixa_preco'].isin(['Médio',␣
 ↪'Baixo'])]['nome'].tolist()
for local in locais_outros:
    faixa_medio_baixo_palavras.extend(processar_texto(local))

# Contabilizando as palavras
frequencia_alto = Counter(faixa_alto_palavras)
frequencia_outros = Counter(faixa_medio_baixo_palavras)

# Obtendo palavras exclusivas da faixa "Alto"
palavras_exclusivas_alto = {palavra: freq for palavra, freq in frequencia_alto.
 ↪items() if palavra not in frequencia_outros}

# Exibindo as palavras exclusivas da faixa "Alto"
print("\nPalavras exclusivas na faixa de preço 'Alto':")
print(sorted(palavras_exclusivas_alto.items(), key=lambda x: x[1],␣
 ↪reverse=True)[:15])  # Exibindo as 10 mais comuns
```

```
Palavras exclusivas na faixa de preço 'Alto':
[('stock', 91), ('blueground', 25), (' ', 6), ('sixth', 6), ('56th', 5),
```

```
('neighborhoodeast', 5), ('86th5th', 5), ('ktown', 5), ('brd', 3), ('29th', 3),
('32', 3), ('viewslincoln', 3), ('sqrluxury', 3), ('welive', 3), ('kan', 3)]
```

6.1 Resposta:

Sim, a palavra "stock" é a que mais aparece nos anúncios de mais alto valor.

7 Passo 3: Previsão de dados

Neste passo, para o problema de previsão de preços, utilizaremos Regressão Linear, pois esse modelo é adequado para modelar a relação entre variáveis contínuas e pode ser eficaz para identificar padrões e tendências no comportamento dos dados. O modelo prevê um valor numérico (preço) com base em uma ou mais variáveis independentes, como área, número de quartos, localização, entre outras.

Além disso, algumas transformações podem ser necessárias para otimizar o modelo, como:

```
Normalização ou padronização de variáveis numéricas para garantir que todas
as variáveis tenham a mesma escala e melhorar a performance do modelo.

Transformação logarítmica em variáveis que apresentam grandes diferenças de
escala (como preço ou área) para reduzir a influência de valores extremos.

Criação de variáveis dummy para variáveis categóricas, como tipo de imóvel
ou localização, para que possam ser incluídas no modelo de regressão.
```

O tipo de problema é regressão, pois estamos tentando prever um valor contínuo (preço).

Utilizaremos o R^2 (coeficiente de determinação) como medida de performance, que indica o quanto da variação dos dados é explicada pelo modelo. Um R^2 próximo de 1 sugere que o modelo é bem ajustado aos dados, enquanto um valor próximo de 0 indica que o modelo não explica bem a variabilidade dos dados. Embora o R^2 seja útil, também podemos considerar outras métricas, como o Erro Quadrático Médio (MSE) ou Erro Absoluto Médio (MAE), para avaliar o modelo de forma mais detalhada, especialmente em relação a outliers.

[28]:
```
print(df_clean.isnull().sum())
```

```
id                              0
nome                            0
host_id                         0
host_name                       0
bairro_group                    0
bairro                          0
latitude                        0
longitude                       0
room_type                       0
price                           0
minimo_noites                   0
numero_de_reviews               0
ultima_review                   0
reviews_por_mes                 0
```

```
calculado_host_listings_count     0
disponibilidade_365               0
faixa_preco                      11
dtype: int64
```

[29]:
```
df_clean.dtypes
```

[29]:
```
id                                  int64
nome                               object
host_id                             int64
host_name                          object
bairro_group                       object
bairro                             object
latitude                          float64
longitude                         float64
room_type                          object
price                               int64
minimo_noites                       int64
numero_de_reviews                   int64
ultima_review                datetime64[ns]
reviews_por_mes                   float64
calculado_host_listings_count       int64
disponibilidade_365                 int64
faixa_preco                      category
dtype: object
```

[30]:
```
#!pip install xgboost

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
from xgboost import XGBRegressor

# Remover colunas irrelevantes
df_clean = df_clean.drop(columns=["id", "nome", "host_id", "host_name",␣
 ↪"ultima_review", "faixa_preco"], errors="ignore")

# Tratar valores ausentes
df_clean["reviews_por_mes"] = df_clean["reviews_por_mes"].fillna(0)

# Codificar variáveis categóricas
df_clean = pd.get_dummies(df_clean, columns=["bairro_group", "bairro",␣
 ↪"room_type"], drop_first=True)

# Separar variáveis
X = df_clean.drop(columns=["price"])
y = df_clean["price"]
```

```
# Remover outliers
mask = (y > y.quantile(0.05)) & (y < y.quantile(0.95))
X = X[mask]
y = y[mask]

# Dividir e treinar
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
modelo = LinearRegression()
modelo.fit(X_train, y_train)

# XGBoost Regressor
xgb = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5,␣
 ↪random_state=42)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)

# Avaliação
print("XGBoost Regressor:")
print("R²:", r2_score(y_test, y_pred_xgb))
print("MAE:", mean_absolute_error(y_test, y_pred_xgb))
print("MSE:", mean_squared_error(y_test, y_pred_xgb))
```

```
XGBoost Regressor:
R²: 0.5505998730659485
MAE: 22.00698471069336
MSE: 810.343505859375
```

[31]:
```
from sklearn.linear_model import Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# Dicionário com os modelos
modelos = {
    'LinearRegression': LinearRegression(),
    'Ridge': Ridge(),
    'Lasso': Lasso(),
    'RandomForest': RandomForestRegressor(random_state=42),
    'GradientBoosting': GradientBoostingRegressor(random_state=42),
}

# Treinar e avaliar cada modelo
for nome, modelo in modelos.items():
    modelo.fit(X_train, y_train)
    y_pred = modelo.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
```

```
    mse = mean_squared_error(y_test, y_pred)

    print(f"\n{nome}")
    print(f"R²: {r2:.4f}")
    print(f"MAE: {mae:.2f}")
    print(f"MSE: {mse:.2f}")
```

```
LinearRegression
R²: 0.5111
MAE: 23.01
MSE: 881.54

Ridge
R²: 0.5103
MAE: 23.04
MSE: 882.98

Lasso
R²: 0.3920
MAE: 26.33
MSE: 1096.35

RandomForest
R²: 0.5428
MAE: 22.03
MSE: 824.47

GradientBoosting
R²: 0.5369
MAE: 22.50
MSE: 835.01
```

7.1 Previsão do novo imóvel:

Após a avaliação comparativa entre diferentes algoritmos de regressão — incluindo Linear Regression, Ridge, Lasso, Random Forest e Gradient Boosting — o modelo XGBoost demonstrou o melhor desempenho. Ele obteve o maior coeficiente de determinação ($R^2 = 0.5505$), indicando melhor capacidade explicativa da variabilidade nos preços. Além disso, apresentou os menores valores de erro médio absoluto (MAE = 22.00) e erro quadrático médio (MSE = 810.34), o que reforça sua superioridade na acurácia preditiva. Com base nesses resultados, o XGBoost foi selecionado como o modelo final para estimar o preço do novo imóvel.

[32]:
```
# Informações do novo imóvel
novo_imovel = {
    'latitude': 40.75362,
    'longitude': -73.98377,
    'minimo_noites': 1,
```

```
    'numero_de_reviews': 45,
    'reviews_por_mes': 0.38,
    'calculado_host_listings_count': 2,
    'disponibilidade_365': 355,
    'bairro_group_Manhattan': 1,
    'bairro_Midtown': 1,
    'room_type_Entire home/apt': 1,
}

# Criar DataFrame com uma linha
X_novo = pd.DataFrame([novo_imovel])

# Identificar colunas faltantes em relação ao treino
colunas_faltantes = [col for col in X_train.columns if col not in X_novo.
 ↪columns]

# Adicionar colunas faltantes com valor 0
faltantes = pd.DataFrame(0, index=X_novo.index, columns=colunas_faltantes)
X_novo = pd.concat([X_novo, faltantes], axis=1)

# Garantir a mesma ordem de colunas que o X_train
X_novo = X_novo[X_train.columns]

# Previsão
preco_previsto = xgb.predict(X_novo)[0]
print(f"Preço previsto para o novo imóvel: ${preco_previsto:.2f}")
```

```
Preço previsto para o novo imóvel: $157.73
```

8 Conclusão

Neste projeto, realizamos uma análise preditiva de preços de imóveis disponibilizados na plataforma Airbnb na cidade de Nova York. Após o pré-processamento dos dados — incluindo limpeza, codificação de variáveis categóricas e remoção de outliers —, diversos modelos de regressão foram testados e comparados com base em métricas como R^2, MAE e MSE.

O modelo XGBoost apresentou o melhor desempenho entre as opções avaliadas, com os seguintes resultados:

R^2: 0.5505

MAE: 22.00

MSE: 810.34

Devido à sua capacidade de capturar relações não lineares e interações entre variáveis, o XGBoost mostrou-se mais eficaz na previsão dos preços em comparação aos modelos lineares.

Além disso, o modelo final foi utilizado para prever o valor de um novo imóvel com base em suas

características específicas, demonstrando sua aplicabilidade prática.