Data_cleaning_and_exploring

February 14, 2023

[242]:
```
# Importamos las librerías necesarias
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
```

[243]:
```
# Cargamos el csv original
raw_file = '~/Desktop/Proyecto/raw/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 exploraremos el *dataset* propuesto y responderemos algunas preguntas que nos hemos planteado.

Trataremos con datos de Airbnb e intentaremos averiguar que datos tienen mayor impacto en el precio de las propiedades ofrecidas en alquiler.

Para eso tendremos que seguir algunos pasos:

1.1 1. Muestreo y exploración inicial de los datos:

1.1.1 ¿Con que datos trabajaremos?

Hemos decidido trabajar con los datos de *Madrid*. Por lo que procederemos a quedarnos con esos datos específicos y posteriormente filtrar las columnas que creemos que serán útiles.

[244]:
```
# Filtramos las filas y nos quedamos solo con las que contienen datos de Madrid
madrid_condition = df['State'].astype(str).str.contains('Madrid')
df = df[madrid_condition]
```

Elegimos las columnas listadas a continuación:

[245]:
```
# Variable con las columnas que vamos a utilizar
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']

# Nos quedamos solo con las columnas elegidas
df = df[columns_to_keep]
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 2. Normalización de las columnas

Algunas columnas tienen datos que no estan normalizados, como por ejemplo el código postal y el barrio.

[246]:
```
# Normalizamos los valores de la columna Zipcode que contiene códigos postales␣
 ↪erróneos y valores nulos
replace_values = {'nan': np.nan, '-': np.nan, '28': np.nan, '-' : np.nan,␣
 ↪'2802\n28012' : '28012', '28002\n28002': '28002', '28051\n28051' : '28051', \
                  'Madrid 28004': '28004', '2815' : '28015', '2805' : '28005'}

df = df.replace({'Zipcode': replace_values})
```

En el caso del barrio, esta columna tenía muchos valores nulos. Nuestra decisión fue rellenar los nulos con los encontrados en la columna Neighbourhood Cleansed y luego descartar esta columna.

[247]:
```
# Cambiamos los valores nulos de la columna Neighbourhood por el valor␣
 ↪correspondiente de la columna Neighbourhood Cleansed
df['Neighbourhood'] = df['Neighbourhood'].fillna(df['Neighbourhood Cleansed'])

# Eliminamos la columna Neighbourhood Cleansed
df = df.drop('Neighbourhood Cleansed', axis = 1)

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

[247]:
```
False    13198
Name: Neighbourhood, dtype: int64
```

1.3 ### Columnas de texto

Vamos a analizar las columnas que son cadenas de texto y a quitar tildes, dobles espacios, ect

[248]:
```
# 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 a conservar o a 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())

# 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]
```

```
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']
```

[248]:

	Neighbourhood	City	State	Property Type	\
1021	Embajadores	Madrid	Comunidad de Madrid	Loft	
1022	Embajadores	Madrid	Comunidad de Madrid	Apartment	
1023	Embajadores	Madrid	Comunidad de Madrid	Apartment	
1024	Embajadores	Madrid	Community of Madrid	Apartment	
1025	Embajadores	Madrid	Comunidad de Madrid	Apartment	
…	…	…	…	…	
492516	Cortes	Centro	Madrid	Apartment	
492517	Cortes	Madrid	Madrid	Apartment	
492518	Cortes	Madrid	Comunidad de Madrid	Apartment	
492519	Cortes	Madrid	Comunidad de Madrid	Apartment	
492520	Cortes	Madrid	Comunidad de Madrid	Apartment	

	Room Type	Bed Type	Cancellation Policy
1021	Entire home/apt	Real Bed	moderate
1022	Entire home/apt	Real Bed	strict
1023	Entire home/apt	Real Bed	moderate
1024	Entire home/apt	Real Bed	strict
1025	Entire home/apt	Real Bed	strict
…	…	…	…
492516	Private room	Futon	moderate
492517	Entire home/apt	Real Bed	flexible
492518	Private room	Real Bed	strict
492519	Entire home/apt	Real Bed	moderate
492520	Entire home/apt	Real Bed	strict

[13198 rows x 7 columns]

1.4 ### Conversión de tipos de datos

Algunos datos estaban en tipos incorrectos para su manejo, también hicimos las correcciones pertinentes.

- Host_Since a tipo fecha:

[249]:
```
# Conversión de las fechas de 'Host Since' en date
df['Host Since'] = pd.to_datetime(df['Host Since'])
```

- ID de la propiedad a numérico en vez de string.

[250]:
```
# Conversión del ID de la entrada en numérico en vez de string
df['ID'] = df['ID'].astype(int)
```

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

[251]:
```
df['Square Meters'] = df['Square Feet'] / 10.764
df['Square Meters']
```

[251]:
```
1021      NaN
1022      NaN
1023      NaN
1024      NaN
1025      NaN
           ..
492516    NaN
492517    NaN
492518    NaN
492519    NaN
492520    NaN
Name: Square Meters, Length: 13198, dtype: float64
```

1.5 ### Property Type

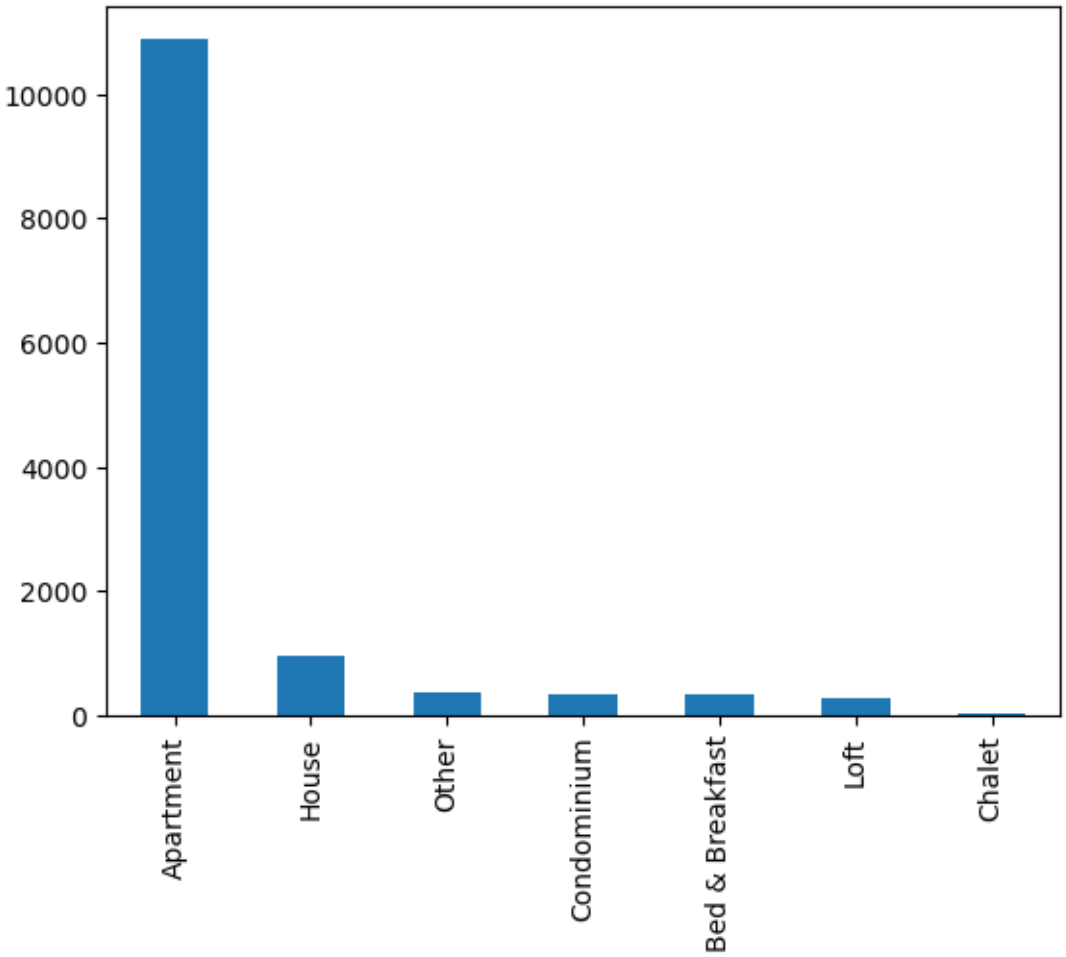
En el caso de esta variable, notamos que habia demasiadas clasificaciones con muy pocos registros, así que decidimos quedarnos con lo más relevante y agrupar el resto como "Otros".

Al ejecutar la siguiente celda se puede ver un gráfico en donde es obvio que la mayoría de las propiedades ofrecidas son *APARTAMENTOS*.

[252]:
```
# Variable con los valores de Property Type que nos interesan (más comunes)
valid_property_types = ['House', 'Apartment', 'Bed & Breakfast', 'Condominium',␣
 ↪'Loft', 'Chalet', 'Hostal']
# Agrupamos los valores de Property Type y creamos un valor Other para el resto
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()
```

[252]: <AxesSubplot: >



1.6 ### Amenities

Aquí realmente basamos nuestras principales dudas sobre el dataset

Estabamos interesadas en conocer si habia alguna amenidad que influyese en el precio o si el numero total de amenidades tenia algun impacto

Primero decidimos revisar cuales eran mas amenidades más frecuentes, cuál era el precio medio y la diferencia de media de precios por amenidad

1. Como primer punto creamos un dataframe llamado “Amenities” que contiene cada amenidad como columna.

[253]:
```
df_amenities = df.Amenities.fillna('').str.get_dummies(sep=',').astype(bool)
df_amenities
```

[253]:

	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	
…	…	…	…	
492516	False	True	False	
492517	True	True	False	
492518	False	False	False	
492519	True	True	False	
492520	True	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	
…	…	…	…	
492516	False	False	False	
492517	False	False	False	
492518	False	False	False	
492519	False	False	False	
492520	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	
…	…	…	…	…	
492516	True	False	False	False	
492517	True	False	True	False	
492518	False	False	False	False	
492519	True	False	False	False	
492520	True	False	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
…     …                 …     …                   …      …
492516  …               False   True                False    True
492517  …               False   True                False    True
492518  …               False  False                False   False
492519  …               False   True                False    True
492520  …               False   True                False    True

        Washer / Dryer  Wheelchair accessible  Window guards  \
1021             False                  False          False
1022             False                  False          False
1023             False                  False          False
1024             False                  False          False
1025             False                  False          False
…                …                      …              …
492516           False                  False          False
492517           False                   True          False
492518           False                  False          False
492519           False                  False          False
492520           False                   True          False

        Wireless Internet  translation missing: en.hosting_amenity_49  \
1021                 True                                         True
1022                 True                                        False
1023                 True                                        False
1024                 True                                        False
1025                 True                                        False
…                   …                                            …
492516               True                                         True
492517               True                                        False
492518               True                                        False
492519               True                                        False
492520               True                                        False

        translation missing: en.hosting_amenity_50
1021                                          True
1022                                         False
1023                                         False
1024                                         False
1025                                         False
…                                            …
492516                                        True
492517                                       False
492518                                        True
492519                                       False
492520                                       False
```

```
[13198 rows x 67 columns]
```

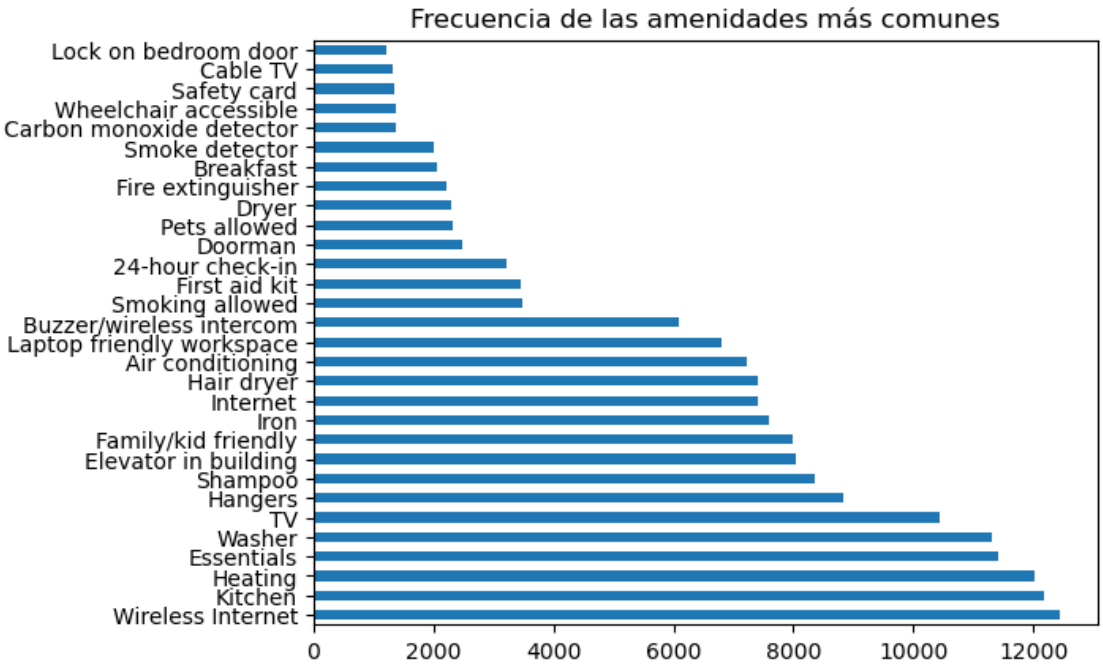
2. Luego calculamos la frecuencia de cada una de las amenities y generamos un gráfico de barras.

[254]:
```
#Calculo de 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 con traslation missing. Las quitaremos␣
 ↪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 las amenidades más comunes")
df_amenities_frequency.transpose()[True].sort_values(ascending=False).head(30).
 ↪plot.barh()
```

[254]:
```
<AxesSubplot: title={'center': 'Frecuencia de las amenidades más comunes'}>
```

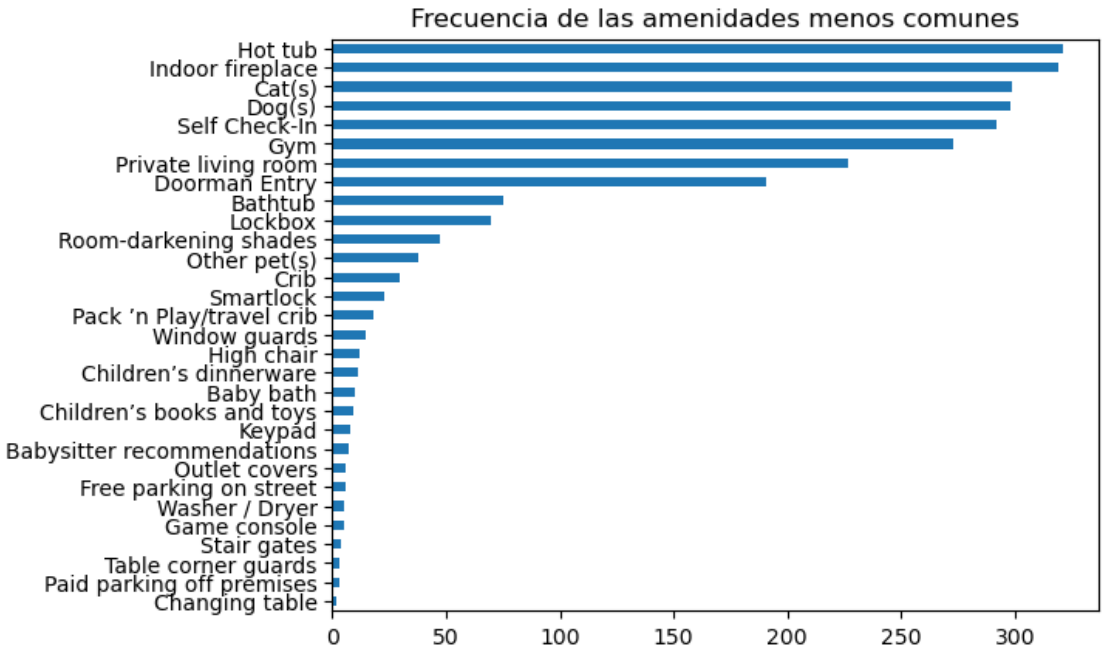


Con esto podemos concluir que la mayoría de las propiedades cuentan con internet inalámbrico,

cocina, calefacción, lavadora, ect.

[255]:
```
plt.title("Frecuencia de las amenidades menos comunes")
df_amenities_frequency.transpose()[True].sort_values(ascending=True).head(30).
↪plot.barh()
```

[255]:
```
<AxesSubplot: title={'center': 'Frecuencia de las amenidades menos comunes'}>
```



Por el contrario es mas raro encontrar Mesa cambiadora, Parking fuera de la localidad, protectores para las esquinas de las mesas, protectores para escaleras…

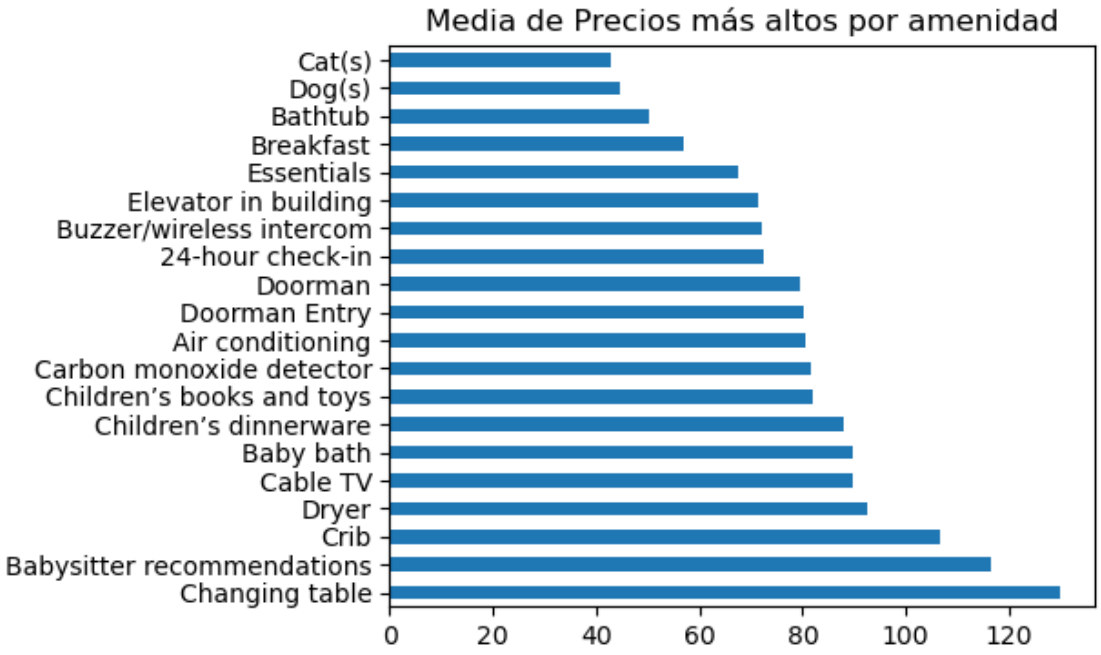
Hay una clara tendencia a no tener propiedades acondicionadas para bebés/niños pequeños.

3. Ahora verificamos la media de precio por tener una amenidad en específico:

[256]:
```
#Añadimos la columna 'Price' a nuestro dataframe.
df_amenities_price = df_amenities.join(df['Price'])
#Inicialicamos un dataframe para 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.
df_amenities_price_mean = df_amenities_price_mean.transpose()
plt.title("Media de Precios más altos por amenidad")
```

```
df_amenities_price_mean[True].head(20).sort_values(ascending=False).plot.
↪barh(figsize=(5, 4))
```

[256]: <AxesSubplot: title={'center': 'Media de Precios más altos por amenidad'}>



Tras ver esto, lo que suele tener más impacto en la media del precio suelen ser cosas asociadas a habitaciones preparadas para infantes como la mesita cambiadora y las recomendaciones de niñeras.

4. Nos parecio bien verificar *cuales amenidades generaban una mayor variación en el precio medio*

[257]:
```
#Añadimos a nuestro DataFrame una columna 'Diff' que registra la diferencia de␣
↪media de precio entre tener el amenity 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)
```

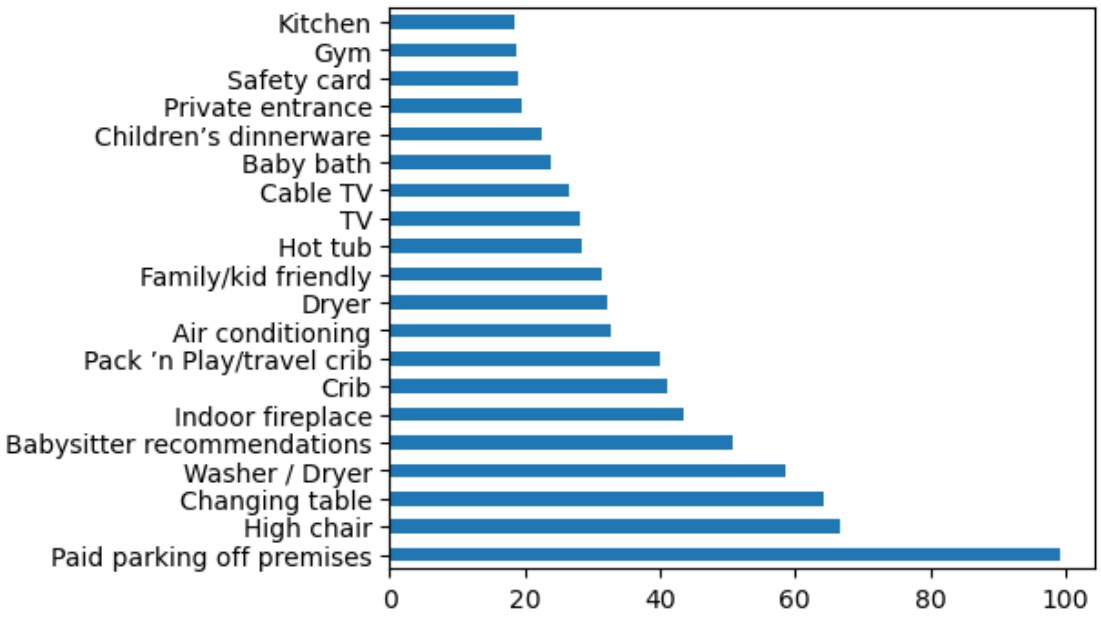
[257]:

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

[258]:
```
df_amenities_price_mean['Diff'].sort_values(ascending=False).head(20).plot.
↪barh(figsize=(5, 4))
```

[258]: <AxesSubplot: >



Tenemos ya que lo mas diferencia hace parece ser el Parking pagado fuera del terreno de la propiedad, la silla alta (para bebés), la mesa cambiadora, lavadora/secadora...

Al hacer esto, tenemos algunas amenities que tienen algun impacto significativo, sin embargo quedaría pendiente analizar estas variables más a fondo para tener conclusiones más fiables.

Puede ser que simplemente sean amenidades muy raras y casualmente estén en inmuebles con un mayor precio.

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

[259]:
```
#Preparamos nuestro dataframe Amenities_Count
df_amenities_count = pd.DataFrame()
#Le añadimos la columna con el conteo de amnidades 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
```

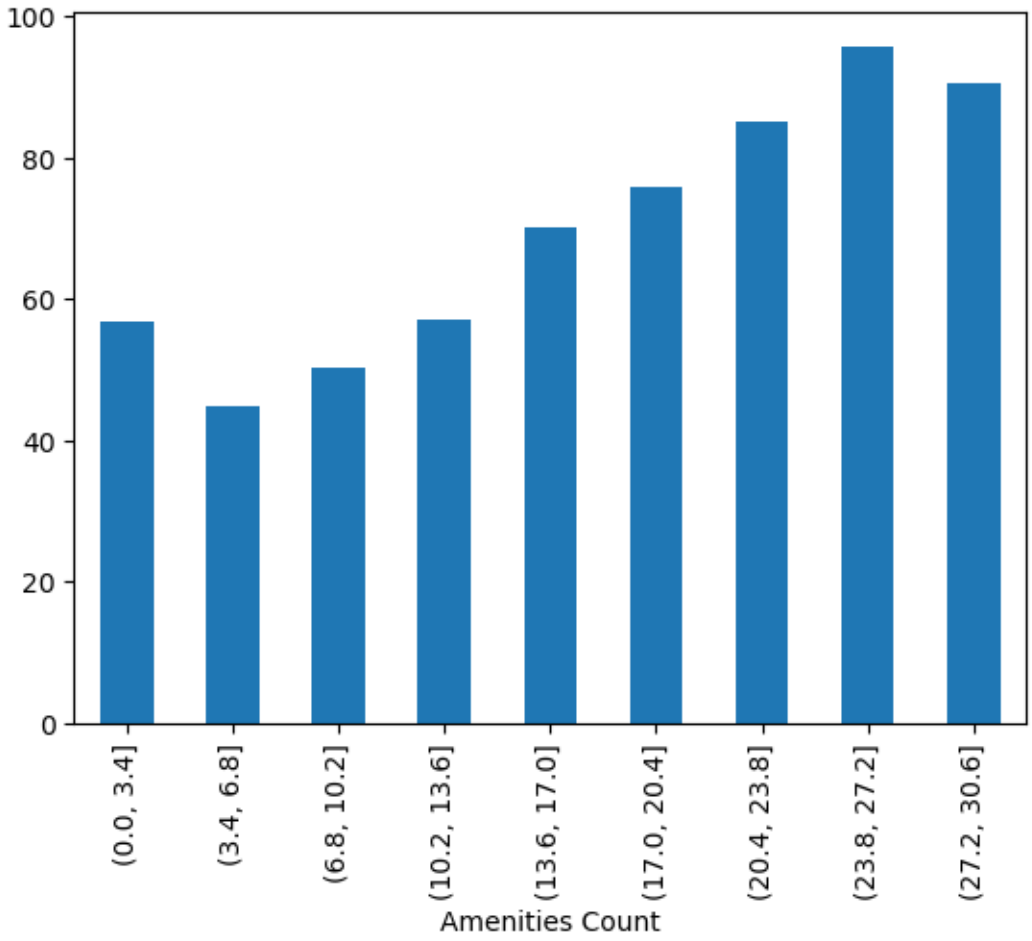
[259]:

	Amenities Count	Price
1021	15	50.0
1022	16	50.0
1023	18	77.0
1024	10	50.0
1025	10	95.0
…	…	…
492516	19	55.0
492517	22	80.0
492518	10	50.0
492519	17	70.0
492520	21	89.0

[13198 rows x 2 columns]

[260]:
```
# Visualizamos la variación del precio medio de las propiedades en función de␣
 ↪cuántas amenities 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()
```

[260]:
```
<AxesSubplot: xlabel='Amenities Count'>
```



Parece ser que el precio varia, alcanzando su minimo las propiedades que tienen 3-7 amenities y se dispara al ofrecer 24-27 amenidades.

La tendencia es subir de precio mientras más amenidades ofrezcas.

1.7 ### 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 propìedad que se encuentra disponible 365 días al año pero solo fue rentada los fines de semana. Es decir, estuvo rentada 104 días de 365.

Eso nos daria una tasa de ocupación del 28,49%

Este dataset no nos da información sobre la tasa de ocupación pero se puede obtener relacionando datos como la media de reviews al mes, el minimo de noches y la disponibilidad anual

Pongamos el caso de una propiedad con un minimo de noches de 2, con 4 reviews al mes y una disponibilidad de 365 días al año.

Si dividimos las (reviews * minimo de noches) * 12 / 365. Esto nos daria el ratio de ocupación, asumiendo que cada individuo que va a la propiedad deja una review.

Decidimos explorar si esto nos daba algun tipo de dato valioso.

Como primer paso definimos una función para calcular esto:

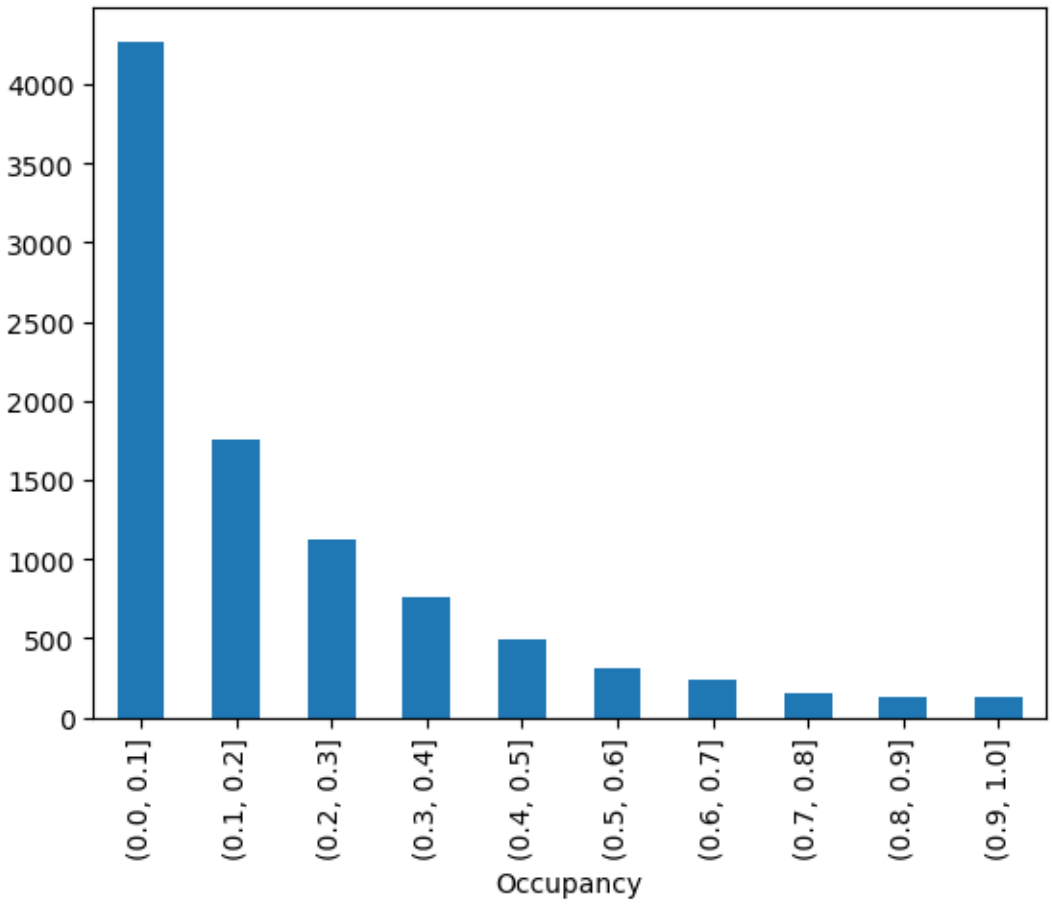
[261]:
```
def calculate_occupancy(reviews_month, min_nights, availability):
    return ((reviews_month * min_nights) * 12 / availability)
```

[262]:
```
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)

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

[262]:
```
<AxesSubplot: xlabel='Occupancy'>
```



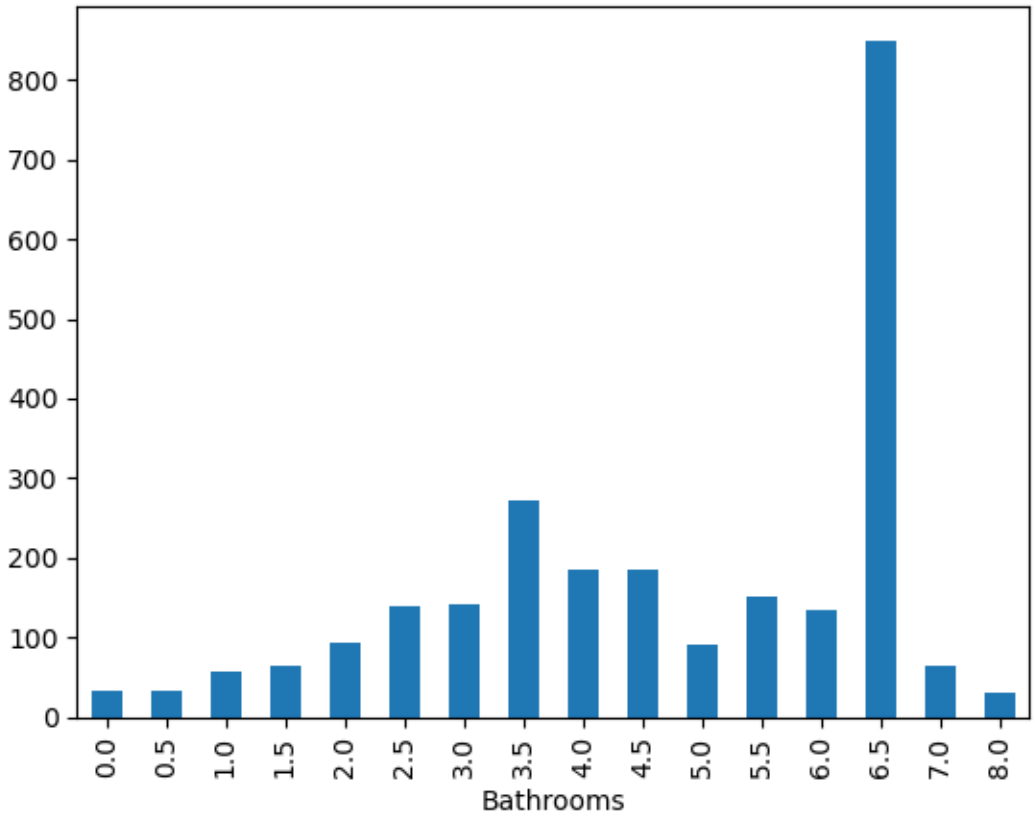
Finalmente decidimos que este dato no era fiable y no continuamos indagando en utilidad.

1.8 ### Bathrooms

Analizamos un poco la variable de los baños para tener un insight de si afectsa el precio final o no.

[263]:
```
#Creamos un plot que nos muestra la media de precio por número de baños.
df.groupby('Bathrooms')['Price'].mean().plot.bar()
```

[263]:
```
<AxesSubplot: xlabel='Bathrooms'>
```

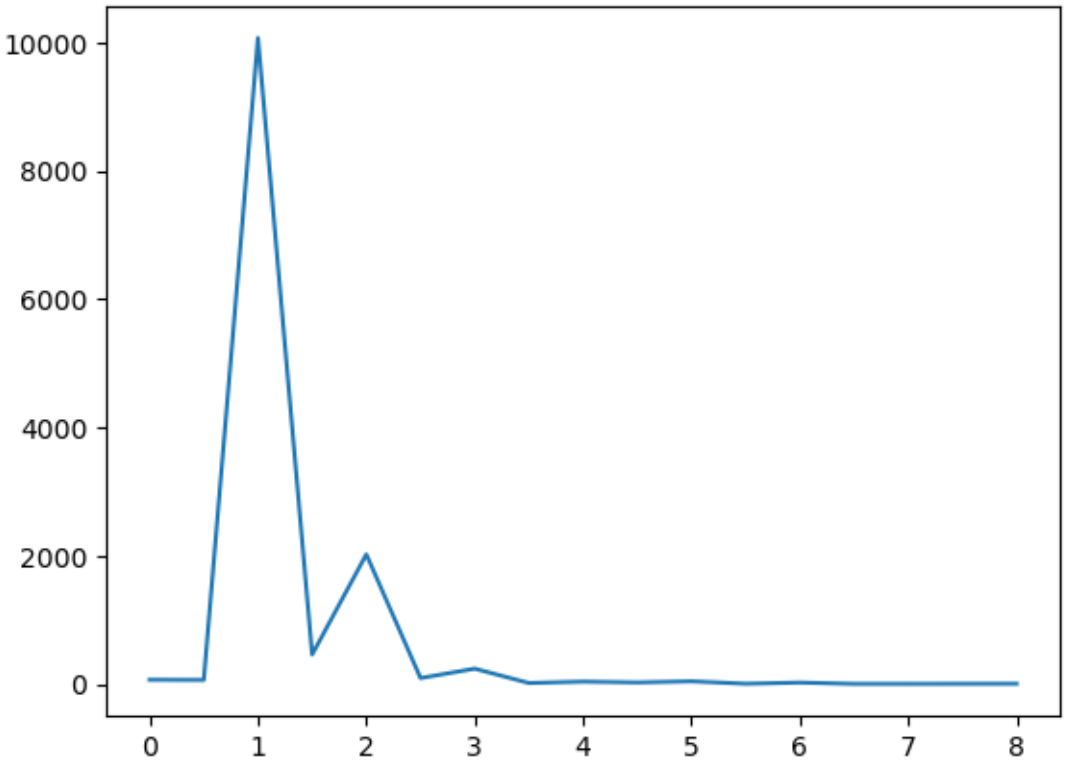


Vemos un poco la distribución de la media de precios, que sube hasta llegar a 3.5. En el caso de muchos baños es bastante probable que solo se encuentren en propiedades muy grandes con precios bastante elevados.

Decidimos ver la distribución de la cantidad de baños por propiedad

[264]:
```
df['Bathrooms'].value_counts().sort_index().plot()
```

[264]:
```
<AxesSubplot: >
```



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

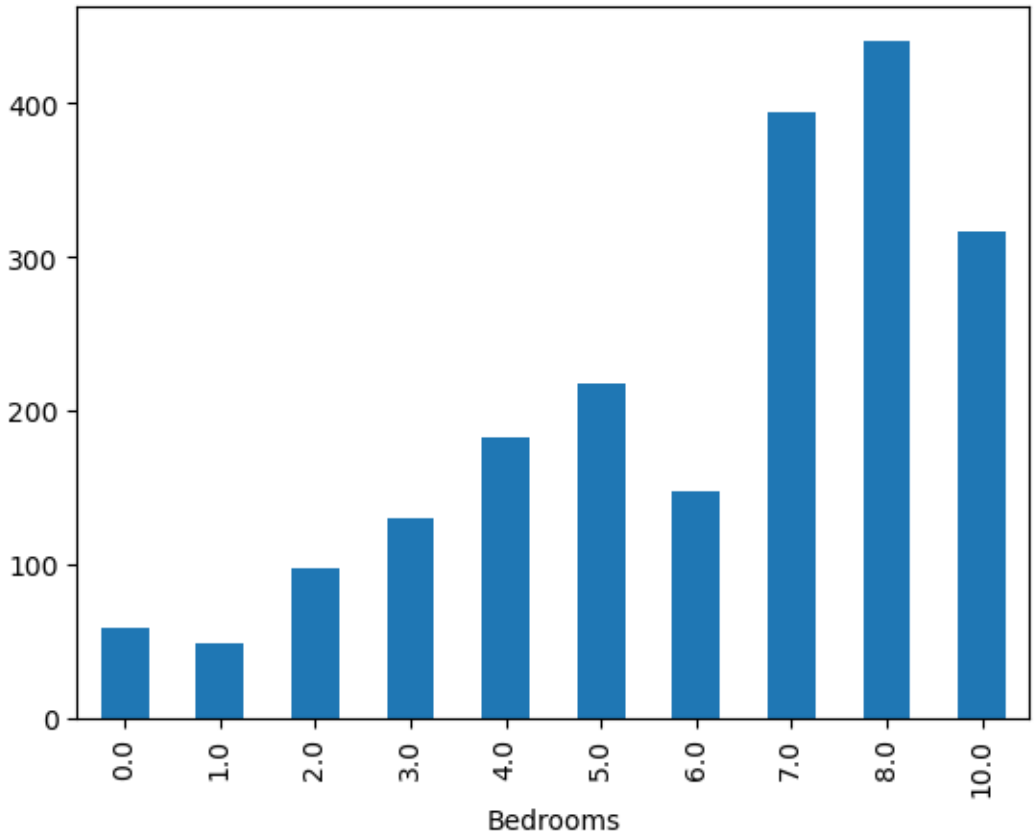
1.9 # Bedrooms

Hicimos un analisis similar al anterior

Primero por precio medio por numero de habitacines:

[265]:
```
df.groupby('Bedrooms')['Price'].mean().plot.bar()
```

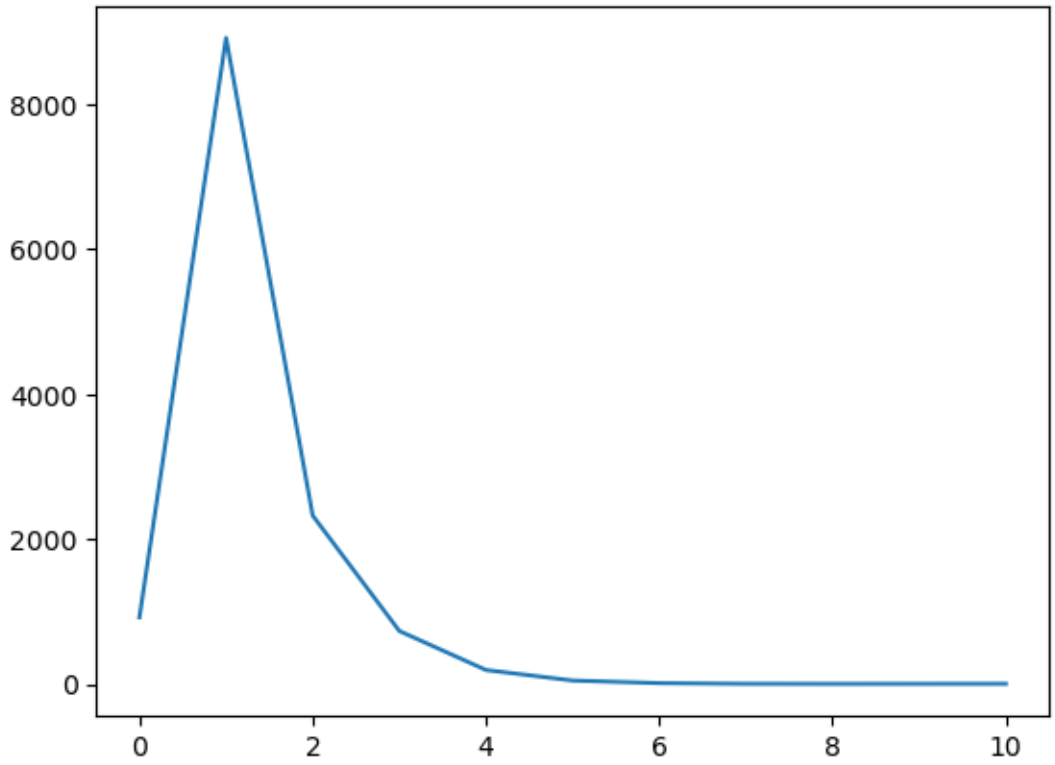
[265]:
```
<AxesSubplot: xlabel='Bedrooms'>
```



Lógicamene el precio sube por cantidad de habitaciones. En el número 6 puede que baje porque sea un hostal con varias habitaciones, lo que lo haría mas barato.

[266]:
```
df['Bedrooms'].value_counts().sort_index().plot()
```

[266]: <AxesSubplot: >



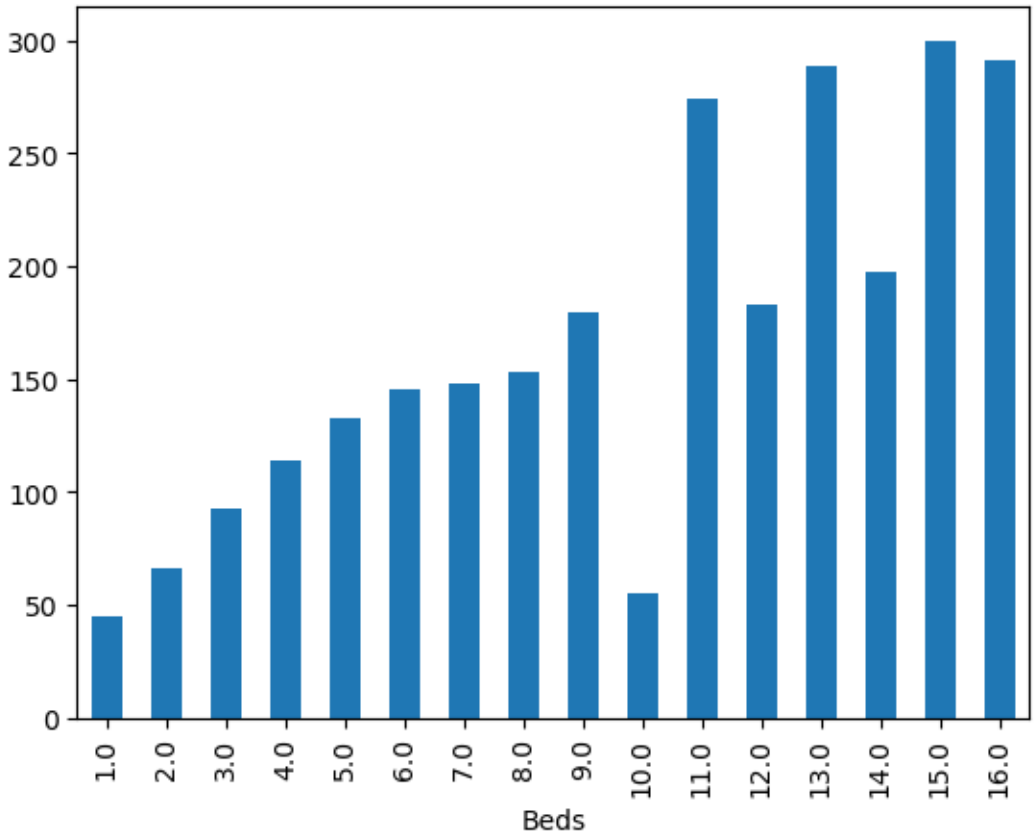
Sin embargo, la mayor parte de las propiedades tienen de 1 a 3 habitaciones.

1.10 ### Beds

Hicimos algo similar tambien con el número de camas.

[267]:
```
df.groupby('Beds')['Price'].mean().plot.bar()
```

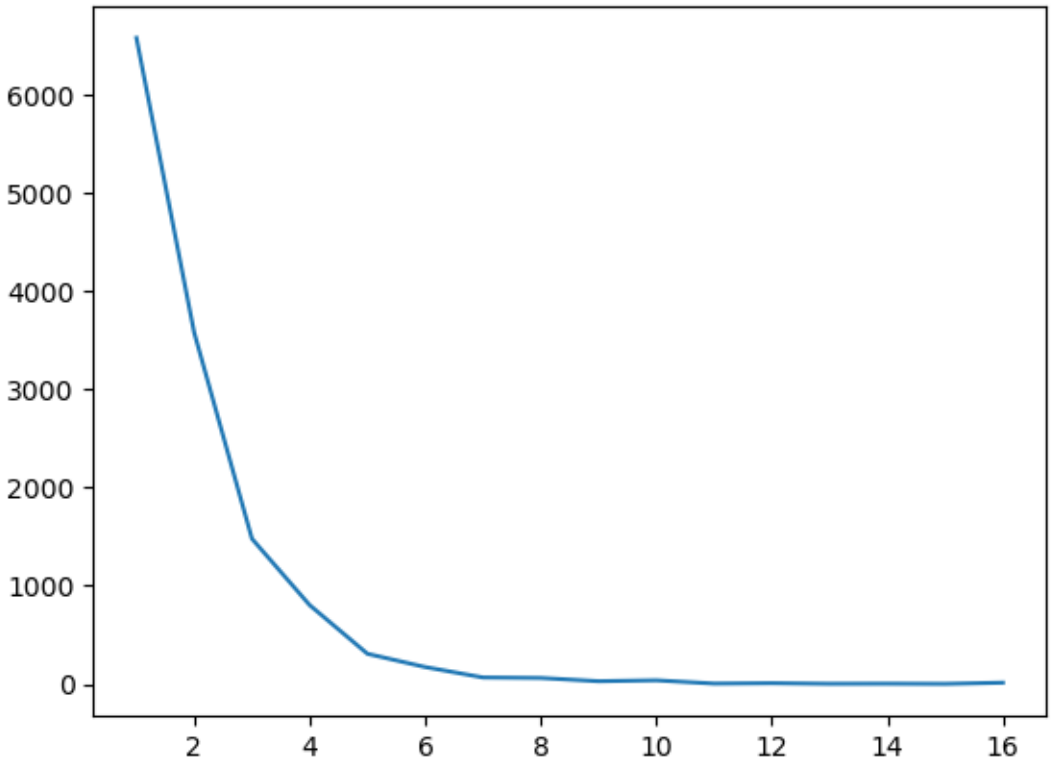
[267]:
```
<AxesSubplot: xlabel='Beds'>
```



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

[268]:
```
df['Beds'].value_counts().sort_index().plot()
```

[268]:
```
<AxesSubplot: >
```



Y que la mayor parte de los alojamientos tienen entre 1-6 camas.

En líneas generales los resultados de *Bathrooms, Bedrooms y Beds* son los esperados. A mayor cantidad de estos features tiende a subir el precio y la mayoría de los alojamientos tienen entre 1-3 baños, camas o habitaciones.

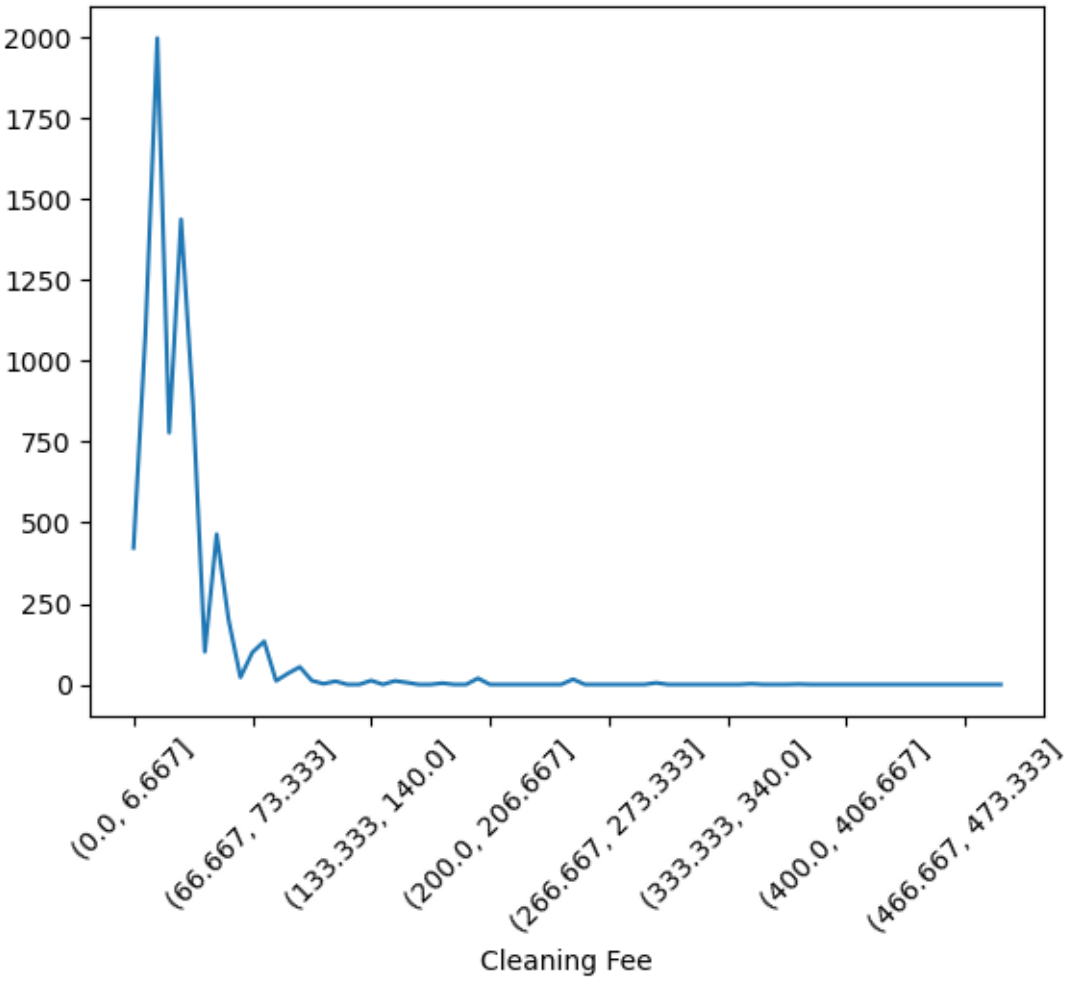
1.11 ### Cleaning Fee

También miramos el Cleaning Fee:

[269]:
```
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)
```

[269]:
```
<AxesSubplot: xlabel='Cleaning Fee'>
```



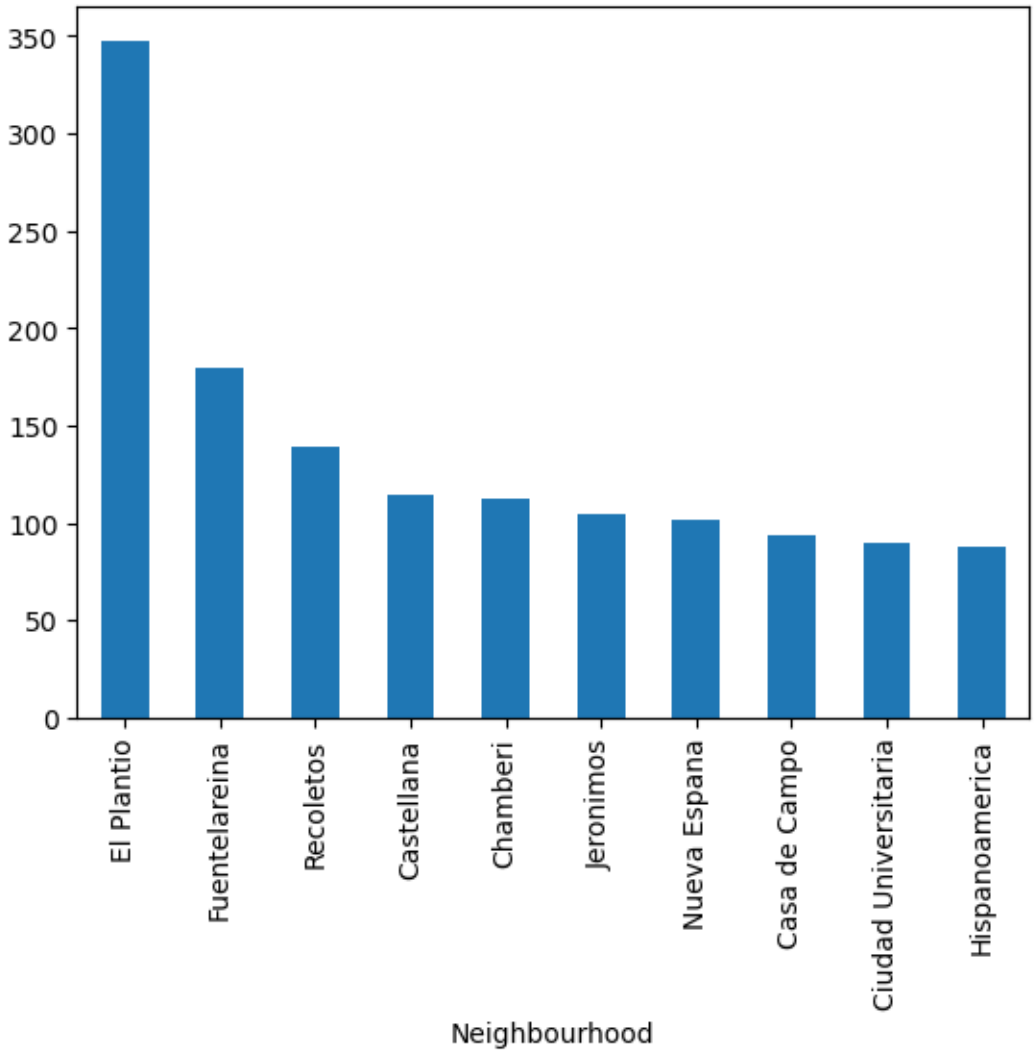
La mayoría tiene un cleaning fee entre 0€ y 75€

1.11.1 Neighbourhood

Decidimos averiguar cuales eran los barrios mas caros por precio por noche:

[270]:
```
df.groupby('Neighbourhood')['Price'].mean().sort_values(ascending=False).
↪head(10).plot.bar()
```

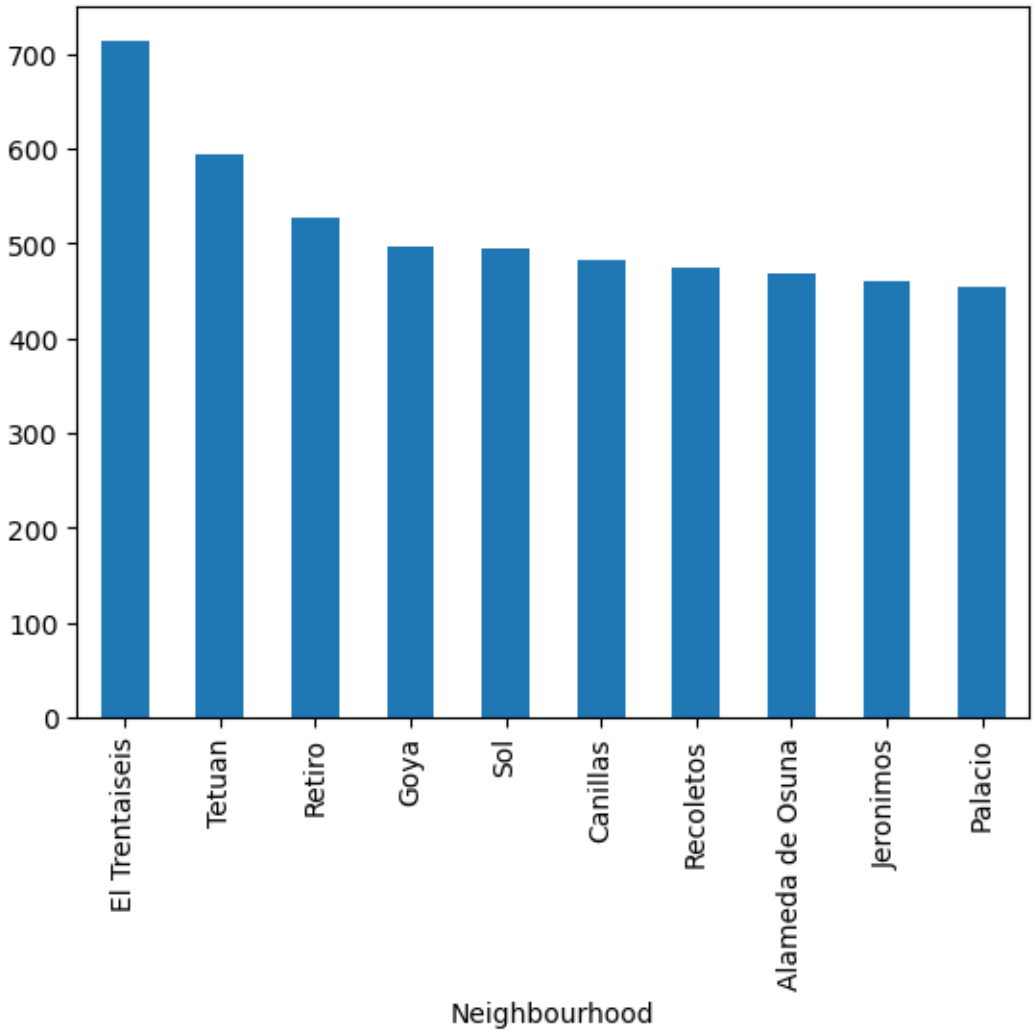
[270]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```



Tambien por precio semanal.

[271]:
```
df.groupby('Neighbourhood')['Weekly Price'].mean().sort_values(ascending=False).
↪head(10).plot.bar()
```

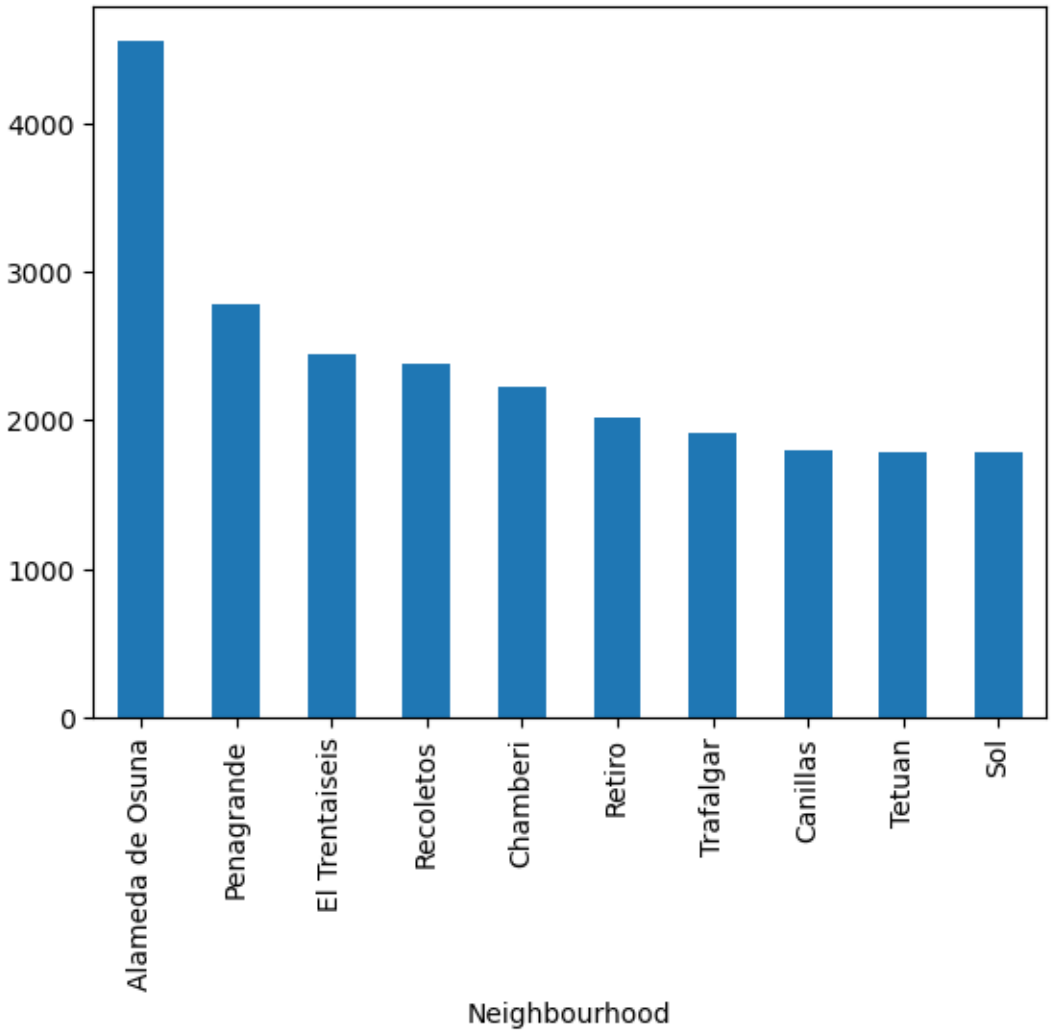
[271]: <AxesSubplot: xlabel='Neighbourhood'>



Y por precio mensual.

[272]:
```
df.groupby('Neighbourhood')['Monthly Price'].mean().
↪sort_values(ascending=False).head(10).plot.bar()
```

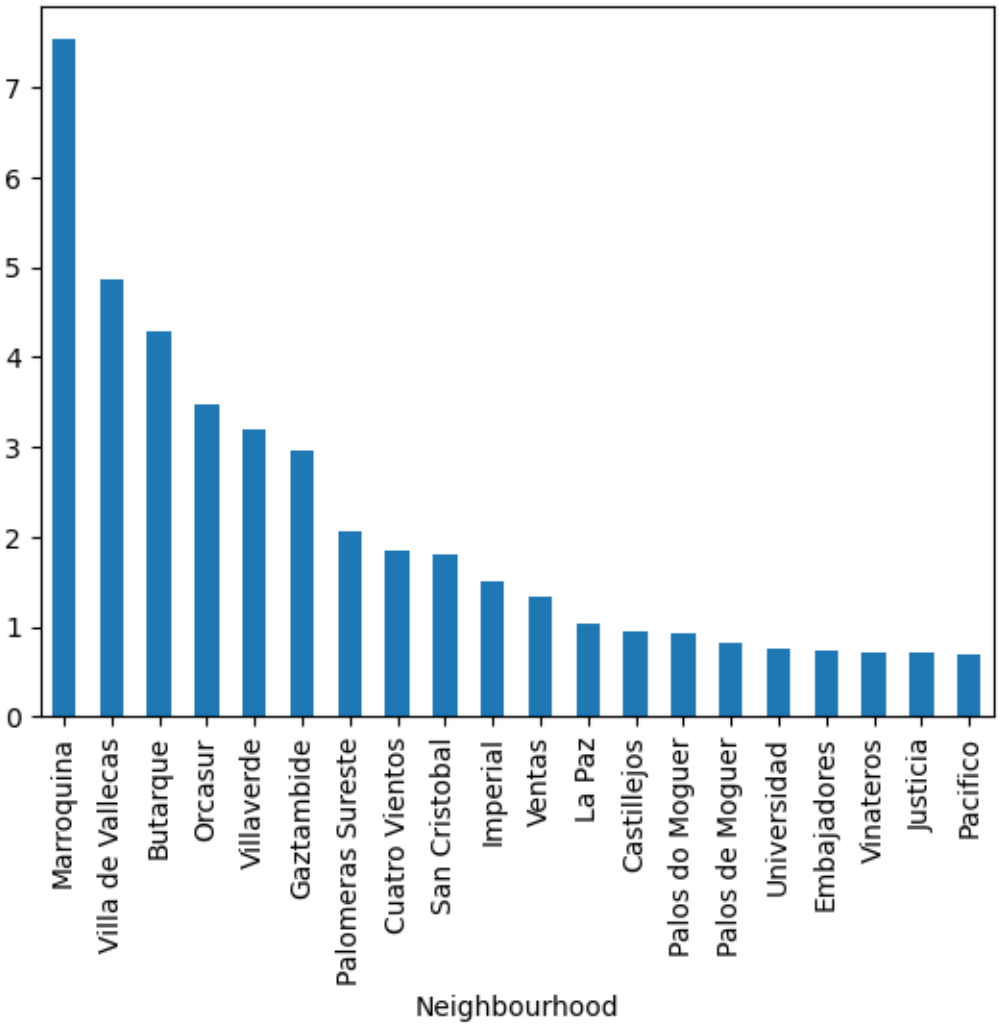
[272]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```



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

[273]:
```
df.groupby('Neighbourhood')['Occupancy'].mean().sort_values(ascending=False).
↪head(20).plot.bar()
```

[273]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```



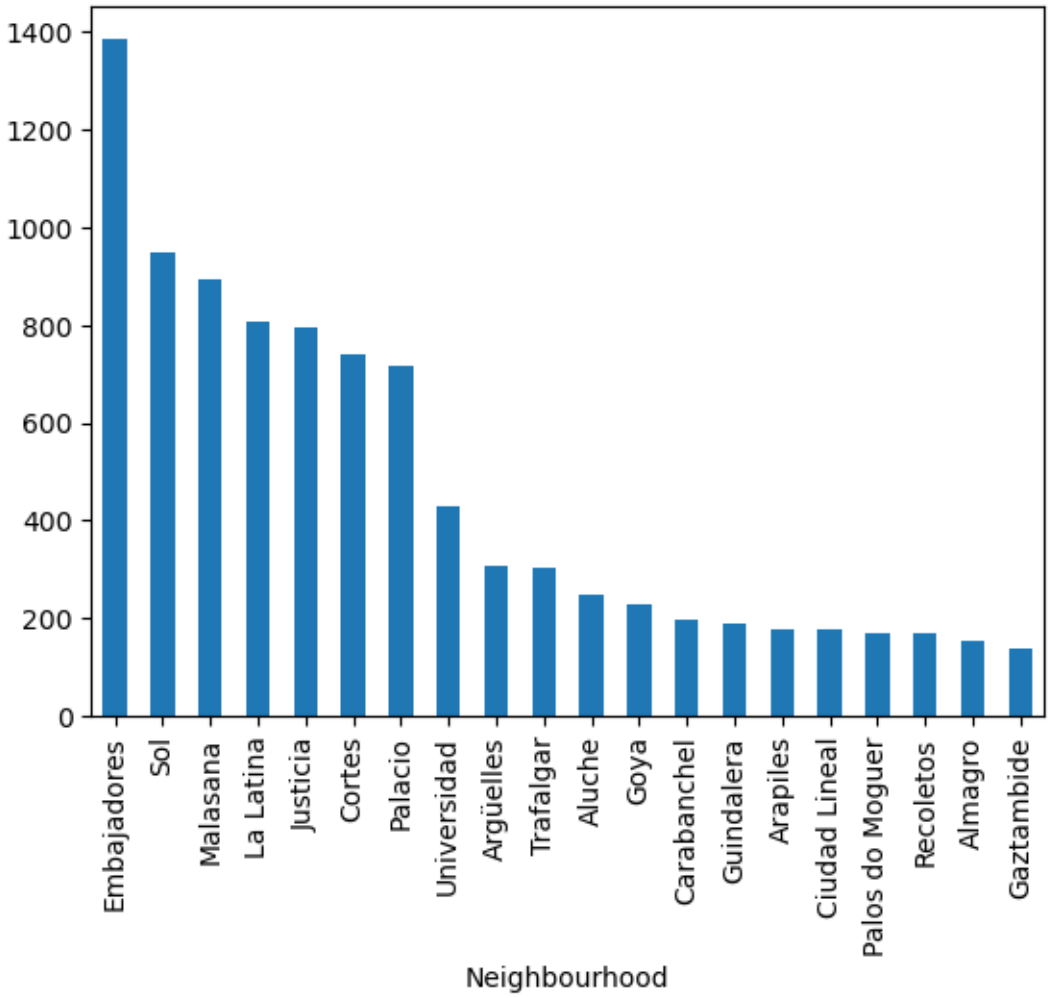
En este caso el ratio de ocupación también puede estar relacionado con la cantidad de propiedades disponibles en ese barrio en especifico.

Con lo cual la tasa de ocupación por barrio sigue sin ser un indicador muy fiable.

Sin embargo nos preguntamos que barrio tenia mas alojamientos en alquiler:

[274]:
```
df.groupby('Neighbourhood').size().sort_values(ascending=False).head(20).plot.
↪bar()
```

[274]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```

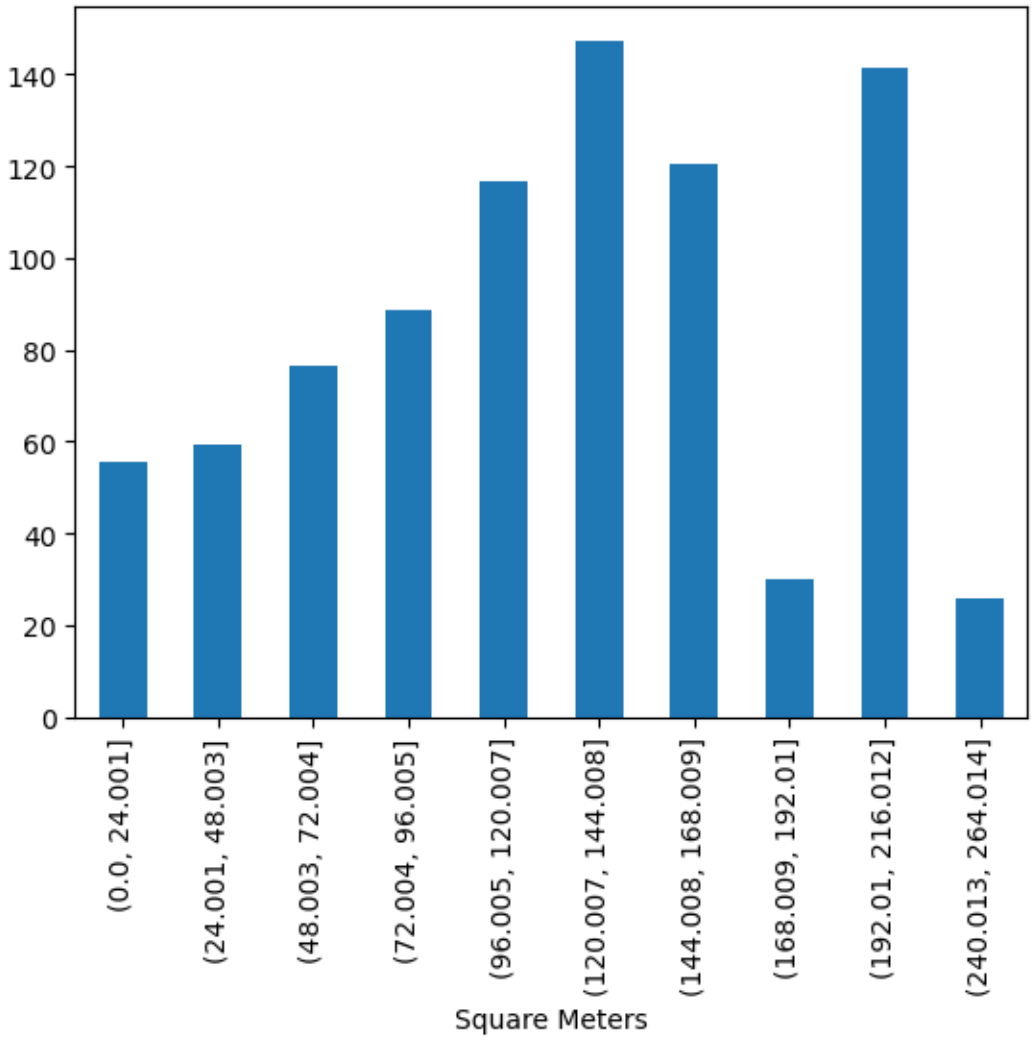


1.11.2 Square Feet / Square Meters

Quisimos averiguar la media de metros cuadrados por propiedad

[275]:
```
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()
```

[275]: <AxesSubplot: xlabel='Square Meters'>

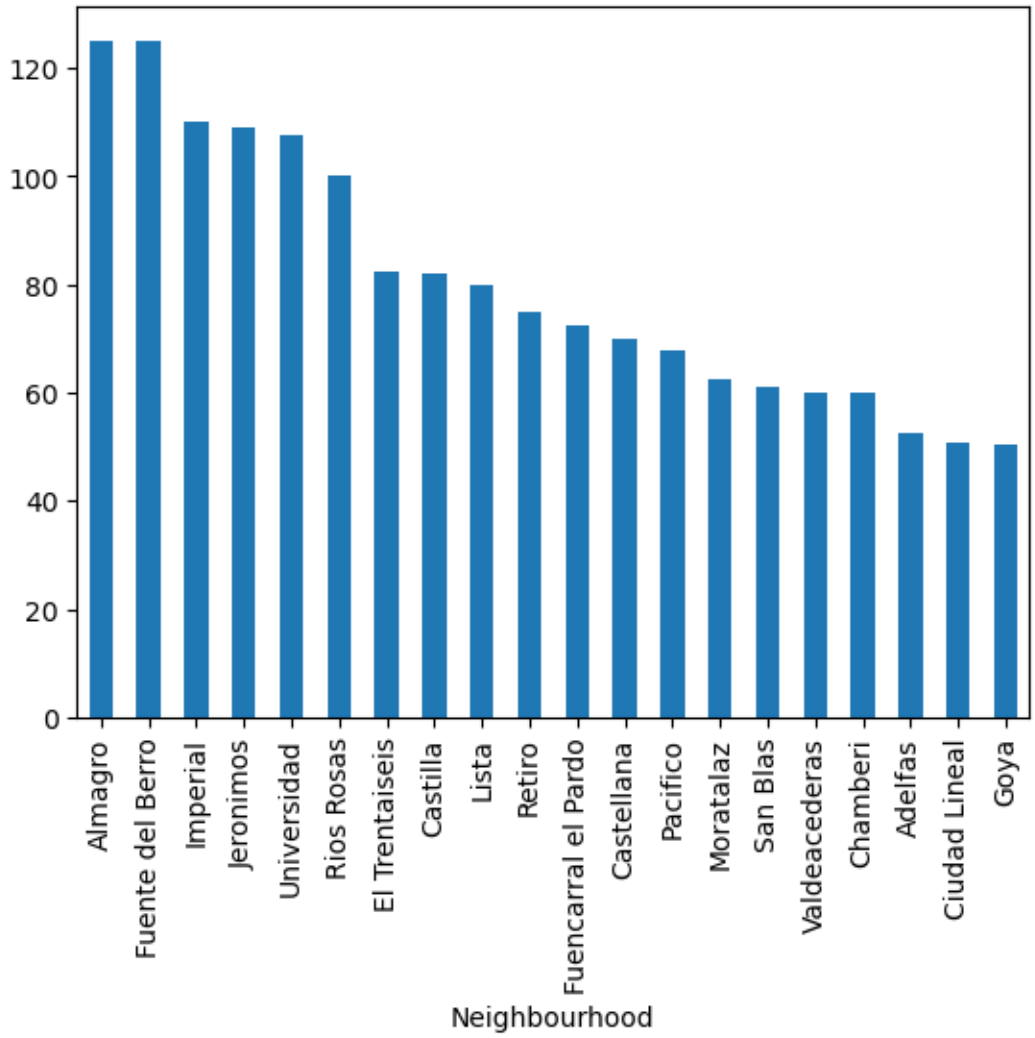


La mayoria podria estar entre 96 y 170 metros cuadrados.

También nos intereso saber cuales eran los varios que ofrecian pisos más grandes:

[276]:
```
df.groupby('Neighbourhood')['Square Meters'].mean().
↪sort_values(ascending=False).head(20).plot.bar()
```

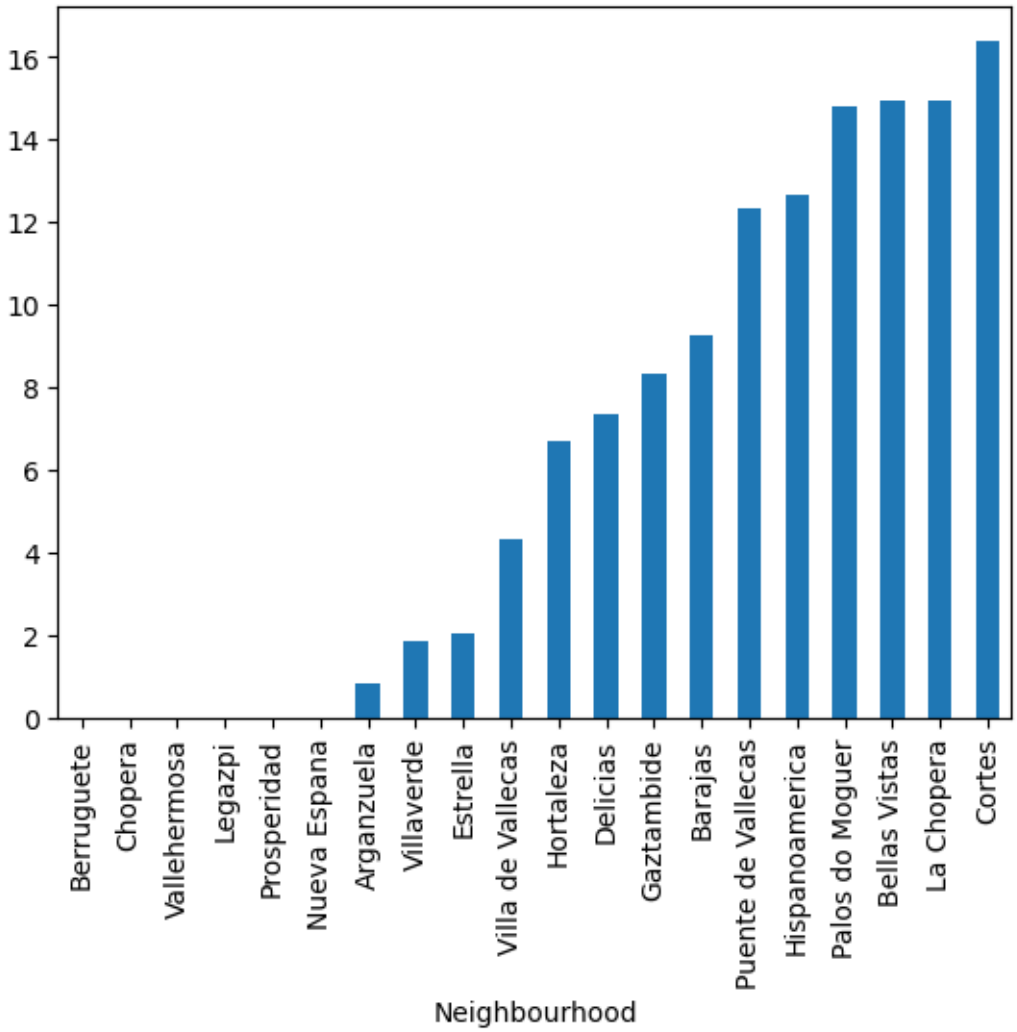
[276]: <AxesSubplot: xlabel='Neighbourhood'>



O mas pequeños

[277]:
```
df.groupby('Neighbourhood')['Square Meters'].mean().dropna().
↪sort_values(ascending=True).head(20).plot.bar()
```

[277]:
```
<AxesSubplot: xlabel='Neighbourhood'>
```



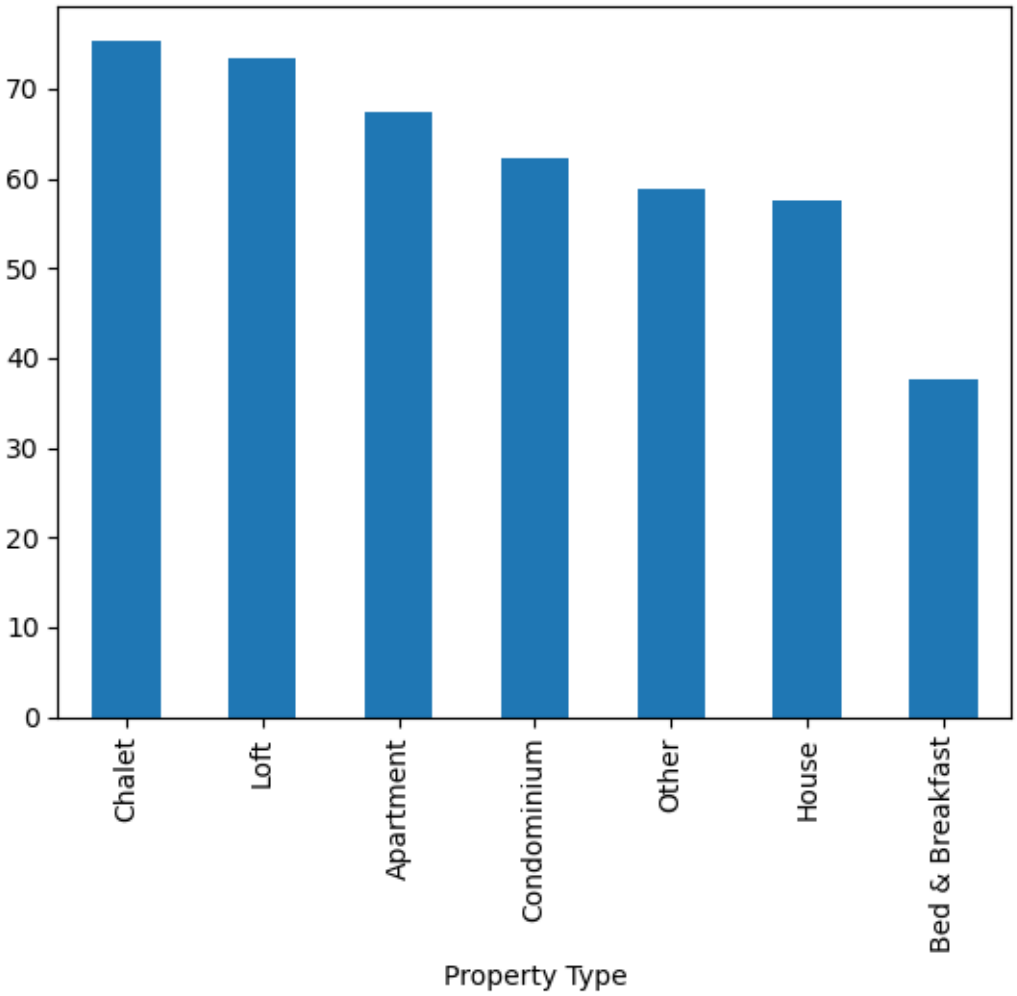
1.11.3 Property Type

Finalmente decidimos explorar los tipos de propiedades ofertados

Primero verificando el precio medio por tipo de propiedad

[278]:
```
df.groupby(['Property Type'])['Price'].mean().sort_values(ascending=False).
↪head(20).plot.bar()
```

[278]:
```
<AxesSubplot: xlabel='Property Type'>
```



Y verificando 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'

[279]:
```
df['Log Price'] = real_price.apply(np.log)
df['Log Price']
```

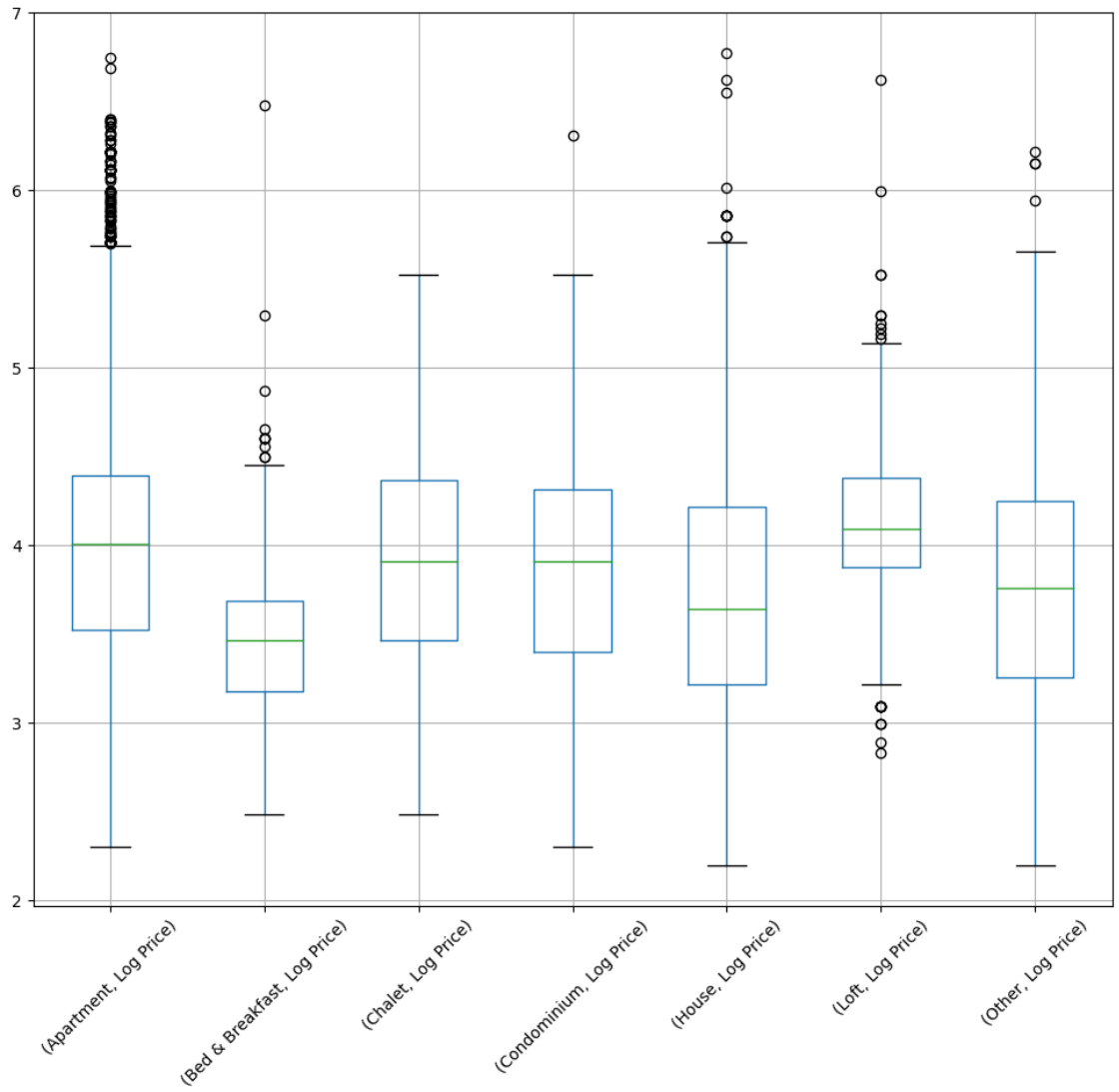
[279]:
```
1021      3.912023
1022      3.912023
1023      4.343805
1024      3.912023
1025      4.553877
            …
492516    4.007333
492517    4.382027
```

```
492518    3.912023
492519    4.248495
492520    4.488636
Name: Log Price, Length: 13198, dtype: float64
```

Hacemos el analisis según el tipo de propiedad:

[280]:
```
df.groupby('Property Type')[['Property Type', 'Log Price']].
  ↪boxplot(subplots=False, figsize=(12, 10), rot=45)
```

[280]: <AxesSubplot: >

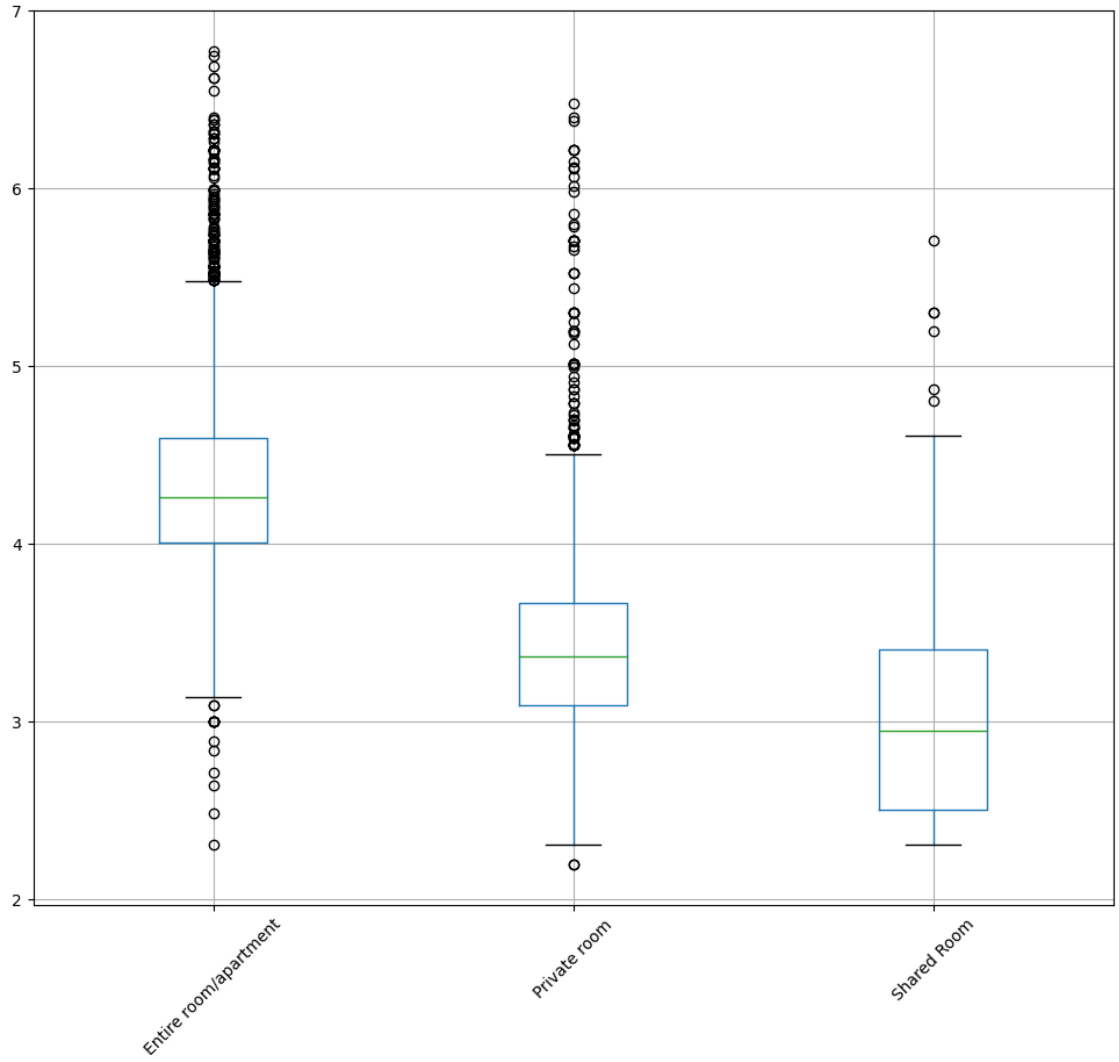


Esto lo que nos dice es que la mediana se mantiene mas o menos constante, entre los 35 y 40 €.

Lo hicimos según el tipo de habitación:

[281]:
```
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
```

[281]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



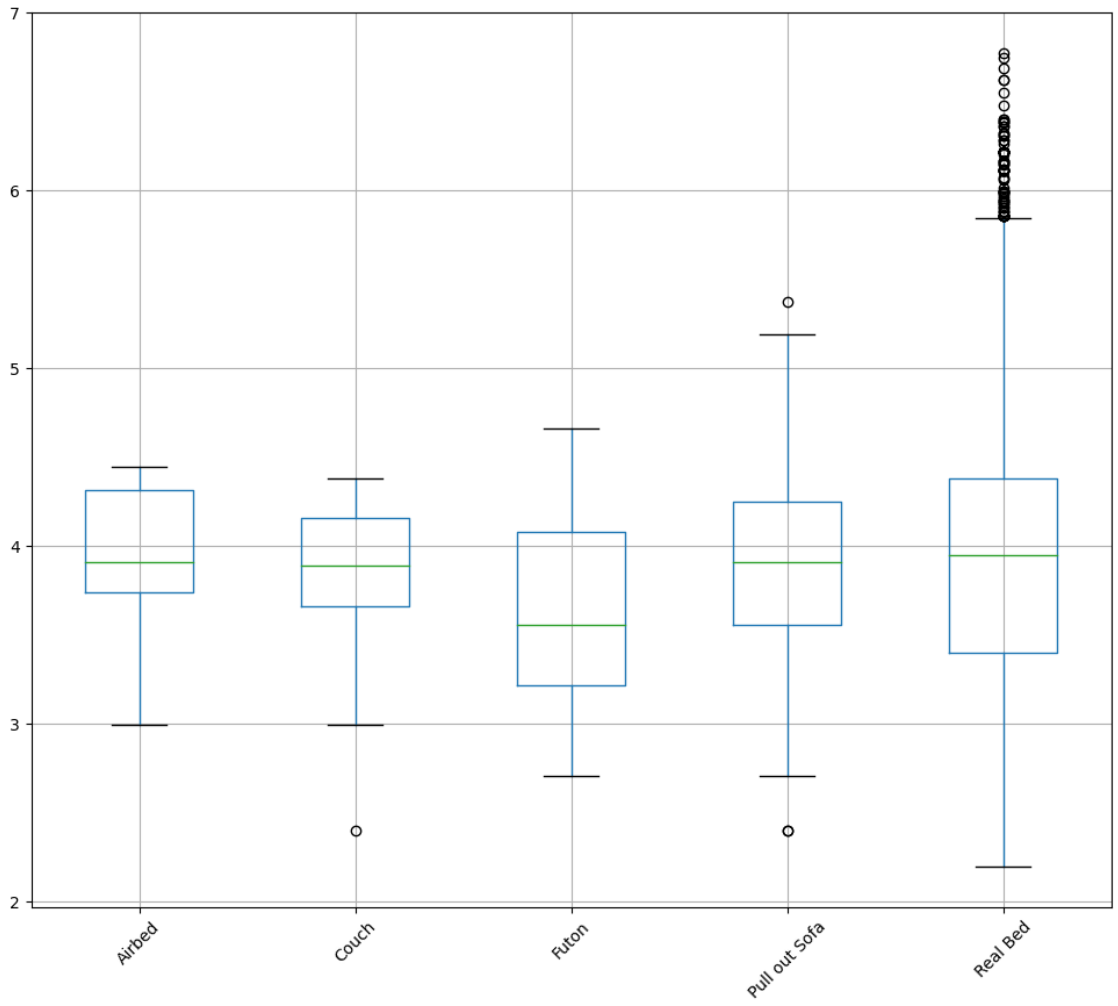
Como era de esperar, un departamento entero es mas costoso que un cuarto privado y un cuarto compartido

Ninguna sorpresa aquí.

Hicimos lo mismo por tipo de cama

[282]:
```
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
```

[282]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



1.12 Conclusión

La conclusión que pudimos sacar es que el precio es una variable sumamente compleja que se ve afectada por varios factores. El barrio, las amenidades, el tamaño del alojamiento, el número de habitaciones.

En ocasiones los datos se han comportado como esperabamos, 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 esto un buen punto de partida para nuestro modelo predictivo.