Machine Learning 101: Predicción de Precios de Alquileres en Madrid

Práctica Final

Alberto Muñoz Freán

In [1]:

```
#En primer lugar, importamos funciones que nos serán útiles a lo largo de todo el proceso.

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn.model_selection import train_test_split

#Modificamos también las opciones de print para ver mejor los datos durante el análisis exp
pd.set_option('max_columns', None)
pd.set_option('max_rows', None)
```

Primero, se cargan los datos del csv en un data frame, de modo que podamos trabajar con ellos:

In [2]:

```
full_df = pd.read_csv('./airbnb-listings-extract.csv', sep=';', decimal='.')
```

Una vez se han cargado con éxito, se procederá al train-test split (80-20) generando dos archivos csv, de modo que el preprocesamiento y análisis exploratorio no afecten a los datos de test:

In [6]:

```
train, test = train_test_split(full_df, test_size=0.2, shuffle=True, random_state=0)

train.to_csv('./train.csv', sep=';', decimal='.', index=False)
test.to_csv('./test.csv', sep=';', decimal='.', index=False)
```

Estos datasets tienen una gran cantidad de columnas. Es muy posible que no todas sean relevantes:

In [31]:

```
#Creamos un data frame (df) con los datos del csv de train:
df = pd.read_csv('./train.csv', sep=';', decimal='.')

#Generamos una lista con el nombre de todas las columnas (variables) del data frame:
df.isna().sum()
```

Out[31]:

```
ID                                  0
Listing Url                         0
Scrape ID                           0
Last Scraped                        0
Name                                1
Summary                           469
Space                            3106
Description                         6
Experiences Offered                 0
Neighborhood Overview            4515
Notes                            7281
Transit                          4581
Access                           5161
Interaction                      5230
House Rules                      4143
Thumbnail Url                    2295
Medium Url                       2295
Picture Url                        18
XL Picture Url                   2295
Host ID                             0
Host URL                            0
Host Name                           3
Host Since                          3
Host Location                      34
Host About                       4160
Host Response Time               1507
Host Response Rate               1507
Host Acceptance Rate            11794
Host Thumbnail Url                  3
Host Picture Url                    3
Host Neighbourhood               3082
Host Listings Count                 3
Host Total Listings Count           3
Host Verifications                  4
Street                              0
Neighbourhood                    4159
Neighbourhood Cleansed              0
Neighbourhood Group Cleansed      818
City                                4
State                             119
Zipcode                           398
Market                             44
Smart Location                      0
Country Code                        0
Country                             1
Latitude                            0
Longitude                           0
Property Type                       0
Room Type                           0
Accommodates                        0
Bathrooms                          44
```

```
Bedrooms                          20
Beds                              37
Bed Type                           0
Amenities                        141
Square Feet                    11350
Price                             15
Weekly Price                    8943
Monthly Price                   8955
Security Deposit                6772
Cleaning Fee                    4850
Guests Included                    0
Extra People                       0
Minimum Nights                     0
Maximum Nights                     0
Calendar Updated                   0
Has Availability               11815
Availability 30                    0
Availability 60                    0
Availability 90                    0
Availability 365                   0
Calendar last Scraped              0
Number of Reviews                  0
First Review                    2539
Last Review                     2540
Review Scores Rating            2661
Review Scores Accuracy          2681
Review Scores Cleanliness       2676
Review Scores Checkin           2688
Review Scores Communication     2677
Review Scores Location          2691
Review Scores Value             2692
License                        11547
Jurisdiction Names             11650
Cancellation Policy                0
Calculated host listings count     4
Reviews per Month               2539
Geolocation                        0
Features                           1
dtype: int64
```

En el caso de 'Square_Feet', el 96% de los valores son nulos, de modo que habría que rellenar la inmensa mayoría de las celdas: es mejor eliminar la columna ante la falta de datos. Lo mismo se aplicará a todas las varibales que presentan más de un 30-35% de nulos.

'Weekly Price' y 'Monthly Price' contienen información redundante con respecto a la variable objetivo, de modo que también serán eliminadas. Lo mismo pasa con 'Geolocation', cuya información ya está contenida en 'Latitude' y 'Longitude'

Todas las varibales relacionadas con 'ID', 'URL', 'Host' o 'Scraping' no contienen información relevante, al igual que 'Name', 'Summary', 'Space' y 'Description'. También se eliminarán del data frame.

A su vez, se filtrará el campo 'City' por "Madrid", que es la ciudad de interés para el modelo. Por esta razón, los campos 'City', 'State', 'Market' y 'Country' dejarán de ser necesarios una vez hecho el filtrado:

In [3]:

```
#Filtramos por 'Madrid':
df1 = df[df['City'].str.contains("Madrid", na = False)]

#Dropeamos las columnas no necesarias:
df_train = df.drop(['Square Feet', 'Has Availability', 'License',
                    'Jurisdiction Names', 'Weekly Price', 'Name',
                    'Monthly Price', 'ID', 'Listing Url', 'Summary',
                    'Scrape ID', 'Last Scraped', 'Thumbnail Url',
                    'Medium Url', 'Picture Url', 'XL Picture Url',
                    'Host ID', 'Host URL', 'Host Name', 'Host Since',
                    'Host Location', 'Host About','Host Response Time',
                    'Host Response Rate', 'Host Acceptance Rate',
                    'Host Thumbnail Url', 'Host Picture Url', 'Description',
                    'Host Neighbourhood', 'Host Listings Count', 'Space',
                    'Host Verifications', 'City', 'State', 'Country Code',
                    'Country', 'Calendar last Scraped', 'Neighborhood Overview',
                    'Notes', 'Transit', 'Access', 'Interaction', 'House Rules',
                    'Security Deposit', 'Cleaning Fee', 'Host Total Listings Count',
                    'Market', 'Calculated host listings count', 'First Review',
                    'Last Review', 'Calendar Updated', 'Geolocation'], axis=1)

#Visualizamos los resultados para verificarlos:
df_train.head()
```

Out[3]:

	Experiences Offered	Street	Neighbourhood	Neighbourhood Cleansed	Neighbourhood Group Cleansed	Zipcode	Sma Locatic
0	none	Jerónimos, Madrid, Comunidad de Madrid 28014, ...	Jerónimos	Jerónimos	Retiro	28014	Madri Spa
1	none	Madrid, Comunidad de Madrid 28012, Spain	NaN	Sol	Centro	28012	Madri Spa
2	none	Carabanchel, Madrid, Comunidad de Madrid 28025...	Carabanchel	Vista Alegre	Carabanchel	28025	Madri Spa
3	none	Madrid, Comunidad de Madrid 28012, Spain	NaN	Embajadores	Centro	28012	Madri Spa
4	none	Gaztambide, Madrid, 28 28015, Spain	Gaztambide	Gaztambide	Chamberí	28015	Madri Spa

El siguiente paso es rellenar las celdas vacías de las columnas restantes, pues los valores nulos no son compatibles con los algoritmos de Machine Learning:

In [4]:

```
df_train.isna().sum()
```

Out[4]:

```
Experiences Offered               0
Street                            0
Neighbourhood                  4159
Neighbourhood Cleansed            0
Neighbourhood Group Cleansed    818
Zipcode                         398
Smart Location                    0
Latitude                          0
Longitude                         0
Property Type                     0
Room Type                         0
Accommodates                      0
Bathrooms                        44
Bedrooms                         20
Beds                             37
Bed Type                          0
Amenities                       141
Price                            15
Guests Included                   0
Extra People                      0
Minimum Nights                    0
Maximum Nights                    0
Availability 30                   0
Availability 60                   0
Availability 90                   0
Availability 365                  0
Number of Reviews                 0
Review Scores Rating           2661
Review Scores Accuracy         2681
Review Scores Cleanliness      2676
Review Scores Checkin          2688
Review Scores Communication    2677
Review Scores Location         2691
Review Scores Value            2692
Cancellation Policy               0
Reviews per Month              2539
Features                          1
dtype: int64
```

'Price' es la variable objetivo, y al tener menos de un 1% de nulos, se dropearán las filas.

Todas las variables numéricas o discretas ('Bathrooms', 'Bedrooms', 'Beds' y los Scores) se completarán usando la media aritmética, a fin de rellenar la información con un valor numérico representativo.

Los valores nulos del resto de variables se completarán con 'Unknown', al ser variables categóricas que contienen principalmente texto:

In [5]:

```
#Creamos un diccionario con los valores 'Unknown' que añadiremos en las variables categóric
values = {'Neighbourhood': 'Unknown', 'Neighbourhood Group Cleansed': 'Unknown', 'Zipcode':
          'Features': 'Unknown'}

#Los valores nulos se rellenan con los valores del diccionario:
df_train2 = df_train.fillna(value=values)

#Rellenamos las celdas de cada columna con su correspondiente media aritmética (Perdón por
#conseguí hacer funcionar la función que pusiste por Slack y me estaba volviendo loco ya):
df_train2['Bathrooms'].fillna((df_train2['Bathrooms'].mean()), inplace=True)
df_train2['Bedrooms'].fillna((df_train2['Bedrooms'].mean()), inplace=True)
df_train2['Beds'].fillna((df_train2['Beds'].mean()), inplace=True)
df_train2['Review Scores Rating'].fillna((df_train2['Review Scores Rating'].mean()), inplac
df_train2['Review Scores Accuracy'].fillna((df_train2['Review Scores Accuracy'].mean()), in
df_train2['Review Scores Cleanliness'].fillna((df_train2['Review Scores Cleanliness'].mean(
df_train2['Review Scores Checkin'].fillna((df_train2['Review Scores Checkin'].mean()), inpl
df_train2['Review Scores Communication'].fillna((df_train2['Review Scores Communication'].m
df_train2['Review Scores Location'].fillna((df_train2['Review Scores Location'].mean()), in
df_train2['Review Scores Value'].fillna((df_train2['Review Scores Value'].mean()), inplace=
df_train2['Reviews per Month'].fillna((df_train2['Reviews per Month'].mean()), inplace=True

#Se eliminan todas las filas que aún contienen nulos (que serán únicamente los de la column
df_train3 = df_train2.dropna()

#Comprobamos que no quedan nulos:
df_train3.isna().sum()
```

Out[5]:

```
Experiences Offered             0
Street                          0
Neighbourhood                   0
Neighbourhood Cleansed          0
Neighbourhood Group Cleansed    0
Zipcode                         0
Smart Location                  0
Latitude                        0
Longitude                       0
Property Type                   0
Room Type                       0
Accommodates                    0
Bathrooms                       0
Bedrooms                        0
Beds                            0
Bed Type                        0
Amenities                       0
Price                           0
Guests Included                 0
Extra People                    0
Minimum Nights                  0
Maximum Nights                  0
Availability 30                 0
Availability 60                 0
Availability 90                 0
Availability 365                0
Number of Reviews               0
Review Scores Rating            0
Review Scores Accuracy          0
Review Scores Cleanliness       0
```

```
Review Scores Checkin           0
Review Scores Communication     0
Review Scores Location          0
Review Scores Value             0
Cancellation Policy             0
Reviews per Month               0
Features                        0
dtype: int64
```

El siguiente paso será calcular la media, la desviación estándar y los cuartiles de las columnas del data frame. Con esto, podremos saber, entre otras cosas, si existen outliers:

In [6]:

```
df_train3.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	
Latitude	11809.0	40.485979	4.695057	-37.851182	40.409757	40.419315	40.4
Longitude	11809.0	-3.784060	14.020002	-123.124429	-3.707543	-3.700771	-3.6
Accommodates	11809.0	3.273605	2.088509	1.000000	2.000000	3.000000	4.0
Bathrooms	11809.0	1.283874	0.659153	0.000000	1.000000	1.000000	1.0
Bedrooms	11809.0	1.342610	0.900693	0.000000	1.000000	1.000000	2.0
Beds	11809.0	2.046555	1.614963	1.000000	1.000000	2.000000	2.0
Price	11809.0	73.712592	71.624844	9.000000	34.000000	55.000000	87.0
Guests Included	11809.0	1.580574	1.153438	0.000000	1.000000	1.000000	2.0
Extra People	11809.0	7.557710	11.160882	0.000000	0.000000	0.000000	15.0
Minimum Nights	11809.0	3.110255	13.435761	1.000000	1.000000	2.000000	3.0
Maximum Nights	11809.0	961.712677	9393.591785	1.000000	365.000000	1125.000000	1125.0
Availability 30	11809.0	8.943094	9.332425	0.000000	0.000000	6.000000	14.0
Availability 60	11809.0	22.939453	19.753442	0.000000	4.000000	20.000000	38.0
Availability 90	11809.0	39.792023	29.649710	0.000000	11.000000	38.000000	65.0
Availability 365	11809.0	202.196799	127.977302	0.000000	78.000000	240.000000	319.0
Number of Reviews	11809.0	22.680752	38.108399	0.000000	1.000000	7.000000	27.0
Review Scores Rating	11809.0	91.626402	8.046474	20.000000	90.000000	91.628179	97.0
Review Scores Accuracy	11809.0	9.409810	0.825232	2.000000	9.000000	9.410040	10.0
Review Scores Cleanliness	11809.0	9.320604	0.883966	2.000000	9.000000	9.320726	10.0
Review Scores Checkin	11809.0	9.624074	0.706710	2.000000	9.623905	10.000000	10.0
Review Scores Communication	11809.0	9.648132	0.672737	2.000000	9.647863	10.000000	10.0
Review Scores Location	11809.0	9.534593	0.677275	2.000000	9.000000	10.000000	10.0
Review Scores Value	11809.0	9.211064	0.846785	2.000000	9.000000	9.211345	10.0
Reviews per Month	11809.0	1.873688	1.656730	0.020000	0.620000	1.873326	2.2

A simple vista, 'Bathrooms' y 'Bedrooms' no deberían contener valores decimales, y 'Bathrooms' no

debería ser nunca cero.

Para modificar esto, los registros con valores de cero se dropearán al ser pocos, y los que contengan decimales pasarán a enteros por redondeo:

In [7]:

```
df_train3['Bathrooms'].value_counts()
```

Out[7]:

```
1.000000    8846
2.000000    1829
1.500000     434
3.000000     258
2.500000      98
4.000000      62
0.500000      56
0.000000      53
5.000000      50
1.285229      43
6.000000      23
3.500000      22
4.500000      19
5.500000       7
8.000000       4
7.000000       3
6.500000       1
7.500000       1
Name: Bathrooms, dtype: int64
```

In [8]:

```
#Redondeamos los datos:
df_train3.Bathrooms = df_train3.Bathrooms.round()
df_train3.Bedrooms = df_train3.Bedrooms.round()

#Eliminamos los resultados de 0 Bathrooms:
df_train4 = df_train3[(df_train3.Bathrooms != 0)]

#Comprobamos el resultado:
df_train4.describe().T
```

```
C:\Users\amfre\anaconda3\lib\site-packages\pandas\core\generic.py:5303: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  self[name] = value
```

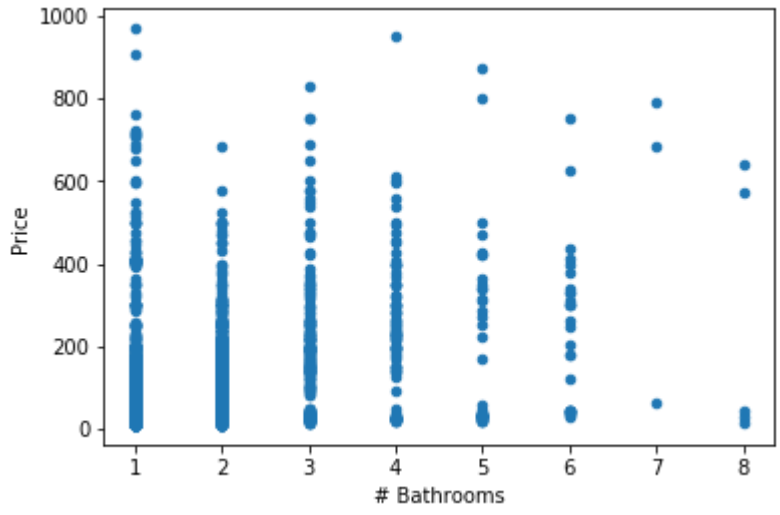
Out[8]:

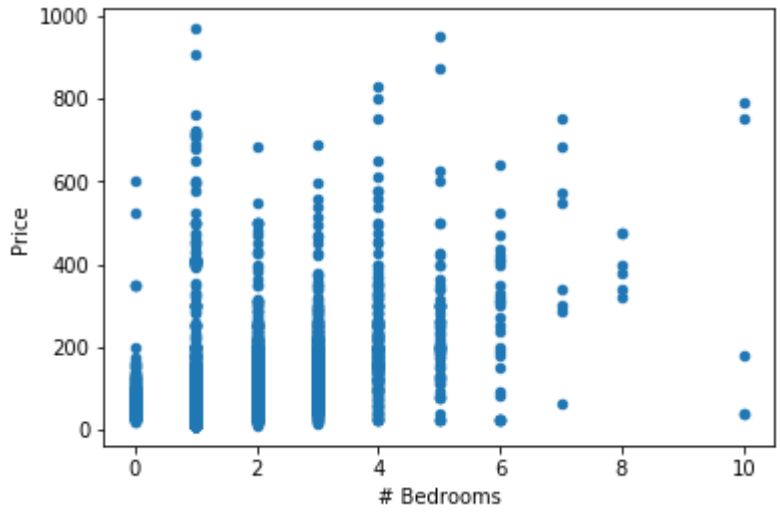
	count	mean	std	min	25%	50%	
Latitude	11700.0	40.487530	4.659978	-37.851182	40.409734	40.419253	4
Longitude	11700.0	-3.802775	14.010026	-123.124429	-3.707487	-3.700770	-
Accommodates	11700.0	3.286496	2.090557	1.000000	2.000000	3.000000	
Bathrooms	11700.0	1.307179	0.663095	1.000000	1.000000	1.000000	
Bedrooms	11700.0	1.344615	0.900680	0.000000	1.000000	1.000000	
Beds	11700.0	2.051262	1.617425	1.000000	1.000000	2.000000	
Price	11700.0	74.071026	71.839908	10.000000	35.000000	56.000000	8
Guests Included	11700.0	1.585385	1.157401	0.000000	1.000000	1.000000	
Extra People	11700.0	7.571197	11.169626	0.000000	0.000000	0.000000	1
Minimum Nights	11700.0	3.119487	13.494250	1.000000	1.000000	2.000000	
Maximum Nights	11700.0	962.398034	9437.155962	1.000000	365.000000	1125.000000	112
Availability 30	11700.0	8.918974	9.300233	0.000000	0.000000	6.000000	1
Availability 60	11700.0	22.903248	19.699418	0.000000	4.000000	20.000000	3
Availability 90	11700.0	39.750342	29.579276	0.000000	12.000000	38.000000	6
Availability 365	11700.0	202.084188	127.937517	0.000000	78.000000	240.000000	31
Number of Reviews	11700.0	22.722991	38.139884	0.000000	1.000000	7.000000	2
Review Scores Rating	11700.0	91.636203	8.035248	20.000000	90.000000	91.628179	9
Review Scores Accuracy	11700.0	9.410999	0.824043	2.000000	9.000000	9.410040	1

	count	mean	std	min	25%	50%	
Review Scores Cleanliness	11700.0	9.322690	0.881448	2.000000	9.000000	9.320726	1
Review Scores Checkin	11700.0	9.624208	0.707370	2.000000	9.623905	10.000000	1
Review Scores Communication	11700.0	9.648844	0.672114	2.000000	9.647863	10.000000	1
Review Scores Location	11700.0	9.534937	0.677247	2.000000	9.000000	10.000000	1
Review Scores Value	11700.0	9.211697	0.847505	2.000000	9.000000	9.211345	1
Reviews per							

In [9]:

```
#Un scatter plot ayuda a comprobar si hay necesidad de un filtrado más fino:
df_train4.plot(kind = 'scatter',x='Bathrooms',y = 'Price')
plt.xlabel('# Bathrooms')
plt.ylabel('Price')
plt.show()

df_train4.plot(kind = 'scatter',x='Bedrooms',y = 'Price')
plt.xlabel('# Bedrooms')
plt.ylabel('Price')
plt.show()
```





Si bien es cierto que hay alojamientos con muchos baños y/o habitaciones que parecen tener un precio muy bajo, se observa que, al aumentar cualquiera de estas dos variables, la mayoría de los puntos se van congregando en precios más altos, de modo que la tendencia de los datos es la esperada.

El siguiente paso es estudiar la relación entre variables. Para ello, se empezará por construir una matriz de correlación:

In [10]:

```
df_train4.corr()
```

Out[10]:

	Latitude	Longitude	Accommodates	Bathrooms	Bedrooms	Beds	P
Latitude	1.000000	-0.489911	0.005862	-0.002729	-0.010983	-0.000363	-0.013
Longitude	-0.489911	1.000000	0.018747	0.028545	0.039851	0.034778	0.005
Accommodates	0.005862	0.018747	1.000000	0.414755	0.725196	0.835325	0.527
Bathrooms	-0.002729	0.028545	0.414755	1.000000	0.514730	0.465723	0.364
Bedrooms	-0.010983	0.039851	0.725196	0.514730	1.000000	0.730143	0.493
Beds	-0.000363	0.034778	0.835325	0.465723	0.730143	1.000000	0.438
Price	-0.013207	0.005278	0.527757	0.364043	0.493902	0.438100	1.000
Guests Included	0.004733	-0.006958	0.564764	0.231342	0.436775	0.456603	0.295
Extra People	0.018611	-0.051027	0.251209	0.081387	0.134007	0.185342	0.103
Minimum Nights	-0.003447	-0.004189	0.006480	0.028969	0.020749	0.007279	0.029
Maximum Nights	0.000019	0.002396	0.001164	-0.003398	0.000263	0.001160	0.000
Availability 30	-0.021711	0.011670	-0.017403	0.039270	0.028712	0.043089	0.108
Availability 60	-0.024223	0.006095	-0.028937	0.010102	0.001215	0.025603	0.066
Availability 90	-0.026437	0.001064	-0.048752	-0.005638	-0.023057	0.002235	0.033
Availability 365	-0.007363	-0.007059	0.079509	0.019817	0.028335	0.090635	0.058
Number of Reviews	0.018706	-0.023955	0.057040	-0.076842	-0.047024	0.016143	-0.056
Review Scores Rating	-0.012385	-0.016872	-0.043301	0.005175	0.011055	-0.034799	0.045
Review Scores Accuracy	-0.011861	-0.012559	-0.041261	0.000307	0.005509	-0.044489	0.043
Review Scores Cleanliness	-0.011331	-0.003492	-0.014363	0.003752	0.009471	-0.012005	0.056
Review Scores Checkin	-0.006468	-0.016704	-0.060194	-0.036262	-0.022997	-0.045040	-0.008
Review Scores Communication	-0.010633	-0.021295	-0.044760	-0.037410	-0.007618	-0.031409	0.003
Review Scores Location	-0.016646	-0.021753	0.008788	-0.000527	-0.028150	-0.023812	0.075
Review Scores Value	-0.012593	-0.016348	-0.049594	-0.004849	-0.006086	-0.039284	0.010
Reviews per Month	0.007018	-0.005588	0.015192	-0.076795	-0.069590	-0.035170	-0.071

Valores de la matriz en el entorno de +/- 0.8 indican una correlación muy elevada entre dos variables. Esto puede suponer una fuente de error cuando se usan ciertos algoritmos de Machine Learning. Para evitarlo, se eliminará una de esas dos variables:

- Las variables Availability 30, 60 y 90 se observan muy relacionadas entre sí, como cabría esperar, pues son diferentes medidas temporales de la misma información. Para evitar redundancias, conservaremos solo la variable 'Availability 30'.

- 'Accomodates' y 'Beds' también presentan una gran relación entre sí, también lógico pues se espera una relación entre el número de huéspedes y las camas de un alojamiento. En este caso, conservaremos 'Accomodates', pues 'Bedrooms' debería aportar prácticamente la misma información que 'Beds', a pesar de que la matriz diga que su correlación no es tan alta.

- 'Accommodates' (Acc) presenta cierta correlación con 'Bathrooms' (Bath) y 'Bedrooms' (Bed). Sería interesante estudiar dicha relación, por ejemplo, creando dos nuevas columnas: 'Bath/Acc' y 'Bed/Acc':

In [11]:

```
#Creamos variables con los datos para rellenar las columnas:
BathAcc = df_train4['Bathrooms']/df_train4['Accommodates']
BedAcc = df_train4['Bedrooms']/df_train4['Accommodates']

#Creamos las columnas y usamos las variables para poblarlas:
df_train4['Bath/Acc'] = BathAcc
df_train4['Bed/Acc'] = BedAcc

#Movemos la columna 'Price' al final del data frame por mera comodidad:
col_name = df_train4['Price']
df_train4.pop('Price')
df_train4.insert(38, "Price", col_name)

#Eliminamos las variables ya expuestas anteriormente:
df_train5 = df_train4.drop(['Availability 60', 'Availability 90', 'Beds'], axis=1)

#Verificamos los resultados:
df_train5.head()
```

```
C:\Users\amfre\anaconda3\lib\site-packages\ipykernel_launcher.py:4: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  after removing the cwd from sys.path.
C:\Users\amfre\anaconda3\lib\site-packages\ipykernel_launcher.py:5: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  """
```

Out[11]:

	Experiences Offered	Street	Neighbourhood	Neighbourhood Cleansed	Neighbourhood Group Cleansed	Zipcode	S Loca
0	none	Jerónimos, Madrid, Comunidad de Madrid 28014, ...	Jerónimos	Jerónimos	Retiro	28014	Ma S
1	none	Madrid, Comunidad de Madrid 28012, Spain	Unknown	Sol	Centro	28012	Ma S
2	none	Carabanchel, Madrid, Comunidad de Madrid 28025...	Carabanchel	Vista Alegre	Carabanchel	28025	Ma S

	Experiences Offered	Street	Neighbourhood	Neighbourhood Cleansed	Neighbourhood Group Cleansed	Zipcode	S Loca
3	none	Madrid, Comunidad de Madrid 28012, Spain	Unknown	Embajadores	Centro	28012	Ma S

Por último, se hará una nueva matriz de correlación para comprobar si la generación de nuevas variables provoca algún cambio significativo:

In [12]:

```
df_train5.corr()
```

Out[12]:

	Latitude	Longitude	Accommodates	Bathrooms	Bedrooms	Guests Included	E: Pec
Latitude	1.000000	-0.489911	0.005862	-0.002729	-0.010983	0.004733	0.018
Longitude	-0.489911	1.000000	0.018747	0.028545	0.039851	-0.006958	-0.051
Accommodates	0.005862	0.018747	1.000000	0.414755	0.725196	0.564764	0.251
Bathrooms	-0.002729	0.028545	0.414755	1.000000	0.514730	0.231342	0.081
Bedrooms	-0.010983	0.039851	0.725196	0.514730	1.000000	0.436775	0.134
Guests Included	0.004733	-0.006958	0.564764	0.231342	0.436775	1.000000	0.359
Extra People	0.018611	-0.051027	0.251209	0.081387	0.134007	0.359523	1.000
Minimum Nights	-0.003447	-0.004189	0.006480	0.028969	0.020749	0.002882	-0.020
Maximum Nights	0.000019	0.002396	0.001164	-0.003398	0.000263	-0.004578	-0.003
Availability 30	-0.021711	0.011670	-0.017403	0.039270	0.028712	-0.052093	0.048
Availability 365	-0.007363	-0.007059	0.079509	0.019817	0.028335	0.061465	0.115
Number of Reviews	0.018706	-0.023955	0.057040	-0.076842	-0.047024	0.100165	0.077
Review Scores Rating	-0.012385	-0.016872	-0.043301	0.005175	0.011055	0.013854	0.024
Review Scores Accuracy	-0.011861	-0.012559	-0.041261	0.000307	0.005509	0.017253	0.013
Review Scores Cleanliness	-0.011331	-0.003492	-0.014363	0.003752	0.009471	0.028722	0.035
Review Scores Checkin	-0.006468	-0.016704	-0.060194	-0.036262	-0.022997	0.003385	0.021
Review Scores Communication	-0.010633	-0.021295	-0.044760	-0.037410	-0.007618	0.009127	0.018
Review Scores Location	-0.016646	-0.021753	0.008788	-0.000527	-0.028150	0.042210	0.044
Review Scores Value	-0.012593	-0.016348	-0.049594	-0.004849	-0.006086	0.008926	0.013
Reviews per Month	0.007018	-0.005588	0.015192	-0.076795	-0.069590	0.054841	0.000
Bath/Acc	-0.017430	0.006321	-0.502138	0.392449	-0.175682	-0.257870	-0.161
Bed/Acc	-0.029518	0.016846	-0.368536	0.116474	0.222588	-0.170779	-0.139
Price	-0.013207	0.005278	0.527757	0.364043	0.493902	0.295415	0.103

No es necesario eliminar más variables por el momento. El siguiente paso es codificar las variables categóricas, de modo que solo incluyan valores numéricos y así sean aptas para su procesamiento. Para hacer esto, se usará Mean Encoder:

In [13]:

```
#Creamos una lista cuyos registros coincidan con el nombre de las variables categóricas a c
categorical = ['Experiences Offered', 'Street', 'Neighbourhood', 'Neighbourhood Cleansed',
               'Zipcode', 'Smart Location', 'Property Type', 'Room Type', 'Bed Type', 'Amen
               'Features']

#Creamos un diccionario donde se irán guardando las medias:
mean_map = {}

#Este bucle irá sustituyendo los valores de las variables categóricas con las medias de la
#en función de los registros de la variable en cuestión:
for c in categorical:
    mean = df_train5.groupby(c)['Price'].mean()
    df_train5[c] = df_train5[c].map(mean)
    mean_map[c] = mean

#Verificamos el resultado:
df_train5.head()
```

Out[13]:

	Experiences Offered	Street	Neighbourhood	Neighbourhood Cleansed	Neighbourhood Group Cleansed	Zipcode	Sm Locat
0	73.964573	126.482759	112.162791	104.600000	70.400568	83.679208	66.524
1	73.964573	60.147268	75.039059	86.061198	73.108539	67.822747	66.524
2	73.964573	35.358974	40.168831	37.074074	39.860140	38.073684	66.524
3	73.964573	60.147268	75.039059	60.921196	73.108539	67.822747	66.524
4	73.964573	32.000000	57.845070	56.541284	73.000000	61.915254	66.524

Ahora que el data frame solo contiene valores numéricos, podemos empezar a probar modelos predictivos para la variable objetivo.

El primero será Random Forest. Una de las propiedades de los algoritmos basados en árboles, es que es posible medir la "importancia" de las variables para el modelo, con lo cual es posible hacer una selección de características aún más exhaustiva:

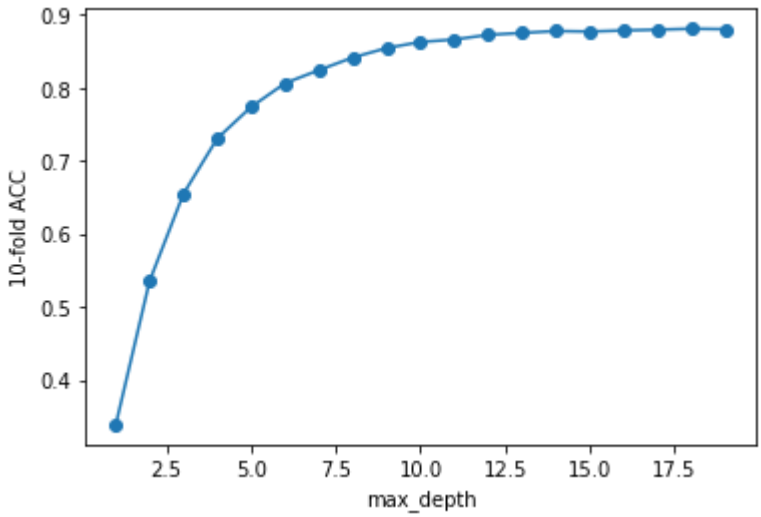
In [24]:

```
#Convertimos los valores del data frame en un formato compatible con el modelo:
data = df_train5.values

y_train = data[:,-1:]    # Este array contiene únicamente la variable objetivo 'Price'.
x_train = data[:,0:35]   # Este array contiene los datos del resto de variables del data fr
```

In [25]:

```
#Importamos las funciones necesarias:
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

#Ejecutamos Grid Search para obtener los parámetros óptimos del modelo (Profundidad del árb
maxDepth = range(1,20)
tuned_parameters = {'max_depth': maxDepth}

grid = GridSearchCV(RandomForestRegressor(random_state=0, n_estimators=200, max_features='s
grid.fit(x_train, y_train.ravel())

#Imprimimos los resultados obtenidos:
print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

#Representamos la gráfica:
scores = np.array(grid.cv_results_['mean_test_score'])
plt.plot(maxDepth,scores,'-o')
plt.xlabel('max_depth')
plt.ylabel('10-fold ACC')

plt.show()
```

```
best mean cross-validation score: 0.881
best parameters: {'max_depth': 18}
```



In [27]:

```
#Calculamos el score del modelo según la pronfundidad establecida:
maxDepthOptimo = 12
bagModel = RandomForestRegressor(max_depth=maxDepthOptimo,n_estimators=200,max_features='sq

print("Train: ",bagModel.score(x_train,y_train))
```

```
Train:  0.9589110126600372
```

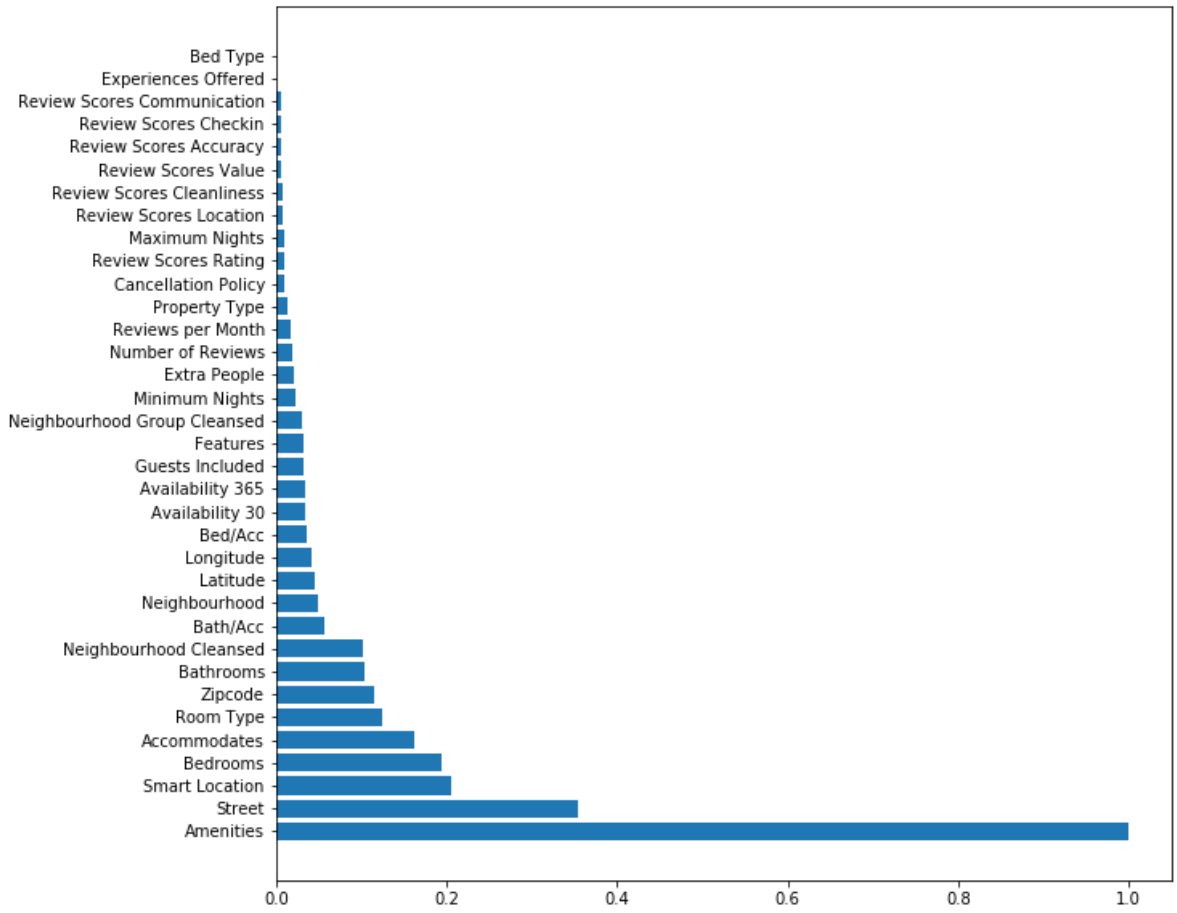
La profundidad óptima según grid search (18) nos da un score de 0.98, pero viendo la gráfica, se puede optar por 12, pues tiene casi la misma Accuracy y puede que generalice mejor. Al probarlo, el score es de 0.95, un score bastante alto teniendo en cuenta que el dataset tiene más de 10.000 registros.

Lo siguiente es observar la importancia de las variables por si es posible dropear alguna, y con ello reducir la complejidad del modelo y mejorar su interpretabilidad. Haciendo esto, se observa que bastantes variables tienen poca o nula importancia en la predicción: 'Experiences Offered', 'Bed Type', 'Review Scores Value', 'Review Scores Checkin', 'Review Scores Communication', 'Review Scores Accuracy', 'Review Scores Cleanliness', 'Review Scores Location', 'Maximum Nights', 'Cancellation Policy', 'Review Scores Rating' y 'Property Type' (ver gráfico debajo).

Al eliminar esas variables el resultado cambia ligeramente, pero la profundidad óptima sigue siendo 18, aunque en este caso, se puede disminuir hasta 11. El nuevo score para profundidad 11 es 0.94. No parece que eliminar tantas características haya tenido gran impacto en el modelo, de hecho, es posible que el modelo haya empeorado ligeramente. Por esta razón, nos quedamos con los valores originales y mantenemos todas las variables:

In [33]:

```
#Eliminamos la variable objetivo:
features = df_train5.columns.drop(['Price'])

#Determinamos los scores de importancia y los normalizamos:
importances = bagModel.feature_importances_
importances = importances / np.max(importances)

#Los ordenamos de forma creciente:
indices = np.argsort(importances)[::-1]

#Representamos los resultados en una gráfica:
plt.figure(figsize=(10,10))
plt.barh(range(x_train.shape[1]),importances[indices])
plt.yticks(range(x_train.shape[1]),features[indices])
plt.show()
```

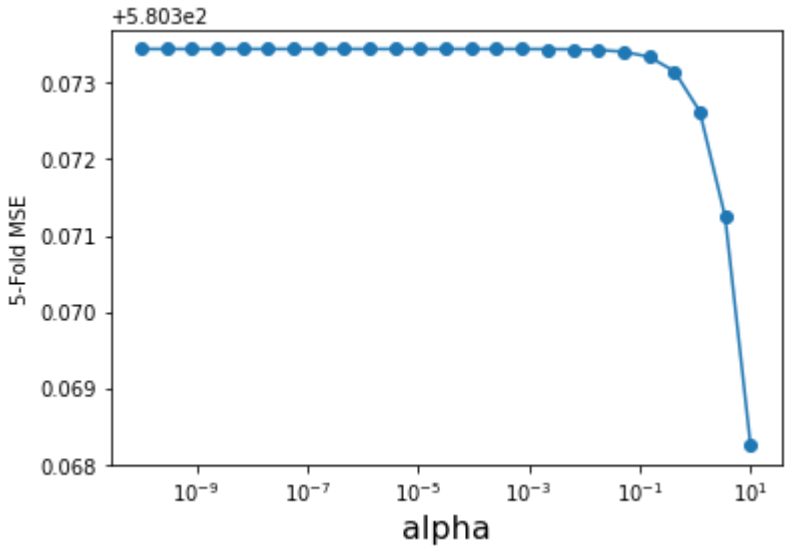


El siguiente método a probar será LASSO, que persigue prevenir el overfitting y mejorar la interpretabilidad:

In [29]:

```
#Importamos las funciones relevantes:
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge

#Ejecutamos GridSearch para obtener los parámetros óptimos (alpha):
alpha_vector = np.logspace(-10,1,25)
param_grid = {'alpha': alpha_vector }
grid = GridSearchCV(Ridge(), scoring='neg_mean_squared_error', param_grid=param_grid, cv=5)
grid.fit(x_train, y_train)
print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

#Representamos los resultados:
scores = -1*np.array(grid.cv_results_['mean_test_score'])
plt.semilogx(alpha_vector,scores,'-o')
plt.xlabel('alpha',fontsize=16)
plt.ylabel('5-Fold MSE')
plt.show()
```

```
best mean cross-validation score: -580.368
best parameters: {'alpha': 10.0}
```



Como último modelo, probaremos con un Boosted Tree:

In [32]:

```
#Importamos las funciones relevantes:
from sklearn.ensemble import GradientBoostingRegressor

#Determinamos los intervalos para iteraciones y Learning Rate:
Niterations = [1500, 2000, 2500]
learningRate = [0.1, 0.05]

#Ejecutamos GridSearch para obtener los parámetros óptimos del modelo (Learning Rate y Núme
param_grid = {'n_estimators': Niterations,'learning_rate':learningRate }
grid = GridSearchCV(GradientBoostingRegressor(random_state=0, max_depth=3), param_grid=para
grid.fit(x_train, y_train.ravel())

#Imprimimos los resultados:
print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))
```

```
best mean cross-validation score: 0.915
best parameters: {'learning_rate': 0.05, 'n_estimators': 1500}
```

Ahora que tenemos los parámetros óptimos de los tres modelos propuestos, podemos empezar a trabajar con los datos de test.

Lo primero es realizar exactamente el mismo preprocesado que han sufrido los datos de train:

In [35]:

```
df_test = pd.read_csv('./test.csv', sep=';', decimal='.')

df_test1 = df_test[df_test['City'].str.contains("Madrid", na = False)]

df_test1 = df_test.drop(['Square Feet', 'Has Availability', 'License',
                     'Jurisdiction Names', 'Weekly Price', 'Name',
                     'Monthly Price', 'ID', 'Listing Url', 'Summary',
                     'Scrape ID', 'Last Scraped', 'Thumbnail Url',
                     'Medium Url', 'Picture Url', 'XL Picture Url',
                     'Host ID', 'Host URL', 'Host Name', 'Host Since',
                     'Host Location', 'Host About','Host Response Time',
                     'Host Response Rate', 'Host Acceptance Rate',
                     'Host Thumbnail Url', 'Host Picture Url', 'Description',
                     'Host Neighbourhood', 'Host Listings Count', 'Space',
                     'Host Verifications', 'City', 'State', 'Country Code',
                     'Country', 'Calendar last Scraped', 'Neighborhood Overview',
                     'Notes', 'Transit', 'Access', 'Interaction', 'House Rules',
                     'Security Deposit', 'Cleaning Fee', 'Host Total Listings Count',
                     'Market', 'Calculated host listings count', 'First Review',
                     'Last Review', 'Calendar Updated', 'Geolocation'], axis=1)

values2 = {'Neighbourhood': 'Unknown', 'Neighbourhood Group Cleansed': 'Unknown', 'Zipcode'
           'Features': 'Unknown'}

df_test2 = df_test1.fillna(value=values2)

df_test2['Bathrooms'].fillna((df_test2['Bathrooms'].mean()), inplace=True)
df_test2['Bedrooms'].fillna((df_test2['Bedrooms'].mean()), inplace=True)
df_test2['Beds'].fillna((df_test2['Beds'].mean()), inplace=True)
df_test2['Review Scores Rating'].fillna((df_test2['Review Scores Rating'].mean()), inplace=
df_test2['Review Scores Accuracy'].fillna((df_test2['Review Scores Accuracy'].mean()), inpl
df_test2['Review Scores Cleanliness'].fillna((df_test2['Review Scores Cleanliness'].mean())
df_test2['Review Scores Checkin'].fillna((df_test2['Review Scores Checkin'].mean()), inplac
df_test2['Review Scores Communication'].fillna((df_test2['Review Scores Communication'].mea
df_test2['Review Scores Location'].fillna((df_test2['Review Scores Location'].mean()), inpl
df_test2['Review Scores Value'].fillna((df_test2['Review Scores Value'].mean()), inplace=Tr
df_test2['Reviews per Month'].fillna((df_test2['Reviews per Month'].mean()), inplace=True)

df_test3 = df_test2.dropna()

df_test3.isna().sum()
```

Out[35]:

```
Experiences Offered             0
Street                          0
Neighbourhood                   0
Neighbourhood Cleansed          0
Neighbourhood Group Cleansed    0
Zipcode                         0
Smart Location                  0
Latitude                        0
Longitude                       0
Property Type                   0
Room Type                       0
Accommodates                    0
Bathrooms                       0
Bedrooms                        0
Beds                            0
```

```
Bed Type                        0
Amenities                       0
Price                           0
Guests Included                 0
Extra People                    0
Minimum Nights                  0
Maximum Nights                  0
Availability 30                 0
Availability 60                 0
Availability 90                 0
Availability 365                0
Number of Reviews               0
Review Scores Rating            0
Review Scores Accuracy          0
Review Scores Cleanliness       0
Review Scores Checkin           0
Review Scores Communication     0
Review Scores Location          0
Review Scores Value             0
Cancellation Policy             0
Reviews per Month               0
Features                        0
dtype: int64
```

In [38]:

```
df_test3.Bathrooms = df_test3.Bathrooms.round()
df_test3.Bedrooms = df_test3.Bedrooms.round()

df_test4 = df_test3[(df_test3.Bathrooms != 0)]

BathAcc = df_test4['Bathrooms']/df_test4['Accommodates']
BedAcc = df_test4['Bedrooms']/df_test4['Accommodates']

df_test4['Bath/Acc'] = BathAcc
df_test4['Bed/Acc'] = BedAcc

col_name = df_test4['Price']
df_test4.pop('Price')
df_test4.insert(38, "Price", col_name)

df_test5 = df_test4.drop(['Availability 60', 'Availability 90', 'Beds'], axis=1)

categorical2 = ['Experiences Offered', 'Street', 'Neighbourhood', 'Neighbourhood Cleansed',
                'Zipcode', 'Smart Location', 'Property Type', 'Room Type', 'Bed Type', 'Amen
                'Features']

mean_map = {}

for c in categorical2:
    mean = df_test5.groupby(c)['Price'].mean()
    df_test5[c] = df_test5[c].map(mean)
    mean_map[c] = mean

#Verificamos el resultado:
df_test5.head()
```

```
C:\Users\amfre\anaconda3\lib\site-packages\pandas\core\generic.py:5303: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  self[name] = value
C:\Users\amfre\anaconda3\lib\site-packages\ipykernel_launcher.py:9: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  if __name__ == '__main__':
C:\Users\amfre\anaconda3\lib\site-packages\ipykernel_launcher.py:10: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
```

```
view-versus-a-copy)
  # Remove the CWD from sys.path while we load stuff.
```

Out[38]:

	Experiences Offered	Street	Neighbourhood	Neighbourhood Cleansed	Neighbourhood Group Cleansed	Zipcode	Sma Locatic
0	73.322845	33.777778	37.583333	26.166667	30.523810	29.058824	64.9719
1	73.322845	89.000000	85.100000	81.485000	72.542537	85.645631	64.9719
2	73.322845	61.888889	76.093484	91.171429	71.119171	71.447368	64.9719
3	73.322845	37.400000	76.093484	38.142857	35.350000	38.880000	64.9719
4	73.322845	59.157407	76.093484	58.939726	72.542537	65.102500	64.9719

Ahora que el data frame de test ha sido preprocesado, comenzaremos por aplicar el modelo de Random Forest:

In [43]:

```
data = df_test5.values

y_test = data[:,-1:]
x_test = data[:,0:35]
```

In [45]:

```
maxDepthOptimo = 12 #Parámetro óptimo según GridSearch
bagModel = RandomForestRegressor(max_depth=maxDepthOptimo,n_estimators=200,max_features='sq

print("Train: ",bagModel.score(x_train,y_train))
print("Test: ",bagModel.score(x_test,y_test))
```

```
C:\Users\amfre\anaconda3\lib\site-packages\ipykernel_launcher.py:2: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples,), for example using ravel().


Train:  0.9566536003366285
Test:  0.9126455033036793
```

In [65]:

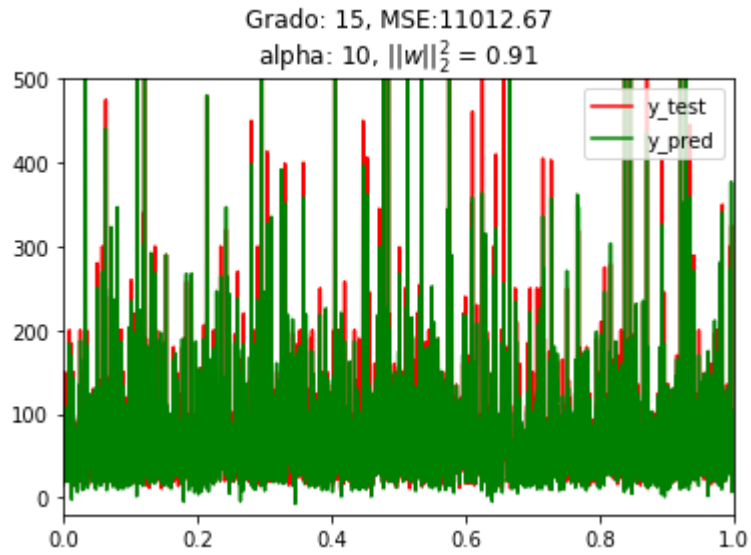
```
from sklearn.metrics import mean_squared_error
MSE_RF = mean_squared_error(y_test, bagModel.predict(x_test))
print("MSE de Random Forest: ", round(MSE_RF,2))
```

```
MSE de Random Forest:  478.43
```

A continuación, probaremos con LASSO:

In [140]:

```
from sklearn.linear_model import Lasso

N_test  = len(y_test)
x = np.linspace(0,1,N_test)
degree = 15

a = 10 #Parámetro óptimo según GridSearch
lasso = Lasso(alpha=a).fit(x_train,y_train)
w = lasso.coef_
norm_w2 = np.dot(w,w.T)

y_hat = lasso.predict(x_test)

error_test = np.mean(np.power(y_test - y_hat,2))

plt.plot(x,y_test,'r',label='y_test')
plt.plot(x,y_hat,'g',label='y_pred')
plt.title('Grado: %i, MSE:%.2f\nalpha: %g, $||w||_2^2$ = %.2g'%(degree,error_test,a,norm_w2
plt.legend()
plt.xlim((0, 1))
plt.ylim((-20, 500))
plt.show()

coef_names = ['w' + str(i) + ': ' for i in range(1,degree+1)]

for f,wi in zip(coef_names,w):
    print(f,wi)

print("MSE de LASSO: ", round(error_test,2))
```




```
w1:  0.0
w2:  0.3516969365541137
w3:  -0.05496181205979861
w4:  -0.06742720928630608
w5:  -0.004241924663845837
```

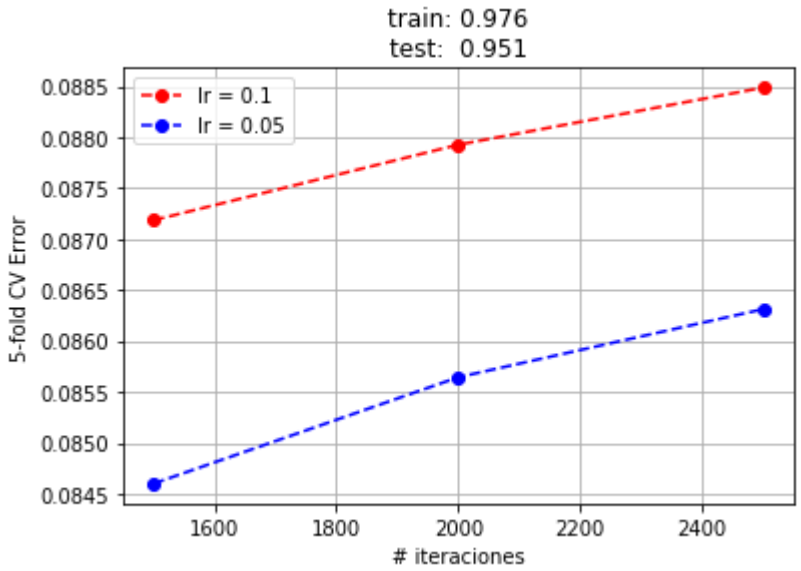
```
w6:  -0.15486895120353442
w7:  0.14014787396300216
w8:  0.0
w9:  -0.0
w10:  -0.021654266842084546
w11:  0.10102635124077125
w12:  0.0
w13:  0.0
w14:  0.0
w15:  0.0
MSE de LASSO:  11012.67
```

Por último, Boosted Tree:

In [57]:

```
lrOptimo = 0.05 #Parámetro óptimo según GridSearch.
neOptimo = 1500 #Parámetro óptimo según GridSearch.
bt = GradientBoostingRegressor(random_state=0, max_depth=3,learning_rate=lrOptimo, n_estima
bt.fit(x_train,y_train)

error = 1-grid.cv_results_['mean_test_score'].reshape(len(learningRate),len(Niterations))
colors = ['r','b','g','k','m']
for i,lr in enumerate(learningRate):
    plt.plot(Niterations,error[i,:],colors[i] + '--o',label='lr = %g'%lr)

plt.legend()
plt.xlabel('# iteraciones')
plt.ylabel('5-fold CV Error')
plt.title('train: %0.3f\ntest:  %0.3f'%(bt.score(x_train,y_train),bt.score(x_test,y_test)))
plt.grid()
plt.show()
```

```
C:\Users\amfre\anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:1454: Dat
aConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



In [64]:

```
from sklearn.metrics import mean_squared_error
MSE_BT = mean_squared_error(y_test, bt.predict(x_test))
print("MSE de Boosted Tree: ", round(MSE_BT,2))
```

```
MSE de Boosted Tree:  268.04
```

Resumen de los resultados (MSE):

- Random Forest: **478.43**
- LASSO: **11012.67**
- Boosted Tree: **268.04**

Aquel modelo con menor Error Cuadrático Medio (MSE) será el más adecuado, pues es el que menos se equivoca en comparación con los datos reales.

Siguiendo este criterio, LASSO es el modelo menos preciso, seguido de Random Forest.

Por tanto, Boosted Tree es el modelo más adecuado de los tres para predecir precios de alojamientos/habitaciones, de acuerdo con los datos de training utilizados y su preprocesado.