Práctica AirBnB - Bloque I - Aprendizaje Automático

Jorge Marcos Puñal

Link al proyecto: https://github.com/JotaEmme/AirBnB-Palma-study/blob/main/Airbnb.ipynb

In [1]:

```
# Librerías de datos
import pandas as pd
import numpy as np

# Librerías gráficas
import seaborn as sns
from matplotlib import pyplot as plt

# Librerías matemáticas
import math
import statistics

# Librerías para el modelo de ML
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor, to_graphviz
from sklearn.feature_selection import SelectKBest, f_regression

# Accuracy
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_err
from sklearn.metrics import balanced_accuracy_score, roc_auc_score, make_sc
from sklearn.model_selection import GridSearchCV # cross validation
```

In [2]:

```
# Carga del archivo
raw_data = pd.read_csv("airbnb.csv", header = 0, index_col = 0)
data = raw_data.copy()
```

Resumen del dataset

In [3]:

```
data.head(2)
```

Out[3]:

id	listing_url	scrape_id	last_scraped	name	des
11547	https://www.airbnb.com/rooms/11547	20200919153121	2020-09-21	My home at the beach	: pe
100831	https://www.airbnb.com/rooms/100831	20200919153121	2020-09-21	HOUSE IN MALLORCA - WiFi(ET-3045)	sp / sit

2 rows × 73 columns

In [4]:

```python
print("Dataset tiene {} filas y {} columnas.".format(*data.shape))
print("Column Information:", data.info())
```

```
Dataset tiene 17608 filas y 73 columnas.
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17608 entries, 11547 to 45499210
Data columns (total 73 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   listing_url                                   17608 non-null  object
 1   scrape_id                                     17608 non-null  int64
 2   last_scraped                                  17608 non-null  object
 3   name                                          17607 non-null  object
 4   description                                   17393 non-null  object
 5   neighborhood_overview                         8213 non-null   object
 6   picture_url                                   17608 non-null  object
 7   host_id                                       17608 non-null  int64
 8   host_url                                      17608 non-null  object
 9   host_name                                     17606 non-null  object
 10  host_since                                    17606 non-null  object
 11  host_location                                 17572 non-null  object
 12  host_about                                    11696 non-null  object
 13  host_response_time                            15862 non-null  object
 14  host_response_rate                            15862 non-null  object
 15  host_acceptance_rate                          16098 non-null  object
 16  host_is_superhost                             17606 non-null  object
 17  host_thumbnail_url                            17606 non-null  object
 18  host_picture_url                              17606 non-null  object
 19  host_neighbourhood                            364 non-null    object
 20  host_listings_count                           17606 non-null  float64
 21  host_total_listings_count                     17606 non-null  float64
 22  host_verifications                            17608 non-null  object
 23  host_has_profile_pic                          17606 non-null  object
 24  host_identity_verified                        17606 non-null  object
 25  neighbourhood                                 8213 non-null   object
 26  neighbourhood_cleansed                        17608 non-null  object
 27  neighbourhood_group_cleansed                  0 non-null      float64
 28  latitude                                      17608 non-null  float64
 29  longitude                                     17608 non-null  float64
 30  property_type                                 17608 non-null  object
 31  room_type                                     17608 non-null  object
 32  accommodates                                  17608 non-null  int64
```

```
 33  bathrooms                                      0 non-null      float64
 34  bathrooms_text                                 17600 non-null  object
 35  bedrooms                                       17333 non-null  float64
 36  beds                                           17511 non-null  float64
 37  amenities                                      17608 non-null  object
 38  price                                          17608 non-null  object
 39  minimum_nights                                 17608 non-null  int64
 40  maximum_nights                                 17608 non-null  int64
 41  minimum_minimum_nights                         17608 non-null  int64
 42  maximum_minimum_nights                         17608 non-null  int64
 43  minimum_maximum_nights                         17608 non-null  int64
 44  maximum_maximum_nights                         17608 non-null  int64
 45  minimum_nights_avg_ntm                         17608 non-null  float64
 46  maximum_nights_avg_ntm                         17608 non-null  float64
 47  calendar_updated                               0 non-null      float64
 48  has_availability                               17608 non-null  object
 49  availability_30                                17608 non-null  int64
 50  availability_60                                17608 non-null  int64
 51  availability_90                                17608 non-null  int64
 52  availability_365                               17608 non-null  int64
 53  calendar_last_scraped                          17608 non-null  object
 54  number_of_reviews                              17608 non-null  int64
 55  number_of_reviews_ltm                          17608 non-null  int64
 56  number_of_reviews_l30d                         17608 non-null  int64
 57  first_review                                   11173 non-null  object
 58  last_review                                    11173 non-null  object
 59  review_scores_rating                           10957 non-null  float64
 60  review_scores_accuracy                         10951 non-null  float64
 61  review_scores_cleanliness                      10953 non-null  float64
 62  review_scores_checkin                          10949 non-null  float64
 63  review_scores_communication                    10951 non-null  float64
 64  review_scores_location                         10950 non-null  float64
 65  review_scores_value                            10949 non-null  float64
 66  license                                        11431 non-null  object
 67  instant_bookable                               17608 non-null  object
 68  calculated_host_listings_count                 17608 non-null  int64
 69  calculated_host_listings_count_entire_homes    17608 non-null  int64
 70  calculated_host_listings_count_private_rooms   17608 non-null  int64
 71  calculated_host_listings_count_shared_rooms    17608 non-null  int64
 72  reviews_per_month                              11173 non-null  float64
dtypes: float64(19), int64(20), object(34)
memory usage: 9.9+ MB
Column Information: None
```

In [5]:

```
data.describe()
```

Out[5]:

	scrape_id	host_id	host_listings_count	host_total_listings_count	neighbo
count	1.760800e+04	1.760800e+04	17606.000000	17606.000000	
mean	2.020092e+13	1.012201e+08	130.200784	130.200784	
std	5.093895e+00	9.453596e+07	246.651175	246.651175	
min	2.020092e+13	4.294200e+04	0.000000	0.000000	
25%	2.020092e+13	2.116003e+07	2.000000	2.000000	
50%	2.020092e+13	8.063674e+07	15.000000	15.000000	
75%	2.020092e+13	1.516679e+08	111.000000	111.000000	
max	2.020092e+13	3.679802e+08	1136.000000	1136.000000	

8 rows × 39 columns

Entendimiento y limpieza del dataset

In [6]:

```
data.columns
```

Out[6]:

```
Index(['listing_url', 'scrape_id', 'last_scraped', 'name', 'description',
       'neighborhood_overview', 'picture_url', 'host_id', 'host_url',
       'host_name', 'host_since', 'host_location', 'host_about',
       'host_response_time', 'host_response_rate', 'host_acceptance_rate',
       'host_is_superhost', 'host_thumbnail_url', 'host_picture_url',
       'host_neighbourhood', 'host_listings_count',
       'host_total_listings_count', 'host_verifications',
       'host_has_profile_pic', 'host_identity_verified', 'neighbourhood',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'latitude'
,
       'longitude', 'property_type', 'room_type', 'accommodates', 'bathroom
s',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_review',
       'last_review', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month'],
      dtype='object')
```

Las columnas seleccionadas son las siguientes:

· latitude

· longitude

· accommodates

· bathrooms_text

· bedrooms

· availability_365

· number_of_reviews

· review_scores_value

· price

In [7]:

```
# Filtramos las columnas que se consideran subjetivamente relevantes
data = data[['latitude', 'longitude', 'accommodates', 'bathrooms_text', 'be
             'availability_365', 'number_of_reviews', 'review_scores_value'
```

In [8]:

```
# Renombramos
data = data.rename({'review_scores_value': 'reviews_value'}, axis=1)
```

In [9]:

```
# Clean numeric fields
data['bathrooms'] = data.bathrooms_text.str.extract('(\d+)')
data['bathrooms'] = data['bathrooms'].apply(pd.to_numeric, errors = 'coerce
data['price'] = data['price'].replace('[\$,]', '', regex = True)
data['price'] = data['price'].apply(pd.to_numeric, errors = 'coerce')
print("Dataset tiene {} filas y {} columnas.".format(*data.shape))
```

```
Dataset tiene 17608 filas y 10 columnas.
```

In [10]:

```
print("Shape: ", data.shape)
print("Column Information:", data.info())
```

```
Shape:  (17608, 10)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17608 entries, 11547 to 45499210
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   latitude           17608 non-null  float64
 1   longitude          17608 non-null  float64
 2   accommodates       17608 non-null  int64  
 3   bathrooms_text     17600 non-null  object 
 4   bedrooms           17333 non-null  float64
 5   availability_365   17608 non-null  int64  
 6   number_of_reviews  17608 non-null  int64  
 7   reviews_value      10949 non-null  float64
 8   price              17608 non-null  float64
 9   bathrooms          17585 non-null  float64
dtypes: float64(6), int64(3), object(1)
memory usage: 1.5+ MB
Column Information: None
```

In [11]:
```python
# Revisamos los NaN por columna
data.isnull().sum()
```

Out[11]:
```
latitude                0
longitude               0
accommodates            0
bathrooms_text          8
bedrooms              275
availability_365        0
number_of_reviews       0
reviews_value        6659
price                   0
bathrooms              23
dtype: int64
```

In [12]:
```python
del data['bathrooms_text']
```

In [13]:
```python
# Reemplazamos los NaN por 1 baño, 1 habitación y 0 reviews
data.fillna({'bathrooms': 1}, inplace = True)
data.fillna({'bedrooms': 1}, inplace = True)
data.fillna({'reviews_value': 0}, inplace = True)

# Hacemos double-check
data.isnull().sum()
```

Out[13]:
```
latitude             0
longitude            0
accommodates         0
bedrooms             0
availability_365     0
number_of_reviews    0
reviews_value        0
price                0
bathrooms            0
dtype: int64
```

In [14]:
```python
# Reordeno para dejar price la última columna
data = data[['latitude', 'longitude', 'accommodates', 'bathrooms', 'bedroom
             'availability_365', 'number_of_reviews', 'reviews_value', 'pri
```

Eliminando propiedades no disponibles todo el año

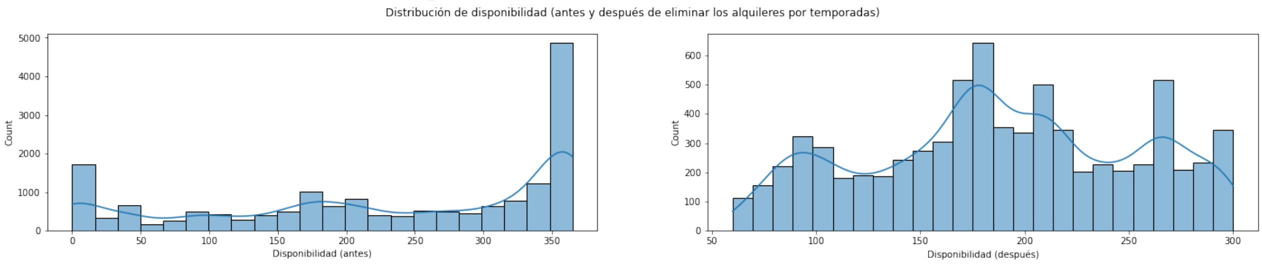
En AirBnB el hospedador puede configurar un calendario de disponibilidad, de tal manera que la propiedad estará disponible unos días o semanas al año. Otras propiedades se encuentran disponibles todo el año, excepto cuando están reservadas. Para una mayor consistencia de los datos se aplicará la definición de "alta disponibilidad" establecida por AirBnB y que se establece en >60 días al año. Dicho lo cual se eliminarán las propiedades con disponibilidad <60 días y también a los nuevos alquileres con disponibilidad >300 días, para así centrar la aproximación.

La curva muestra una distribución bi-modal en disponibilidad, demostrando dos agrupaciones en los tipos de alquileres. Los picos de disponibilidad por debajo de 50 días indican alquileres de corto plazo, mientras que en el pico sobre los 365 dias indican que son propiedades dedicadas íntegramente al alquiler. Una vez se han limpiado los datos, eliminare "availability_365" por su irrelevancia, tan solo revela los alquileres del próximo año.

In [15]:

```
# Distribución de availability_365
fig, axs = plt.subplots(ncols=2, figsize=(24, 4))
fig.suptitle('Distribución de disponibilidad (antes y después de eliminar l

# Antes de la limpieza
x_axis = data['availability_365'].dropna()
sns.histplot(pd.Series(x_axis, name='Disponibilidad (antes)'), kde=True, ax

# Eliminar 60 < disponibilidad < 300 días
data = data.query('60 <= availability_365 <= 300')
print("Dataset tiene {} filas y {} columnas.".format(*data.shape))

# Después de la limpieza
x_axis = data['availability_365'].dropna()
sns.histplot(pd.Series(x_axis, name='Disponibilidad (después)'), kde=True,
data = data.drop('availability_365', axis = 1)
print("Dataset tiene {} filas y {} columnas.".format(*data.shape))
```

```
Dataset tiene 7343 filas y 9 columnas.
Dataset tiene 7343 filas y 8 columnas.
```



Eliminando grandes propiedades

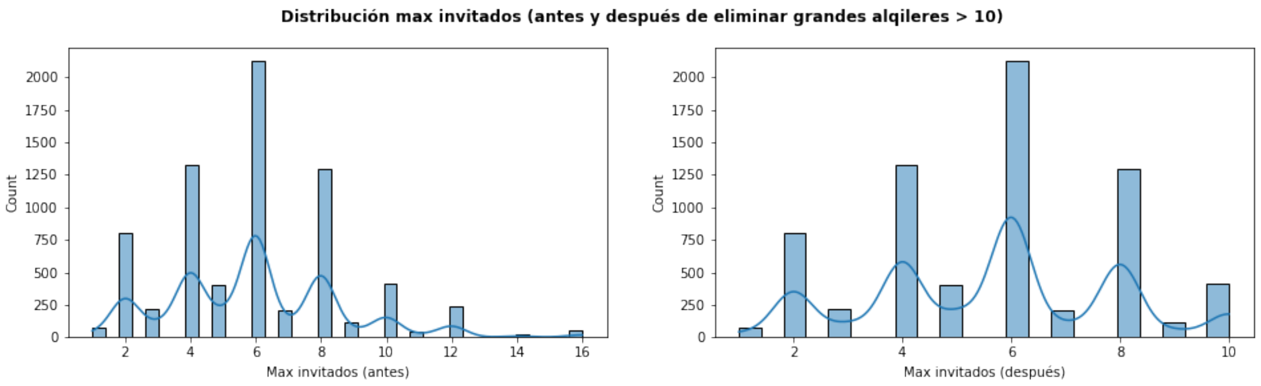
Para este modelo eliminaré las propiedades con capacidad determinada por >10 huéspedes.

In [16]:

```
# Distribución de huéspedes
fig, axs = plt.subplots(ncols=2, figsize=(16, 4))
fig.suptitle('Distribución max invitados (antes y después de eliminar granc
             weight='bold', fontsize=12)

# Antes de la limpieza
x_axis = data['accommodates'].dropna()
sns.histplot(pd.Series(x_axis, name='Max invitados (antes)'), kde=True, ax=

# Eliminar huéspedes > 10
condition = data[data['accommodates'] > 10]
rows_to_drop = condition.index
print("{} filas eliminadas.".format(condition.shape[0]))
data = data.drop(rows_to_drop, axis=0)
print("Dataset tiene {} filas y {} columnas.".format(*data.shape))

# Después de la limpieza
x_axis = data['accommodates'].dropna()
sns.histplot(pd.Series(x_axis, name='Max invitados (después)'), kde=True, a
```

```
364 filas eliminadas.
Dataset tiene 6979 filas y 8 columnas.
```

Out[16]: <AxesSubplot:xlabel='Max invitados (después)', ylabel='Count'>



Eliminando las propiedades más caras

La gráfica izquierda muestra una distribución right-skewed con un long tail, dado por las propiedades de mayor importe. Para este modelo eliminaré los alquileres por encima de los 400€/noche para mantener su comparabilidad.

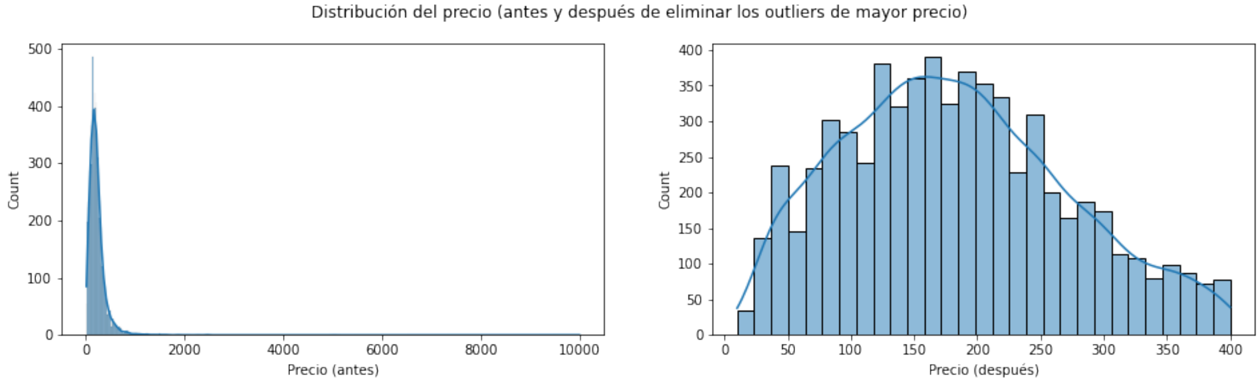
In [17]:

```python
# Distribución del precio
fig, axs = plt.subplots(ncols = 2, figsize = (16, 4))
fig.suptitle('Distribución del precio (antes y después de eliminar los outl

# Antes de la limpieza
x_axis = data['price'].dropna()
sns.histplot(pd.Series(x_axis, name = 'Precio (antes)'), kde = True, ax = a

# Eliminar precios > 400€/noche
condition = data[data['price'] > 400]
rows_to_drop = condition.index
print("{} filas eliminadas.".format(condition.shape[0]))
data = data.drop(rows_to_drop, axis = 0)
print("Dataset tiene {} filas y {} columnas.".format(*data.shape))

# Después de la limpieza
x_axis = data['price'].dropna()
sns.histplot(pd.Series(x_axis, name = 'Precio (después)'), kde = True, ax =
```

```
631 filas eliminadas.
Dataset tiene 6348 filas y 8 columnas.
```

Out[17]:

```
<AxesSubplot:xlabel='Precio (después)', ylabel='Count'>
```

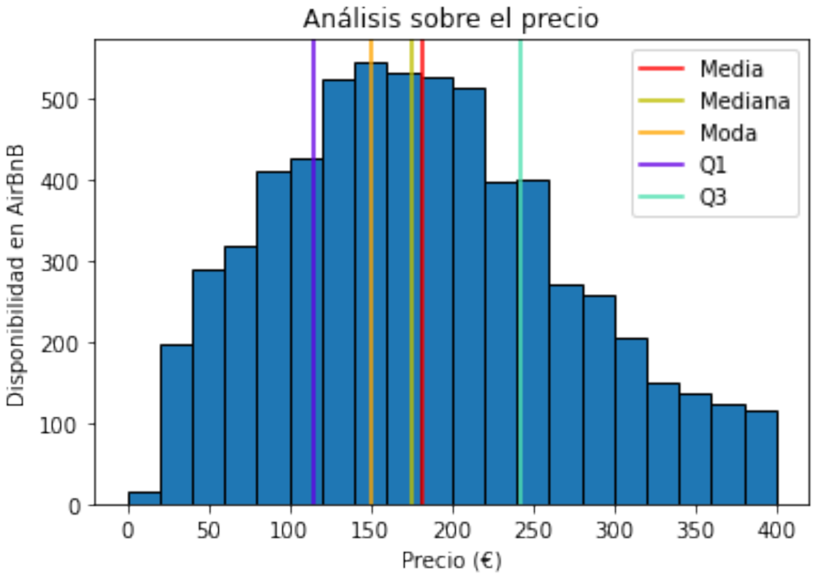


Estudio estadístico sobre precio

In [18]:

```python
# Media
average_price = round(np.mean(data['price']), 3)

# Mediana
median_price = np.median(data['price'])

# Moda
mode_price = statistics.mode(data['price'])

# Varianza
variance_price = round(np.var(data['price']), 3)

# Desviación estandar
stdev_price = round(np.std(data['price']), 3)

# Primer cuartil
q1 = np.percentile(data.price, 25)

# Tercer cuartil
q3 = np.percentile(data.price, 75)

print('Media:\t\t\t', average_price)
print('Mediana:\t\t', median_price)
print('Moda:\t\t\t', mode_price)
print('Varianza:\t\t', variance_price)
print('Desviación estandar:\t', stdev_price)
print('Primer cuartil:\t\t', q1)
print('Tercer cuartil:\t\t', q3)

# Plot hystogram
plt.hist(data.price, range = (0, 400), bins = 20, edgecolor = 'black')
plt.title("Análisis sobre el precio")
plt.xlabel('Precio (€)')
plt.ylabel('Disponibilidad en AirBnB')
plt.axvline(average_price, color = 'r', linewidth=1.5, label = "Media")
plt.axvline(median_price, color = 'y', linewidth=1.5, label = "Mediana")
plt.axvline(mode_price, color = 'orange', linewidth=1.5, label = "Moda")
plt.axvline(q1, color = '#6400e4', linewidth=1.5, label="Q1")
plt.axvline(q3, color = '#4fe0b0', linewidth=1.5, label="Q3")
plt.legend()

plt.show()
```

```
Media:                    182.286
Mediana:                  175.0
Moda:                     150.0
Varianza:                 7861.528
Desviación estandar:      88.665
Primer cuartil:           115.0
Tercer cuartil:           242.355
```



Representación sobre plano

· Mapa de calor: Representa zonas calientes por densidad de oferta.

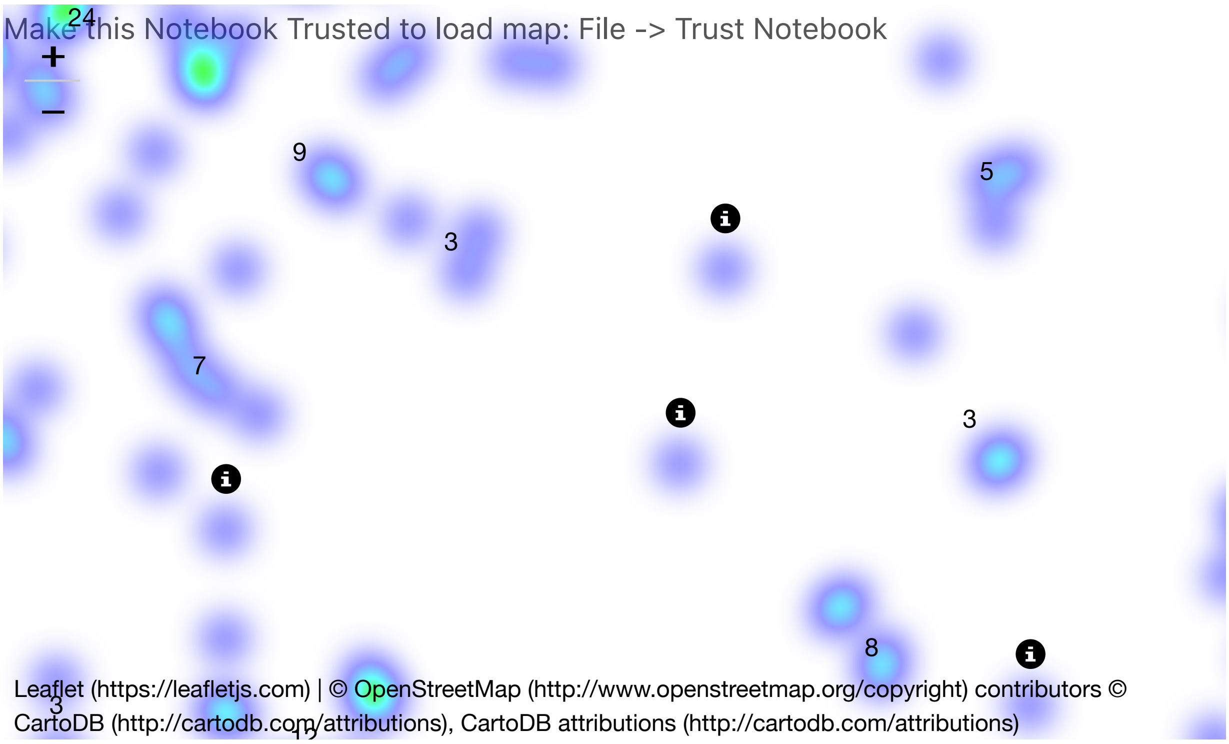
· Clustering: Densidad de ofertas publicadas.

· Código de colores: Representando diferentes precios.

In [19]:

```python
import folium
from folium.plugins import MarkerCluster, HeatMap

init_lat = data['latitude'].mean()
init_long = data['longitude'].mean()

m = folium.Map(
    tiles='CartoDB Dark_Matter',
    location=[init_lat, init_long],
    zoom_start=13,
    max_bounds=True
    )
marker_cluster = MarkerCluster().add_to(m)
for index, row in data.iterrows():
    lat_air = row.latitude
    long_air = row.longitude
    price = row.price
    if (price > 0 ) & (price < 30):
        folium.Marker([lat_air, long_air],
                      tooltip=price,
                      icon=folium.Icon(color='green'),
                      clustered_marker = True
                        ).add_to(marker_cluster)
    elif (price < 50) & (price >= 30):
        folium.Marker([lat_air, long_air],
                    clustered_marker = True,
                    tooltip=price,
                    icon=folium.Icon(color='orange')
                 ).add_to(marker_cluster)
    elif (price < 90) & (price >= 50):
        folium.Marker([lat_air, long_air],
                    clustered_marker = True,
                    tooltip=price,
                    icon=folium.Icon(color='lightred')
                 ).add_to(marker_cluster)
    else:
        folium.Marker([lat_air, long_air],
                    clustered_marker = True,
                    tooltip=price,
                    icon=folium.Icon(color='red')
                 ).add_to(marker_cluster)

# convert to (n, 2) nd-array format for heatmap
privateArr = data[['latitude', 'longitude']].values

# plot heatmap
m.add_child(HeatMap(privateArr, radius=12))
```

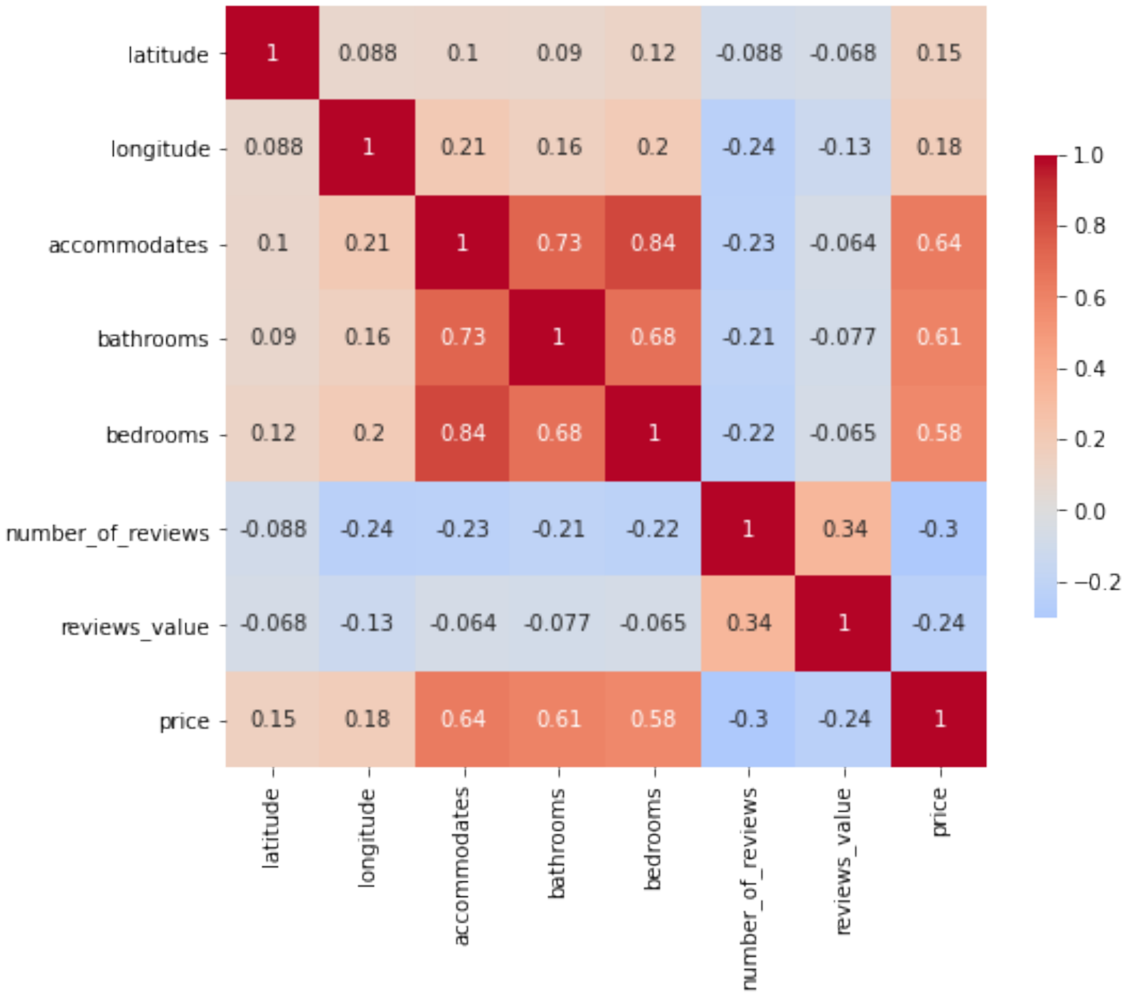
Out[19]:



In [20]:

```
# Crea matriz de correlación
_, ax = plt.subplots(figsize=(8, 8))

# Ploteo con Seaborn
sns.heatmap(data.corr(), cmap='coolwarm', center=0, square=True, cbar_kws=
```

Out[20]: <AxesSubplot:>

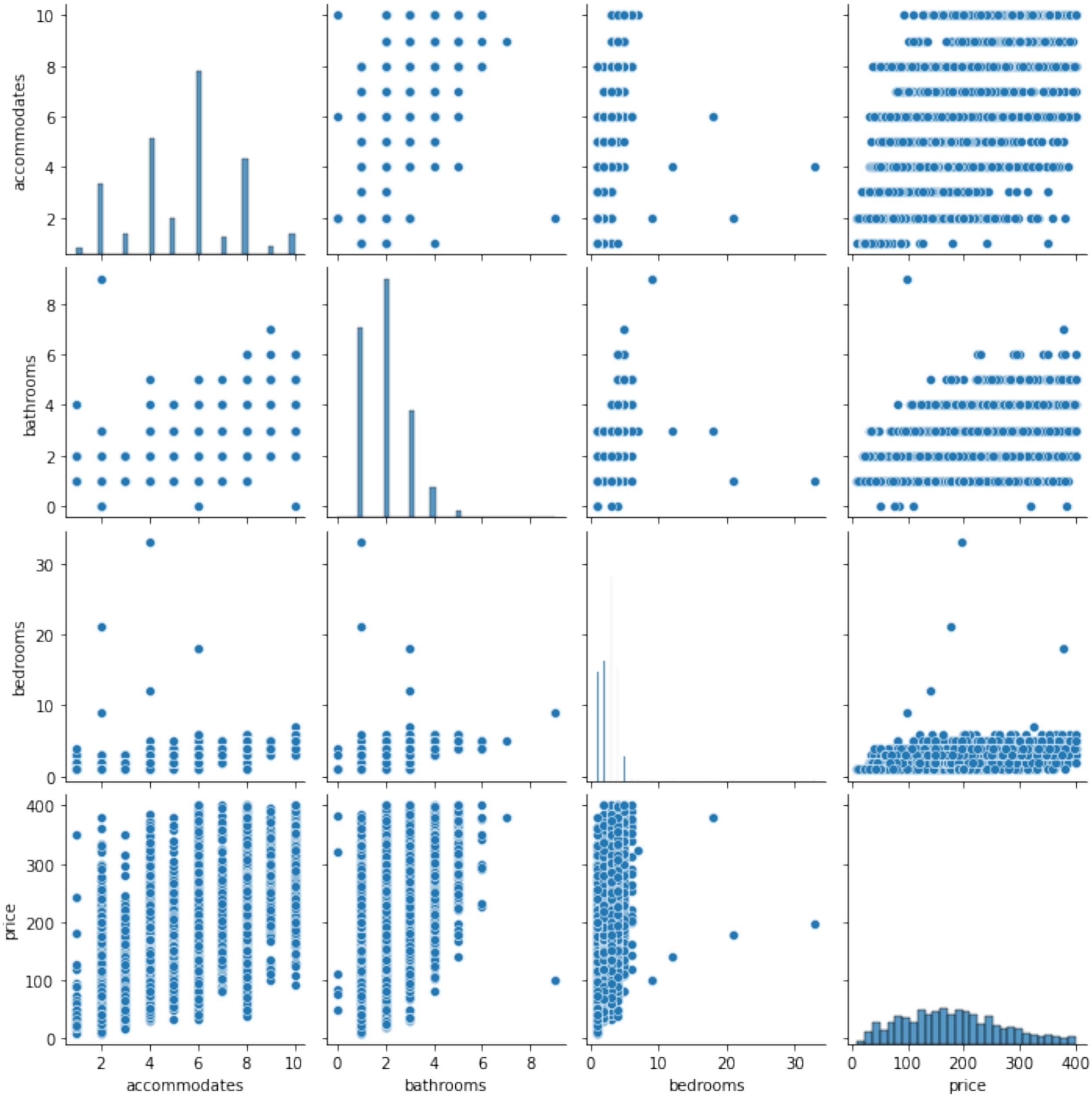


In [21]:

```
# Filtra aquellos valores que tengan un 0.5 > abs > 1, los fusiona y elimi
highly_corr = data.corr()[(np.abs(data.corr()) > 0.5) & (np.abs(data.corr(

# Saco en pantalla los valores únicos
print(highly_corr['variable'].unique())

# Ploteo con Seaborn
sns.pairplot(data[highly_corr['variable'].unique()])
```

['accommodates' 'bathrooms' 'bedrooms' 'price']

Out[21]: <seaborn.axisgrid.PairGrid at 0x7ffe7eb25340>



In [22]:

```
# Datos finales con los que trabajaremos el modelo
data
```

Out[22]:

id	latitude	longitude	accommodates	bathrooms	bedrooms	number_of_reviews
159218	39.73689	2.89745	3	1.0	1.0	268
160482	39.65414	2.62546	6	2.0	3.0	62
166820	39.67932	2.50136	2	1.0	1.0	159
193426	39.89730	3.07411	6	2.0	3.0	82
210156	39.83375	3.10918	6	2.0	3.0	92
...	...	...	...	...	...	...
45459698	39.84032	3.12234	5	1.0	3.0	0
45475973	39.55374	2.61833	2	1.0	1.0	0
45478905	39.58170	2.67153	2	1.0	1.0	0
45493152	39.75437	2.90504	6	2.0	3.0	0
45496032	39.54550	2.39348	2	1.0	1.0	0

6348 rows × 8 columns

Preparación del modelo

In [23]:

```
# Preparo los datos para el modelo
Xraw = data.iloc[:, 0:7]
y = data.iloc[:, 7]
```

In [24]:

```
# Función para el cálculo del error
def RMSE(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

Random Forest

Planteo un modelo de regresión basado en el algoritmo Random Forest, bajo el criterio de que tenemos muchas características a ser evaluadas y este modelo es bueno trabajando con variables independientes.

Selección de filtros

Dado que el número de variables a analizar es considerable, podemos intentar ajustarlas todas o buscar y ajustar las más relevantes. Para hacer esto último podemos seleccionar las mejores fetures basadas en test estadísticos, eliminando todo excepto las de mayor resultado k.

In [25]:

```
results = []
count = 0
minim_abs = 99999999999
nmax = 9

N = Xraw.shape[1]
print('|----|----------------------|')
print('| k  |        RMSE          |')
for i in range (1,N):
    X = Xraw.copy()
    X = SelectKBest(f_regression,k=i).fit_transform(X,y)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
    random_forest_regression = RandomForestRegressor()
    random_forest_regression.fit(X_train,y_train)
    y_test_pred = random_forest_regression.predict(X_test)
    results.append((mean_squared_error(y_test,y_test_pred))**(1/2))
    print('|----|----------------------|')
    print('|',i,'  | ',(mean_squared_error(y_test,y_test_pred))**(1/2),'|')
    #Exit at nmax items not decreasing
    if len(results) != 0:
        if (mean_squared_error(y_test,y_test_pred))**(1/2) > minim_abs:
            count += 1
        else:
            minim_abs = (mean_squared_error(y_test,y_test_pred))**(1/2)
            count = 0
    else:
        count = 0
    if count > nmax:
        print('|----|----------------------|')
        print('| breaking after nmax items|')
        break
print('|----|----------------------|')

error_RF = RMSE(y_test, y_test_pred)
```

```
|----|----------------------|
| k  |        RMSE          |
|----|----------------------|
| 1  |  68.30887410266877 |
|----|----------------------|
| 2  |  64.95680837685326 |
|----|----------------------|
| 3  |  64.69709491762613 |
|----|----------------------|
| 4  |  66.03296757253719 |
|----|----------------------|
| 5  |  66.455409681193 |
|----|----------------------|
| 6  |  65.5799570786101 |
|----|----------------------|
```

In [26]:

```
k = results.index(min(results)) + 1
print('El mínimo elemento en la lista es', k, 'y es', min(results))
```

```
El mínimo elemento en la lista es 3 y es 64.69709491762613
```

Model pre-filtered and score

In [27]:

```
N_ESTIM = 1000
X = Xraw
selection = SelectKBest(f_regression,k=k)
X = selection.fit_transform(X,y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, r
random_forest_regression = RandomForestRegressor(n_jobs=2,n_estimators=N_ES
random_forest_regression.fit(X_train,y_train)
y_test_predict = random_forest_regression.predict(X_test)
y_train_predict = random_forest_regression.predict(X_train)
print('Valor de K: ', k)
print('Número de Random Forests: ',N_ESTIM)
rf_results=pd.DataFrame({'algorithm':['Random Forest'],
                         'Training error': [mean_absolute_error(y_train,y_t
                         'Test error':[mean_absolute_error(y_test,y_test_pr
                         'Train_r2_score':[r2_score(y_train,y_train_predict
                         'Test_r2_Score':[r2_score(y_test,y_test_predict)]]
print(rf_results)
```

```
Valor de K:   3
Número de Random Forests:  1000
       algorithm  Training error  Test error  Train_r2_score  Test_r2_Score
0  Random Forest       50.190713   51.066497        0.477261       0.484015
```

GridSearch para RandomForest

In [28]:

```
#Grid model object
rf_grid = RandomForestRegressor()

param_grid = {'n_estimators':[1000],
              'max_leaf_nodes':[3,6,9],
              'ccp_alpha':[0.9,0.8,0.7]}

optimal_params = GridSearchCV(estimator = rf_grid,
                              param_grid=param_grid,
                              verbose=2,
                              n_jobs = 3,
                              cv = 3)

optimal_params.fit(X_train, y_train)

print(optimal_params.best_score_)
print(optimal_params.best_params_)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done  27 out of  27 | elapsed:   41.7s finished
0.4536852609588826
{'ccp_alpha': 0.7, 'max_leaf_nodes': 9, 'n_estimators': 1000}
```

Decision Tree

In [29]:

```
# Decision Tree
X = Xraw
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, r
regtree = DecisionTreeRegressor(min_samples_split=2, min_samples_leaf=4, ma
dt = regtree.fit(X_train,y_train)
train_predict=dt.predict(X_train)
test_predict=dt.predict(X_test)
dt_results=pd.DataFrame({'algorithm':['Decision Tree'],
                         'Training error': [mean_absolute_error(y_train,tra
                         'Test error':[mean_absolute_error(y_test,test_pred
                         'Train_r2_score':[r2_score(y_train,train_predict)]
                         'Test_r2_Score':[r2_score(y_test,test_predict)]})
print(dt_results)

error_DeTr = RMSE(y_test, test_predict)
```

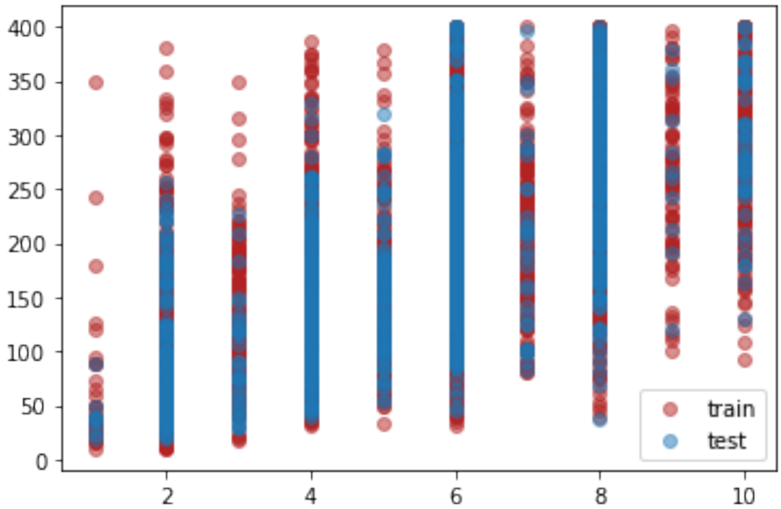
```
       algorithm  Training error  Test error  Train_r2_score  Test_r2_Score
0  Decision Tree       43.758183   49.451333        0.584901       0.500202
```

Ejemplo de representación de train y test entre las variables accommodates y price

In [30]:

```
plt.scatter(X_train['accommodates'], y_train, label='train', c='firebrick',
plt.scatter(X_test['accommodates'], y_test, label='test', alpha=0.5)
plt.legend()
```

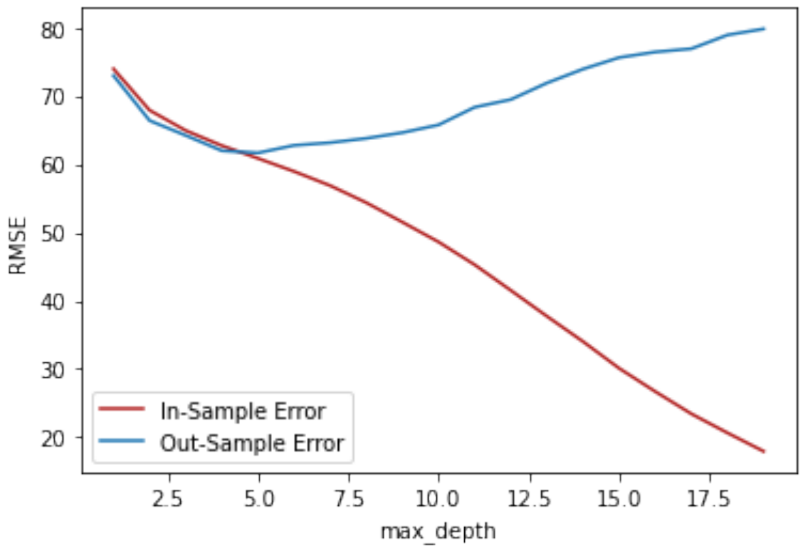
Out[30]: <matplotlib.legend.Legend at 0x7ffe7f603fd0>



Al aplicar decission tree, se puede jugar con tanto con su profundidad como con su número de hojas. Comparamos gráficamente como evoluciona el RMSE en función de la profundidad del arbol.

In [31]:
```
max_depths = range(1, 20)
in_sample_errors = []
out_sample_errors = []
for max_depth in max_depths:
    tree = DecisionTreeRegressor(max_depth=max_depth).fit(X_train, y_train)
    y_pred_train = tree.predict(X_train)
    y_pred_test = tree.predict(X_test)
    in_sample_errors.append(RMSE(y_train, y_pred_train))
    out_sample_errors.append(RMSE(y_test, y_pred_test))

plt.plot(max_depths, in_sample_errors, c='firebrick', label='In-Sample Erro
plt.plot(max_depths, out_sample_errors, label='Out-Sample Error')

plt.xlabel('max_depth')
plt.ylabel('RMSE')
plt.legend(loc='best')
```

Out[31]: <matplotlib.legend.Legend at 0x7ffe7f42d0a0>



XGBoost

In [32]:
```
xgb_round1 = XGBRegressor()
```

In [33]:
```
param_grid = {
            'learn_rate':[0.1,0.05,0.01],
            'min_child_weight':[3,4,5],
            'gamma':[i/10.0 for i in range(3,6)],
            'subsample':[i/10.0 for i in range(6,11)],
            'colsample_bytree':[i/10.0 for i in range(6,11)],
            'max_depth': [9,10,11],
            'n_estimators':[300,500,1000]
}
```

In [34]:

```
optimal_params = GridSearchCV(
                estimator = xgb_round1,
                param_grid=param_grid,
                verbose=2, # NOTE: If you want to see what Grid Search is doi
                n_jobs = 2,
                cv = 3
               )

optimal_params.fit(X_train, y_train, early_stopping_rounds=10, eval_set=[(X

print(optimal_params.best_score_)
print(optimal_params.best_params_)
```

```
Fitting 3 folds for each of 6075 candidates, totalling 18225 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  37 tasks      | elapsed:    8.9s
[Parallel(n_jobs=2)]: Done 158 tasks      | elapsed:   28.0s
[Parallel(n_jobs=2)]: Done 361 tasks      | elapsed:  1.0min
[Parallel(n_jobs=2)]: Done 644 tasks      | elapsed:  1.9min
[Parallel(n_jobs=2)]: Done 1009 tasks      | elapsed:  2.9min
[Parallel(n_jobs=2)]: Done 1454 tasks      | elapsed:  4.1min
[Parallel(n_jobs=2)]: Done 1981 tasks      | elapsed:  5.6min
[Parallel(n_jobs=2)]: Done 2588 tasks      | elapsed:  7.3min
[Parallel(n_jobs=2)]: Done 3277 tasks      | elapsed:  9.1min
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed: 11.1min
[Parallel(n_jobs=2)]: Done 4897 tasks      | elapsed: 13.4min
[Parallel(n_jobs=2)]: Done 5828 tasks      | elapsed: 15.8min
[Parallel(n_jobs=2)]: Done 6841 tasks      | elapsed: 18.8min
[Parallel(n_jobs=2)]: Done 7934 tasks      | elapsed: 22.0min
[Parallel(n_jobs=2)]: Done 9109 tasks      | elapsed: 25.3min
[Parallel(n_jobs=2)]: Done 10364 tasks      | elapsed: 28.9min
[Parallel(n_jobs=2)]: Done 11701 tasks      | elapsed: 33.0min
[Parallel(n_jobs=2)]: Done 13118 tasks      | elapsed: 37.4min
[Parallel(n_jobs=2)]: Done 14617 tasks      | elapsed: 42.0min
[Parallel(n_jobs=2)]: Done 16196 tasks      | elapsed: 47.1min
[Parallel(n_jobs=2)]: Done 17857 tasks      | elapsed: 52.4min
[Parallel(n_jobs=2)]: Done 18225 out of 18225 | elapsed: 53.5min finished
```

```
[19:49:04] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:541:
Parameters: { learn_rate } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip th
rough this
  verification. Please open an issue if you find above cases.


[0]	validation_0-rmse:151.62137
[1]	validation_0-rmse:115.65456
[2]	validation_0-rmse:92.50348
[3]	validation_0-rmse:78.45510
[4]	validation_0-rmse:70.22861
[5]	validation_0-rmse:65.54177
[6]	validation_0-rmse:62.93241
[7]	validation_0-rmse:61.63047
[8]	validation_0-rmse:60.77943
[9]	validation_0-rmse:60.30464
[10]	validation_0-rmse:60.11770
[11]	validation_0-rmse:60.00397
[12]	validation_0-rmse:59.98875
[13]	validation_0-rmse:60.02403
[14]	validation_0-rmse:60.03123
[15]	validation_0-rmse:60.02966
[16]	validation_0-rmse:60.13486
[17]	validation_0-rmse:60.19898
[18]	validation_0-rmse:60.25098
[19]	validation_0-rmse:60.16614
[20]	validation_0-rmse:60.22897
[21]	validation_0-rmse:60.21773
0.5233638198362006
{'colsample_bytree': 1.0, 'gamma': 0.3, 'learn_rate': 0.1, 'max_depth': 9,
'min_child_weight': 5, 'n_estimators': 300, 'subsample': 1.0}
```

Final XGBoost model after parameters

In [35]:

```python
# Final model after adjusting parameters.
xg = XGBRegressor(seed=42,
                  objective='reg:squarederror',
                  gamma=0.3,
                  learn_rate=0.1,
                  max_depth=9,
                  min_child_weight=5,
                  subsample=1,
                  colsample_bytree=1,
                  n_estimators=300)

xg.fit(X_train, y_train,
       verbose=False,
       early_stopping_rounds=10,
       eval_set=[(X_test, y_test)])
```

```
[19:49:05] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:541:
Parameters: { learn_rate } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip th
rough this
  verification. Please open an issue if you find above cases.
```

Out[35]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.6, gamma=0.3, gpu_id=-1
,
             importance_type='gain', interaction_constraints='', learn_rate
=0.1,
             learning_rate=0.300000012, max_delta_step=0, max_depth=9,
             min_child_weight=5, missing=nan, monotone_constraints='()',
             n_estimators=300, n_jobs=4, num_parallel_tree=1, random_state=
42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=42,
             subsample=1, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

In [36]:

```python
# XGB Regressor
train_predict=xg.predict(X_train)
test_predict=xg.predict(X_test)
xg_results=pd.DataFrame({'algorithm':['XGBoost'],
                         'Training error': [mean_absolute_error(y_train, tr
                         'Test error':[mean_absolute_error(y_test, test_pre
                         'Train_r2_score':[r2_score(y_train, train_predict)
                         'Test_r2_Score':[r2_score(y_test, test_predict)]})
print(xg_results)

error_XGB = RMSE(y_test, test_predict)
```

```
  algorithm  Training error  Test error  Train_r2_score  Test_r2_Score
0   XGBoost       30.945974   46.334156        0.782058       0.557431
```

In [37]:

```python
pd.concat([rf_results, dt_results, xg_results], axis=0, ignore_index=True)
```

Out[37]:

	algorithm	Training error	Test error	Train_r2_score	Test_r2_Score
0	Random Forest	50.190713	51.066497	0.477261	0.484015
1	Decision Tree	43.758183	49.451333	0.584901	0.500202
2	XGBoost	30.945974	46.334156	0.782058	0.557431

In [38]:

```python
print('Los errores con los distintos tipos de modelos optimizados son los s
print('Error del modelo RF: {e} k$'.format(e=round(error_RF, 2)))
print('Error medio RF: {e} $'.format(e=round((error_RF/len(X_test))*1000, 2
print('\nError del modelo DT: {e} k$'.format(e=round(error_DeTr, 2)))
print('Error medio DT: {e} $'.format(e=round((error_DeTr/len(X_test))*1000,
print('\nError del modelo XGBoost: {e} k$'.format(e=round(error_XGB, 2)))
print('Error medio XGBoost: {e} $'.format(e=round((error_XGB/len(X_test))*1
```

```
Los errores con los distintos tipos de modelos optimizados son los siguient
es:

Error del modelo RF: 65.58 k$
Error medio RF: 51.64 $

Error del modelo DT: 62.91 k$
Error medio DT: 49.53 $

Error del modelo XGBoost: 59.2 k$
Error medio XGBoost: 46.61 $
```

Conclusiones

El mejor modelo encontrado haciendo uso de diferentes técnicas es el XGBoost, con un error medio de 46,61$.

Es realmente complejo estimar el valor de una propiedad basándonos en determinadas variables, por ejemplo, fueron descartados los ammenities debido a que incluían elementos tan dispares como piscina y tostador o parking y secador del pelo, con lo que la manera más exhaustiva sería asignar un valor a cada ammenitie, pero complicaría en exceso esta tarea.

Es interesante cómo al implementar los algoritmos y al optimizar sus hiperparámetros se ha disminuido el tiempo de procesamiento de 81 minutos a unos 53 minutos.