Data_cleaning_and_exploring

February 15, 2023

[13]:
```
# Importamos las librerías necesarias
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
```

[14]:
```
# Cargamos el fichero original
raw_file = '../airbnb-listings.csv'
df = pd.read_csv(raw_file, delimiter=";", low_memory=False)
```

1 Proyecto Final para el Bootcamp Mujeres en Tech - KeepCoding

---

En este proyecto vamos a explorar el *dataset* propuesto y a responder algunas preguntas que nos hemos planteado. Trataremos con datos de la plataforma de Airbnb e intentaremos averiguar qué características tienen mayor impacto en el precio de las propiedades ofrecidas en alquiler.

Para ello, hemos seguido los siguientes pasos.

1.1 Muestreo y exploración inicial de los datos

1.1.1 ¿Con qué datos trabajaremos?

Hemos decidido trabajar con los datos correspondientes a la *Comunidad de Madrid*, por lo que procederemos a quedarnos con esos datos concretos y descartar el resto de las filas. Posteriormente, vamos a filtrar las columnas que creemos que nos serán útiles.

[15]:
```
# Seleccionamos las filas correspondientes a la Comunidad de Madrid y las␣
 ↪columnas con las que vamos a trabajar
rows_to_keep = df['State'].astype(str).str.contains('Madrid')
```

```
columns_to_keep = ['ID', 'Host ID', 'Host Since', 'Neighbourhood',␣
 ↪'Neighbourhood Cleansed', 'City', 'State', 'Zipcode', 'Latitude',␣
 ↪'Longitude', 'Amenities', 'Property Type', 'Room Type', 'Bathrooms',␣
 ↪'Bedrooms', 'Beds', 'Bed Type', 'Square Feet', 'Cleaning Fee', 'Availability␣
 ↪365', 'Review Scores Location', 'Cancellation Policy', 'Accommodates',␣
 ↪'Reviews per Month', 'Minimum Nights', 'Price', 'Monthly Price', 'Weekly␣
 ↪Price']
df = df.loc[rows_to_keep, columns_to_keep]
```

A continuación, visualizamos las columnas elegidas.

[16]:
```
print(df.columns)
```

```
Index(['ID', 'Host ID', 'Host Since', 'Neighbourhood',
       'Neighbourhood Cleansed', 'City', 'State', 'Zipcode', 'Latitude',
       'Longitude', 'Amenities', 'Property Type', 'Room Type', 'Bathrooms',
       'Bedrooms', 'Beds', 'Bed Type', 'Square Feet', 'Cleaning Fee',
       'Availability 365', 'Review Scores Location', 'Cancellation Policy',
       'Accommodates', 'Reviews per Month', 'Minimum Nights', 'Price',
       'Monthly Price', 'Weekly Price'],
      dtype='object')
```

1.2 Normalización de las columnas

Observamos que algunas columnas tienen datos que no estan normalizados, por ejemplo, el código postal y el barrio, así que procedemos a analizarlas y normalizarlas para un mejor tratamiento del dato.

[17]:
```
# Normalizamos los valores de la columna 'Zipcode' que contiene códigos␣
 ↪postales erróneos y valores nulos
zp_normalization = {'nan':np.nan, '-':np.nan, '28':np.nan, '2802\n28012':
 ↪'28012', '28002\n28002':'28002', '28051\n28051':'28051', 'Madrid 28004':
 ↪'28004', '2815':'28015', '2805':'28005', '20126':np.nan, '2804':'28004',␣
 ↪'27013':'28013', '2015':'28015', '27004':'28004', '25008':'28008', '20013':
 ↪'28013', '280013':'28013'}
df = df.replace({'Zipcode': zp_normalization})
```

En el caso de la columna Neighbourhood, esta tenía muchos valores nulos, por lo que decidimos rellenar los nulos con los encontrados en la columna Neighbourhood Cleansed y luego descartar esta última columna.

[18]:
```
# Cambiamos los valores nulos de la columna 'Neighbourhood' por el valor␣
 ↪correspondiente de la columna 'Neighbourhood Cleansed', eliminamos esta␣
 ↪última
df['Neighbourhood'] = df['Neighbourhood'].fillna(df['Neighbourhood Cleansed'])
df = df.drop('Neighbourhood Cleansed', axis = 1)

# Comprobamos que no quedan valores nulos
df['Neighbourhood'].isna().value_counts()
```

[18]:
```
False    13198
Name: Neighbourhood, dtype: int64
```

1.2.1 Columnas de texto

En el caso de las columnas de texto, es necesario normalizar los caracteres que pueden dar problemas de codificación y buscar errores de tipografía o estilo. Decidimos analizar las columnas que son cadenas de texto para quitar tildes, dobles espacios, etc.

[19]:
```
# Analizamos qué columnas necesitan normalización textual
for column in df.columns:
    if df[column].dtype == object:
        print(column)

# Variable con dichas columnas
str_columns = ['Neighbourhood', 'City', 'State', 'Property Type', 'Room Type',␣
 ↪'Bed Type', 'Cancellation Policy']

# Analizamos los caracteres no alfabéticos que contienen dichas columnas para␣
 ↪decidir cuáles conservar y cuáles eliminar
for column in df.columns:
    if column in str_columns:
        temp_df = df[df[column].astype(str).str.contains('-|#|\.|,|;|:|_|&|/
 ↪|Ç|\'|%|[|]|{|}|=|\?|¿|\(|\)|¡|!|\$|`\´|>|<|\||ª|º\\|@|\d|  ')]
        print(temp_df[column].unique())
```

```
ID
Host Since
Neighbourhood
City
State
Zipcode
Amenities
Property Type
Room Type
Bed Type
Cancellation Policy
['Fuencarral-el Pardo' 'Fuencarral-El Pardo']
['Delicias-Madrid' 'Madrid, Comunidad de Madrid, ES' 'Centro, Madrid'
 'las matas  madrid' 'Madrid, Comunidad de Madrid, ESPANA'
 'Madrid, Vallecas (Fontarrón)' 'Aravaca (Madrid)' 'Chueca, Madrid']
['Madrid, Spain' 'España,Madrid']
['Bed & Breakfast' 'Camper/RV']
['Entire home/apt']
['Pull-out Sofa']
['super_strict_60' 'super_strict_30']
```

[20]:
```
# Función para eliminar los caracteres no alfa-numéricos y los dobles espacios
def no_alfa_num(text):
    characters = '-|_|\(|\)|  '
    for character in text:
        match = re.search(characters, text)
        if match:
            text = text.replace(match.group(0), ' ')
    return text

# Función para normalizar tildes y eñes
def normalize(text):
    characters = (('á', 'a'), ('é', 'e'), ('í', 'i'), ('ó', 'o'), ('ú', 'u'),␣
 ↪('ñ', 'n'))
    for a, b in characters:
        text = text.replace(a, b).replace(a.upper(), b.upper())
    return text

# Normalización de las columnas de texto
for column in df.columns:
    if column in str_columns:
        column_normalized = list(map(normalize, list(map(no_alfa_num,␣
 ↪df[column].astype(str)))))
        df[column] = column_normalized

df[str_columns].head(10)
```

[20]:

	Neighbourhood	City	State	Property Type	Room Type	Bed Type	Cancellation Policy
1021	Embajadores	Madrid	Comunidad de Madrid	Loft	Entire home/apt	Real Bed	moderate
1022	Embajadores	Madrid	Comunidad de Madrid	Apartment	Entire home/apt	Real Bed	strict
1023	Embajadores	Madrid	Comunidad de Madrid	Apartment	Entire home/apt	Real Bed	moderate
1024	Embajadores	Madrid	Community of Madrid	Apartment	Entire home/apt	Real Bed	strict
1025	Embajadores	Madrid	Comunidad de Madrid	Apartment	Entire home/apt	Real Bed	strict
1026	La Latina	Madrid	Community of Madrid	Apartment	Entire home/apt	Real Bed	moderate
1027	Embajadores	Madrid	Community of Madrid	Apartment	Entire home/apt	Real Bed	moderate
1028	La Latina	Madrid	Comunidad de Madrid	Apartment	Private room	Real Bed	moderate
1029	Embajadores	Madrid	Comunidad de Madrid	Apartment			
1030	La Latina	Madrid	Community of Madrid	Apartment			

```
1029  Entire home/apt  Real Bed            moderate
1030  Entire home/apt  Real Bed            moderate
```

1.2.2 Conversión de tipos de datos

A continuación, observamos también que algunos tipos datos eran incorrectos para su manejo, por lo que también hicimos las correcciones pertinentes.

- Convertimos la columna Host_Since a tipo fecha:

[24]:
```
# Conversión de las fechas de 'Host Since' en date
df['Host Since'] = pd.to_datetime(df['Host Since'])
```

- Convertimos la columna ID a tipo numérico (en vez de string).

[25]:
```
# Conversión del ID en numérico
df['ID'] = df['ID'].astype(int)
```

- El dataset scrapeado tenia una columna en pies cudrados, decidimos convertirla a metros cuadrados para tener una mejor comprensión de la métrica.

[26]:
```
# Conversión de Square Feet en metros cuadrados
df['Square Meters'] = df['Square Feet'] / 10.764
```

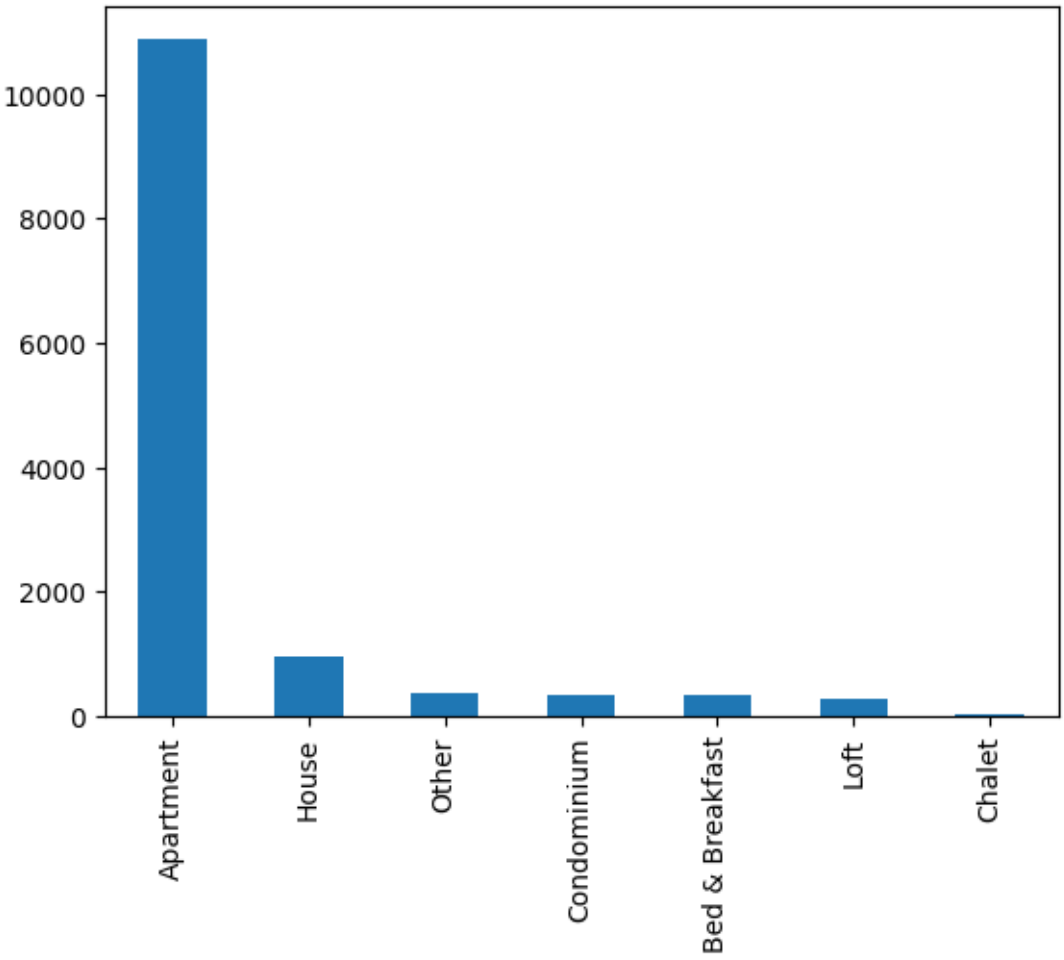
1.2.3 Property Type

En el caso de la columna Property Type, notamos que había demasiadas clasificaciones con muy pocos registros, así que decidimos quedarnos con las más relevantes y agrupar el resto de tipos de propiedades como "Otros".

Al ejecutar la siguiente celda, visualizamos un gráfico donde podemos observar, claramente, que la mayoría de las propiedades ofrecidas son *apartamentos*.

[27]:
```
# Agrupamos los valores de la columna 'Property Type' por los más comunes y␣
 ↪almacenamos el resto en el valor 'Other'
valid_property_types = ['House', 'Apartment', 'Bed & Breakfast', 'Condominium',␣
 ↪'Loft', 'Chalet', 'Hostal']
property_types = df['Property Type']
property_types = property_types.map(lambda value: value if value in␣
 ↪valid_property_types else 'Other')
df['Property Type'] = property_types

# Visualizamos el resultado
df['Property Type'].value_counts().plot.bar()
```

[27]: <AxesSubplot: >



1.2.4 Amenities

En la columna de Amenities es donde basamos nuestras principales dudas sobre el dataset.

Estábamos interesadas en saber si había algún servicio que influyese notablemente en el precio o si el número total de servicios tenía algún impacto real en el mismo, así que decidimos analizarla en profundidad.

En primer lugar, decidimos revisar cuáles eran los servicios más frecuentes, cuál era el precio medio de los alojamientos y la diferencia del precio medio en función del servicio.

1. Como primer punto, creamos un dataframe llamado "Amenities" que contiene cada servicio como columna.

[28]:
```
df_amenities = df.Amenities.fillna('').str.get_dummies(sep=',').astype(bool)
df_amenities.head(10)
```

[28]:

	24-hour check-in	Air conditioning	Baby bath	\
1021	False	False	False	
1022	True	True	False	
1023	False	True	False	
1024	False	True	False	
1025	False	True	False	
1026	False	True	False	
1027	False	True	False	
1028	False	False	False	
1029	False	True	False	
1030	False	True	False	

	Babysitter recommendations	Bathtub	Breakfast	\
1021	False	False	False	
1022	False	False	False	
1023	False	False	False	
1024	False	False	False	
1025	False	False	False	
1026	False	False	False	
1027	False	False	False	
1028	False	False	False	
1029	False	False	False	
1030	False	False	False	

	Buzzer/wireless intercom	Cable TV	Carbon monoxide detector	Cat(s)	\
1021	True	False	False	False	
1022	True	False	False	False	
1023	True	False	True	False	
1024	False	True	False	False	
1025	False	False	False	False	
1026	False	False	True	False	
1027	True	False	False	False	
1028	False	False	False	False	
1029	False	False	False	False	
1030	True	True	False	False	

	…	Suitable for events	TV	Table corner guards	Washer	\
1021	…	False	True	False	False	
1022	…	False	True	False	True	
1023	…	False	True	False	True	
1024	…	False	True	False	True	
1025	…	False	True	False	True	
1026	…	False	False	False	True	
1027	…	False	False	False	True	
1028	…	False	False	False	True	
1029	…	False	True	False	True	
1030	…	False	False	False	True	

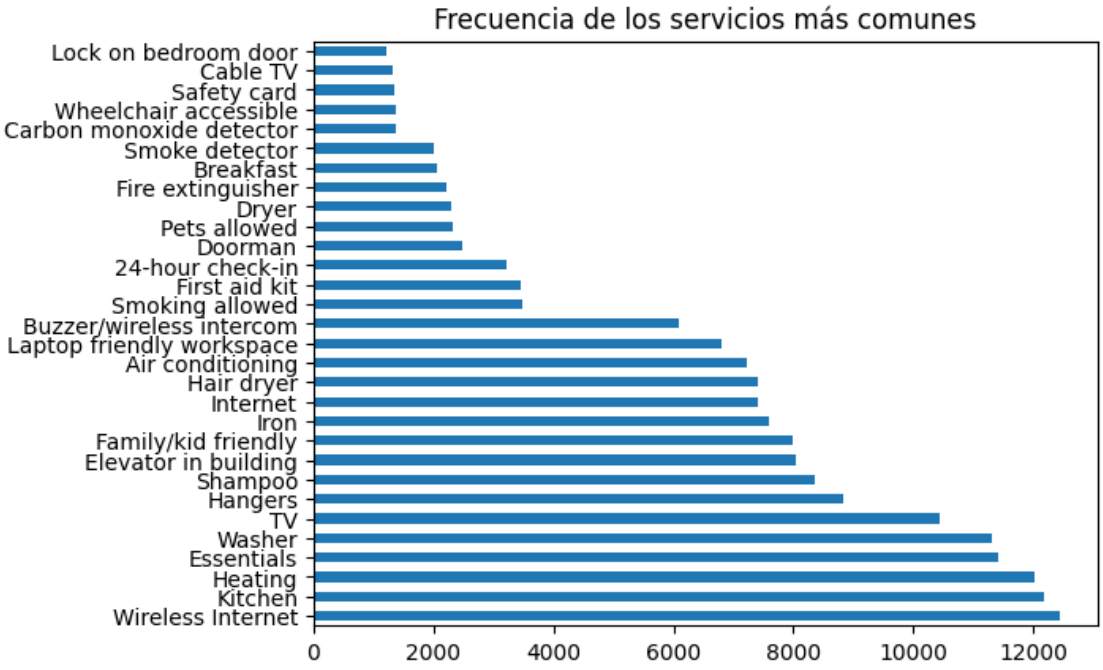
```
      Washer / Dryer  Wheelchair accessible  Window guards  Wireless Internet  \
1021           False                  False          False               True
1022           False                  False          False               True
1023           False                  False          False               True
1024           False                  False          False               True
1025           False                  False          False               True
1026           False                  False          False               True
1027           False                  False          False               True
1028           False                  False          False               True
1029           False                  False          False               True
1030           False                  False          False               True

      translation missing: en.hosting_amenity_49  \
1021                                        True
1022                                       False
1023                                       False
1024                                       False
1025                                       False
1026                                       False
1027                                       False
1028                                        True
1029                                       False
1030                                       False

      translation missing: en.hosting_amenity_50
1021                                        True
1022                                       False
1023                                       False
1024                                       False
1025                                       False
1026                                       False
1027                                       False
1028                                        True
1029                                       False
1030                                       False

[10 rows x 67 columns]
```

2. Después, calculamos la *frecuencia de cada uno de los servicios* y generamos un gráfico de barras para visualizarlo mejor.

[29]:
```
# Calculamos la frecuencia de las Amenities
df_amenities_frequency = pd.DataFrame()
for column in df_amenities.columns:
    df_amenities_frequency[column] = df_amenities[column].value_counts()
```

```
# Detectamos que tenemos dos variables llamadas 'traslation missing...', las␣
 ↪quitamos dado que no aportan un valor real a nuestro análisis
df_amenities_frequency = df_amenities_frequency.drop('translation missing: en.
 ↪hosting_amenity_49', axis = 1)
df_amenities_frequency = df_amenities_frequency.drop('translation missing: en.
 ↪hosting_amenity_50', axis = 1)

# Plot de las frecuencias de las Amenities
plt.title("Frecuencia de los servicios más comunes")
df_amenities_frequency.transpose()[True].sort_values(ascending=False).head(30).
 ↪plot.barh()
```

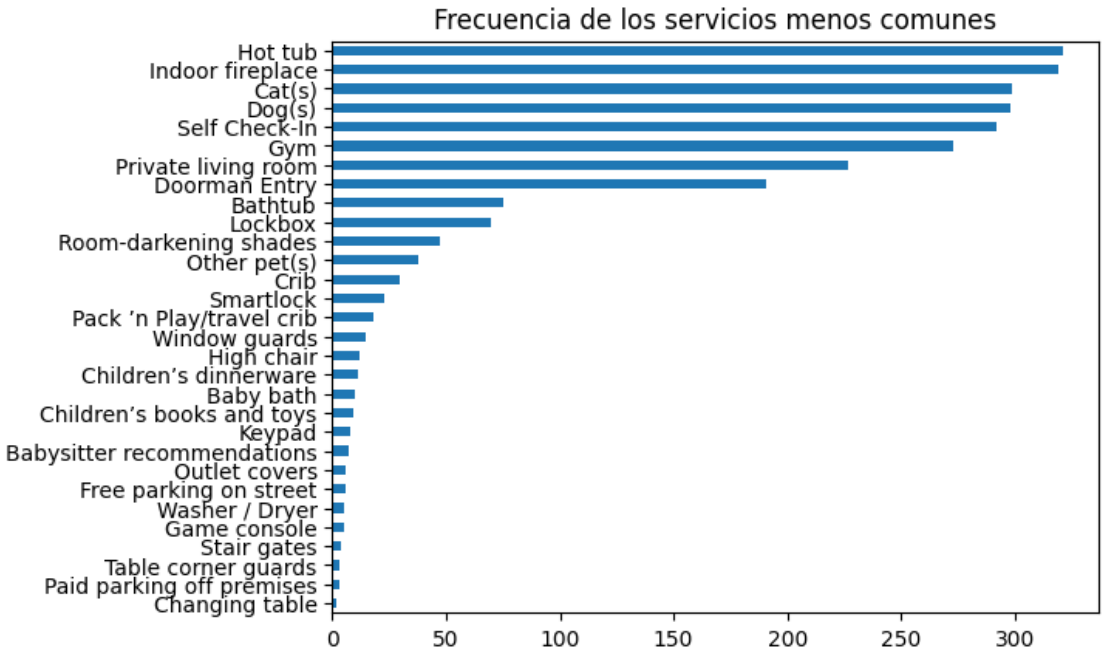
[29]: `<AxesSubplot: title={'center': 'Frecuencia de los servicios más comunes'}>`



Con esto, podemos concluir que la mayoría de las propiedades cuentan con internet inalámbrico, cocina, calefacción, lavadora, etc.

[30]:
```
# Generamos otro plot para visualizar las frecuencias más bajas
plt.title("Frecuencia de los servicios menos comunes")
df_amenities_frequency.transpose()[True].sort_values(ascending=True).head(30).
 ↪plot.barh()
```

[30]: `<AxesSubplot: title={'center': 'Frecuencia de los servicios menos comunes'}>`

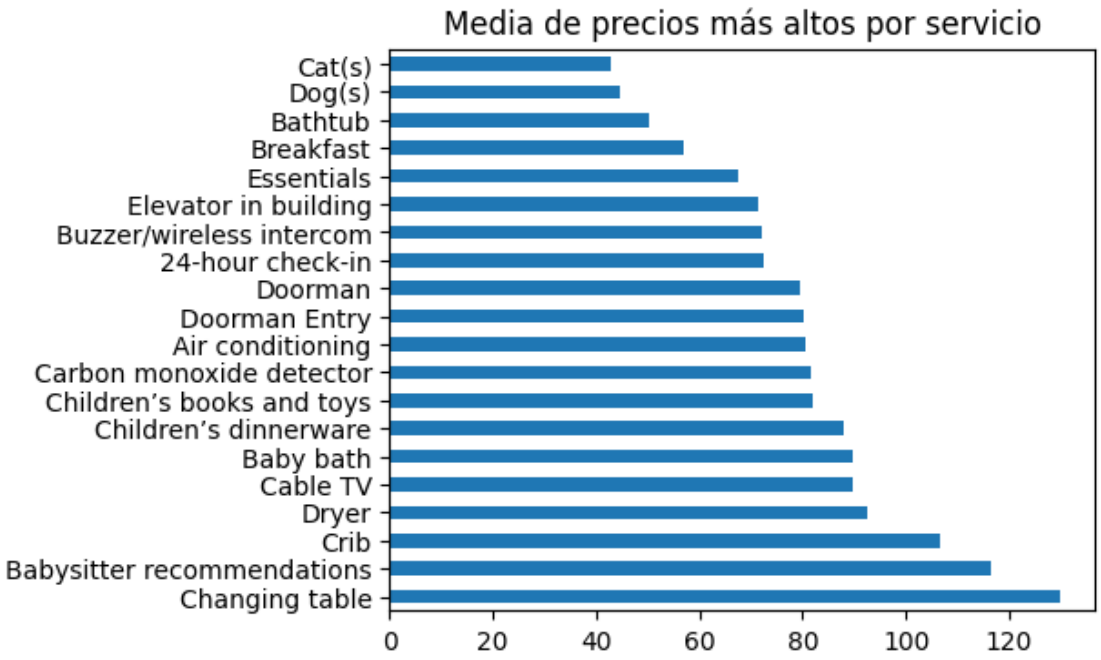


Por el contrario, es más raro encontrar mesa cambiadora, parking fuera de las instalaciones, protectores para las esquinas de las mesas, protectores para escaleras, etc. Es decir, hay una clara tendencia a no tener las propiedades acondicionadas para bebés y niños pequeños.

3. A continuación, verificamos la *media del precio por tener un servicio en concreto.*

[32]:
```
# Añadimos la columna 'Price' a nuestro dataframe
df_amenities_price = df_amenities.join(df['Price'])

# Inicialicamos un dataframe para añadir la media del precio
df_amenities_price_mean = pd.DataFrame()

# Añadimos columnas que nos den la media por amenity
for column in df_amenities.columns:
    df_amenities_price_mean[column] = df_amenities_price.
 ↪groupby(column)['Price'].mean()

# Hacemos la traspuesta para poder agrupar en el eje correcto cada una de las␣
 ↪amenities y visualizamos el resultado
df_amenities_price_mean = df_amenities_price_mean.transpose()
plt.title("Media de precios más altos por servicio")
df_amenities_price_mean[True].head(20).sort_values(ascending=False).plot.
 ↪barh(figsize=(5, 4))
```

[32]:
```
<AxesSubplot: title={'center': 'Media de precios más altos por servicio'}>
```



Tras ver esto, concluimos que lo que suele tener más impacto en la media del precio suelen ser los servicios asociados a las habitaciones acondicionadas para niños o bebés, como la mesita cambiadora y las recomendaciones de niñeras.

4. Entonces nos pareció bien verificar *qué servicios generaban una mayor variación en el precio medio.*

[33]:
```
# Añadimos una columna 'Diff' que registra la diferencia de la media del precio␣
↪entre tener el servicio o no
df_amenities_price_mean['Diff'] = df_amenities_price_mean[True] -␣
↪df_amenities_price_mean[False]
df_amenities_price_mean.sort_values(by='Diff', ascending=False).head(20)
```

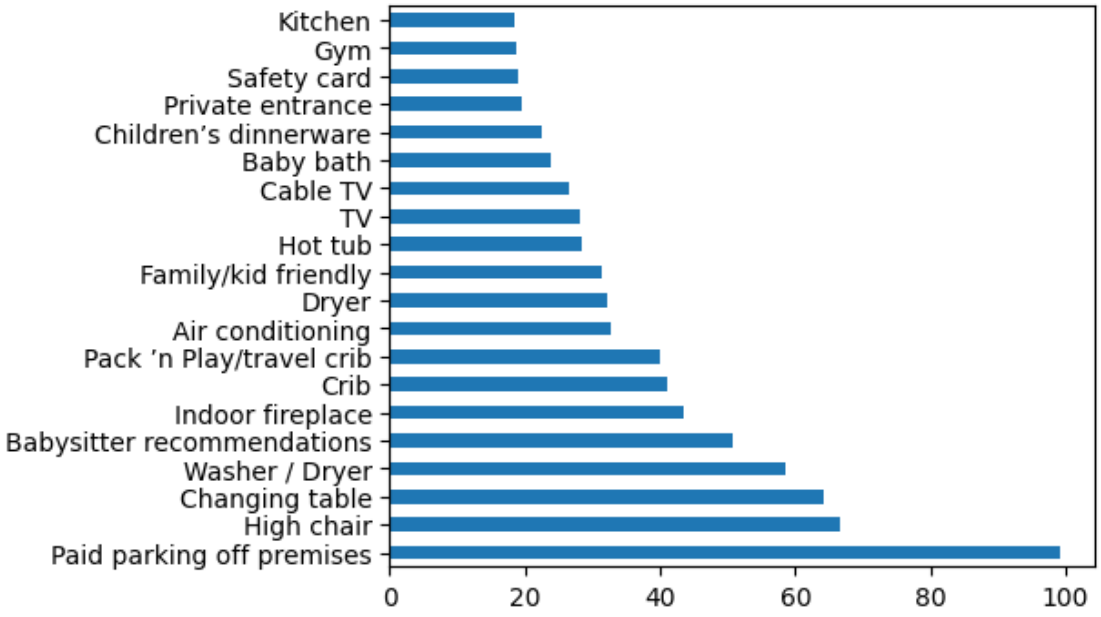
[33]:

24-hour check-in	False	True	Diff
Paid parking off premises	65.710754	165.000000	99.289246
High chair	65.672611	132.416667	66.744055
Changing table	65.723591	130.000000	64.276409
Washer / Dryer	65.711089	124.400000	58.688911
Babysitter recommendations	65.706342	116.571429	50.865087
Indoor fireplace	64.679176	108.263323	43.584147
Crib	65.640094	106.633333	40.993239
Pack 'n Play/travel crib	65.678840	105.611111	39.932271
Air conditioning	47.771644	80.541984	32.770339
Dryer	60.152421	92.365427	32.213006
Family/kid friendly	46.767816	78.156481	31.388665
Hot tub	65.041029	93.575000	28.533971

```
TV                          43.395585   71.653079   28.257494
Cable TV                    63.088428   89.615970   26.527541
Baby bath                   65.715229   89.600000   23.884771
Children's dinnerware       65.714676   88.090909   22.376233
Private entrance            65.156850   84.803618   19.646767
Safety card                 63.799004   82.795235   18.996231
Gym                         65.348533   84.007353   18.658820
Kitchen                     48.669625   67.154497   18.484872
```

[34]:
```
# Visualizamos dicha diferencia en un gráfico
df_amenities_price_mean['Diff'].sort_values(ascending=False).head(20).plot.
↪barh(figsize=(5, 4))
```

[34]: <AxesSubplot: >



Vemos ahora que lo que genera más diferencia parece ser el parking fuera de la propiedad, la silla alta (para bebés), la mesa cambiadora, la lavadora y secadora…

Tras hacer este pequeño análisis, observamos que hay algunos servicios que tienen un impacto significativo. Sin embargo, quedaría pendiente analizar estas variables más a fondo para tener conclusiones más fiables.

Otra conclusión podría ser que, simplemente, sean servicios muy raros y que, casualmente, estén en los inmuebles con un precio mayor.

5. Otro aspecto que decidimos explorar es la *relación de la cantidad de servicios con el precio.*

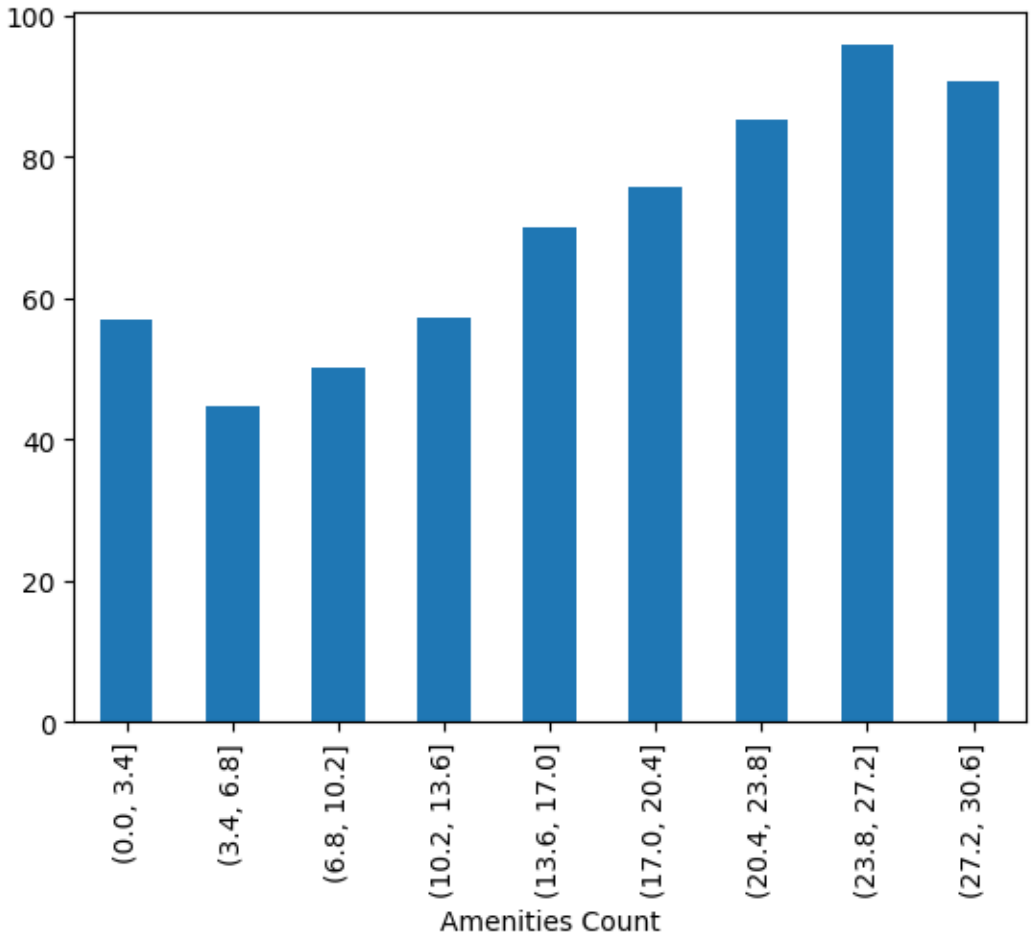
[35]:
```
# Inicializamos el dataframe Amenities_Count
df_amenities_count = pd.DataFrame()

# Añadimos una columna con el conteo de servicios y el precio
df_amenities_count['Amenities Count'] = df['Amenities'].str.split(',').
 ↪fillna('').map(lambda x: len(x))
df_amenities_count = df_amenities_count.join(df['Price'])
df_amenities_count.head(10)
```

[35]:

	Amenities Count	Price
1021	15	50.0
1022	16	50.0
1023	18	77.0
1024	10	50.0
1025	10	95.0
1026	16	69.0
1027	9	70.0
1028	16	30.0
1029	13	57.0
1030	12	59.0

[36]:
```
# Visualizamos la variación del precio medio de las propiedades en función de␣
 ↪cuántos servicios totales tienen, agrupando el conteo en 10 grupos
steps = 10
max = df_amenities_count['Amenities Count'].max()
stept = max / steps
steps = np.arange(0, max, stept)
groups = pd.cut(df_amenities_count['Amenities Count'], steps)
df_amenities_count.groupby(groups)['Price'].mean().plot.bar()
```

[36]:
```
<AxesSubplot: xlabel='Amenities Count'>
```



Parece ser que el precio varía: se observa el mínimo en las propiedades que tienen entre 3 y 7 servicios, y se dispara y alcanza el máximo al ofrecer entre 24 y 27 servicios.

Es decir, la tendencia es subir el precio medio del alojamiento mientras más servicios se ofrezcan.

1.2.5 Ratio de Ocupación.

La tasa de ocupación es un término que se utiliza para saber cuánto tiempo pasa un espacio rentado en relación a cuánto tiempo pasa disponible.

En este caso, pongamos el ejemplo de una propiedad que se encontraba disponible 365 días al año, pero solo fue rentada los fines de semana. Es decir, estuvo rentada 104 días de 365. Eso nos daría una tasa de ocupación del 28,49%.

Este dataset no nos da información sobre la tasa de ocupación, pero la podemos obtener relacionando datos como la media de reseñas al mes, el mínimo de noches a alquilar y la disponibilidad anual.

Pongamos el caso de una propiedad con un mínimo de noches a alquilar de 2, con 4 reseñas al mes y una disponibilidad de 365 días al año.

Dividiendo las (reseñas * mínimo de noches) * 12 / disponibilidad, obtenemos el ratio de ocupación, asumiendo que cada individuo que va a la propiedad deja una reseña.

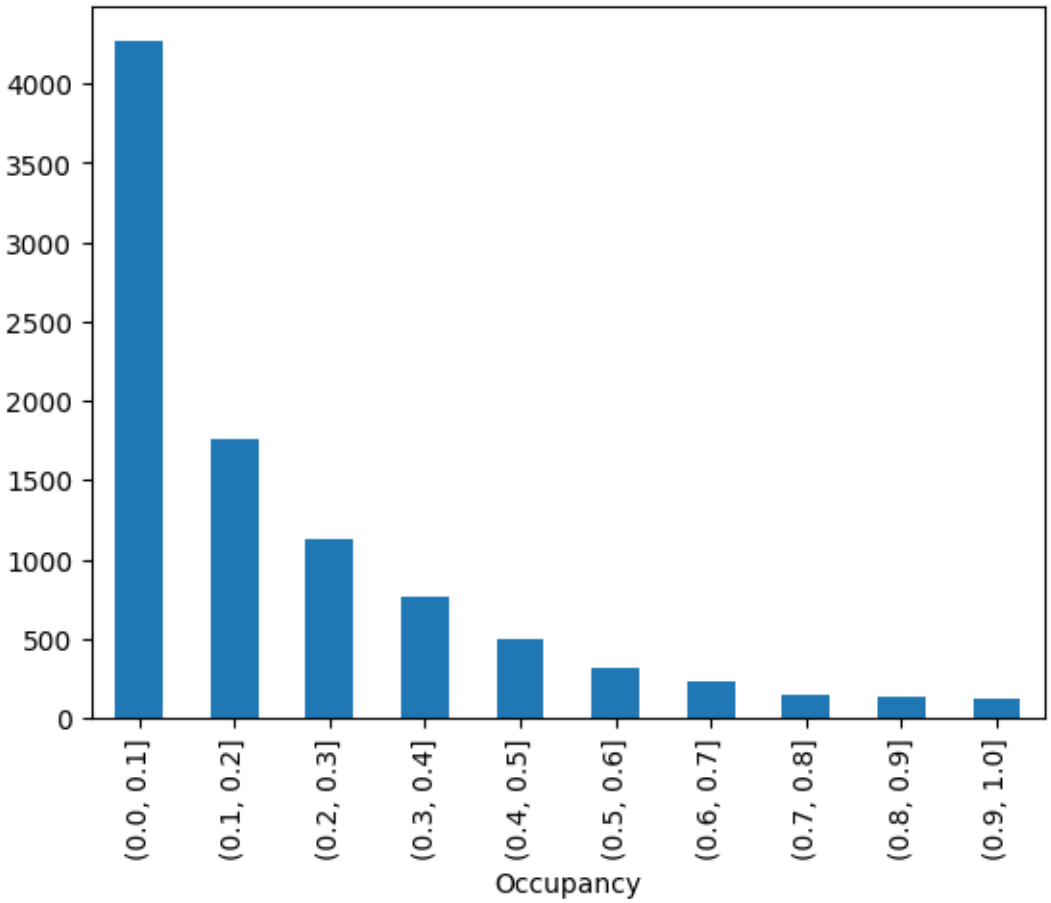
Decidimos explorar si esto nos daba algún tipo de información valiosa.

Como primer paso, definimos una función para calcular dicha tasa de ocupación.

[48]:
```
# Función para calcular la tasa de ocupación
def calculate_occupancy(reviews_month, min_nights, availability):
    return ((reviews_month * min_nights) * 12 / availability)
```

[49]:
```
# Creamos una nueva columna con la tasa de ocupación calculada según la fórmula
reviews = df['Reviews per Month'].fillna(0)
nights = df['Minimum Nights'].fillna(0)
availability = df['Availability 365'].fillna(0).map(lambda value: value if␣
 ↪value != 0 else 9999999)
df['Occupancy'] = calculate_occupancy(reviews, nights, availability)

# Visualizamos el resultado
steps = np.arange(0, 1.01, 0.1)
groups = pd.cut(df['Occupancy'], steps)
df.groupby(groups).size().plot.bar()
```

[49]:
```
<AxesSubplot: xlabel='Occupancy'>
```



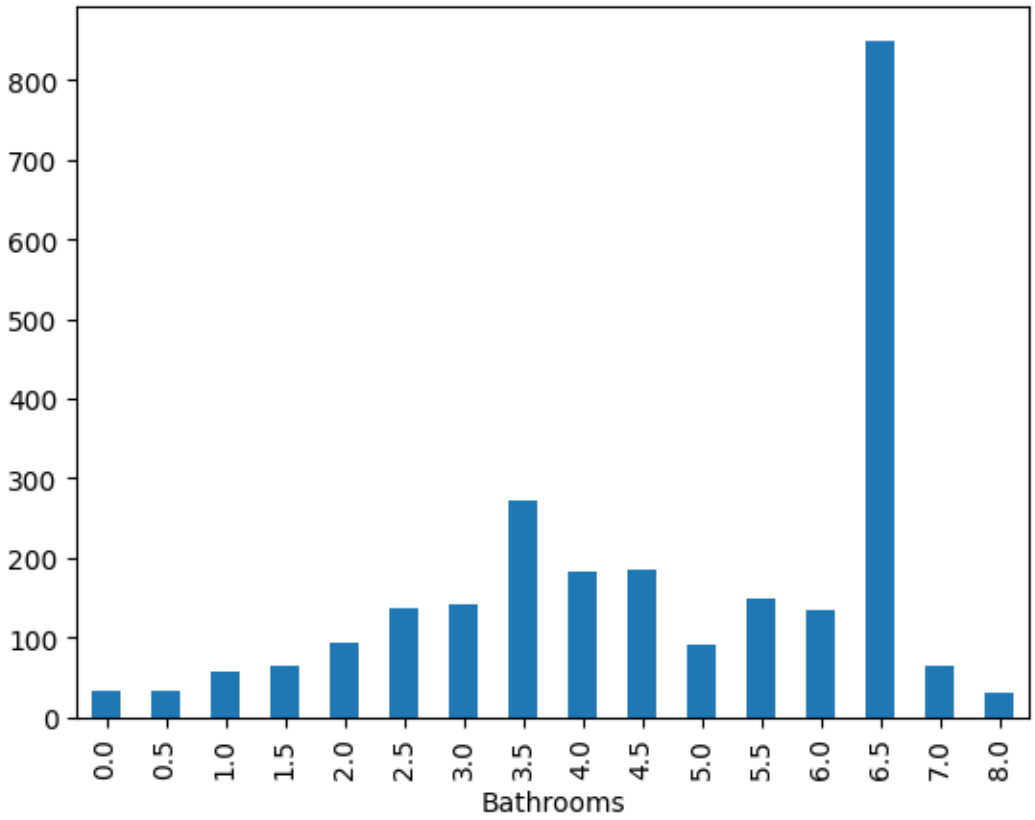
Finalmente, decidimos que este dato no era fiable y no continuamos indagando en su utilidad.

1.2.6 Bathrooms

Analizamos un poco la columna de los baños para tener un insight de si la cantidad de baños afecta al precio final o no.

[24]:
```
# Creamos un plot que nos muestra la media de precio por número de baños
df.groupby('Bathrooms')['Price'].mean().plot.bar()
```

[24]:
```
<AxesSubplot: xlabel='Bathrooms'>
```

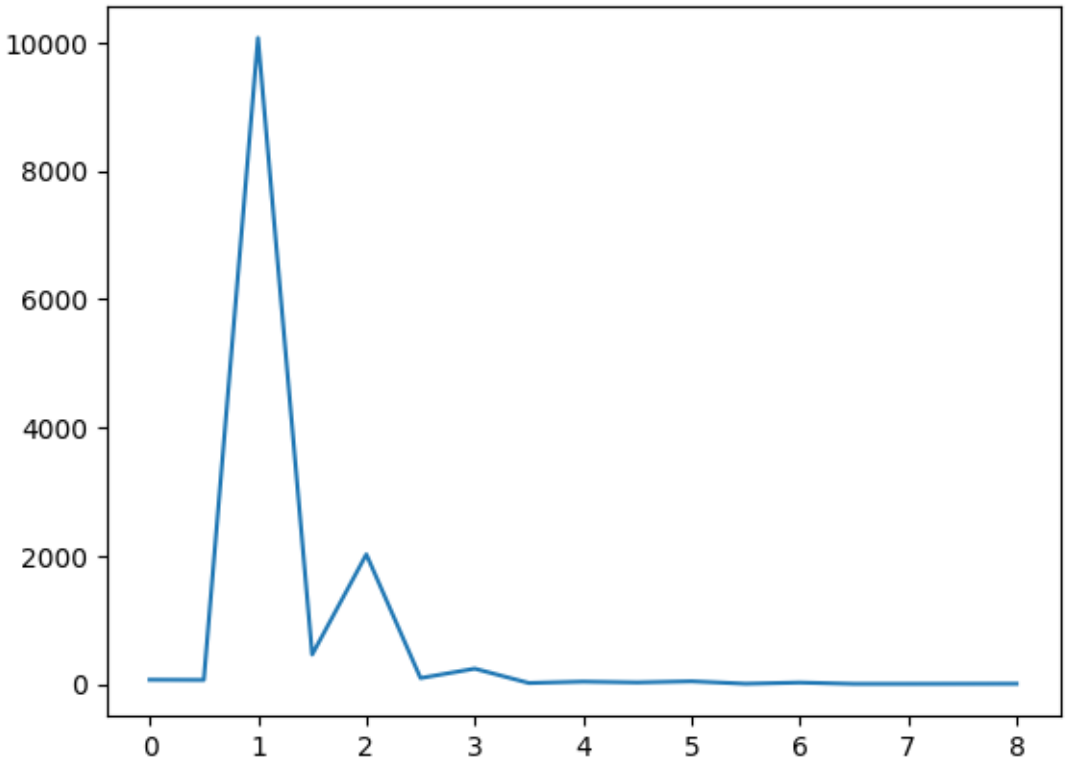


Vemos la distribución de la media de precios, que sube hasta llegar a 3.5 baños. En el caso de más baños es bastante probable que solo se encuentren en propiedades muy grandes con precios bastante elevados, por lo que los datos no serían muy útiles.

Decidimos ver la distribución de la cantidad de baños por propiedad.

[38]:
```
df['Bathrooms'].value_counts().sort_index().plot()
```

[38]:
```
<AxesSubplot: >
```



La mayoría de las propiedades tienen de 1 a 2 baños.

Finalmente, concluimos que este dato no nos va a ser de utilidad.

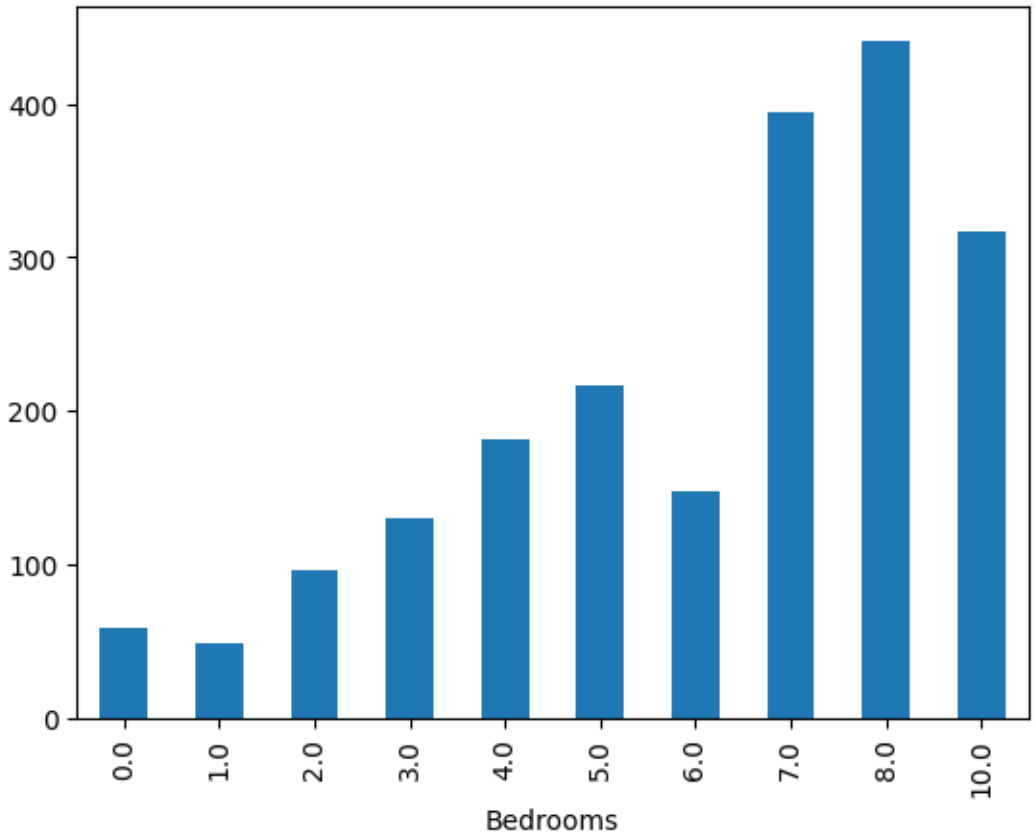
1.2.7 Bedrooms

Hacemos un análisis similar al anterior.

Primero, vemos el precio medio por número de habitacines.

[39]:
```
df.groupby('Bedrooms')['Price'].mean().plot.bar()
```

[39]:
```
<AxesSubplot: xlabel='Bedrooms'>
```

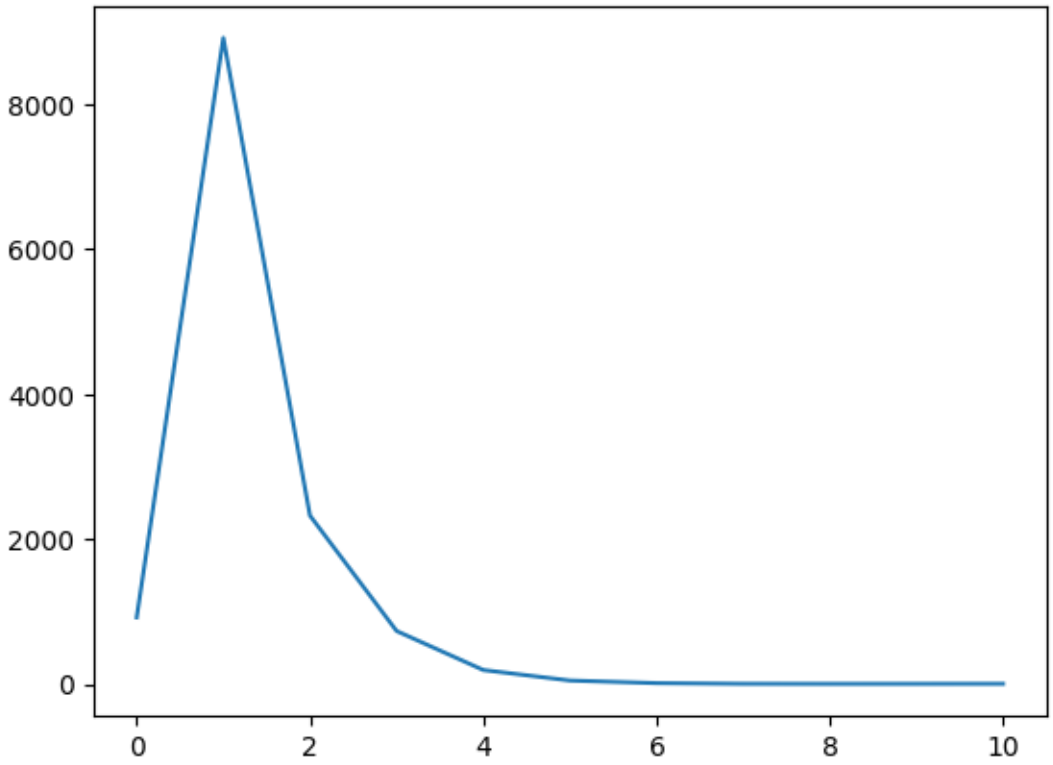


Lógicamente, el precio sube por cantidad de habitaciones. En el dato de 6 habitaciones, puede que baje porque sea un hostal con varias habitaciones, por ejemplo, lo que lo haría más barato.

Vemos también la distribución de la cantidad de habitaciones por propiedad.

[40]:
```
df['Bedrooms'].value_counts().sort_index().plot()
```

[40]:
```
<AxesSubplot: >
```



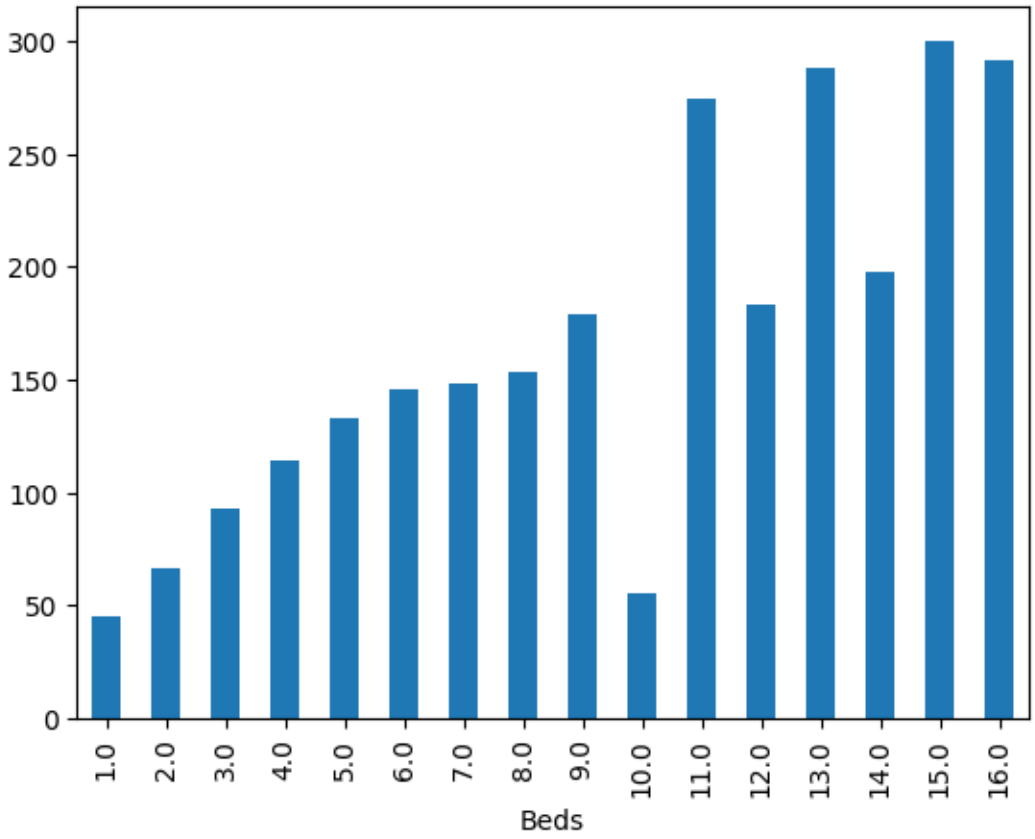
Sin embargo, la mayor parte de las propiedades tienen de 1 a 3 habitaciones. Tampoco le vemos mucha utilidad a estos datos.

1.2.8 Beds

Hacemos otro análisis similar con el número de camas.

[41]:
```
df.groupby('Beds')['Price'].mean().plot.bar()
```

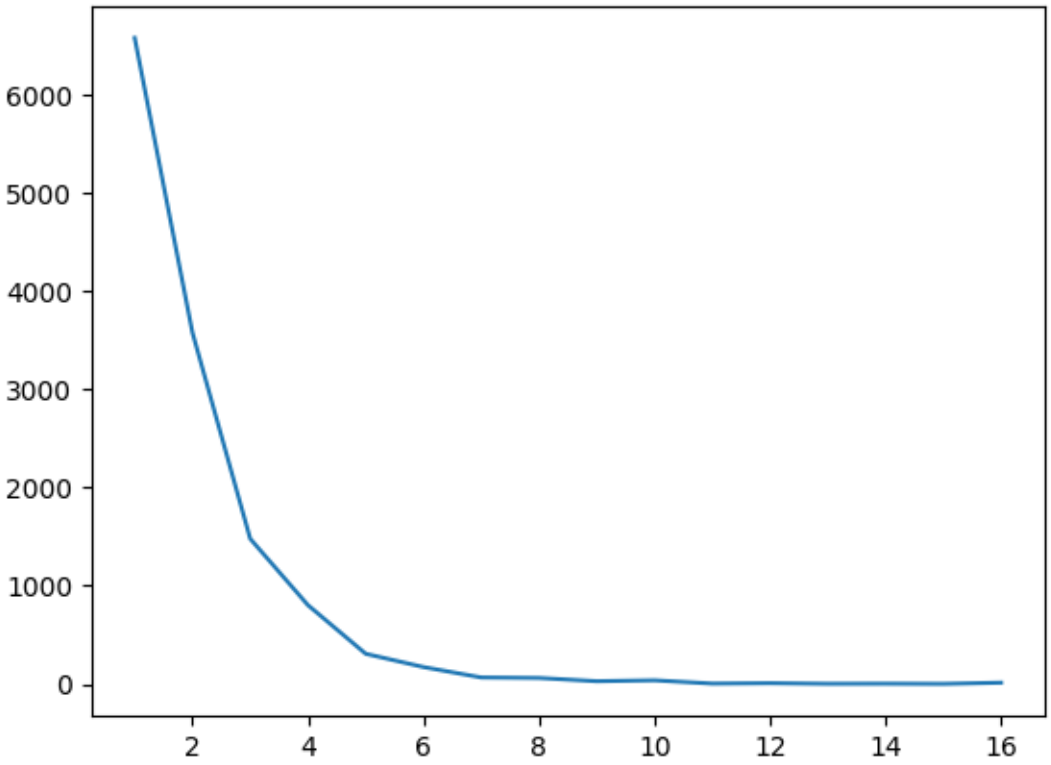
[41]:
```
<AxesSubplot: xlabel='Beds'>
```



Vemos una progresión en el precio a medida que aumentan las camas, a pesar de algunas excepciones que tendrían que estudiarse más detalladamente.

[42]:
```
df['Beds'].value_counts().sort_index().plot()
```

[42]:
```
<AxesSubplot: >
```



Y también vemos que la mayor parte de los alojamientos tienen entre 1 y 6 camas.

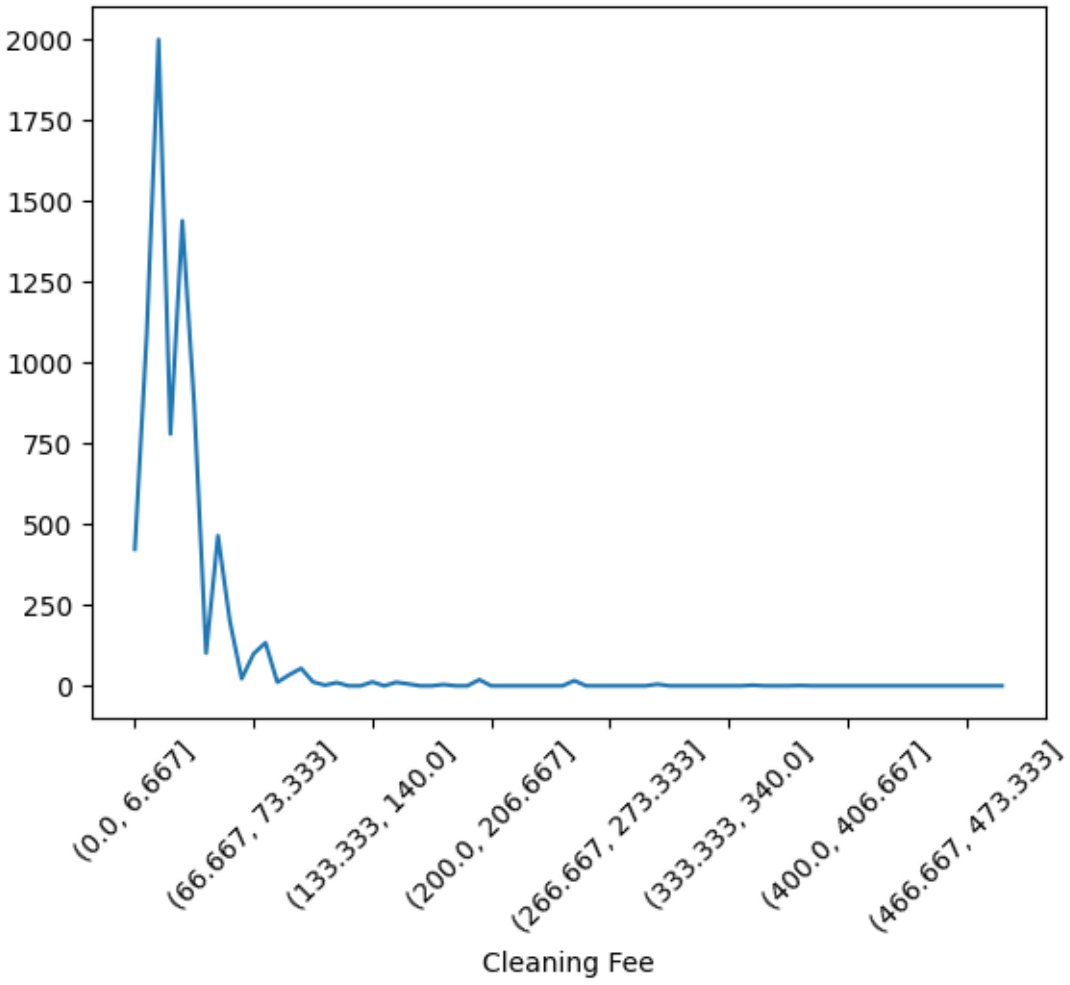
En líneas generales, los resultados del análisis de las columnas *Bathrooms, Bedrooms y Beds* son los esperados. A mayor cantidad de estas utilidades, mayor es el precio, y la mayoría de los alojamientos tienen entre 1 y 3 baños, camas o habitaciones.

1.2.9 Cleaning Fee

Por otra parte, decidimos analizar también la columna Cleaning Fee.

[43]:
```
df['Cleaning Fee'] = df['Cleaning Fee'].fillna(0)

steps = 75
stept = df['Cleaning Fee'].max() / steps
steps = np.arange(0, df['Cleaning Fee'].max(), stept)
groups = pd.cut(df['Cleaning Fee'], steps)
df.groupby(groups).size().plot(rot=45)
```

[43]:
```
<AxesSubplot: xlabel='Cleaning Fee'>
```



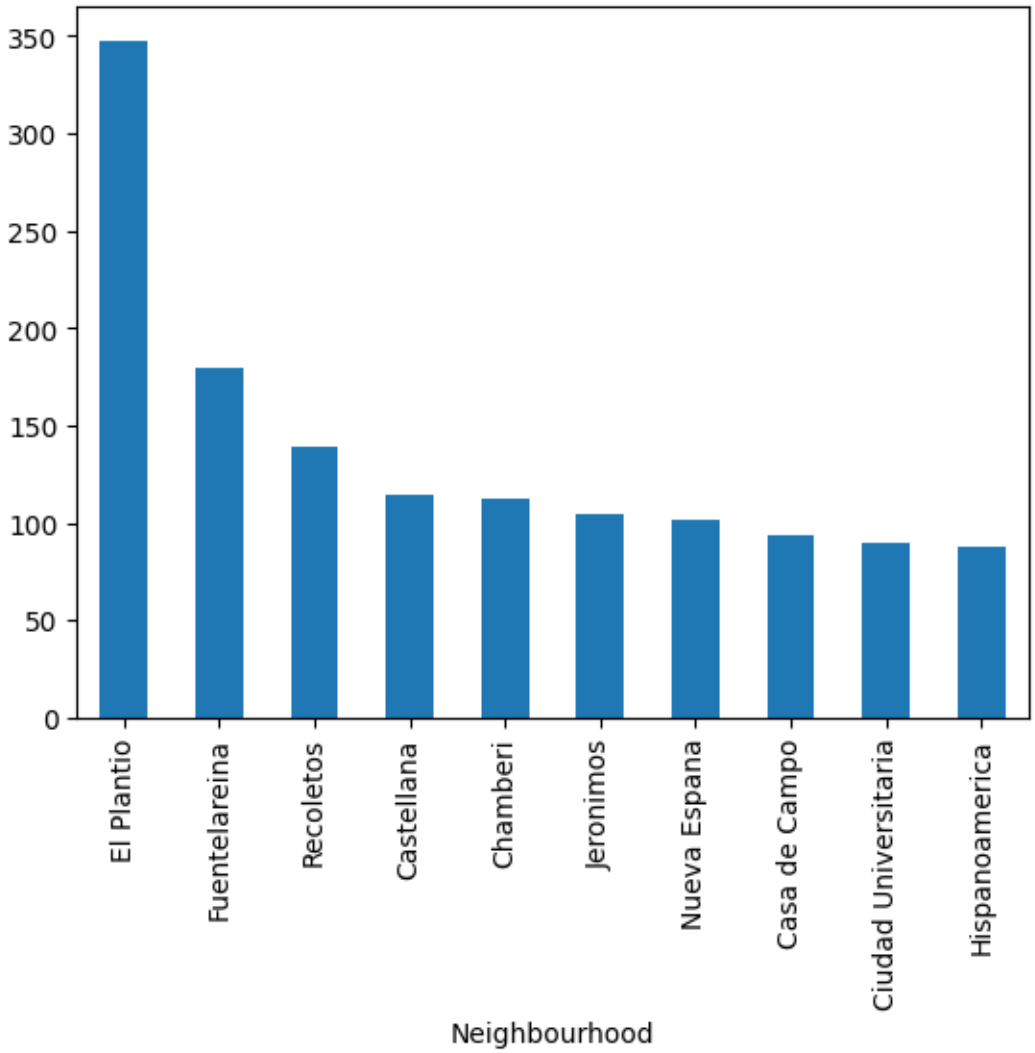
Observamos que la mayoría de los alojamientos tiene un cleaning fee entre 0€ y 75€.

1.2.10 Neighbourhood

Para continuar, decidimos investigar cuáles eran los barrios más caros teniendo en cuenta, primero, el precio por noche.

[44]:
```
df.groupby('Neighbourhood')['Price'].mean().sort_values(ascending=False).
↪head(10).plot.bar()
```

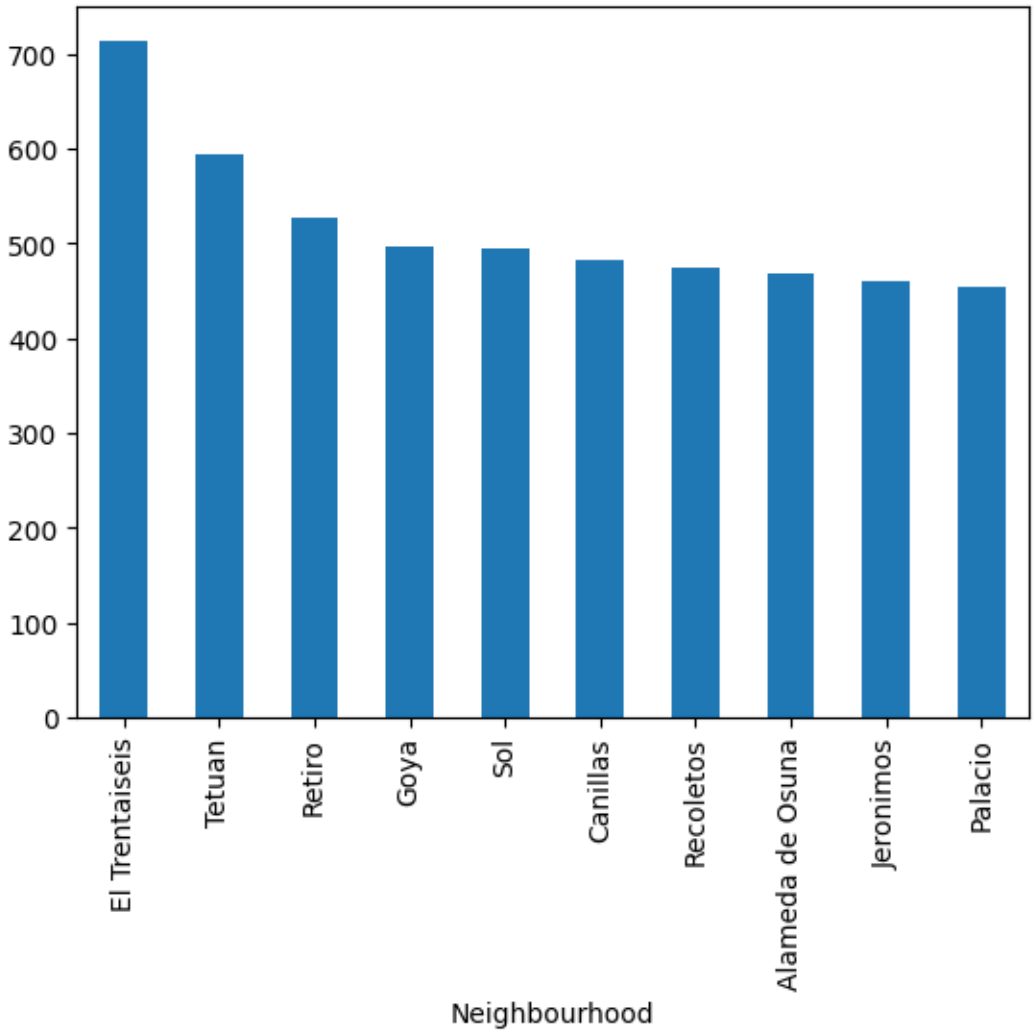
[44]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```



Después, el precio por semana.

[33]:
```
df.groupby('Neighbourhood')['Weekly Price'].mean().sort_values(ascending=False).
↪head(10).plot.bar()
```

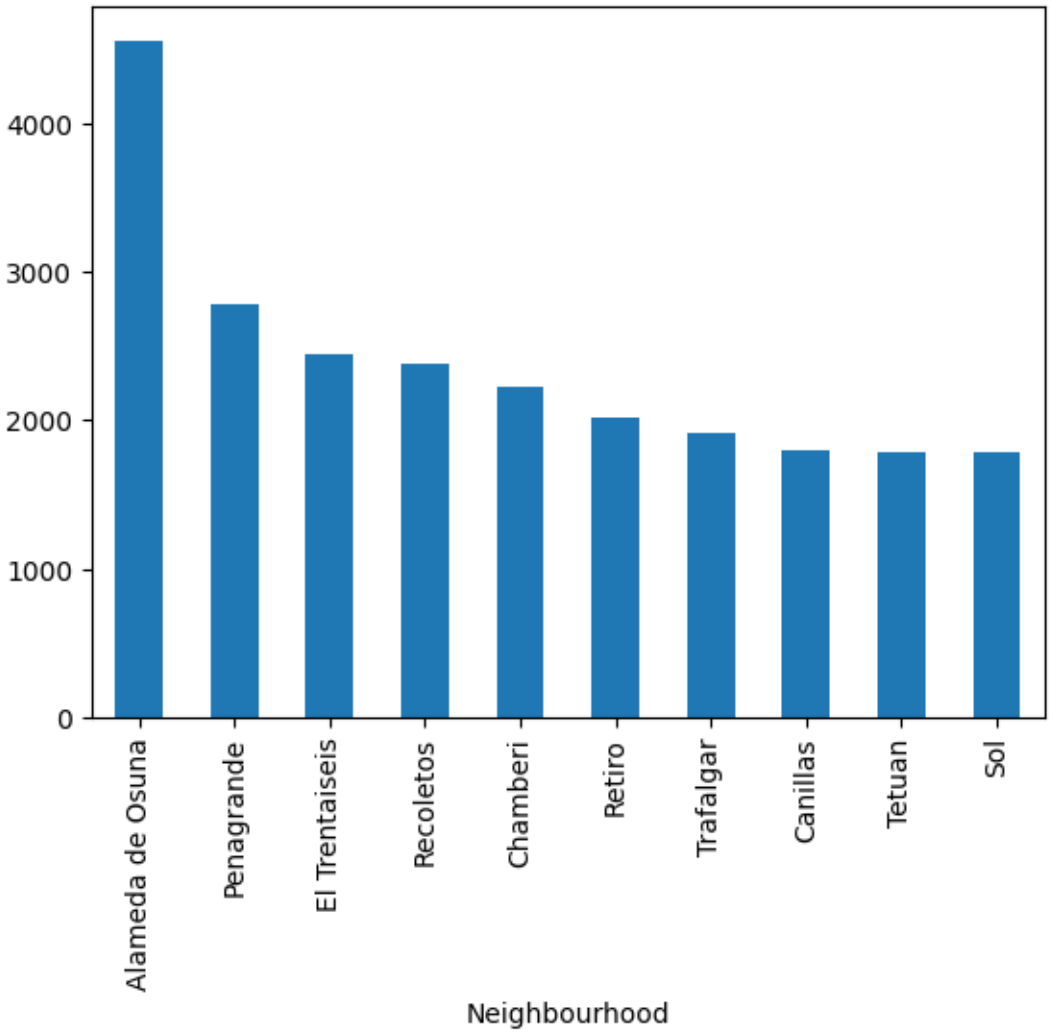
[33]: <AxesSubplot: xlabel='Neighbourhood'>



Y, finalmente, el precio por mes.

[45]:
```
df.groupby('Neighbourhood')['Monthly Price'].mean().
↪sort_values(ascending=False).head(10).plot.bar()
```

[45]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```

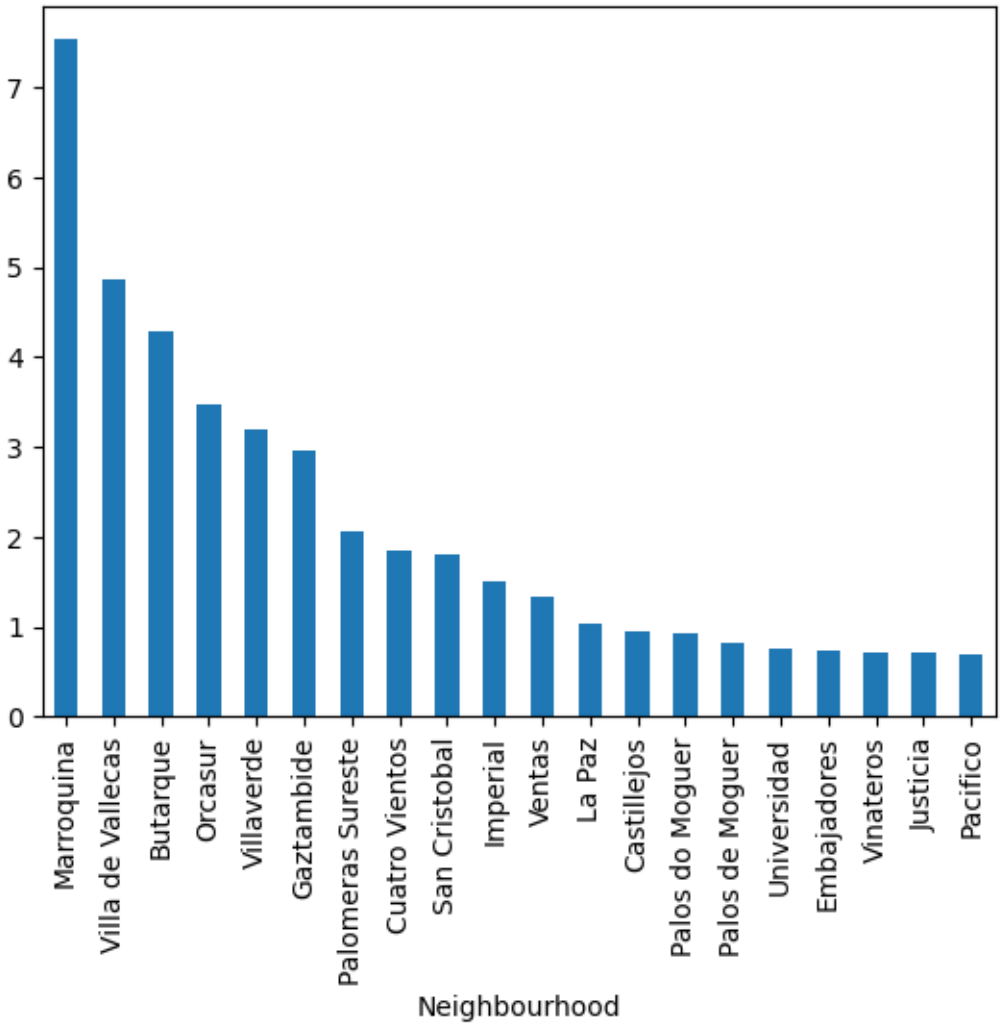


Como se puede observar, los resultados de estos análisis hace que sea necesario investigar más a fondo cuál es el motivo de tantas diferencias.

También nos preguntamos sobre los barrios con un ratio de ocupación más alto.

[50]:
```
df.groupby('Neighbourhood')['Occupancy'].mean().sort_values(ascending=False).
↪head(20).plot.bar()
```

[50]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```

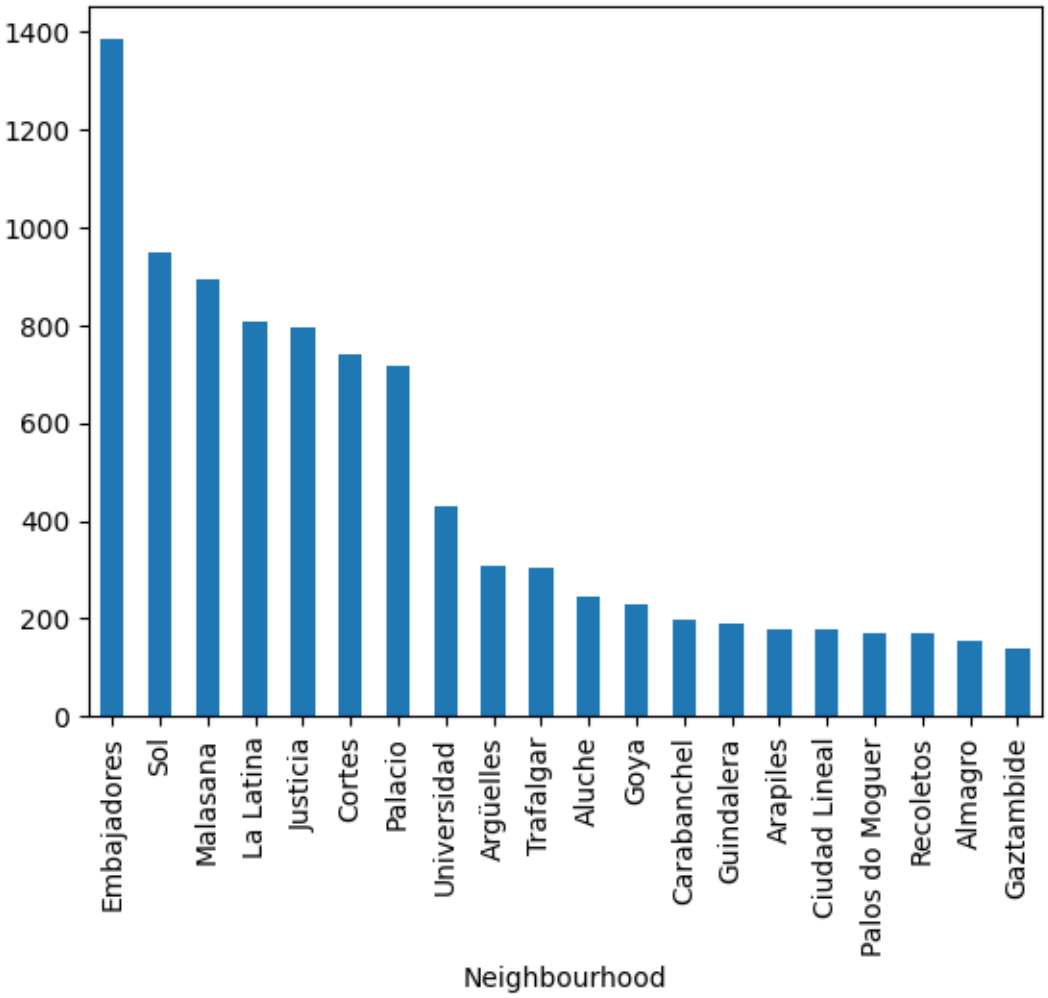


En este caso, el ratio de ocupación también puede estar relacionado con la cantidad de propiedades disponibles en ese barrio en concreto. Con lo cual, la tasa de ocupación por barrio sigue sin ser un indicador muy fiable.

Sin embargo, nos preguntamos qué barrio tenía más alojamientos en alquiler.

[51]:
```
df.groupby('Neighbourhood').size().sort_values(ascending=False).head(20).plot.
↪bar()
```

[51]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```

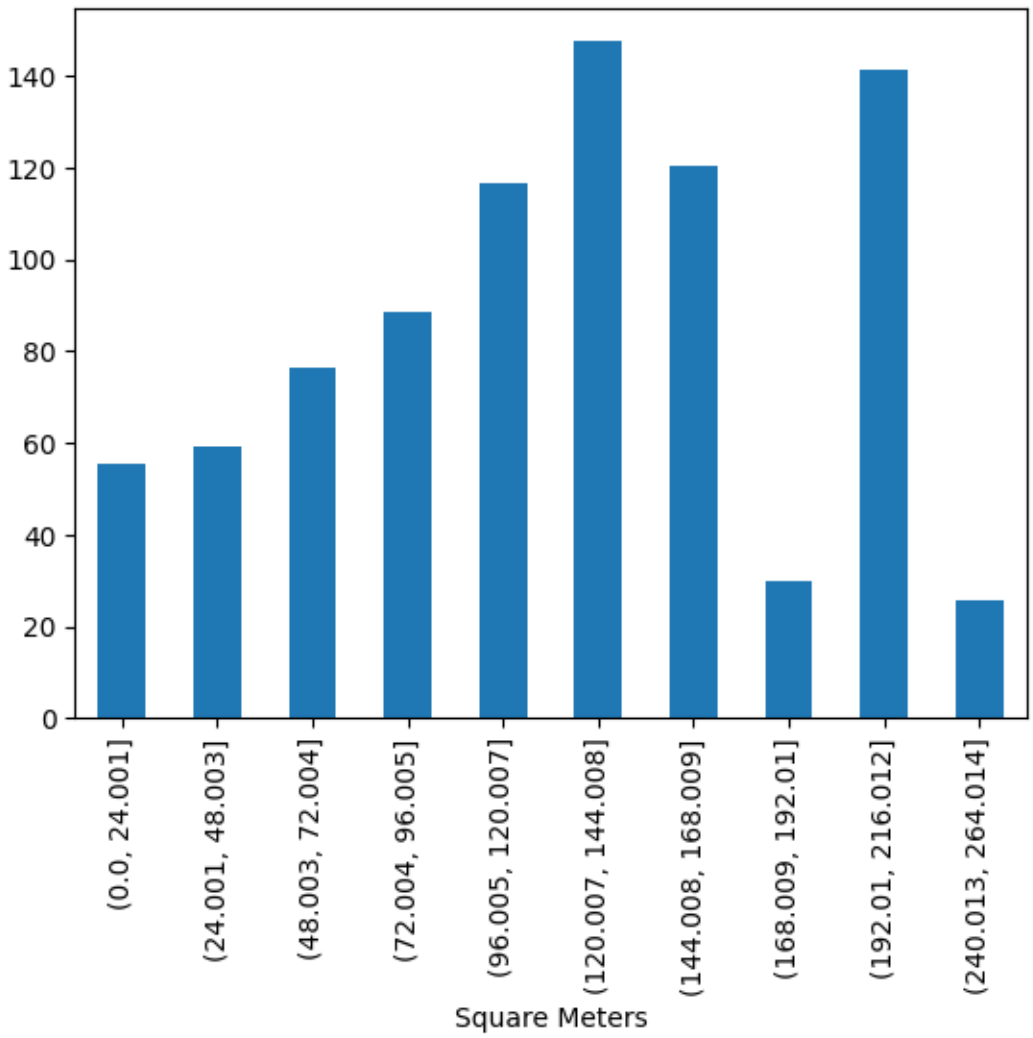


1.2.11 Square Feet / Square Meters

En otra línea, quisimos averiguar la media de metros cuadrados por propiedad.

[52]:
```
steps = 20
stept = df['Square Meters'].max() / steps
steps = np.arange(0, df['Square Meters'].max(), stept)
groups = pd.cut(df['Square Meters'], steps)
df.groupby(groups)['Price'].mean().dropna().plot.bar()
```

[52]:
```
<AxesSubplot: xlabel='Square Meters'>
```

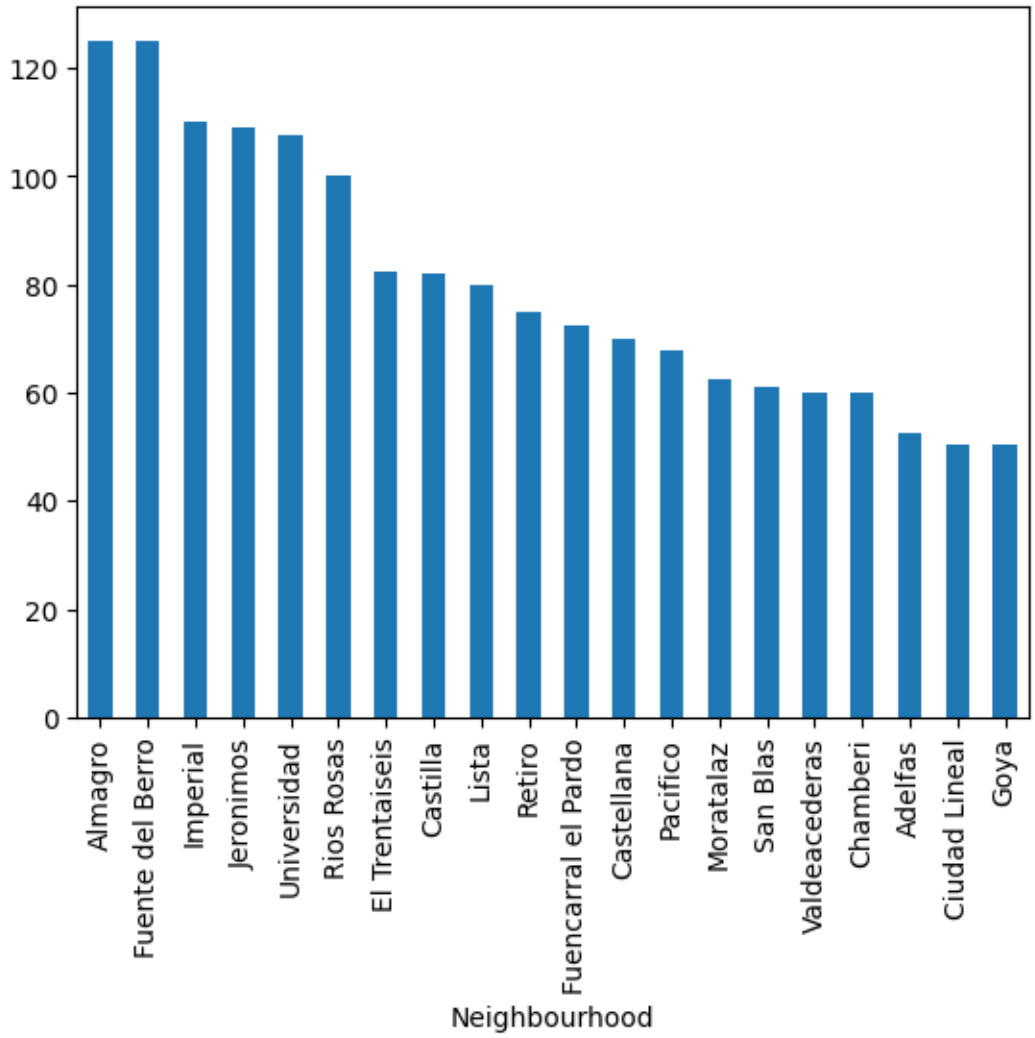


La mayoría de propiedades está entre 96 y 170 metros cuadrados.

Tras esto, nos interesó saber cuáles eran los barrios que ofrecían pisos más grandes.

[53]:
```
df.groupby('Neighbourhood')['Square Meters'].mean().
  ↪sort_values(ascending=False).head(20).plot.bar()
```

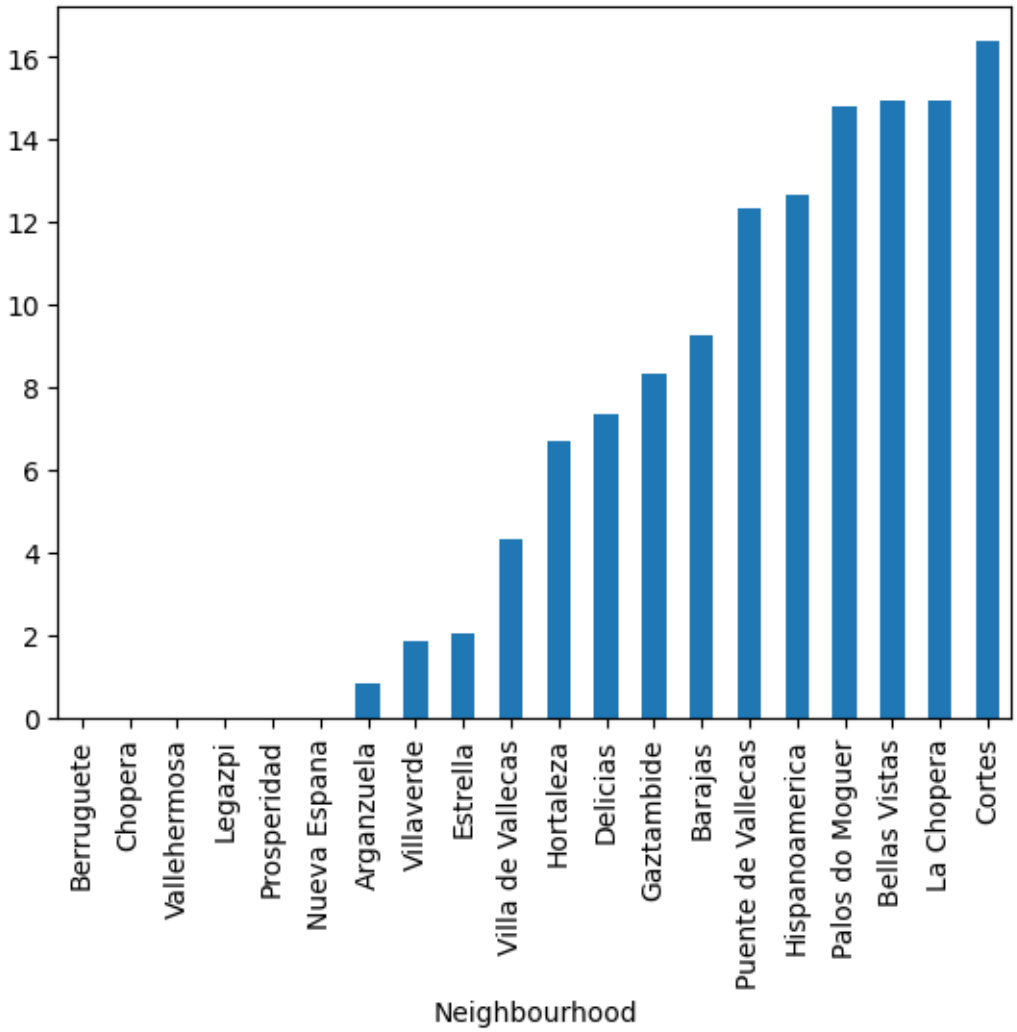
[53]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```



Y más pequeños.

[54]:
```
df.groupby('Neighbourhood')['Square Meters'].mean().dropna().
  ↪sort_values(ascending=True).head(20).plot.bar()
```

[54]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```



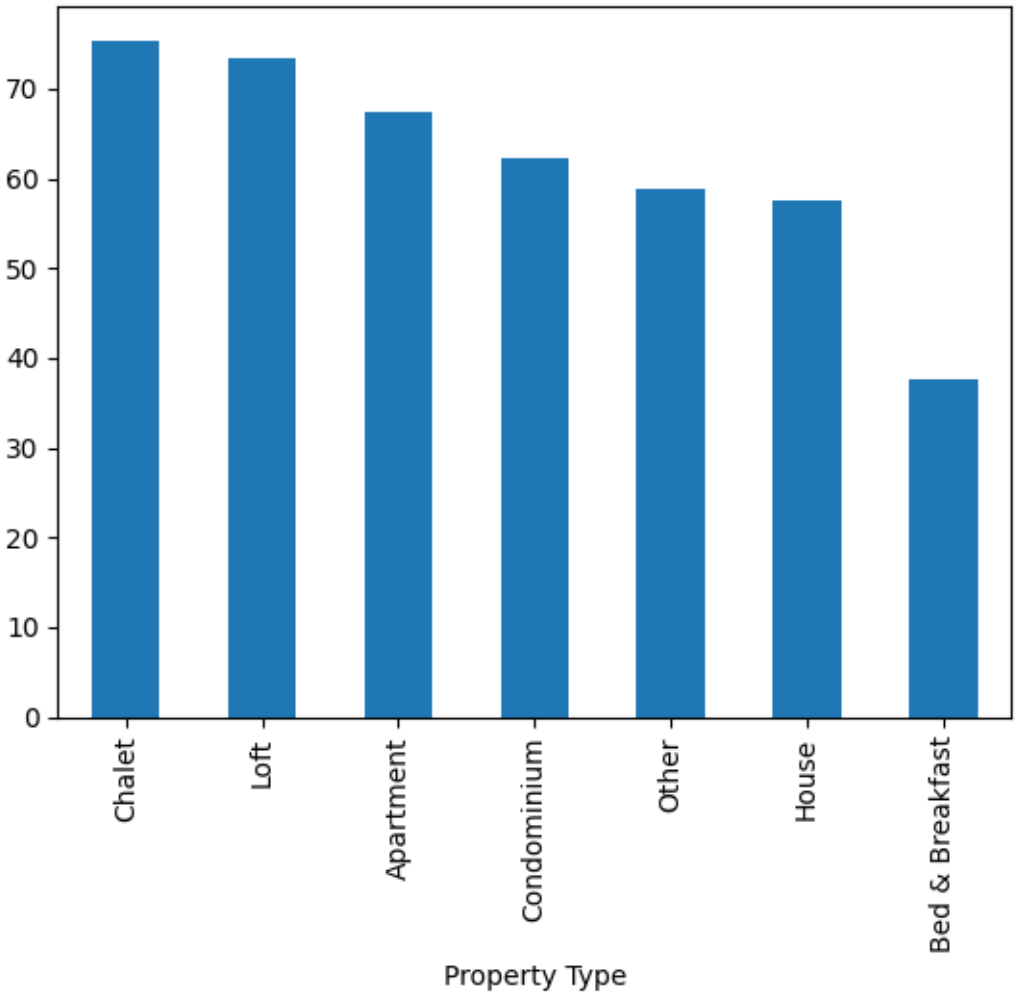
1.2.12 Property Type

Finalmente, decidimos explorar los tipos de propiedades ofertados.

Primero verificamos el precio medio por tipo de propiedad.

[55]:
```
df.groupby(['Property Type'])['Price'].mean().sort_values(ascending=False).
↪head(20).plot.bar()
```

[55]: <AxesSubplot: xlabel='Property Type'>



Y verificamos también las variaciones de la mediana por tipo de propiedad. Para esto, decidimos usar el boxplot, por lo cual, para tener una mejor visualición, creamos la columna 'Log Price' con el logaritmo de la columna 'Price'.

[56]:
```
df['Log Price'] = df['Price'].apply(np.log)
df['Log Price']
```

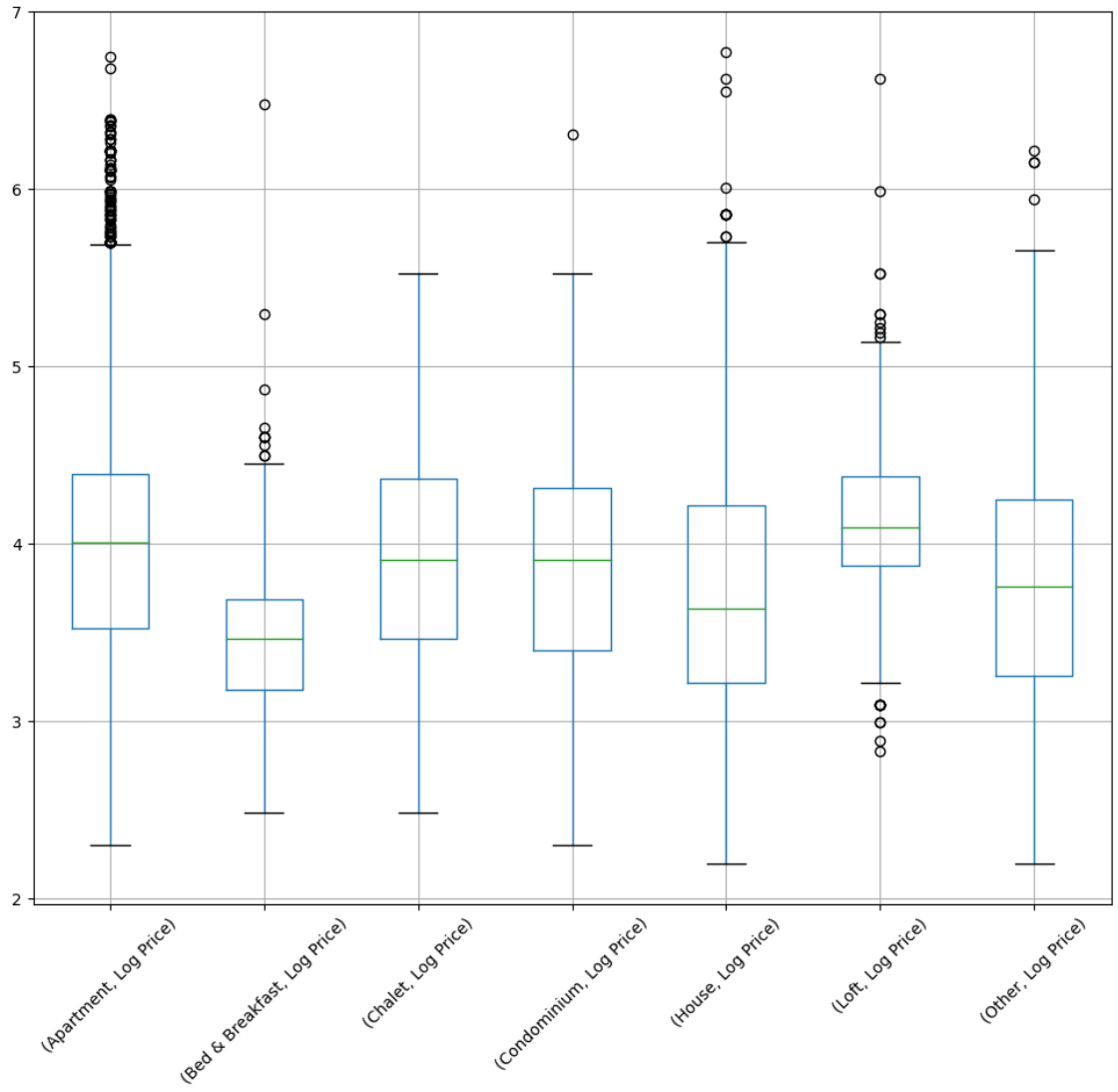
[56]:
```
1021       3.912023
1022       3.912023
1023       4.343805
1024       3.912023
1025       4.553877
             ...
492516     4.007333
492517     4.382027
492518     3.912023
```

```
492519    4.248495
492520    4.488636
Name: Log Price, Length: 13198, dtype: float64
```

Hacemos el análisis según el tipo de propiedad.

[57]:
```
df.groupby('Property Type')[['Property Type', 'Log Price']].
  ↪boxplot(subplots=False, figsize=(12, 10), rot=45)
```

[57]: <AxesSubplot: >

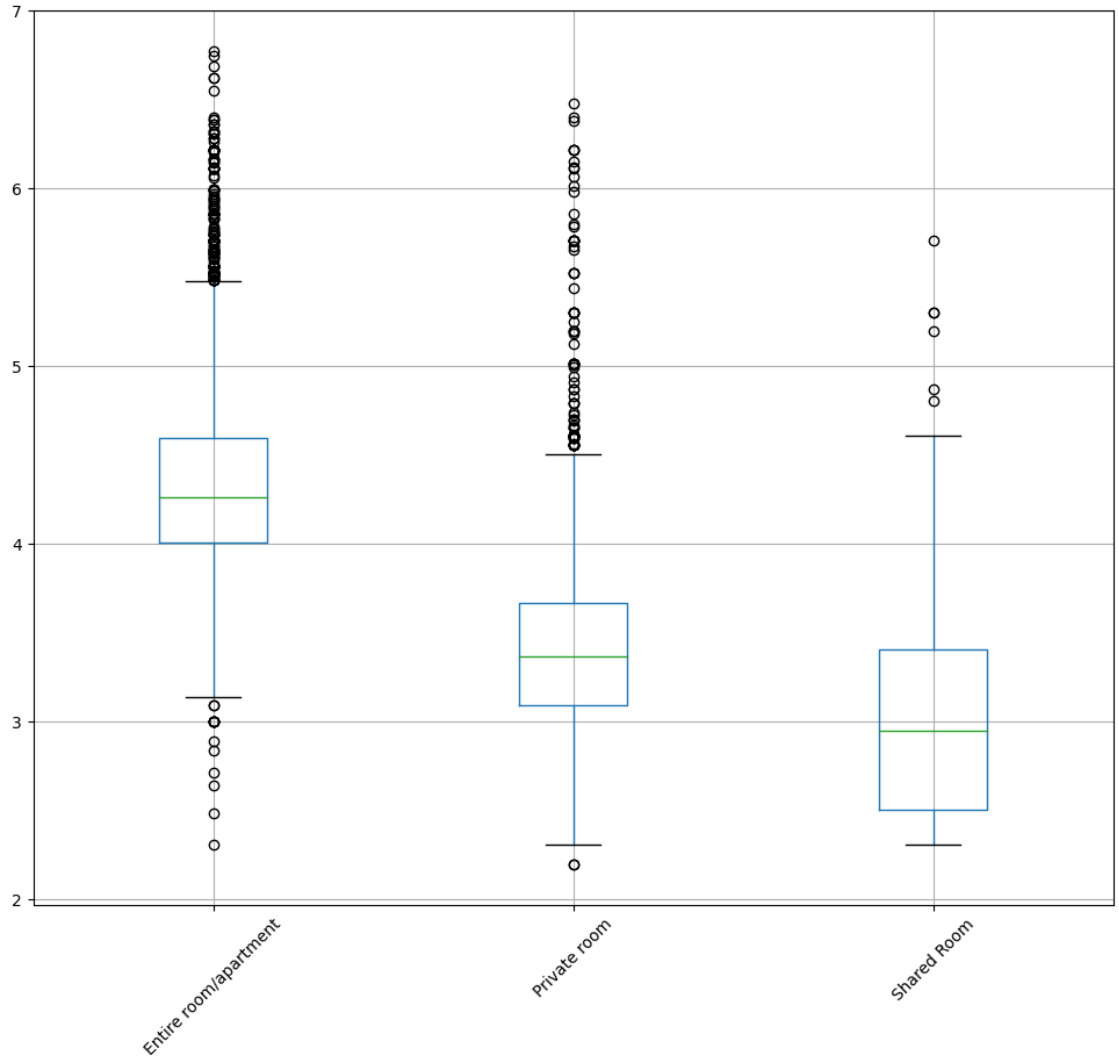


Esto lo que nos indica es que la mediana se mantiene más o menos constante, entre los 35€ y 40€.

Hacemos el análisis también según el tipo de habitación.

[58]:
```
new_labels = ['Entire room/apartment', 'Private room', 'Shared Room'] # lista␣
 ↪con las nuevas etiquetas
df.groupby('Room Type')[['Room Type', 'Log Price']].boxplot(subplots=False,␣
 ↪figsize=(12, 10), rot=45)
plt.xticks([1, 2, 3], new_labels)
plt.show
```

[58]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Como era de esperar, un departamento entero es más costoso que un cuarto privado y un cuarto compartido. Ninguna sorpresa aquí.

Hacemos lo mismo por tipo de cama.

[67]:
```
new_labels = ['Airbed', 'Couch', 'Futon', 'Pull out Sofa', 'Real Bed']
df.groupby('Bed Type')[['Log Price']].boxplot(subplots=False, figsize=(12, 10),␣
 ↪rot=45)
plt.xticks([1, 2, 3, 4, 5], new_labels)
plt.show()
```

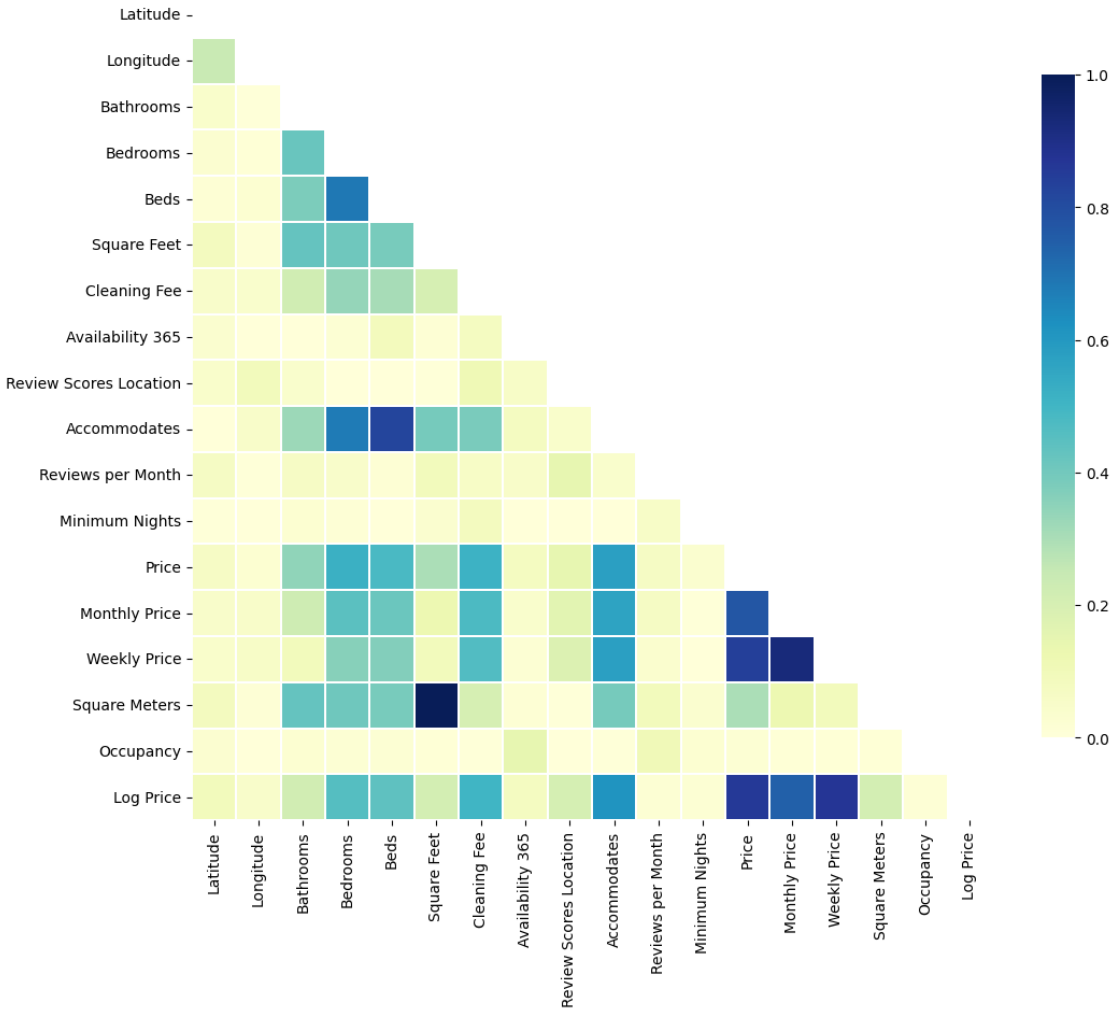
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In [67], line 2
      1 new_labels = ['Airbed', 'Couch', 'Futon', 'Pull out Sofa', 'Real Bed']
----> 2 df.groupby('Bed Type')[['Log Price']].boxplot(subplots=False,␣
 ↪figsize=(12, 10), rot=45)
      3 plt.xticks([1, 2, 3, 4, 5], new_labels)
      4 plt.show()

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pandas\cor·\groupby\gene
 ↪py:1415, in DataFrameGroupBy.__getitem__(self, key)
   1406 if isinstance(key, tuple) and len(key) > 1:
   1407     # if len == 1, then it becomes a SeriesGroupBy and this is actually
   1408     # valid syntax, so don't raise warning
   1409     warnings.warn(
   1410         "Indexing with multiple keys (implicitly converted to a tuple "
   1411         "of keys) will be deprecated, use a list instead.",
   1412         FutureWarning,
   1413         stacklevel=find_stack_level(),
   1414     )
-> 1415 return super().__getitem__(key)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pandas\cor·\base.
 ↪py:238, in SelectionMixin.__getitem__(self, key)
    236     if len(self.obj.columns.intersection(key)) != len(set(key)):
    237         bad_keys = list(set(key).difference(self.obj.columns))
--> 238         raise KeyError(f"Columns not found: {str(bad_keys)[1:-1]}")
    239     return self._gotitem(list(key), ndim=2)
    241 elif not getattr(self, "as_index", False):

KeyError: "Columns not found: 'Log Price'"
```

1.3 Correlación entre las diferentes columnas

1. Primero observamos las correlaciones en un mapa de calor del dataframe con los datos de los airbnb

[60]:
```
corr = np.abs(df.drop(['ID', 'Host ID'], axis=1).corr(numeric_only=True))
mask = np.zeros_like(corr, dtype= bool)
```

```
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(12,10))

sns.heatmap(corr, mask= mask, vmin = 0.0, vmax = 1.0, center = 0.5, linewidths␣
 ↪= .1, cmap= "YlGnBu", cbar_kws={"shrink": .8})

plt.show()
```

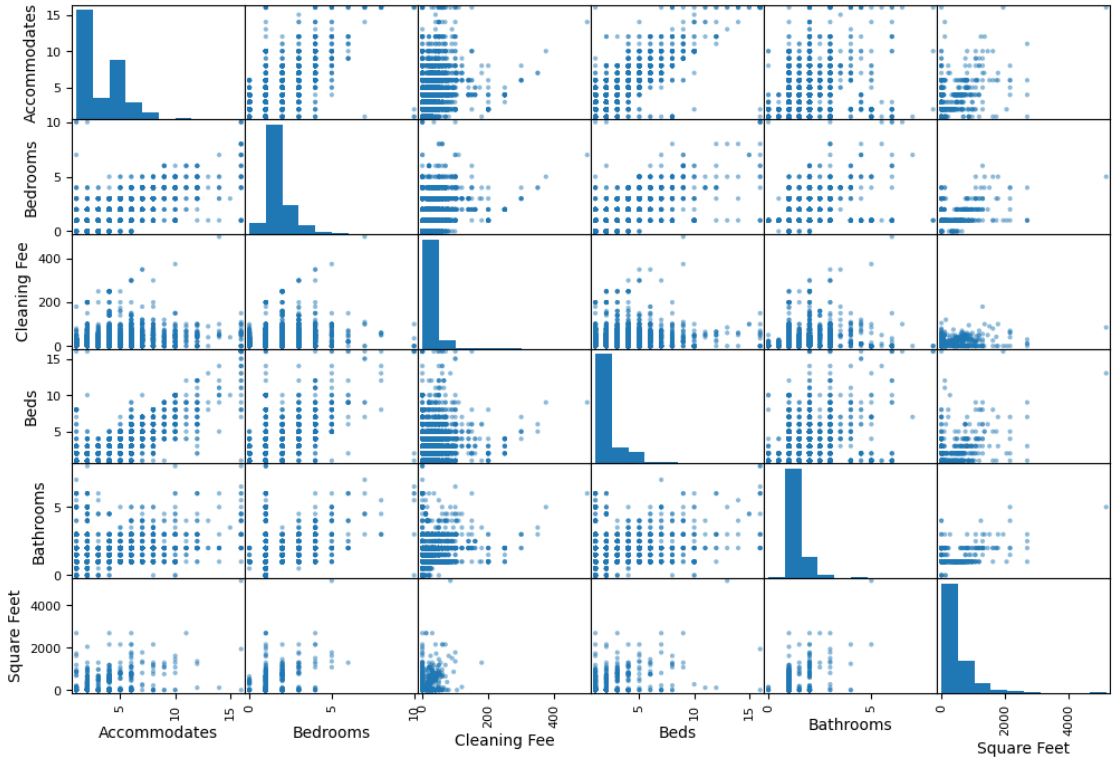


2. Obtengo la matriz de correlaciones, y someto al campo 'Price' con el resto de campos a ver en qué grado se correlacionan. Este valor oscila entre -1 y 1. Obviamente, un cierto campo consigo mismo tiene una correlación de 1.

[61]:
```
corr_matrix        = df.corr(numeric_only=True)
corr_matrix['Price'].sort_values(ascending=False)
```

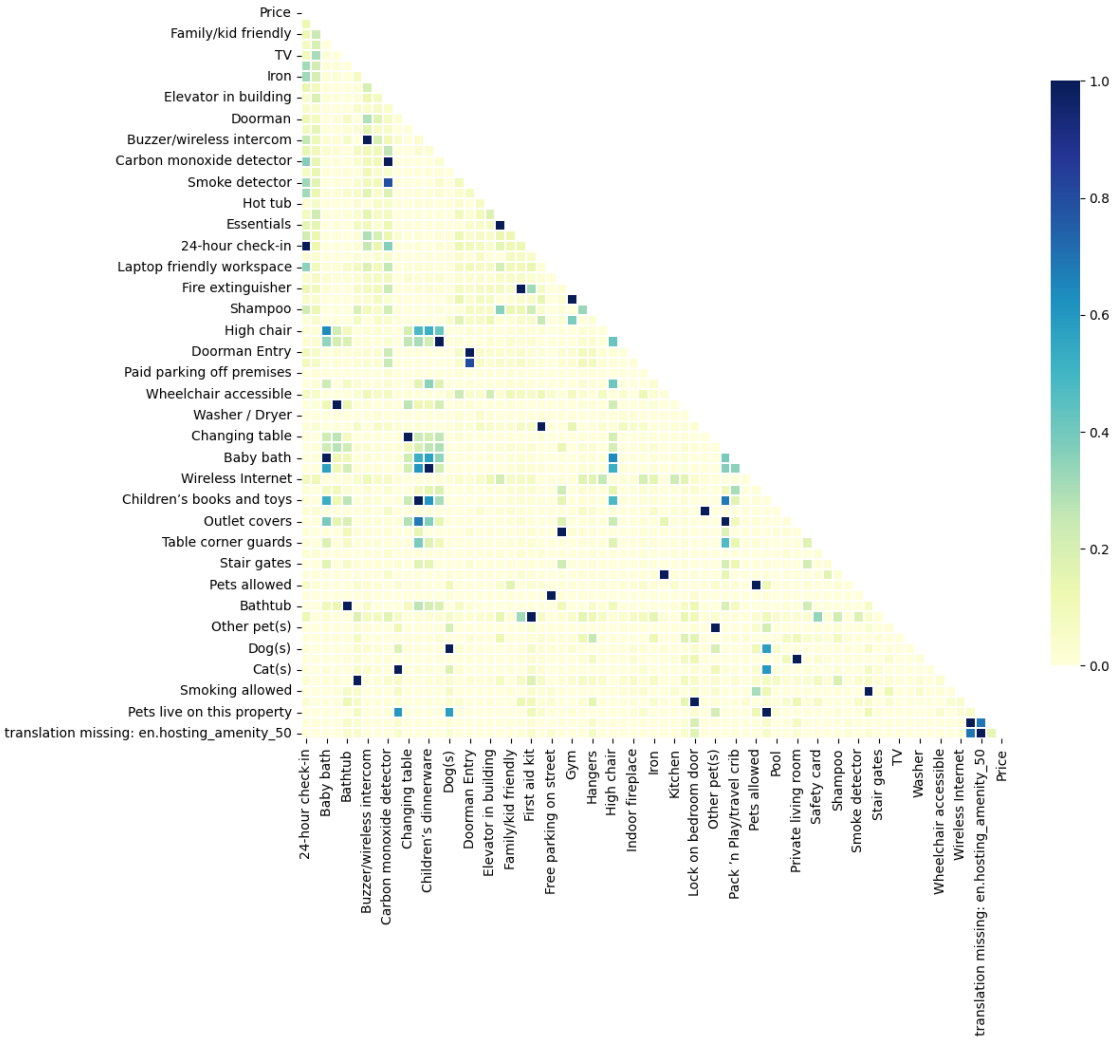
[61]:
```
Price                     1.000000
Log Price                 0.856872
Weekly Price              0.842458
Monthly Price             0.773039
Accommodates              0.577302
Bedrooms                  0.522443
Cleaning Fee              0.515124
Beds                      0.483216
Bathrooms                 0.346237
Square Feet               0.299669
Square Meters             0.299669
Review Scores Location    0.146184
Availability 365          0.074876
Latitude                  0.065767
Minimum Nights            0.031449
Occupancy                -0.022367
Longitude                -0.026428
ID                       -0.032347
Reviews per Month        -0.066801
Host ID                  -0.074860
Name: Price, dtype: float64
```

[62]:
```
attributes = ['Accommodates', 'Bedrooms', 'Cleaning Fee', 'Beds', 'Bathrooms',␣
 ↪'Square Feet']

scatter_matrix(df[attributes], figsize=(12, 8))
plt.show()
```



3. Correlaciones en un mapa de calor del dataframe con los datos de amenities y el precio

[63]:
```
amenities_columns = df_amenities.columns.tolist()
concat = pd.concat([df_amenities, df[['ID', 'Price']]], axis=1)

corr = np.abs(concat).corr(numeric_only=False)
mask = np.zeros_like(corr, dtype= bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(12,10))

sns.heatmap(corr.sort_values(by=['Price'], ascending=False), mask= mask, vmin =␣
 ↪0.0, vmax = 1.0, center = 0.5, linewidths = .1, cmap= "YlGnBu",␣
 ↪cbar_kws={"shrink": .8})

plt.show()
```



1.4 Conclusión

La conclusión que podemos sacar tras el análisis expuesto es que el precio es una variable sumamente compleja que se ve afectada por muchos factores, como el barrio, los servicios ofrecidos, el tamaño del alojamiento, el número de habitaciones, etc.

En ocasiones, los datos se han comportado como esperábamos, en otros casos no ha sido así.

Por supuesto, se podría hacer un análisis más profundo sobre algunas variables y añadir algunos KPI's más para el desarrollo. Sin embargo, consideramos este análisis un buen punto de partida para nuestro modelo predictivo.

Para finalizar esta parte del proyecto, vamos a eliminar algunas de las columnas creadas a lo largo del análisis que no nos son útilies para la continuación del trabajo y que solo molestarían.

Por otra parte, también vamos a crear una nuevo dataframe con los servicios (correspondientes a la columna original 'Amenities') que, tras todo el análisis, consideramos más relevantes para seguir

trabajando con ellos.

Al final, encontraremos el código para generar dos nuevos ficheros en formato csv: uno con el dataframe de Airbnb limpio y otro con el dataframe de los citados servicios. Ambos ficheros serán utilizados posteriormente.

[64]:
```
# Descartamos las columnas que no utilizaremos más adelante
df = df.drop('Square Meters', axis = 1)
df = df.drop('Log Price', axis = 1)
```

[65]:
```
# Creamos un nuevo dataframe con una columna por cada uno de los elementos de␣
 ↪la columna 'Amenities', eliminamos esta última del dataframe original
df_amenities = df.Amenities.fillna("").str.get_dummies(sep=',').astype(bool).
 ↪join(df['ID'])
df = df.drop('Amenities', axis=1)

# Nos quedamos con las columnas de 'Amenities' que consideramos relevantes
amenities_to_keep = ['ID', 'Self Check-In', 'Smartlock', 'Air conditioning',␣
 ↪'Elevator in building', 'Essentials', 'Internet', 'Heating', 'Pets allowed',␣
 ↪'Smoking allowed', 'Pool', 'TV', 'Kitchen', 'Gym']
df_amenities = df_amenities[amenities_to_keep]
```

[66]:
```
# Finalmente, si se desea generar los ficheros finales limpios, se puede␣
 ↪ejecutar esta celda
df.to_csv('../airbnb-listings_cleaned.csv', sep=';', index=False)
df_amenities.to_csv('../amenities.csv', sep=';', index=False)
```