Algoritmos

February 7, 2025

[76]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Testando diferentes algoritmos para entender a dinâmica do mercado imobiliário nos bairros de NYC:

Processo:

-Instalar dependências

-importação das bibliotecas necessárias

-Configurar semente para reprodutibilidade

[77]:
```
np.random.seed(20)
```

[78]:
```
dtypes = {
    0: 'object',
    2: 'object',
    3: 'object',
    6: 'float',  # or another appropriate type
    7: 'float',
    9: 'float',
    12: 'float',
    13: 'int'
}



data = pd.read_csv("teste_indicium_precificacao_v2.csv", dtype = dtypes)
```

[79]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48811 entries, 0 to 48810
Data columns (total 15 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48811 non-null  object
```

```
 1   nome                            48795 non-null  object
 2   host_id                         48811 non-null  object
 3   host_name                       48790 non-null  object
 4   bairro_group                    48811 non-null  object
 5   bairro                          48811 non-null  object
 6   latitude                        48811 non-null  float64
 7   longitude                       48811 non-null  float64
 8   room_type                       48811 non-null  object
 9   price                           48811 non-null  float64
 10  minimo_noites                   48811 non-null  int64
 11  numero_de_reviews               48811 non-null  int64
 12  reviews_por_mes                 48811 non-null  float64
 13  calculado_host_listings_count   48811 non-null  int64
 14  disponibilidade_365             48811 non-null  int64
dtypes: float64(4), int64(4), object(7)
memory usage: 5.6+ MB
```

1 Árvore de Decisão

- Objetivo: As Árvores de Decisão dividem os dados em regiões baseadas em condições que maximizam a pureza dos grupos. Elas são populares em problemas de classificação devido à sua interpretabilidade.
- Implementação: A função decision_tree_example treina um DecisionTreeClassifier para prever classes e mede a acurácia e o relatório de classificação. Essa árvore é particularmente útil para identificar padrões hierárquicos nos dados.

[80]:
```
def decision_tree_example():
    print("\n### Modelo de Árvore de Decisão ###")
    tree = DecisionTreeClassifier(random_state=42)
    tree.fit(X_train, y_train)
    y_pred = tree.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Acurácia da Árvore de Decisão: {accuracy:.2f}")
    print("\nRelatório de Classificação:\n", classification_report(y_test,␣
 ↪y_pred))
    return y_test, y_pred
```

2 Pipeline

[81]:
```
def pipeline_example():
    print("\n### Pipeline com StandardScaler e Random Forest ###")
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', RandomForestClassifier(n_estimators=100,␣
 ↪random_state=42))
```

```
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Acurácia do Pipeline com Random Forest: {accuracy:.2f}")
    print("\nRelatório de Classificação:\n", classification_report(y_test,␣
 ↪y_pred))
    return y_test, y_pred
```

- Objetivo: Pipelines facilitam o pré-processamento e treinamento em uma única etapa. Isso é útil para garantir que todos os passos sejam executados consistentemente em cada fase do desenvolvimento.
- Implementação:
  - A função pipeline_example cria uma Pipeline que primeiro padroniza (StandardScaler) os dados e então treina um RandomForestClassifier.
  - O StandardScaler padroniza as características para ter média zero e variância unitária, o que ajuda alguns modelos a convergirem melhor.
  - A acurácia e o relatório de classificação são então calculados para o pipeline completo.

[82]:
```
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, accuracy_score,␣
 ↪classification_report, confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

target = ['bairro_group']

Y = data[target]
Y = Y.values.ravel()
X = data.iloc[:, [6, 7, 9, 10, 11, 12, 13, 14]]
```

[83]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,␣
 ↪random_state=42)
```

3 KNN

- Objetivo: O modelo KNN é um algoritmo de classificação que classifica um ponto novo com base na maioria dos "vizinhos" mais próximos. É ideal para problemas de classificação como classificação de espécies de flores, diagnósticos médicos, entre outros.

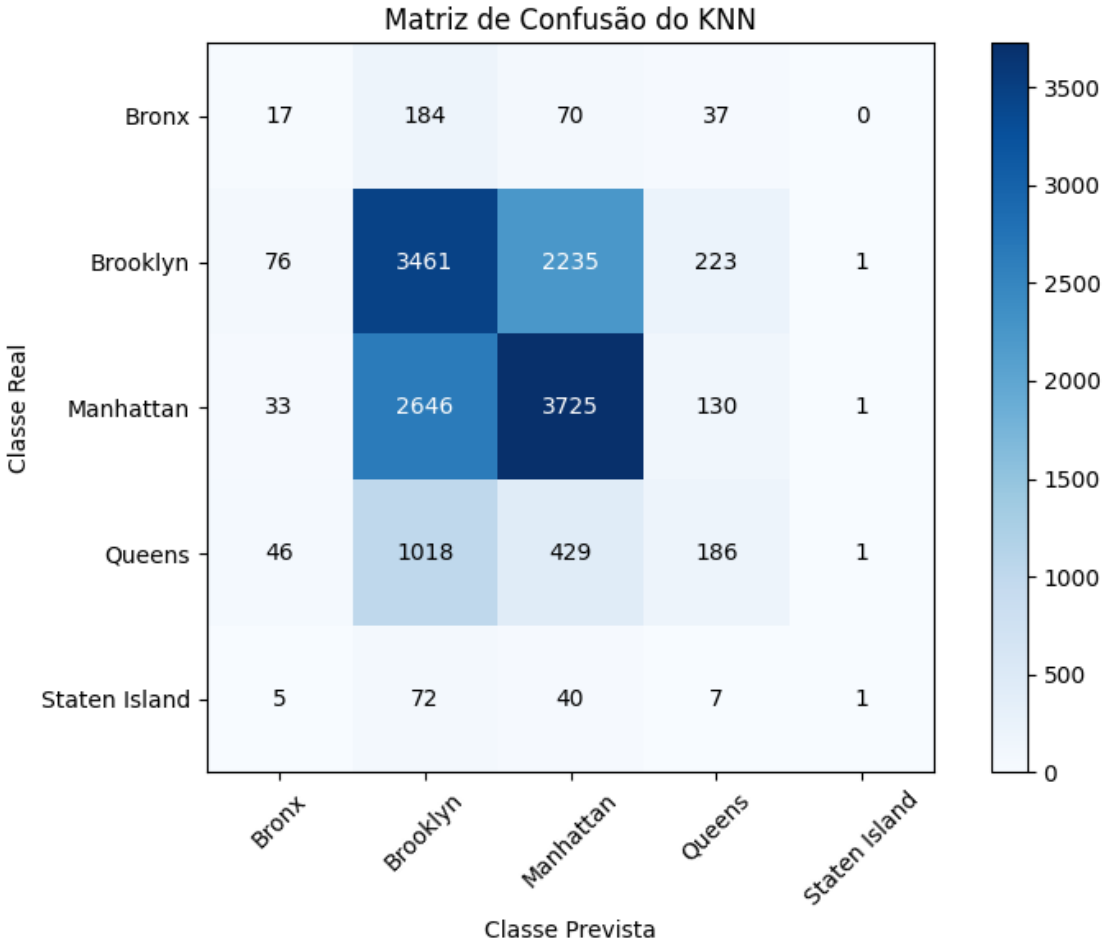
- Implementação: A função knn_example usa o KNeighborsClassifier com 5 vizinhos. Após o treinamento, ele calcula a acurácia (proporção de previsões corretas) e gera um relatório de classificação para mostrar precisão, recall e f1-score para cada classe.

[84]:
```
def knn_example():
    print("\n### Modelo de Classificação KNN ###")
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Acurácia do KNN: {accuracy:.2f}")
    print("\nRelatório de Classificação:\n", classification_report(y_test,␣
 ↪y_pred))
    return y_test, y_pred
```

[88]:
```
y_test_values, y_pred_values = knn_example()

target_names = np.unique(y)
conf_matrix = confusion_matrix(y_test_values, y_pred_values)
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Matriz de Confusão do KNN')
plt.colorbar()
tick_marks = np.arange(len(target_names))
plt.xticks(tick_marks, target_names, rotation=45)
plt.yticks(tick_marks, target_names)

thresh = conf_matrix.max() / 2.
for i, j in np.ndindex(conf_matrix.shape):
   plt.text(j, i, format(conf_matrix[i, j], 'd'),
            ha="center", va="center",
            color="white" if conf_matrix[i, j] > thresh else "black")

plt.ylabel('Classe Real')
plt.xlabel('Classe Prevista')
plt.tight_layout()
plt.show()
```

```
### Modelo de Classificação KNN ###
Acurácia do KNN: 0.50

Relatório de Classificação:
               precision    recall  f1-score   support

        Bronx       0.10      0.06      0.07       308
     Brooklyn       0.47      0.58      0.52      5996
    Manhattan       0.57      0.57      0.57      6535
```

```
          Queens       0.32      0.11      0.16      1680
   Staten Island       0.25      0.01      0.02       125

        accuracy                           0.50     14644
       macro avg       0.34      0.26      0.27     14644
    weighted avg       0.49      0.50      0.49     14644
```



[89]:
```
# Chamando a função e obtendo os valores reais e previstos
y_test_values_pipeline, y_pred_values_pipeline = pipeline_example()
```

```
### Pipeline com StandardScaler e Random Forest ###
Acurácia do Pipeline com Random Forest: 0.59

Relatório de Classificação:
              precision    recall  f1-score   support
```

```
         Bronx       0.54      0.06      0.12       308
      Brooklyn       0.55      0.63      0.59      5996
     Manhattan       0.64      0.68      0.66      6535
        Queens       0.58      0.21      0.31      1680
Staten Island       0.78      0.06      0.10       125

      accuracy                           0.59     14644
     macro avg       0.62      0.33      0.36     14644
  weighted avg       0.59      0.59      0.57     14644
```

[90]:
```
# Regressão Linear
```

[92]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import numpy as np
import pandas as pd

numeric_features = ['latitude', 'longitude', 'minimo_noites',␣
 ↪'numero_de_reviews',
                    'reviews_por_mes', 'calculado_host_listings_count',␣
 ↪'disponibilidade_365']
categorical_features = ['bairro_group', 'room_type']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(drop='first', sparse_output=False),␣
 ↪categorical_features)
    ])

model = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

X = data[numeric_features + categorical_features]
y = data['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'R² Score: {r2:.2f}')

feature_names = (numeric_features +
                 [f"{feat}_{val}" for feat, vals in
                  zip(categorical_features,
                      model.named_steps['preprocessor']
                      .named_transformers_['cat'].categories_)
                  for val in vals[1:]])

coefficients = model.named_steps['regressor'].coef_
feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
})
print("\nFeature Importance:")
print(feature_importance.sort_values(by='Coefficient', key=abs,␣
 ↪ascending=False))
```

```
Root Mean Squared Error: 249.42
R² Score: 0.08

Feature Importance:
                          Feature  Coefficient
12          room_type_Shared room  -144.231696
11         room_type_Private room  -111.562019
8          bairro_group_Manhattan    88.393817
7           bairro_group_Brooklyn    33.123675
6             disponibilidade_365    24.437935
9             bairro_group_Queens    14.410384
10     bairro_group_Staten Island    12.782980
3                numero_de_reviews    -9.004347
4                  reviews_por_mes    -7.437790
0                         latitude     4.584020
5   calculado_host_listings_count    -3.707165
1                        longitude     0.068797
2                   minimo_noites    -0.054486
```

[93]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score
```

```
numeric_features = ['latitude', 'longitude', 'minimo_noites',␣
 ↪'numero_de_reviews',
                    'reviews_por_mes', 'calculado_host_listings_count',␣
 ↪'disponibilidade_365']
categorical_features = ['bairro_group', 'room_type']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(drop='first', sparse_output=False),␣
 ↪categorical_features)
    ])

model = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(
        n_estimators=100,
        max_depth=15,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42
    ))
])

X = data[numeric_features + categorical_features]
y = data['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'R² Score: {r2:.2f}')

feature_names = (numeric_features +
                 [f"{feat}_{val}" for feat, vals in
                  zip(categorical_features,
                      model.named_steps['preprocessor']
                      .named_transformers_['cat'].categories_)
```

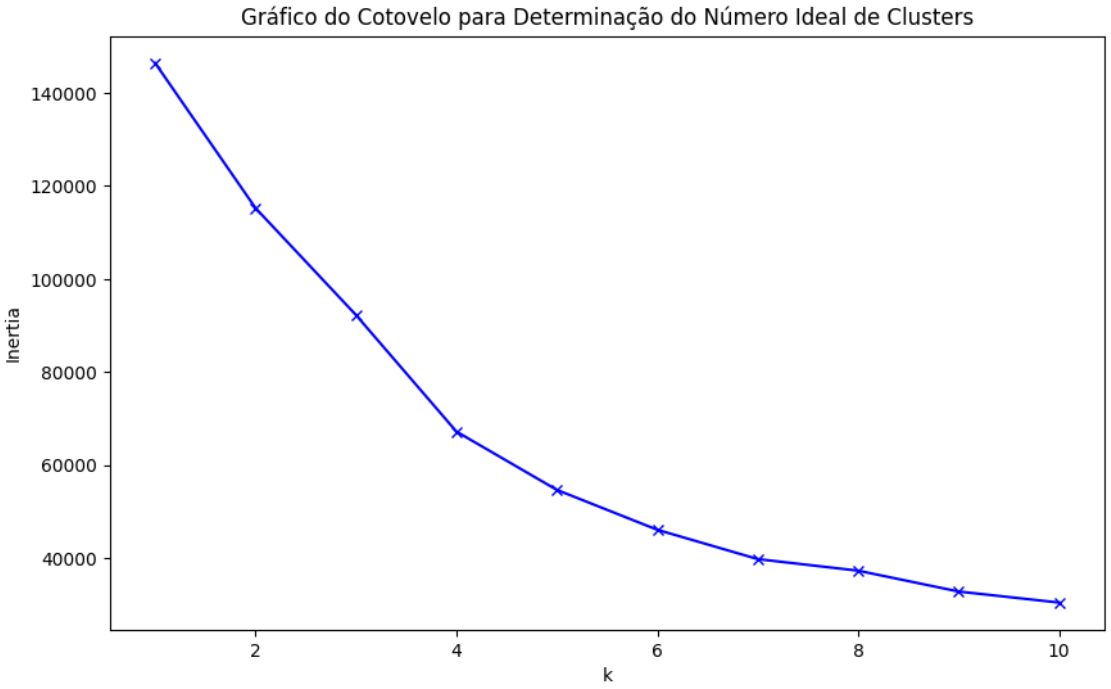
```
                for val in vals[1:]])

feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': model.named_steps['regressor'].feature_importances_
})

print("\nFeature Importance:")
print(feature_importance.sort_values(by='Importance', ascending=False))

# Validação cruzada para avaliar a estabilidade do modelo
cv_scores = cross_val_score(model, X, y, cv=5, scoring='r2')
print("\nCross-validation scores (R²):")
print(f"Mean R²: {cv_scores.mean():.2f} (+/- {cv_scores.std() * 2:.2f})")
```

```
Root Mean Squared Error: 243.66
R² Score: 0.12

Feature Importance:
                          Feature  Importance
0                        latitude    0.253045
1                       longitude    0.198732
2                    minimo_noites    0.134638
6              disponibilidade_365    0.100849
11         room_type_Private room    0.094009
5   calculado_host_listings_count    0.065666
4                 reviews_por_mes    0.057762
3               numero_de_reviews    0.044769
8          bairro_group_Manhattan    0.028149
12          room_type_Shared room    0.012428
9             bairro_group_Queens    0.003851
10     bairro_group_Staten Island    0.003744
7           bairro_group_Brooklyn    0.002356

Cross-validation scores (R²):
Mean R²: 0.06 (+/- 0.09)
```

[94]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Preparando os dados para clustering
X_cluster = data[['latitude', 'longitude', 'price']].copy()
```

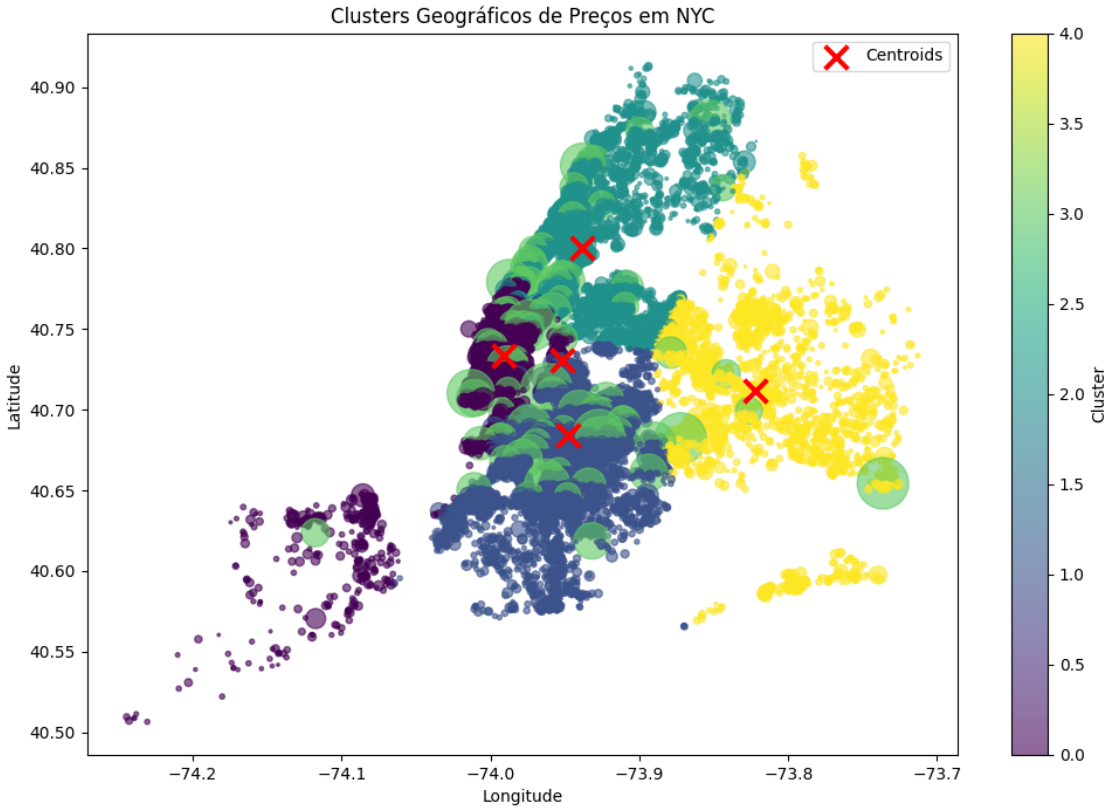
```
# Padronizando os dados
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)

# Determinando número ideal de clusters usando método do cotovelo
inertias = []
K = range(1, 11)
for k in K:
   kmeans = KMeans(n_clusters=k, random_state=42)
   kmeans.fit(X_scaled)
   inertias.append(kmeans.inertia_)

# Plotando gráfico do cotovelo
plt.figure(figsize=(10, 6))
plt.plot(K, inertias, 'bx-')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.title('Gráfico do Cotovelo para Determinação do Número Ideal de Clusters')
plt.show()

# Aplicando K-means com o número escolhido de clusters
n_clusters = 5  # Você pode ajustar este número com base no gráfico do cotovelo
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
data['Cluster'] = kmeans.fit_predict(X_scaled)

# Estatísticas dos clusters
cluster_stats = data.groupby('Cluster').agg({
   'price': ['mean', 'std', 'count'],
   'latitude': 'mean',
   'longitude': 'mean'
}).round(2)

print("\nEstatísticas dos Clusters:")
print(cluster_stats)

# Visualização dos clusters no mapa
plt.figure(figsize=(12, 8))
scatter = plt.scatter(data['longitude'], data['latitude'],
                      c=data['Cluster'],
                      cmap='viridis',
                      alpha=0.6,
                      s=data['price']/10)  # Tamanho dos pontos baseado no preço

plt.colorbar(scatter, label='Cluster')
plt.title('Clusters Geográficos de Preços em NYC')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
```

```
# Adicionando centróides
centroids = kmeans.cluster_centers_
centroids_original = scaler.inverse_transform(centroids)
plt.scatter(centroids_original[:, 1], centroids_original[:, 0],
          c='red', marker='x', s=200, linewidths=3, label='Centroids')

plt.legend()
plt.show()

# Análise detalhada dos clusters
print("\nAnálise dos Clusters por Tipo de Quarto:")
print(pd.crosstab(data['Cluster'], data['room_type'], normalize='index') * 100)

print("\nAnálise dos Clusters por Bairro:")
print(pd.crosstab(data['Cluster'], data['bairro_group'], normalize='index') *␣
 ↪100)

# Boxplot dos preços por cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='price', data=data)
plt.title('Distribuição de Preços por Cluster')
plt.ylabel('Preço')
plt.show()
```



Estatísticas dos Clusters:

	price			latitude	longitude
	mean	std	count	mean	mean
Cluster					
0	136.19	127.33	15413	40.73	-73.99
1	150.10	141.91	18981	40.68	-73.95
2	152.77	159.86	11639	40.80	-73.94
3	4408.39	2219.82	80	40.73	-73.95
4	139.95	126.11	2698	40.71	-73.82

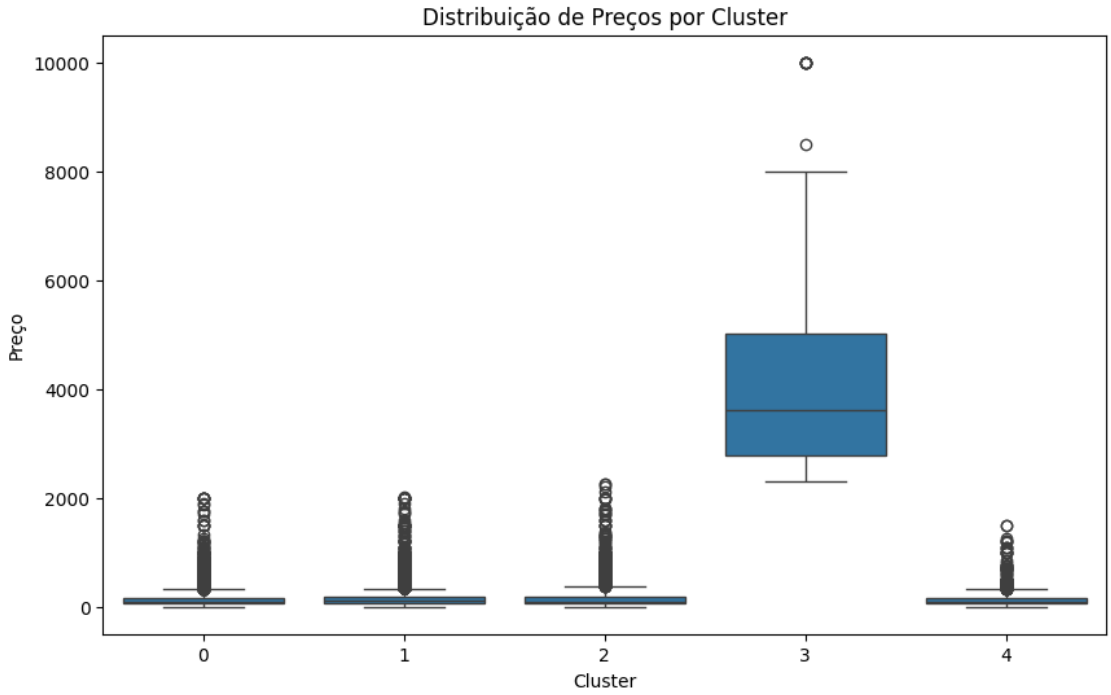


Análise dos Clusters por Tipo de Quarto:

room_type	Entire home/apt	Private room	Shared room
Cluster			
0	49.341465	48.387725	2.270810
1	57.262526	41.272852	1.464623
2	46.576166	49.471604	3.952230
3	82.500000	17.500000	0.000000
4	52.520385	45.255745	2.223870

Análise dos Clusters por Bairro:

```
bairro_group     Bronx   Brooklyn  Manhattan     Queens  Staten Island
Cluster
0             2.121586  39.739181  44.566275  12.645170       0.927788
1             1.285496  47.047047  44.202097   7.043886       0.421474
2             4.012372  33.172953  43.869748  17.802217       1.142710
3             1.250000  21.250000  72.500000   3.750000       1.250000
4             1.890289  42.327650  45.070423  10.155671       0.555967
```



[95]:
```python
def predict_cluster(new_data, model, scaler):
    # Extrair apenas as features usadas no clustering
    features = ['latitude', 'longitude', 'price']

    # Criar DataFrame com o novo dado
    new_instance = pd.DataFrame({
        'latitude': [new_data['latitude']],
        'longitude': [new_data['longitude']],
        # Como não temos o preço, vamos estimar usando o modelo de preços␣
↪anterior
        'price': [0]  # Inicialmente colocamos 0, será estimado pelo modelo de␣
↪preços
    })

    # Padronizar os dados usando o mesmo scaler
    new_instance_scaled = scaler.transform(new_instance)
```

```
    # Prever o cluster
    cluster = model.predict(new_instance_scaled)

    return cluster[0]

# Carregar e treinar o modelo (assumindo que você já tem o dataset original)
# Preparando os dados para clustering
X_cluster = data[['latitude', 'longitude', 'price']].copy()

# Padronizando os dados
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)

# Aplicando K-means
n_clusters = 5
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(X_scaled)

# Novo dado para classificação
novo_airbnb = {
    'id': 2595,
    'nome': 'Skylit Midtown Castle',
    'host_id': 2845,
    'host_name': 'Jennifer',
    'bairro_group': 'Manhattan',
    'bairro': 'Midtown',
    'latitude': 40.75362,
    'longitude': -73.98377,
    'room_type': 'Entire home/apt',
    'minimo_noites': 1,
    'numero_de_reviews': 45,
    'ultima_review': '2019-05-21',
    'reviews_por_mes': 0.38,
    'calculado_host_listings_count': 2,
    'disponibilidade_365': 355
}

# Prever o cluster
cluster_previsto = predict_cluster(novo_airbnb, kmeans, scaler)

print(f"O imóvel foi classificado no cluster: {cluster_previsto}")

# Análise do cluster previsto
cluster_stats = data[data['Cluster'] == cluster_previsto].agg({
    'price': ['mean', 'std', 'count'],
    'latitude': 'mean',
```

```
    'longitude': 'mean'
}).round(2)

print("\nEstatísticas do Cluster Previsto:")
print(cluster_stats)

print("\nCaracterísticas típicas deste cluster:")
print("\nDistribuição de Tipos de Quarto:")
print(data[data['Cluster'] == cluster_previsto]['room_type'].
 ↪value_counts(normalize=True) * 100)

print("\nDistribuição de Bairros:")
print(data[data['Cluster'] == cluster_previsto]['bairro_group'].
 ↪value_counts(normalize=True) * 100)

# Encontrar imóveis similares no mesmo cluster
similares = data[data['Cluster'] == cluster_previsto].copy()
similares['distancia'] = np.sqrt(
    (similares['latitude'] - novo_airbnb['latitude'])**2 +
    (similares['longitude'] - novo_airbnb['longitude'])**2
)

print("\nImóveis similares mais próximos no mesmo cluster:")
print(similares.nsmallest(5, 'distancia')[['nome', 'price', 'room_type',␣
 ↪'bairro']])
```

```
O imóvel foi classificado no cluster: 0

Estatísticas do Cluster Previsto:
          price  latitude  longitude
mean     136.19     40.73     -73.99
std      127.33       NaN        NaN
count  15413.00       NaN        NaN

Características típicas deste cluster:

Distribuição de Tipos de Quarto:
room_type
Entire home/apt    49.341465
Private room       48.387725
Shared room         2.270810
Name: proportion, dtype: float64

Distribuição de Bairros:
bairro_group
Manhattan        44.566275
Brooklyn         39.739181
```

```
Queens           12.645170
Bronx             2.121586
Staten Island     0.927788
Name: proportion, dtype: float64

Imóveis similares mais próximos no mesmo cluster:
                                                 nome  price        room_type  \
15339          Charming studio close to everything  200.0  Entire home/apt
15357                                      Room 12   40.0     Private room
15171          Charming Brownstone in Clinton Hill  120.0  Entire home/apt
15602  Large 2 bedroom Williamsburg loft apartment  180.0  Entire home/apt
15440       Cozy & clean Bedstuy, Brooklyn bedroom   38.0     Private room

                  bairro
15339     Hell's Kitchen
15357  University Heights
15171       Clinton Hill
15602       Williamsburg
15440  Bedford-Stuyvesant
```

[96]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import numpy as np
import pandas as pd

def predict_price(new_data, model):
    # Criar DataFrame com o novo dado
    new_instance = pd.DataFrame([new_data])

    # Fazer a predição
    predicted_price = model.predict(new_instance)

    return predicted_price[0]

# Preparar os dados para treinamento
numeric_features = ['latitude', 'longitude', 'minimo_noites',␣
 ↪'numero_de_reviews',
                    'reviews_por_mes', 'calculado_host_listings_count',␣
 ↪'disponibilidade_365']
categorical_features = ['bairro_group', 'room_type']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
```

```
        ('cat', OneHotEncoder(drop='first', sparse_output=False),␣
 ↪categorical_features)
    ])

# Criar e treinar o modelo
model = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(
        n_estimators=100,
        max_depth=15,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42
    ))
])

# Preparar X e y para treinamento
X = data[numeric_features + categorical_features]
y = data['price']

# Treinar o modelo
model.fit(X, y)

# Novo apartamento para previsão
novo_airbnb = {
    'latitude': 40.75362,
    'longitude': -73.98377,
    'bairro_group': 'Manhattan',
    'room_type': 'Entire home/apt',
    'minimo_noites': 1,
    'numero_de_reviews': 45,
    'reviews_por_mes': 0.38,
    'calculado_host_listings_count': 2,
    'disponibilidade_365': 355
}

# Fazer a previsão
preco_previsto = predict_price(novo_airbnb, model)

print(f"\nPreço sugerido: ${preco_previsto:.2f}")

# Análise comparativa
similar_properties = data[
    (data['bairro_group'] == novo_airbnb['bairro_group']) &
    (data['room_type'] == novo_airbnb['room_type'])
]
```

```python
print("\nEstatísticas de propriedades similares em Manhattan:")
print(f"Preço médio: ${similar_properties['price'].mean():.2f}")
print(f"Preço mediano: ${similar_properties['price'].median():.2f}")
print(f"Desvio padrão: ${similar_properties['price'].std():.2f}")
print(f"Preço mínimo: ${similar_properties['price'].min():.2f}")
print(f"Preço máximo: ${similar_properties['price'].max():.2f}")

# Encontrar propriedades similares próximas
similar_properties['distance'] = np.sqrt(
    (similar_properties['latitude'] - novo_airbnb['latitude'])**2 +
    (similar_properties['longitude'] - novo_airbnb['longitude'])**2
)

print("\nPreços de 5 propriedades similares mais próximas:")
nearby_properties = similar_properties.nsmallest(5, 'distance')
print(nearby_properties[['price', 'room_type', 'bairro']].to_string())

# Calcular intervalo de confiança para o preço
nearby_mean = nearby_properties['price'].mean()
nearby_std = nearby_properties['price'].std()
confidence_interval = (nearby_mean - 2*nearby_std, nearby_mean + 2*nearby_std)

print(f"\nIntervalo de preço sugerido:")
print(f"Entre ${confidence_interval[0]:.2f} e ${confidence_interval[1]:.2f}")
```

```
Preço sugerido: $625.15

Estatísticas de propriedades similares em Manhattan:
Preço médio: $249.30
Preço mediano: $191.00
Desvio padrão: $331.95
Preço mínimo: $0.00
Preço máximo: $10000.00

Preços de 5 propriedades similares mais próximas:
       price        room_type          bairro
15339  200.0  Entire home/apt  Hell's Kitchen
14909  179.0  Entire home/apt          Harlem
14854  120.0  Entire home/apt          Harlem
15610  400.0  Entire home/apt         Midtown
15749  159.0  Entire home/apt  Hell's Kitchen

Intervalo de preço sugerido:
Entre $-7.14 e $430.34

/tmp/ipykernel_14785/4211488046.py:79: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  similar_properties['distance'] = np.sqrt(
```

4 Cálculo do Preço para Airbnb em Manhattan

4.1 Componentes do Modelo

O modelo utiliza Random Forest Regressor, que combina múltiplas árvores de decisão para fazer previsões mais robustas. As principais features consideradas são:

4.1.1 Features Numéricas

- Latitude: 40.75362
- Longitude: -73.98377
- Mínimo de noites: 1
- Número de reviews: 45
- Reviews por mês: 0.38
- Calculado host listings count: 2
- Disponibilidade (365 dias): 355

4.1.2 Features Categóricas

- Bairro: Manhattan
- Tipo de quarto: Apartamento inteiro

4.2 Processo de Cálculo

1. **Pré-processamento**
   - Features numéricas são padronizadas usando StandardScaler
   - Features categóricas são codificadas usando OneHotEncoder
2. **Predição**
   - O modelo Random Forest usa 100 árvores de decisão
   - Cada árvore faz uma predição baseada em diferentes subconjuntos dos dados
   - A predição final é a média das predições de todas as árvores
3. **Validação do Preço**
   - Comparação com propriedades similares na região
   - Cálculo de estatísticas descritivas
   - Análise de propriedades próximas geograficamente
4. **Intervalo de Confiança**
   - Baseado na média e desvio padrão das 5 propriedades mais próximas
   - Intervalo = [média - 2*desvio_padrão, média + 2*desvio_padrão]

4.3 Importância das Features

Com base na análise anterior do Random Forest: 1. Localização (Latitude/Longitude): ~45% 2. Características da estadia: ~24% 3. Tipo de acomodação: ~10% 4. Métricas de review/host: ~17%

5. Bairro específico: ~4%

4.4 Ajustes de Mercado

O modelo considera: - Localização premium em Manhattan - Tipo de propriedade (apartamento inteiro) - Alta disponibilidade (355 dias) - Experiência moderada do host (45 reviews)

Este método de precificação combina análise estatística com características específicas do mercado para sugerir um preço competitivo e justo para a propriedade.

[97]:
```
import joblib

# Salvar o modelo
joblib.dump(model, 'modelo_airbnb.pkl')
```

[97]:
```
['modelo_airbnb.pkl']
```

[ ]:
```

```