Práctica final(primera parte) de Aprendizaje automático

por Ramon Saleta-Piersanti Besga

Índice

Para guiar un poco la lectura del cuaderno en **.ipynb** comenzaré seccionando las partes en las que consiste la propia práctica:

1. Exploración de datos: Breve exploración de las diferentes variables de la práctica para seleccionar las relevantes.
2. Tratamiento de datos: Conocido como ***Data Wrangling*** dicho apartado consistirá en detectar la cantidad de datos faltantes(NaN) o aquellos que entorpezcan la predicción(outliers) y tomar una estrategia al respecto. Aquí también se transformarán variables, ya sea categorizarlas o aplicar algún otro procedimiento.
3. Aplicación de modelos: Se utilizarán distintos **modelos predictivos** comprendidos en el marco del **Aprendizaje Automático** a fín de encontrar la mejor predictabilidad.

Problema a resolver

A partir del archivo 'airbnb.csv' el cual tiene **17608 entradas de hospedajes**(filas) con una cantidad de **74 variables**(columnas) se propone predecir un **precio orientativo** para un **hospedaje recién introducido** en el sistema. Para dicho fin, antes hay que cribar el *Dataset* y sanearlo de posibles problemas.

---

1. Exploración de datos

En primer lugar, se importa las librerías a utilizar.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
```

Se carga el *Dataset* de Airbnb.

In [2]:
```
df=pd.read_csv('airbnb.csv')
df.head()
```

Out[2]:

	id	listing_url	scrape_id	last_scraped	name	descrip
0	11547	https://www.airbnb.com/rooms/11547	20200919153121	2020-09-21	My home at the beach	Sun, joy, quality, b & peac

	id	listing_url	scrape_id	last_scraped	name	descrip
1	100831	https://www.airbnb.com/rooms/100831	20200919153121	2020-09-21	HOUSE IN MALLORCA - WiFi(ET-3045)	<b spaceH situate c
2	105891	https://www.airbnb.com/rooms/105891	20200919153121	2020-09-20	VILLAGE HOUSE WITH POOL: IDEAL FOR FAMILIES	The hous street o outskirts c
3	106833	https://www.airbnb.com/rooms/106833	20200919153121	2020-09-20	Villa with a big pool in Mallorca	<b spaceThi restored ol
4	130669	https://www.airbnb.com/rooms/130669	20200919153121	2020-09-20	Room great apartment	Locate reside neighbour and 1

5 rows × 74 columns

Para apreciar las dimensiones del *Dataset*, se usará la propiedad .shape, el primer índice corresponde a las filas(entradas) mientras que el segundo a las columnas(variables).

In [3]:
```
df.shape
```

Out[3]:
```
(17608, 74)
```

Efectivamente, el archivo contiene un registro de 17608 entradas con 74 variables distintas. De entre todas esas variables, se debe seleccionar aquellas relevantes ante el problema. Para tal fin, se realizará una rápida exploración de los nombres de las variables y que formato tienen.

In [4]:
```
# Realizamos una lista con los nombres de las columnas, puesto que el DataFrame trab
colnames=df.columns
```

In [5]:
```
# Se irá representando las columnas de 10 en 10 hasta abarcar las 74.
df[colnames[:10]].head()
```

Out[5]:

	id	listing_url	scrape_id	last_scraped	name	descrip
0	11547	https://www.airbnb.com/rooms/11547	20200919153121	2020-09-21	My home at the beach	Sun, joy, quality, b & peac
1	100831	https://www.airbnb.com/rooms/100831	20200919153121	2020-09-21	HOUSE IN MALLORCA - WiFi(ET-3045)	<b spaceH situate c

	id	listing_url	scrape_id	last_scraped	name	descrip
2	105891	https://www.airbnb.com/rooms/105891	20200919153121	2020-09-20	VILLAGE HOUSE WITH POOL: IDEAL FOR FAMILIES	The hous street o outskirts c
3	106833	https://www.airbnb.com/rooms/106833	20200919153121	2020-09-20	Villa with a big pool in Mallorca	<b space Thi restored ol
4	130669	https://www.airbnb.com/rooms/130669	20200919153121	2020-09-20	Room great apartment	Locatec reside neighbour and 1

En un principio, las 10 primeras variables contiene información poco útil. Serán eliminadas del *Dataset*.

In [6]:
```
df[colnames[10:20]].head()
```

Out[6]:

	host_name	host_since	host_location	host_about	host_response_time	host_response_rate	host_a
0	Daniel	2009-10-02	Balearic Islands, Spain	.	within an hour	100%	
1	Miguel	2011-04-23	Mallorca	Somos una pareja con los mismos gustos e inter...	NaN	NaN	
2	Bartomeu	2011-05-01	Ariany, Balearic Islands, Spain	Hola!. Resido en una casa de campo de un puebl...	within a few hours	100%	
3	Xisco	2011-05-02	Palma de Mallorca, Balearic Islands, Spain	I'm Xisco. I love Mallorcan way of life.	within a day	100%	
4	Nick	2011-05-30	Palma de Mallorca, Balearic Islands, Spain	NaN	within a day	100%	

Listado rápido sobre las variables 11-20:

- host_name: El nombre del hospedador no es relevante a nivel numérico. Podría ser un indicador sobre si alguien es de Mallorca, pero requeriría mucho filtrado.
- host_since: Podría ofrecer información útil a nivel de servicio, se debería convertir el texto a variable date_time.
- host_location: En principio no interesa. Si el hospedador es de la zona, puede ser valorable.
- host_about: Breve descripción, *a priori* descartable.
- host_response_time: Se puede categorizar, interesante.

- host_response_rate: Si se quita el '%' es una variable cuantificable a tener en cuenta.
- host_acceptance_rate: Lo mismo que en el caso anterior.
- host_is_superhost: El 'Super hospedador' es una categoría a tener en cuenta.
- host_thumbnail_url: Para el estudio de la práctica no nos interesa evaluar una imagen.
- host_picture_url: Lo mismo que en el caso anterior.

In [7]:
```
df[colnames[20:30]].head()
```

Out[7]:

	host_neighbourhood	host_listings_count	host_total_listings_count	host_verifications	host_has_prof
0	NaN	0.0	0.0	['email', 'phone', 'reviews', 'jumio', 'govern...	
1	NaN	1.0	1.0	['email', 'phone', 'facebook', 'reviews', 'jum...	
2	NaN	2.0	2.0	['email', 'phone', 'facebook', 'reviews', 'jum...	
3	NaN	1.0	1.0	['email', 'phone', 'reviews', 'jumio', 'offlin...	
4	NaN	3.0	3.0	['email', 'phone', 'reviews']	

Listado rápido sobre las variables 21-30:

- host_neighbourhood: El barrio del hospedador. No interesa.
- host_listings_count: Cantidad de veces que dicha persona ha sido hospedadora. Importante.
- host_total_listings_count: Lo mismo que la anterior.
- host_verifications: Verificación del hospedador, parece relacionada con 'host_identity_verified'.
- host_has_profile_pic: Se puede categorizar, interesante.
- host_identity_verified: Categoría verdadero/falso relacionada con 'host_verifications', usaremos esta.
- neighbourhood: Barrio de la vivienda. Relacionada con 'neighbourhood_cleansed'
- neighbourhood_cleansed: Municipio de la vivienda. Hay 53 en Mallorca, requiere mucha computación pero es relevante.
- neighbourhood_group_cleansed: Es una variable repleta de NaN.
- latitude: Puede ser interesante si se transforma juntamente con 'longitude'. Posible uso en *Decision Trees*.

In [8]:
```
df[colnames[30:40]].head()
```

Out[8]:

	longitude	property_type	room_type	accommodates	bathrooms	bathrooms_text	bedrooms	be
0	2.48182	Entire apartment	Entire home/apt	2	NaN	1 bath	1.0	

	longitude	property_type	room_type	accommodates	bathrooms	bathrooms_text	bedrooms	be
1	3.16255	Entire house	Entire home/apt	8	NaN	3 baths	4.0	
2	3.07165	Entire townhouse	Entire home/apt	6	NaN	2 baths	3.0	
3	3.30121	Entire villa	Entire home/apt	4	NaN	1 bath	2.0	
4	2.60333	Private room in apartment	Private room	2	NaN	1 bath	1.0	

Listado rápido sobre las variables 31-40:

- longitude: Puede ser interesante si se transforma juntamente con 'latitude'. Posible uso en *Decision Trees*.
- property_type: Tipo de vivienda, relacionada con 'room_type'.
- room_type: Similar a la anterior, hay 4 categorías. En todo caso se usará esta.
- accommodates: 'Aforo' de la vivienda. Interesante.
- bathrooms: Repleta de NaN.
- bathrooms_text: Si se quita la parte de texto, es una variable relevante.
- bedrooms: Dormitorios, es interesante. Relacionada con 'beds' y seguramente sea preferible escoger esa.
- beds: Camas, al estar relacionada con 'bedrooms' seguramente sea más interesante, puesto que admite más valores.
- amenities: Comodidades, sin entrar en detalle sobre cuales hay, se puede probar a cuantificar las presentes.
- price: Precio, requiere quitar el signo de dollar($) y cuidar la coma de los miles. Es el *Output* de la práctica.

In [9]:
```
df[colnames[40:50]].head()
```

Out[9]:

	minimum_nights	maximum_nights	minimum_minimum_nights	maximum_minimum_nights	minim
0	5	60	5	5	
1	7	365	7	7	
2	6	365	6	6	
3	5	365	5	5	
4	2	365	2	2	

Listado rápido sobre las variables 41-50:

Las 8 primeras variables ofrecen información muy similar a primera vista, se tomará en todo caso 'minimum_nights_avg_ntm' y 'maximum_nights_avg_ntm'

- calendar_updated: Parece a priori repleta de NaN.
- has_availability: Disponibilidad, categoría de servicio, al ser dinámica puede no ser útil.

In [10]:
```
df[colnames[50:60]].head()
```

Out[10]:

	availability_30	availability_60	availability_90	availability_365	calendar_last_scraped	number_of_re
0	13	43	73	311	2020-09-21	
1	0	0	0	0	2020-09-21	
2	23	53	83	325	2020-09-20	
3	20	50	80	355	2020-09-20	
4	30	60	90	365	2020-09-20	

Listado rápido sobre las variables 51-60:

- availability_30: Muestra la disponibilidad de la vivienda a 30 días vista, sus homólogas 60, 90 y 365 funcionan igual. No es relevante para nuestro problema.
- calendar_last_scraped: Especifica cuando una calendario se ha visualizado por última vez. No interesa.
- number_of_reviews: Cantidad de valoraciones de la entrada en cuestión. Relevante, puede tener su utilidad.
- number_of_reviews_ltm: Reviews en el último año(hace 12 meses). A priori poco útil.
- number_of_reviews_l30d: Reviews en los últimos 30 días, Como el anterior.
- first_review: primera valoración. poco útil.
- last_review: última valoración. Podría servir como filtro sobre la actividad de la vivienda.

In [11]:
```
df[colnames[60:70]].head()
```

Out[11]:

	review_scores_rating	review_scores_accuracy	review_scores_cleanliness	review_scores_checkin	revi
0	96.0	10.0	9.0	10.0	
1	100.0	10.0	10.0	10.0	
2	97.0	10.0	10.0	10.0	
3	98.0	10.0	10.0	10.0	
4	NaN	NaN	NaN	NaN	

Listado rápido sobre las variables 61-70:

- Los distintos 'review_score' pueden ofrecer información relevante, dada su cantidad eligiremos 'review_scores_rating'.
- license: En principio no es relevante.
- instant_bookable: Reserva instantánea, puede servir.
- calculated_host_listings_count: Podría guardar relación con los 'host_listing' anteriores. Podría servir.

In [12]:
```
df[colnames[70:74]].head()
```

Out[12]:

	calculated_host_listings_count_entire_homes	calculated_host_listings_count_private_rooms	calculated
0	1	0	
1	1	0	
2	2	0	
3	1	0	
4	0	2	

Listado rápido sobre las variables 71-74:

- Los distintos 'calculated_host_listing' vienen agrupados en 'calculated_host_listings_count' usaremos esa variable.
- reviews_per_month: Valoraciones mensuales, relevante.

En base a las distintas variables de interés se reducirá el *dataset* al siguiente conjunto:

1. host_since
2. host_response_time
3. host_response_rate
4. host_acceptance_rate
5. host_is_superhost
6. host_listings_count
7. host_total_listings_count
8. host_has_profile_pic
9. reviews_per_month
10. host_identity_verified
11. review_scores_rating
12. instant_bookable
13. room_type
14. accommodates
15. bathrooms_text
16. bedrooms
17. beds
18. amenities
19. calculated_host_listings_count
20. number_of_reviews
21. last_review
22. neighbourhood_cleansed
23. latitude
24. longitude
25. price

De éstas 25 variables, se tiene el precio como salida y lat/lon como posible representación.

Cabe añadir que el objetivo de la práctica es proponer un precio a una vivienda acabada de introducir al sistema, por lo que, entre dichas variables, las siguientes están relacionadas con el

servicio:

1. host_since
2. host_response_time
3. host_response_rate
4. host_acceptance_rate
5. host_is_superhost
6. host_listings_count
7. host_total_listings_count
8. host_has_profile_pic
9. reviews_per_month
10. host_identity_verified
11. review_scores_rating
12. instant_bookable
13. calculated_host_listings_count
14. number_of_reviews
15. last_review

Mientras que por otro lado, están presentes las relacionadas con la característica relacionadas con la vivienda:

1. room_type
2. accommodates
3. bathrooms_text
4. bedrooms
5. beds
6. amenities
7. neighbourhood_cleansed
8. latitude
9. longitude
10. price

Por lo tanto, aunque es cierto que el precio del alojamiento implica ambas categorías de variables, para una casa recién registrada carecerá de las 15 variables relacionadas con el servicio. En consecuencia, se tomará un *Dataset* reducido con las 10 últimas variables.

In [13]:
```
sel_names=[
    'neighbourhood_cleansed',
    'room_type',
    'accommodates',
    'bathrooms_text',
    'bedrooms',
    'beds',
    'amenities',
    'latitude',
    'longitude',
    'price'
]
```

---

2. Tratamiento de datos

Para tocar lo mínimo posible el *Dataset* original, se utilizará las siguientes clases vistas en un notebook de clase.

In [14]:
```
from sklearn.base import TransformerMixin

### aux functions

class SelectColumns(TransformerMixin):
    def __init__(self, columns: list) -> pd.DataFrame:
        if not isinstance(columns, list):
            raise ValueError('Specify the columns into a list')
        self.columns = columns
    def fit(self, X, y=None): # we do not need to specify the target in the transfor
        return self
    def transform(self, X):
        return X[self.columns]

class DropColumns(TransformerMixin):
    def __init__(self, columns: list) -> pd.DataFrame:
        if not isinstance(columns, list):
            raise ValueError('Specify the columns into a list')
        self.columns = columns
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X.drop(self.columns, axis=1)
```

Utilizamos la clase SelectColumns con las columnas elegidas para generar un *Dataset* de dimensionalidad reducida.

In [15]:
```
sel_col=SelectColumns(sel_names)
df2=sel_col.fit_transform(df)
df2.head()
```

Out[15]:

	neighbourhood_cleansed	room_type	accommodates	bathrooms_text	bedrooms	beds	amen
0	Calvià	Entire home/apt	2	1 bath	1.0	1.0	["Ov "V "Cc ma "Dishes
1	Santa Margalida	Entire home/apt	8	3 baths	4.0	7.0	["First aid "Hair dr "Ir "Was
2	Maria de la Salut	Entire home/apt	6	2 baths	3.0	4.0	["Sm ala "Ov "V "Garde t
3	Sant Llorenç des Cardassar	Entire home/apt	4	1 bath	2.0	4.0	["Pool", " parkin premi "Air co
4	Palma de Mallorca	Private room	2	1 bath	1.0	2.0	["P "Washer", conditioni "Kitc

A primera vista se aprecia elementos de texto, enteros y números en coma flotante, mediante .unique() se puede evaluar qué tipos hay.

In [16]:
```
kind=df2.dtypes
kind.unique()
```

Out[16]:
```
array([dtype('O'), dtype('int64'), dtype('float64')], dtype=object)
```

Efectivamente, el *Dataset* reducido consta de variables numéricas enteras, en coma flotante y texto plano(denotado por el dtype 'o').

Interesa transformar las variables en texto para que puedan adquirir algún significado en la predicción.

In [17]:
```
colnames=df2.columns
df2[colnames[df2.dtypes=='O']].head()
```

Out[17]:

	neighbourhood_cleansed	room_type	bathrooms_text	amenities	price
0	Calvià	Entire home/apt	1 bath	["Oven", "Wifi", "Coffee maker", "Dishes and s...	$89.00
1	Santa Margalida	Entire home/apt	3 baths	["First aid kit", "Hair dryer", "Iron", "Washe...	$175.00
2	Maria de la Salut	Entire home/apt	2 baths	["Smoke alarm", "Oven", "Wifi", "Garden or bac...	$140.00
3	Sant Llorenç des Cardassar	Entire home/apt	1 bath	["Pool", "Free parking on premises", "Air cond...	$200.00
4	Palma de Mallorca	Private room	1 bath	["Pool", "Washer", "Air conditioning", "Kitche...	$110.00

Las 5 variables acabadas de mostrar, requieren transformar el texto para ser útiles a nivel computacional. En primer lugar, interesa saber la cantidad de NaN presentes en cada variable por si se requiere algún tipo de procesado adicional.

In [18]:
```
# Contador para determinar si hay presencia de Not a Number(NaN).
k=0

# Listas para poner tanto el nombre de la variable como el de la probabilidad.
# Servirá en caso de querer quitar la variable en caso de tener mucha presencia de N
Col_na=[]
Col_probna=[]
for name in colnames:
    l=len(df2[df2[name].isna()==True])
    if(l>0):
        k+=1
        prob=float(len(df2[df2[name].isna()==True]))/176.08
        print( k,name, l, str(round(prob,2))+'%')
        Col_na.append(name)
        Col_probna.append(prob)
```

```
1 bathrooms_text 8 0.05%
2 bedrooms 275 1.56%
3 beds 97 0.55%
```

De las 10 variables, solamente 3 tienen NaN. Se aprecia que respecto a la cantidad total de entradas, es una cantidad pequeña(aproximadamente un 2% del total de valores).

De las variables tipo texto, solamente requiere tratar con cuidado 'bathrooms_text'.

En primer lugar se transforma el *Output*, el precio.

In [19]:
```
# Cantidad de entradas en el dataset.
l=len(df2)

# Array de Numpy para ir poniendo los valores.
y=np.zeros(l)
# Bucle para eliminar el dólar a cada entrada y a los miles, quitar el caracter ','.
for i in range(l):
    a=df2.price[i][1:]
    y[i]=float( a.replace(',',''))
# Convertimos el array con el nombre de columna deseado y eliminamos la columna de t
Y=pd.Series(y,name='num_price')
df2=pd.concat([df2,Y], axis=1)
df2=df2.drop(columns='price')
```

Una vez hecho el *Chunk* anterior comprobamos que efectivamente, la nueva columna es numérica.

In [20]:
```
df2['num_price'].dtype
```

Out[20]:
```
dtype('float64')
```

Una vez transformado el precio, el municipio servirá para estimar un precio promedio de vivienda. La otra opción es categorizar los 53 municipios y eso supondría un coste computacional elevado.(Añadir 53 variables categóricas)

In [21]:
```
# Array para ir poniendo los distintos resultados
precio_medio=np.zeros(l)

# Se coge las columnas a utilizar para no tener que cargar el dataset completo.
precio=df2[['num_price']]
municipios=df2['neighbourhood_cleansed']

# Bucle para ir asignando el precio medio según el municipio.
for i in range(l):
    municipio=municipios[i]
    precio_medio[i]=round(precio[municipios==municipio].mean(),2)

# Convertimos el array con el nombre de columna deseado y eliminamos la columna de t
Y=pd.Series(precio_medio,name='neighbour_price')
df2=pd.concat([df2,Y], axis=1)
df2=df2.drop(columns='neighbourhood_cleansed')
```

In [22]:
```
df2['neighbour_price'].dtype
```

Out[22]:
```
dtype('float64')
```

Para comprobar que se ha realizado correctamente, se puede mirar la cantidad única de valores presentes en la nueva columna.

In [23]:
```
len(df2['neighbour_price'].unique())
```

Out[23]:
```
53
```

Dicho resultado es el esperado, 53 valores diferentes de precios medios, de acuerdo con el precio de cada municipio.

El siguiente paso es categorizar las habitaciones, se puede eliminar una categoría que sería la ausencia de todas las demás.

In [24]:
```
df2['room_type'].unique()
```

Out[24]:
```
array(['Entire home/apt', 'Private room', 'Hotel room', 'Shared room'],
      dtype=object)
```

Tenemos 4 categorías, la última de ellas la eliminaremos.

In [25]:
```
# Variable dummy que coge las categorías presentes en un vector.
dummy_1=pd.get_dummies(df2['room_type'])
# Se elimina una de las variables, la ausencia de las demás categorías, por compleme
# Similar a coger una columna con True y False y eliminar la variable dummy de False
room_type=dummy_1.drop(columns='Shared room')
room_type=room_type.rename(columns={'Entire home/apt': 'apartment', 'Hotel room': 'h
room_type.head()
```

Out[25]:

	apartment	hot_room	priv_room
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	0	0	1

En principio hay 4 posibilidades, pertenecer a una de las variables(3 casos) o no pertenecer a ninguna(las tres variables valen 0).

Por lo tanto, se puede comprobar que una porción de ellas no pertenecen a ninguna categoría.

In [26]:
```
room_type[(room_type['apartment']==0)&(room_type['hot_room']==0)&(room_type['priv_ro
```

Out[26]:

	apartment	hot_room	priv_room
426	0	0	0
515	0	0	0
1005	0	0	0
1517	0	0	0
1882	0	0	0

La ausencia de categoría explica también una categoría. Dicho lo anterior, se procede a anexionar las columnas categóricas anteriores.

In [27]:
```
df2=pd.concat([df2,room_type], axis=1)
df2=df2.drop(['room_type'],axis=1)
```

Elipsis para mostrar como está quedando el *dataset*

In [28]:
```
df2.head()
```

Out[28]:

	accommodates	bathrooms_text	bedrooms	beds	amenities	latitude	longitude	num_price

	accommodates	bathrooms_text	bedrooms	beds	amenities	latitude	longitude	num_price
0	2	1 bath	1.0	1.0	["Oven", "Wifi", "Coffee maker", "Dishes and s...	39.51888	2.48182	89.0
1	8	3 baths	4.0	7.0	["First aid kit", "Hair dryer", "Iron", "Washe...	39.76347	3.16255	175.0
2	6	2 baths	3.0	4.0	["Smoke alarm", "Oven", "Wifi", "Garden or bac...	39.66044	3.07165	140.0
3	4	1 bath	2.0	4.0	["Pool", "Free parking on premises", "Air cond...	39.61600	3.30121	200.0
4	2	1 bath	1.0	2.0	["Pool", "Washer", "Air conditioning", "Kitche...	39.56478	2.60333	110.0

La variable 'amenities' en vez de categorizar cada elemento, una simplificación consiste en contar qué cantidad hay en cada entrada. Dicha simplificación tiene su parte buena, cuantificar su cantidad, y su parte mala, se le da la misma importancia a cada 'comodidad'. En otras palabras, se ofrece a piscina o wifi la misma importancia.

In [29]:
```
# Lista para ir anexionando cada valor.
count_amenities=[]

# Bucle para contar cuantos elementos hay separados por comas.
for i in range(len(df2)):
    count=len(df2['amenities'][i].split(','))
    count_amenities.append(count)

# Se convierte la lista a Series con el nombre de columna deseado y eliminamos la co
count_amenities=pd.Series(count_amenities, name='count_amenities')
df2=pd.concat([df2,count_amenities], axis=1)
df2=df2.drop(columns='amenities')
```

In [30]:
```
df2.head()
```

Out[30]:

	accommodates	bathrooms_text	bedrooms	beds	latitude	longitude	num_price	neighbour_pri
0	2	1 bath	1.0	1.0	39.51888	2.48182	89.0	359.8
1	8	3 baths	4.0	7.0	39.76347	3.16255	175.0	185.3
2	6	2 baths	3.0	4.0	39.66044	3.07165	140.0	177.1
3	4	1 bath	2.0	4.0	39.61600	3.30121	200.0	222.2
4	2	1 bath	1.0	2.0	39.56478	2.60333	110.0	162.1

La siguiente variable es la relativa a los baños. Se realizará una exploración de elementos únicos.

In [31]:
```
df2['bathrooms_text'].unique()
```

Out[31]:
```
array(['1 bath', '3 baths', '2 baths', '1 private bath', '0 baths',
       '1.5 baths', '2 shared baths', '7 baths', '4.5 baths', '2.5 baths',
       '4 baths', '5 baths', 'Shared half-bath', '1.5 shared baths',
       '3.5 baths', '1 shared bath', '6 baths', '8 baths', nan,
       '6.5 baths', '9.5 baths', '5.5 baths', '7.5 baths',
       '12 shared baths', '4 shared baths', '8.5 baths', '0 shared baths',
       '2.5 shared baths', 'Half-bath', '12 baths', '11 baths',
       '13 baths', '9 baths', '3 shared baths', '3.5 shared baths',
       '12.5 baths', '32 baths', '19 baths', 'Private half-bath',
       '13 shared baths', '10 baths', '14 baths', '16 baths'],
      dtype=object)
```

Como hay categoría '0 baños', la presencia de NaN corresponde a una cantidad no especificada. Se transformará el resto de entradas y luego se buscará una estrategia con las problemáticas.

In [32]:
```
len(df2[df2['bathrooms_text'].isna()==True])
```

Out[32]:
```
8
```

Hay 8 entradas con NaN. Para transformar el texto, mejor evitar acceder a dichas entradas.

Se puede categorizar y cuantificar 'bathrooms_text' en relación a 'shared'=compartido, 'private'=privado y 'bathroom' no especificado.

In [33]:
```
# Tres listas que usaremos para categorizar/couantificar los distintos tipos de baño
bathroom=[]
shared=[]
private=[]
# Lista con True/False para no trastear las entradas con NaN.
lista=df2['bathrooms_text'].isna()
for i in range(len(df2)):
    if (lista[i]==False):
        count=df2['bathrooms_text'][i]
        count=count.lower()
        if 'shared' in count:
            if 'half' in count:
                bathroom.append(0.)
                shared.append(0.5)
                private.append(0.)
            else:
                bathroom.append(0.)
                shared.append(float(count.split(' ')[0]))
                private.append(0.)
        elif 'private' in count:
            if 'half' in count:
                bathroom.append(0.)
                shared.append(0)
                private.append(0.5)
            else:
                bathroom.append(0.)
                shared.append(0.)
                private.append(float(count.split(' ')[0]))
        else:
            if 'half' in count:
                bathroom.append(0.5)
                shared.append(0)
                private.append(0)
            else:
                bathroom.append(float(count.split(' ')[0]))
```

```
                shared.append(0.)
                private.append(0.)
    else:
        bathroom.append(np.nan)
        shared.append(np.nan)
        private.append(np.nan)
d = {'bathrooms': bathroom, 'shared': shared, 'private': private}

baths=pd.DataFrame(data=d)

df2=pd.concat([df2,baths], axis=1)
df2=df2.drop(columns='bathrooms_text')
```

Se puede comprobar que efectivamente, 8 entradas no tienen baños asociados.

In [34]:
```
df2[df2['bathrooms'].isna()==True]
```

Out[34]:

	accommodates	bedrooms	beds	latitude	longitude	num_price	neighbour_price	apartmer
185	4	2.0	2.0	39.57012	2.65751	59.0	162.14	
1882	2	1.0	1.0	39.56264	2.61869	100.0	162.14	
12548	1	NaN	1.0	39.55556	2.60402	250.0	162.14	
15415	0	NaN	NaN	39.56952	2.64516	0.0	162.14	
16275	4	NaN	NaN	39.35877	3.22180	165.0	242.91	
16287	8	NaN	NaN	39.48155	3.08651	330.0	253.75	
16619	2	1.0	1.0	39.57356	2.65960	49.0	162.14	
16723	4	NaN	NaN	39.88919	3.01769	275.0	256.81	

Se puede comprobar la cantidad de baños en cada categoría.

In [35]:
```
print('Bathrooms: '+str(len(df2[df2['bathrooms']>0]))+'\n'+'Shared: '+str(len(df2[df
```

```
Bathrooms: 16330
Shared: 720
Private: 514
```

Las categorías 'shared' y 'private' suman entre las dos un 7% de la cantidad de datos. Para simplificar el tratamiento de outliers en el futuro, se ha considerado que juntar las 3 categorías.

In [36]:
```
y=np.array(df2['bathrooms'])

for i in range(l):
     if (lista[i]==False):
            y[i]=df2['bathrooms'][i]+df2['shared'][i]+df2['private'][i]

baths=pd.DataFrame(data=y)
df2=pd.concat([df2,baths], axis=1)
df2=df2.drop(columns=['bathrooms','shared','private'])
df2=df2.rename(columns={0:'bathrooms'})
```

In [37]:
```
df2.head()
```

Out[37]:

	accommodates	bedrooms	beds	latitude	longitude	num_price	neighbour_price	apartment	h
0	2	1.0	1.0	39.51888	2.48182	89.0	359.82	1	
1	8	4.0	7.0	39.76347	3.16255	175.0	185.36	1	

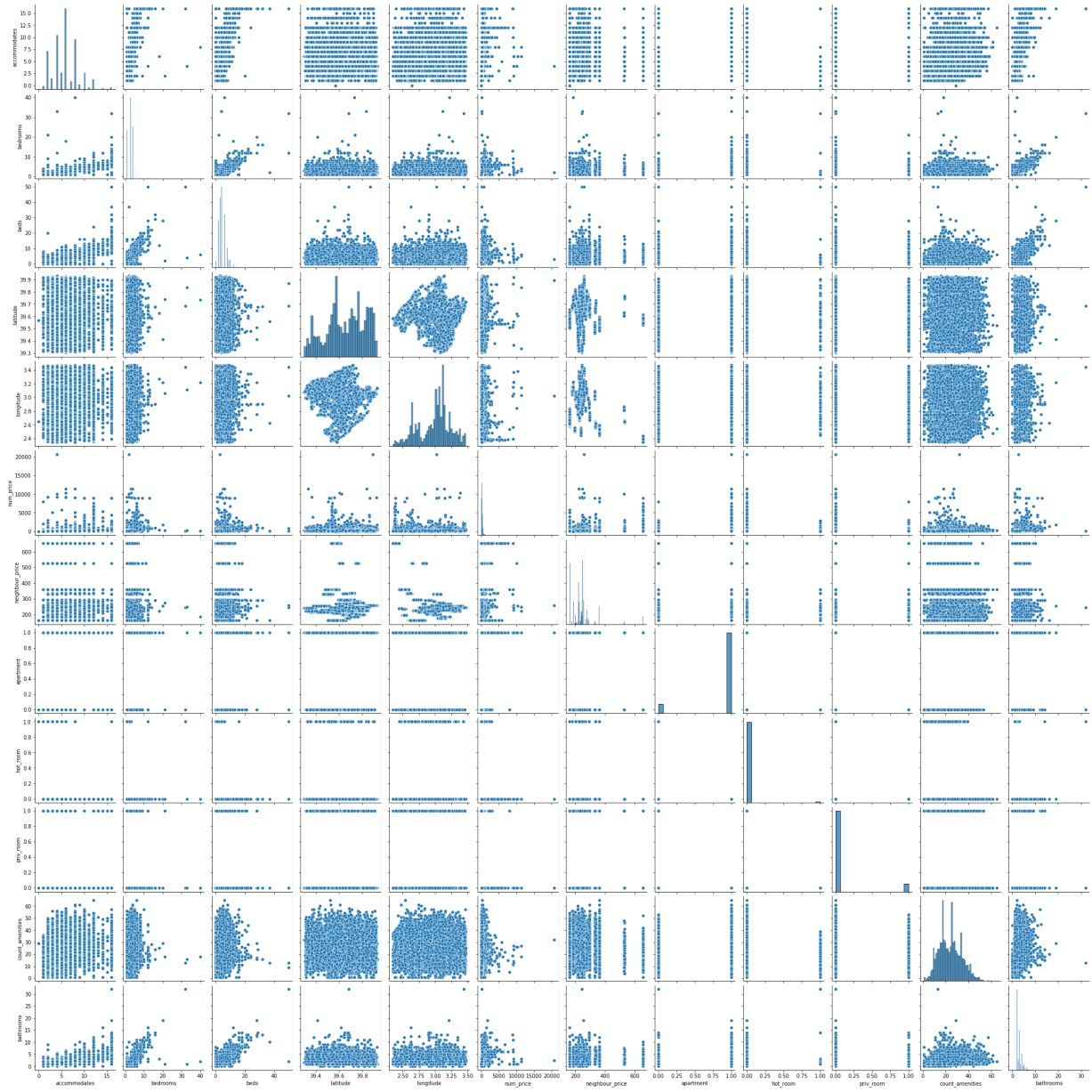
	accommodates	bedrooms	beds	latitude	longitude	num_price	neighbour_price	apartment	h
2	6	3.0	4.0	39.66044	3.07165	140.0	177.13	1	
3	4	2.0	4.0	39.61600	3.30121	200.0	222.28	1	
4	2	1.0	2.0	39.56478	2.60333	110.0	162.14	0	

El Dataset ya se encuentra transformado, las variables de texto ya no están presentes y se han categorizado/cuantificado.

Habiendo hecho este primer procesamiento de datos, quedan 12 variables.

In [38]:
```
sn.pairplot(df2)
```

Out[38]:
```
<seaborn.axisgrid.PairGrid at 0x1f11035f908>
```

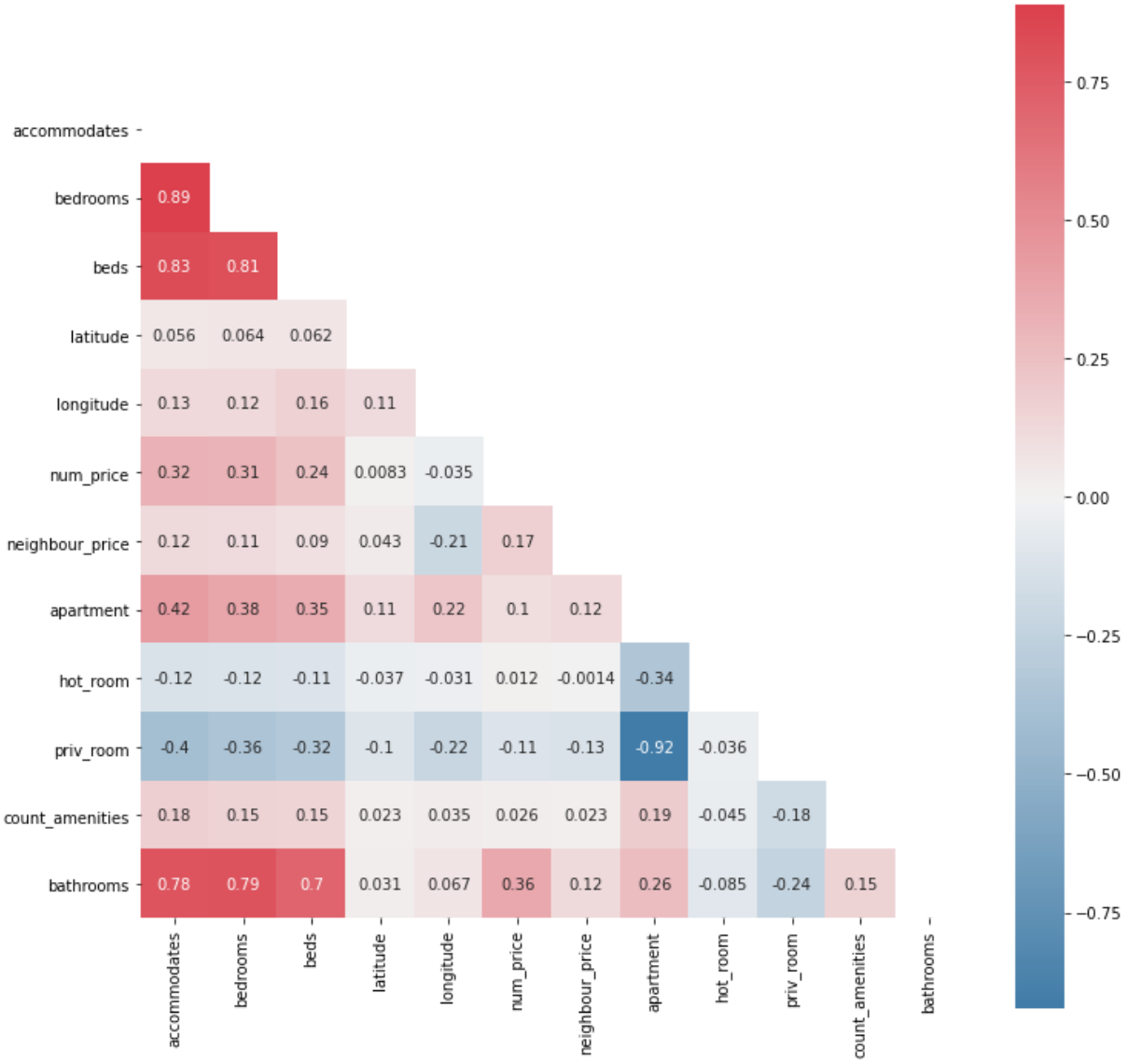


A continuación, se hará uso de la matriz de correlación para comprobar que variables son colineales.

In [39]:
```
corr_matrix=df2.corr()

f, ax = plt.subplots(figsize=(12, 12))
```

```
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

cmap = sn.diverging_palette(240, 10,n=9,as_cmap=True)

sn.heatmap(corr_matrix,mask=mask,cmap=cmap,center=0,square=True,annot=True,ax=ax)
```

Out[39]: <AxesSubplot:>



En la tabla de correlaciones se aprecia mucha correlación entre 'beds','bedrooms' y 'accomodates'; mientras que 'priv_room' y 'apartment' están anticorrelacionadas.

Una posibilidad es reducir dichas variables, quedarse con una de ellas en cada conjunto. Del primer grupo, 'accomodates' se resume en el aforo del alojamiento así que será la variable elegida. De las anticorrelacionadas, con 'apartment'.

In [40]:
```
df2=df2.drop(columns=['beds','bedrooms','priv_room'])
df2.head()
```

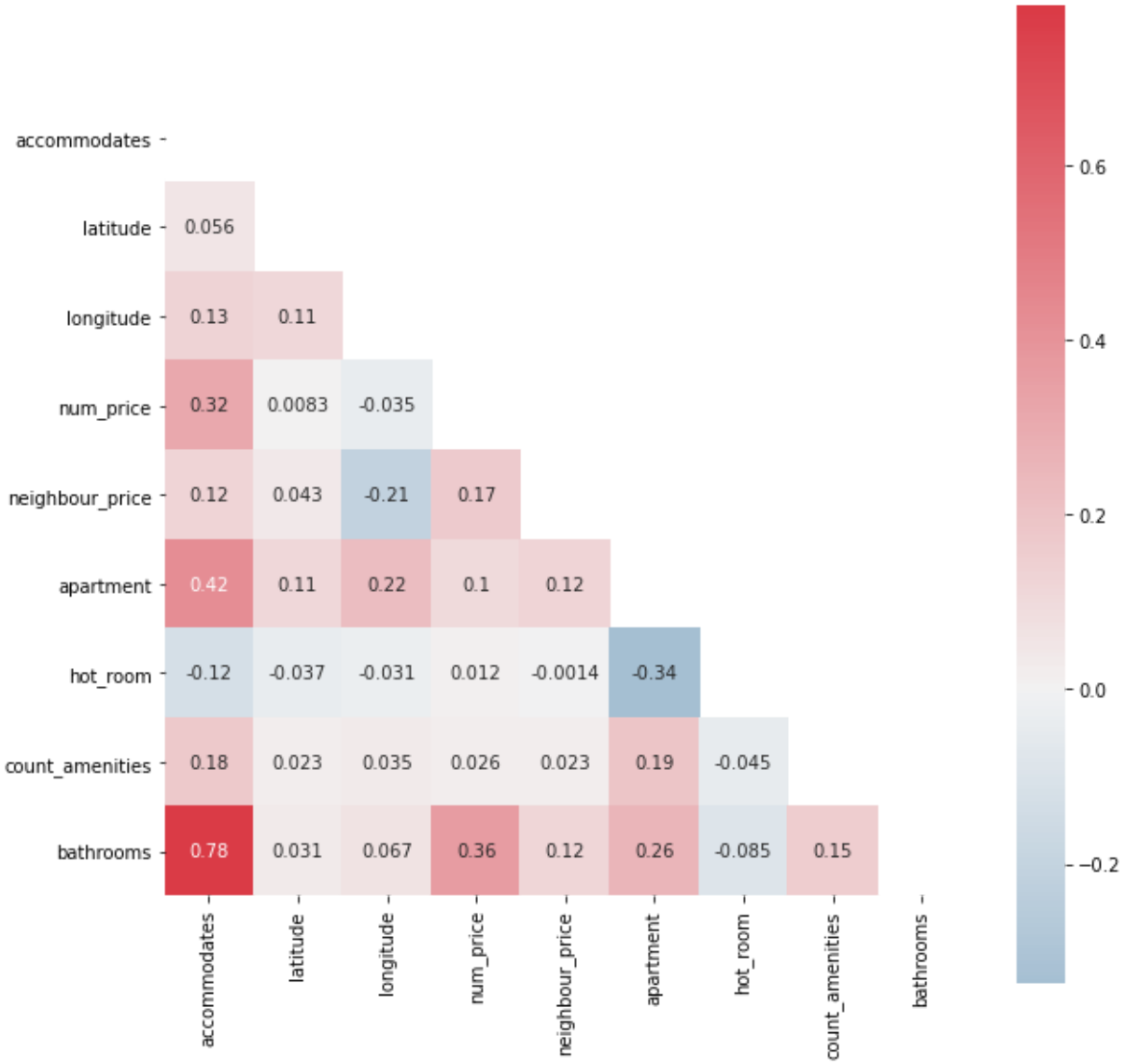
Out[40]:

	accommodates	latitude	longitude	num_price	neighbour_price	apartment	hot_room	count_ar
0	2	39.51888	2.48182	89.0	359.82	1	0	
1	8	39.76347	3.16255	175.0	185.36	1	0	
2	6	39.66044	3.07165	140.0	177.13	1	0	
3	4	39.61600	3.30121	200.0	222.28	1	0	

	accommodates	latitude	longitude	num_price	neighbour_price	apartment	hot_room	count_ar
4	2	39.56478	2.60333	110.0	162.14	0	0	

In [41]:
```
corr_matrix=df2.corr()

f, ax = plt.subplots(figsize=(10, 10))

mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

cmap = sn.diverging_palette(240, 10,n=9,as_cmap=True)

sn.heatmap(corr_matrix,mask=mask,cmap=cmap,center=0,square=True,annot=True,ax=ax)
```

Out[41]: <AxesSubplot:>



Los baños y el aforo también tienen una correlación alta, pero se ha puesto como límite cortar a 0.8 para encontrar un compromiso.

Quitadas un par de variables, falta comprobar si aún quedan datos indeterminados.

In [42]:
```
colnames=df2.columns

# Contador para determinar si hay presencia de Not a Number(NaN).
k=0
```

```
# Listas para poner tanto el nombre de la variable como el de la probabilidad.
# Servirá en caso de querer quitar la variable en caso de tener mucha presencia de N
Col_na=[]
Col_probna=[]
for name in colnames:
    l=len(df2[df2[name].isna()==True])
    if(l>0):
        k+=1
        prob=float(len(df2[df2[name].isna()==True]))/176.08
        print( k,name, l, str(round(prob,2))+'%')
        Col_na.append(name)
        Col_probna.append(prob)
if (k==0):
    print('El dataset no contiene variables con NaN')
```

```
1 bathrooms 8 0.05%
```

Ahora mismo, solamente la columna de 'bathrooms' contiene NaNs. optaremos por ofrecerle el valor de la mediana.

In [43]:
```
# Lista con los valores True/False en caso de haber o no NaNs.
lista=df2['bathrooms'].isna()
mediana=df2['bathrooms'].median()
for i in range(len(df2)):
    if (lista[i]==True):
        df2.loc[i, ('bathrooms')]=mediana
# Línea para comprobar que la variable en cuestión no tiene NaNs.
len(df2[df2['bathrooms'].isna()==True])
```

Out[43]:
```
0
```

In [44]:
```
df2.shape
```

Out[44]:
```
(17608, 9)
```

se ha logrado en todo momento mantener todas las entradas del dataset original, aunque obviamente la dimensionalidad se ha visto considerablemente reducida.

El siguiente paso consiste en eliminar los puntos *outliers*, de esta manera se da mayor importancia a los puntos realmente presentes en el *Dataset*. Se utilizarán solamente las variables cuantitativas.

In [45]:
```
Col_outliers=['accommodates', 'num_price', 'count_amenities', 'bathrooms']
Col_outliers
```

Out[45]:
```
['accommodates', 'num_price', 'count_amenities', 'bathrooms']
```

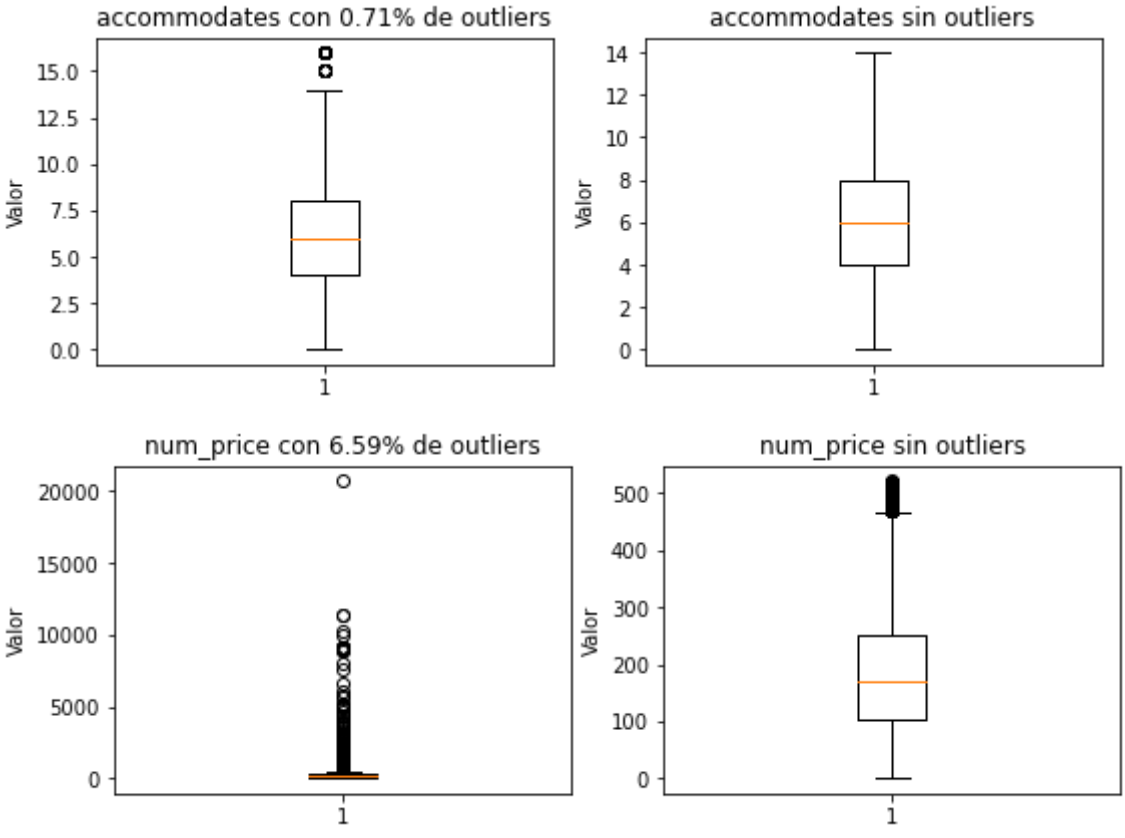
In [46]:
```
out=0
print( 'variable'+'\t'+'75%+IQR\t'+'amount\t'+'% of data')
for name in Col_outliers:
    statistics=df2[name].describe()

    iqr=1.5*(statistics[6]-statistics[4])

    sup_limit=statistics[6]+iqr

    sup_limit
    out_k=len(df2[df2[name]>sup_limit])
    out+=out_k
    print( name+'\t'+str(round(sup_limit,2))+'\t'+str(out_k)+'\t'+ str(round(out_k/1
```

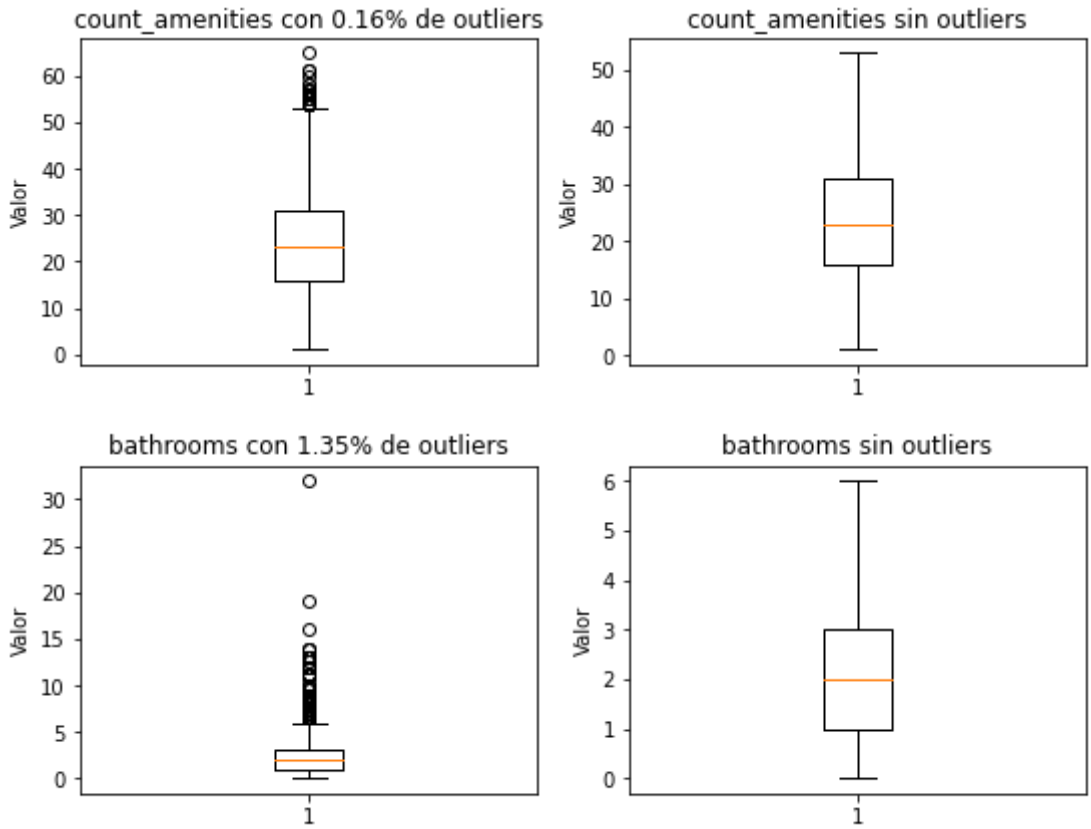
```
variable        75%+IQR amount  % of data
```

```
accommodates     14.0    125     0.71
num_price        522.5   1160    6.59
count_amenities  53.5    29      0.16
bathrooms        6.0     238     1.35
```

Una opción es considerar como *outlier* aquellos puntos más alejados del 75%+IQR(tercer quartil+ Rango Intercuartil), la variable más afectada es justamente el precio. Mediante un diagrama de cajas y bigotes se puede apreciar el resultado anteriormente expuesto.

In [47]:
```
df3=df2.copy()

for name in Col_outliers:
    statistics=df2[name].describe()

    iqr=1.5*(statistics[6]-statistics[4])

    sup_limit=statistics[6]+iqr

    out_k=len(df2[df2[name]>sup_limit])
    df3=df3[df3[name]<=sup_limit]

    fig1, (ax1,ax2) = plt.subplots(ncols=2, nrows=1,figsize=(9,3))

    #plt.subplots_adjust(hspace=0.5,wspace=0.5)

    ax1.boxplot(df2[name])
    ax1.set_title(name+' con '+ str(round(out_k/176.08,2))+'% de outliers')
    ax1.set_ylabel('Valor')

    ax2.boxplot(df3[name])
    ax2.set_title(name+' sin '+'outliers')
    ax2.set_ylabel('Valor')
    plt.show()
df3.reset_index(drop=True,inplace=True)
```





En las anteriores gráficas se muestran las distribuciones antes y después de haber eliminado los outliers. Así y todo se aprecia que, aunque se hayan quitado del *Dataset* previo, las gráficas de la derecha son una orientación sobre cual es el resultado final de los datos.

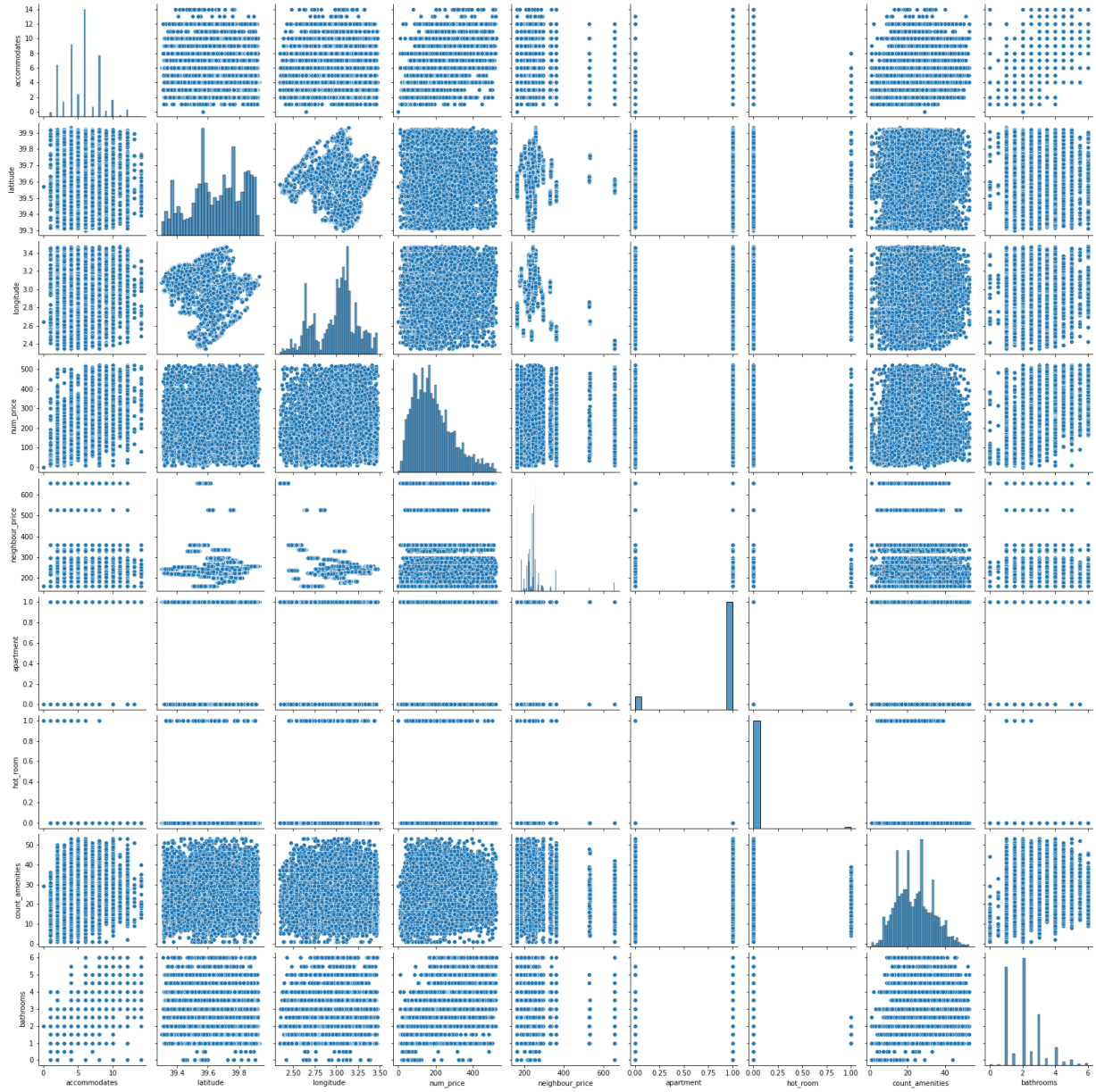
In [48]:
```
print(df3.shape)
```

```
(16340, 9)
```

El *Dataset* se ha visto reducido una cantidad de aproximada de 1300 entradas. Ahora se tiene un 92% del total.

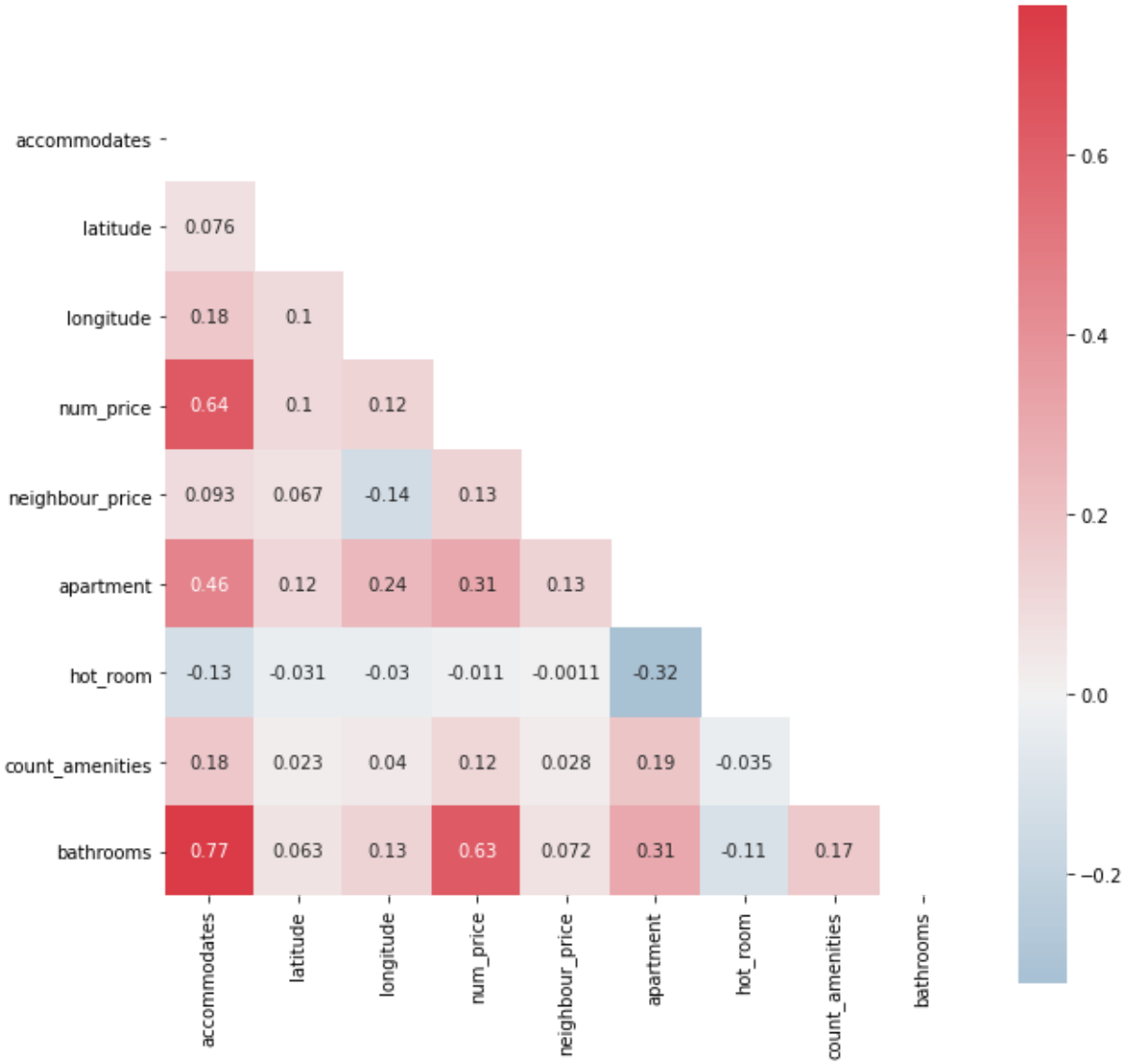
In [49]:
```
sn.pairplot(df3)
```

Out[49]:
```
<seaborn.axisgrid.PairGrid at 0x1f120299518>
```



In [50]:
```
corr_matrix=df3.corr()

f, ax = plt.subplots(figsize=(10, 10))

mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

cmap = sn.diverging_palette(240, 10,n=9,as_cmap=True)

sn.heatmap(corr_matrix,mask=mask,cmap=cmap,center=0,square=True,annot=True,ax=ax)
```

Out[50]: <AxesSubplot:>



En ambas matrices de correlación tenemos que tanto 'bathrooms' como 'accommodates' tienen una correlación significativa con el precio.

---

3. Aplicación de modelos

Para acabar la práctica, falta aplicar los modelos predictivos. En primer lugar hay que normalizar los datos de entrada a fin de que todas las variables presentes tengan el mismo peso.

In [51]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
```

In [52]:
```
# Dataset al cual se elimina la columna de precio y posteriormente se normaliza entr
df3_norm=df3.copy()
colnames=df3_norm.columns
colnames=colnames.drop('num_price')
# La variable de salida se mantiene intacta, sin alterar.
y3=df3[['num_price']]
scaler = MinMaxScaler()
df3_norm[colnames] = scaler.fit_transform(df3_norm[colnames])

X3=df3_norm[colnames]
y=y3
```

Se visualizará los datos para apreciar como han cambiado las entradas.

In [53]:
```
X3
```

Out[53]:

	accommodates	latitude	longitude	neighbour_price	apartment	hot_room	count_amenities
0	0.142857	0.345024	0.119856	0.401527	1.0	0.0	0.423077
1	0.571429	0.734078	0.722795	0.047164	1.0	0.0	0.365385
2	0.428571	0.570195	0.642283	0.030448	1.0	0.0	0.653846
3	0.285714	0.499507	0.845609	0.122156	1.0	0.0	0.173077
4	0.142857	0.418035	0.227480	0.000000	0.0	0.0	0.115385
...	...	...	...	...	...	...	...
16335	0.428571	0.736591	0.691210	0.229647	1.0	0.0	0.634615
16336	0.642857	0.948623	0.611123	0.192294	1.0	0.0	0.269231
16337	0.428571	0.719603	0.494712	0.177852	1.0	0.0	0.480769
16338	0.142857	0.387367	0.041611	1.000000	0.0	0.0	0.519231
16339	0.428571	0.226824	0.361065	0.108791	1.0	0.0	0.326923

16340 rows × 8 columns

In [54]:
```
y
```

Out[54]:

	num_price
0	89.0
1	175.0
2	140.0
3	200.0
4	110.0
...	...
16335	195.0
16336	110.0
16337	179.0
16338	42.0
16339	100.0

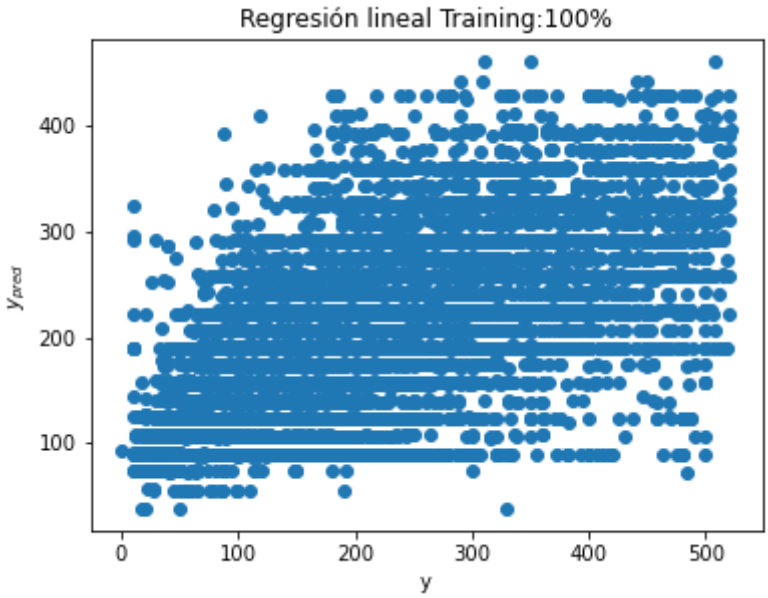
16340 rows × 1 columns

La transformación mix/max ha sido exitosa. Se puede proceder a realizar la regresión lineal con 'bathrooms' y 'acccommodates', las variables con más correlación a 'num_price'.

In [55]:
```
# Los inputs de interés
X=X3[['bathrooms','accommodates']]
# Llamamos la clase LinearRegression y se realiza el ajuste.
lin=LinearRegression().fit(X,y)
# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((lin.predict(X)-np.array(y))**2))
# Se muestra por pantalla ambos valores.
```

```
print( 'R2: '+'\t'+str(round(lin.score(X,y),3))+'\n'+'RMSE: '+'\t'+str(round(RMSE,2)
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(y,lin.predict(X))
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Regresión lineal Training:100%')
plt.show()
```

```
R2:     0.457
RMSE:   78.69
```

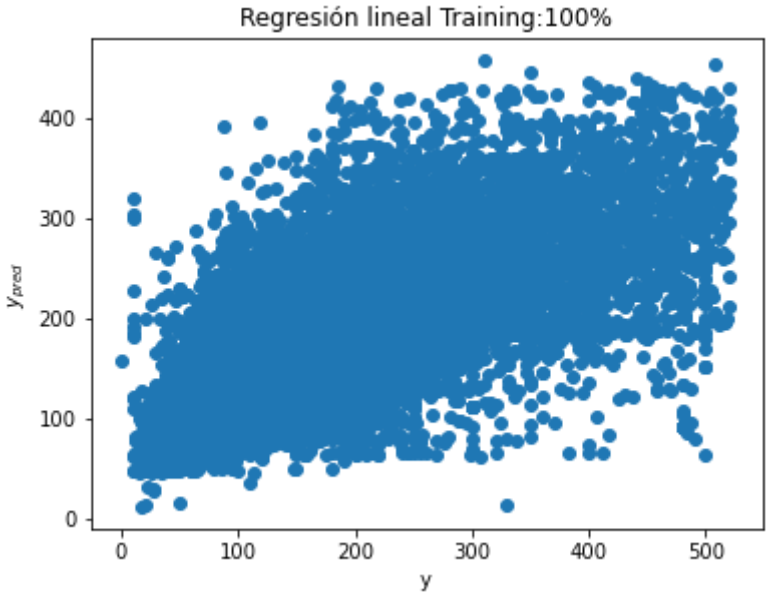


La regresión lineal arroja un R2 de 0.46 aproximadamente y un error de 79 dólares, en caso de redondear.

Se puede probar a utilizar todo el dataset de X a ver qué sucede.

In [56]:
```
# Los inputs de interés
X=X3
# Llamamos la clase LinearRegression y se realiza el ajuste.
lin=LinearRegression().fit(X,y)
# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((lin.predict(X)-np.array(y))**2))
# Se muestra por pantalla ambos valores.
print( 'R2: '+'\t'+str(round(lin.score(X,y),3))+'\n'+'RMSE: '+'\t'+str(round(RMSE,2)
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(y,lin.predict(X))
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Regresión lineal Training:100%')
plt.show()
```

```
R2:     0.473
RMSE:   77.54
```



Añadiendo las demás variables no ha variado mucho el resultado. Ha reducido un poco el error y R2 al añadir variables siempre tiende a subir. Dada la poca reducción en el error, el esfuerzo computacional de trabajar con las variables adicionales no vale la pena bajo este modelo.

Por consistencia con los demás modelos que se utilizarán. La evaluación de la regresión lineal se hará a través de todas las variables del dataset.

Se probará una estrategia train+test a ver si mejora las previsiones.

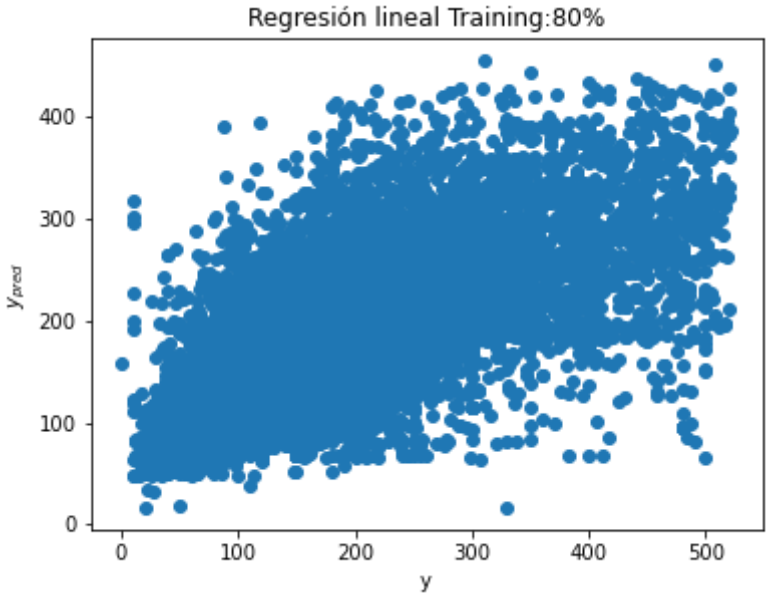
In [57]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Listas para poner los valores de interés y así realizar posteriormente una tabla.
model_name=[]
RMSE_training=[]
RMSE_testing=[]
```

Con validación cruzada logramos un valor similar a la regresión lineal global. En este caso, uno de los 5-fold ha ofrecido un R2 de casi 0.51.

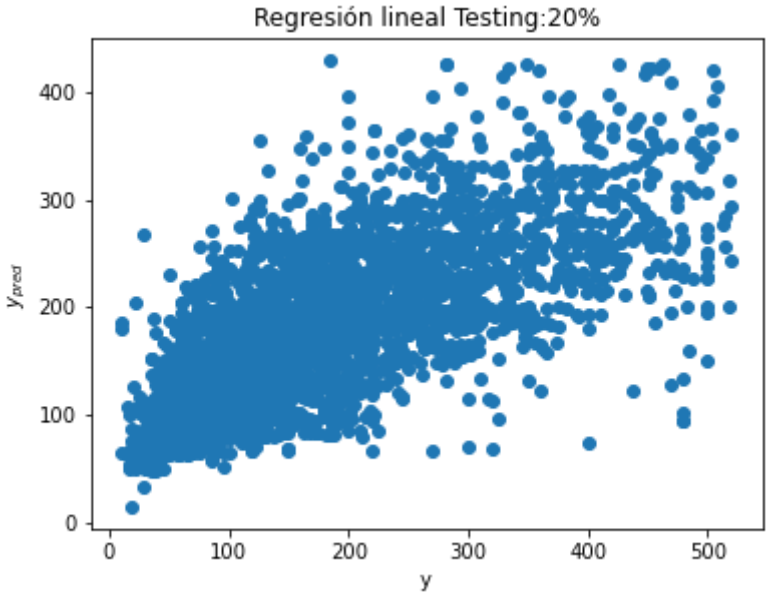
In [58]:
```
model_name.append('LinRegress')

# Llamamos la clase LinearRegression y se realiza el ajuste.
lin=LinearRegression().fit(X_train,y_train)
# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((lin.predict(X_train)-np.array(y_train))**2))
RMSE_training.append(round(RMSE,2))
# Se muestra por pantalla ambos valores.
print( 'R2: '+'\t'+str(round(lin.score(X_train,y_train),3))+'\n'+'RMSE: '+'\t'+str(r
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(y_train,lin.predict(X_train))
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Regresión lineal Training:80%')
plt.show()

# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((lin.predict(X_test)-np.array(y_test))**2))
RMSE_testing.append(round(RMSE,2))
```

```
# Se muestra por pantalla ambos valores.
print( 'R2: '+'\t'+str(round(lin.score(X_test,y_test),3))+'\n'+'RMSE: '+'\t'+str(rou
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(y_test,lin.predict(X_test))
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Regresión lineal Testing:20%')
plt.show()
```

```
R2:     0.472
RMSE:   77.56
```




```
R2:     0.473
RMSE:   77.51
```



En el caso de la regresión lineal, utilizar todo el *dataset* o separarlo en entrenamiento y test ofrece un resultado similar.

Otro método es el árbol de decisión.

In [59]:
```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV

# Parameters established for looking the best model among the possiblities.
parameters = {'max_depth':[6,9,12,15,18], 'min_samples_split':[2,3,4,5,6], 'max_feat
```

```
tree=DecisionTreeRegressor()
# Application of the gridsearchCV
clf=GridSearchCV(tree,parameters, cv=5)

clf.fit(X,y)
```

Out[59]:
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [6, 9, 12, 15, 18],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [2, 3, 4, 5, 6],
                         'random_state': [42]})
```

In [60]:
```
best_model=clf.best_estimator_
best_model.get_params()
```

Out[60]:
```
{'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 6,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 42,
 'splitter': 'best'}
```

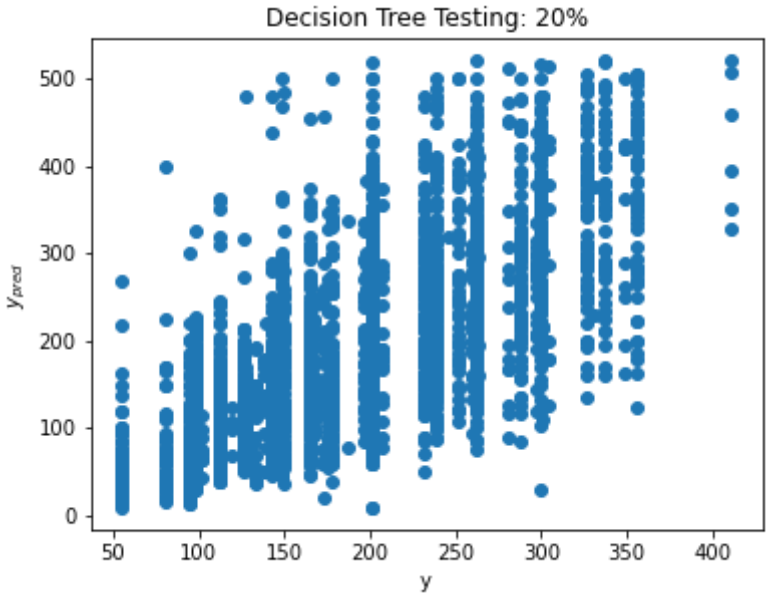
In [61]:
```
best_model.get_n_leaves()
```

Out[61]:
```
60
```

In [62]:
```
model_name.append('DecisTree')

# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((best_model.predict(X_train)-np.array(y_train).T)**2))
RMSE_training.append(round(RMSE,2))
# Se muestra por pantalla ambos valores.
print( 'R2: '+'\t'+str(round(best_model.score(X_train,y_train),3))+'\n'+'RMSE: '+'\t
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(best_model.predict(X_train),y_train)
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Decision Tree Training: 80%')
plt.show()


# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((best_model.predict(X_test)-np.array(y_test).T)**2))
RMSE_testing.append(round(RMSE,2))
# Se muestra por pantalla ambos valores.
print( 'R2: '+'\t'+str(round(best_model.score(X_test,y_test),3))+'\n'+'RMSE: '+'\t'+
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(best_model.predict(X_test),y_test)
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Decision Tree Testing: 20%')
plt.show()
```

```
R2:     0.509
RMSE:   74.84
```


```
R2:     0.501
RMSE:   75.42
```



Con el gridsearch hemos encontrado un modelo de arbol de decisión que ha ofrecido un 0.51 de R2 y el error cometido es de aproximadamente 2 dólares por debajo de la regresión lineal.

In [63]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Parameters established for looking the best model among the possiblities.
parameters = {'max_depth':[6,9,12,15,18], 'min_samples_split':[2,3,4,5,6], 'max_feat

forest=RandomForestRegressor()
# Application of the gridsearchCV
clf=GridSearchCV(forest,parameters, cv=5)

yr=np.ravel(y)
yr

clf.fit(X,yr)
```

Out[63]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [6, 9, 12, 15, 18],
                         'max_features': ['auto', 'sqrt', 'log2'],
```

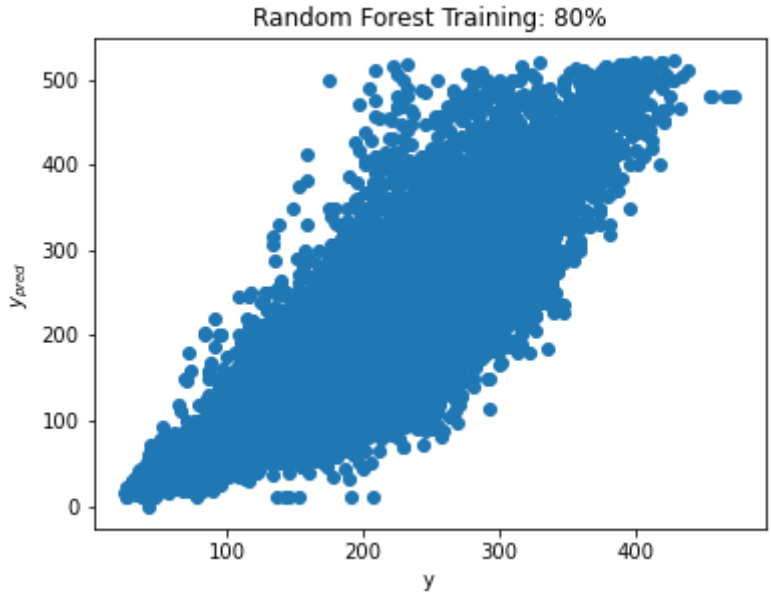
```
                                'min_samples_split': [2, 3, 4, 5, 6],
                                'random_state': [42]})
```

In [64]:
```
best_model=clf.best_estimator_
best_model.get_params()
```

Out[64]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 15,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

In [65]:
```
model_name.append('RandForest')

# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((best_model.predict(X_train)-np.array(y_train).T)**2))
RMSE_training.append(round(RMSE,2))
# Se muestra por pantalla ambos valores.
print( 'R2: '+'\t'+str(round(best_model.score(X_train,y_train),3))+'\n'+'RMSE: '+'\t
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(best_model.predict(X_train),y_train)
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Random Forest Training: 80%')
plt.show()


# Se tomará el error como el RMSE(raíz de la media de errores al cuadrado).
RMSE=np.sqrt(np.mean((best_model.predict(X_test)-np.array(y_test).T)**2))
RMSE_testing.append(round(RMSE,2))
# Se muestra por pantalla ambos valores.
print( 'R2: '+'\t'+str(round(best_model.score(X_test,y_test),3))+'\n'+'RMSE: '+'\t'+
# Scatter plot para graficar la comparativa.
plt.figure(figsize=(6,4.5))
plt.scatter(best_model.predict(X_test),y_test)
plt.xlabel('y')
plt.ylabel('$y_{pred}$')
plt.title('Random Forest Testing: 20%')
plt.show()
```

```
R2:     0.719
RMSE:   56.59
```




```
R2:     0.715
RMSE:   56.98
```

El modelo de Random Forest ha resultado generar una predicción con un una reducción de error de cerca de 20 dólares respecto los 2 anteriores modelos.

Dicha reducción de error refleja a su vez una R2 mucho mayor, cerca de 0.72.

Una vez probados los distintos modelos, Se hará una tabla para comentar los resultados finales.

In [72]:
```
d = {'Modelo': model_name, 'Error training($)': RMSE_training, 'Error test($)': RMSE

results=pd.DataFrame(data=d)
results
```

Out[72]:

	Modelo	Error training($)	Error test($)
0	LinRegress	77.56	77.51
1	DecisTree	74.84	75.42
2	RandForest	56.59	56.98

Conclusiones

De acuerdo con los 3 modelos aplicados, se ha optado por realizar una tabla mostrando solamente el error realizado.

Antes de realizar un veredicto, comentar virtudes/defectos de cada modelo utilizado:

1. Regresión Lineal: Es el caso más rápido, resoluble matemáticamente. Dada la alta correlación con 'bathrooms' y 'accommodates' son las mínimas para usar en dicho modelo.
2. Árbol de Decision: Dada su naturaleza de "categorizar" a través de la reducción de entropía, si no se pone un límite puede sobreentrenarse, se requiere hacer uso de Gridsearch y aplicar validación cruzada(5-fold para proporción 80/20).
3. Random Forest: Genera aleatoriamente 100 árboles de decisión y la salida se elige de acuerdo con la categoría mayoritaria. Es el modelo con mayor coste computacional utilizado.

Dicho todo lo anterior, en el caso que el tiempo de computación sea un problema, la regresión lineal ofrece un error similar al árbol de decisión, difieren en 3 dólares.

En caso de querer ofrer la mejor predicción, Random forest ha sido el modelo con mejor ejecución. En base al precio propuesto por la predicción, será la que ofrezca menor error.

Trabajo futuro

Como trabajo futuro se me ocurren diferentes posibilidades que se han descartado para simplificar el modelo. Entre las distintas posibilidades:

1. Tener en cuenta las variables de servicios descartadas: Aunque hay alguna orientada a la cantidad de reseñas, la de poner foto del hospedador o que esté verificado pueden aportar algo.
2. Categorizar la columna 'amenities': Como se ha comentado anteriormente, no ofrece el mismo servicio tener piscina que tener plancha o aire acondicionado. Dicha discriminación puede aportar en lo que se refiere a métodos como Árbol de Decisión o Random forest. Por otra parte, puede conllevar un tiempo computacional elevado.