April 14, 2023

1 Lab 2: Building a Modeling Data Set

[1]:
```
import os
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
```

In this lab, you will complete the following tasks to build a modeling data set:

1. Load the Airbnb "listings" data set and identify the number of rows & columns
2. Remove features that are not currently useful for analysis; Modify features to make sure they are machine-comprehensible
3. Build a new regression label column by winsorizing outliers
4. Replace all missing values with means
5. Identify two features with the highest correlation with the label
6. Build appropriate bivariate plots between the highest correlated features and the label

1.1 Part 1. Load the Data

We will once again be working with the Airbnb NYC "listings" data set. Use the specified path and name of the file to load the data. Save it as a Pandas DataFrame called `df`.

[2]:
```
# Do not remove or edit the line below:
filename = os.path.join(os.getcwd(), "data", "listings.csv.gz")
```

Task: load the data and save it to DataFrame `df`.

[3]:
```
# YOUR CODE HERE

### Solution:
df = pd.read_csv(filename)
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2728:
DtypeWarning: Columns (67) have mixed types.Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Task: Display the shape of `df` -- that is, the number of rows and columns.

[4]:
```
# YOUR CODE HERE

### Solution:
df.shape
```

[4]:
```
(38277, 74)
```

Task: Get a peek at the data by displaying the first few rows, as you usually do.

[5]:
```
# YOUR CODE HERE

### Solution:
df.head()
```

[5]:

	id	listing_url	scrape_id	last_scraped	name	description	neighborhood_overview	picture_url	host_id
0	2595	https://www.airbnb.com/rooms/2595	20211204143024	2021-12-05	Skylit Midtown Castle	Beautiful, spacious skylit studio in the heart...	Centrally located in the heart of Manhattan ju...	https://a0.muscache.com/pictures/f0813a11-40b2...	2845
1	3831	https://www.airbnb.com/rooms/3831	20211204143024	2021-12-05	Whole flr w/private bdrm, bath & kitchen(pls r...	Enjoy 500 s.f. top floor in 1899 brownstone, w...	Just the right mix of urban center and local n...	https://a0.muscache.com/pictures/e49999c2-9fd5...	4869
2	5121	https://www.airbnb.com/rooms/5121	20211204143024	2021-12-05	BlissArtsSpace!	The space HELLO EVERYONE AND THANK...	NaN	https://a0.muscache.com/pictures/2090980c-b68e...	7356
3	5136	https://www.airbnb.com/rooms/5136	20211204143024	2021-12-05	Spacious Brooklyn Duplex, Patio + Garden	We welcome you to stay in our lovely 2 br dupl...	NaN	https://a0.muscache.com/pictures/miso/Hosting-...	7378
4	5178	https://www.airbnb.com/rooms/5178	20211204143024	2021-12-05	Large Furnished Room Near B'way	Please dont expect the luxury here just a bas...	Theater district, many restaurants around here.	https://a0.muscache.com/pictures/12065/f070997...	8967

```
                                host_url  ... review_scores_communication  \
0  https://www.airbnb.com/users/show/2845  ...                        4.79
1  https://www.airbnb.com/users/show/4869  ...                        4.80
2  https://www.airbnb.com/users/show/7356  ...                        4.91
3  https://www.airbnb.com/users/show/7378  ...                        5.00
4  https://www.airbnb.com/users/show/8967  ...                        4.42

  review_scores_location review_scores_value license instant_bookable  \
0                   4.86                4.41     NaN                 f
1                   4.71                4.64     NaN                 f
2                   4.47                4.52     NaN                 f
3                   4.50                5.00     NaN                 f
4                   4.87                4.36     NaN                 f

  calculated_host_listings_count calculated_host_listings_count_entire_homes  \
0                              3                                            3
1                              1                                            1
2                              2                                            0
3                              1                                            1
4                              1                                            0

  calculated_host_listings_count_private_rooms  \
0                                            0
1                                            0
2                                            2
3                                            0
4                                            1

  calculated_host_listings_count_shared_rooms reviews_per_month
0                                           0              0.33
1                                           0              4.86
2                                           0              0.52
3                                           0              0.02
4                                           0              3.68

[5 rows x 74 columns]
```

1.2 Part 2. Feature Selection and Engineering

We won't need the data fields that contain free, unstructured text. For example, we wont need the columns that contain apartment descriptions supplied by the host, customer reviews, or descriptions of the neighborhoods in which a listing is located.

The code cell below contains a list containing the names of *unstructured text* columns.

[6]:
```
unstr_text_colnames = ['description', 'name', 'neighborhood_overview',␣
↪'host_about', 'host_name', 'host_location']
```

Task: Drop the columns with the specified names, *in place* (that is, make sure this change

applies to the original DataFrame `df`, instead of creating a temporary new DataFrame with fewer columns).

[7]:
```
# YOUR CODE HERE

### Solution:
df.drop(unstr_text_colnames, axis = 1, inplace=True)
```

Task: Display the shape of the data to verify that the new number of columns is what you expected.

[8]:
```
# YOUR CODE HERE

### Solution:
df.shape
```

[8]:
```
(38277, 68)
```

We will furthermore get rid of all the columns which contain website addresses (URLs).

Task: Create a list which contains the names of columns that contain URLs. Save the resulting list to variable `url_colnames`.

Tip: There are different ways to accomplish this, including using Python list comprehensions

[9]:
```
# YOUR CODE HERE

### Solution:
url_colnames = [x for x in list(df.columns) if '_url' in x]
```

Task: Drop the columns with the specified names contained in list `url_colnames` in place (that is, make sure this change applies to the original DataFrame `df`, instead of creating a temporary new DataFrame object with fewer columns).

[10]:
```
# YOUR CODE HERE

### Solution:
df.drop(url_colnames, axis = 1, inplace=True)
```

Task: Another property of this dataset is that the `price` column contains values that are listed as <currency_name><numeric_value>. For example, it contains values that look like this: `$120`.

Let's look at the first 15 unique values of this column.

Display the first 15 unique values of the `price` column:

[11]:
```
# YOUR CODE HERE

# solution:
df['price'].unique()[:15]
```

[11]:
```
array(['$150.00', '$75.00', '$60.00', '$275.00', '$68.00', '$98.00',
       '$89.00', '$65.00', '$62.00', '$90.00', '$199.00', '$96.00',
       '$299.00', '$140.00', '$175.00'], dtype=object)
```

In order for us to use the prices for modeling, we will have to transform all values of this `price` feature into regular floats. We will first need to remove the dollar signs (in this case, the platform forces the currency to be the USD, so we do not need to worry about targeting, say, the Japanese

Yen sign, nor about converting the values into USD). Furthermore, we need to remove commas from all values that are in the thousands or above: for example, `$2,500$`. Here is how to do both:

[12]:
```
df['price'] = df['price'].str.replace(',', '')
df['price'] = df['price'].str.replace('$', '')
df['price'] = df['price'].astype(float)
```

Let's display the first few unique values again, to make sure they are transformed:

[13]:
```
df['price'].unique()[:15]
```

[13]:
```
array([150.,  75.,  60., 275.,  68.,  98.,  89.,  65.,  62.,  90., 199.,
        96., 299., 140., 175.])
```

Well done! Our transformed dataset looks like this:

[14]:
```
df.head()
```

[14]:

	id	scrape_id	last_scraped	host_id	host_since	host_response_time	host_response_rate	host_acceptance_rate	host_is_superhost	host_neighbourhood	...	review_scores_communication	review_scores_location	review_scores_value	license	instant_bookable	calculated_host_listings_count	calculated_host_listings_count_entire_homes
0	2595	20211204143024	2021-12-05	2845	2008-09-09	within a day	80%	17%	f	Midtown	...	4.79	4.86	4.41	NaN	f	3	3
1	3831	20211204143024	2021-12-05	4869	2008-12-07	a few days or more	9%	69%	f	Clinton Hill	...	4.80	4.71	4.64	NaN	f	1	1
2	5121	20211204143024	2021-12-05	7356	2009-02-03	within an hour	100%	100%	f	Bedford-Stuyvesant	...	4.91	4.47	4.52	NaN	f	2	0
3	5136	20211204143024	2021-12-05	7378	2009-02-03	within a day	100%	25%	f	Greenwood Heights	...	5.00	4.50	5.00	NaN	f	1	1
4	5178	20211204143024	2021-12-05	8967	2009-03-03	within a day	100%	100%	f	Hell's Kitchen	...	4.42	4.87	4.36	NaN	f	1	0

```
  calculated_host_listings_count_private_rooms  \
0                                            0
1                                            0
2                                            2
3                                            0
4                                            1

   calculated_host_listings_count_shared_rooms  reviews_per_month
0                                            0               0.33
1                                            0               4.86
2                                            0               0.52
3                                            0               0.02
4                                            0               3.68

[5 rows x 63 columns]
```

1.3 Part 3. Create a (Winsorized) Label Column

Assume that your goal is to use this dataset to fit a regression model that predicts the price under which a given space is listed.

Task: Create a new version of the `price` column, named `label_price`, in which we replace the top and bottom 1% outlier values with the corresponding percentile value. Add this new column to the DataFrame `df`.

Remember, you will first need to load the `stats` module from the `scipy` package:

[15]:
```
# YOUR CODE HERE - import the necessary package

### Solution:
import scipy.stats as stats
```

[16]:
```
# YOUR CODE HERE  - add the new column to the DataFrame

### Solution:
df['label_price'] = stats.mstats.winsorize(df['price'], limits=[0.01, 0.01])
```

Let's verify that a new column got added to the DataFrame:

[17]:
```
df.head()
```

[17]:
```
     id       scrape_id last_scraped  host_id  host_since  host_response_time  \
0  2595  20211204143024   2021-12-05     2845  2008-09-09        within a day
1  3831  20211204143024   2021-12-05     4869  2008-12-07  a few days or more
2  5121  20211204143024   2021-12-05     7356  2009-02-03      within an hour
3  5136  20211204143024   2021-12-05     7378  2009-02-03        within a day
4  5178  20211204143024   2021-12-05     8967  2009-03-03        within a day

  host_response_rate host_acceptance_rate host_is_superhost  \
0                80%                  17%                 f
1                 9%                  69%                 f
```

```
2               100%                100%                f
3               100%                 25%                f
4               100%                100%                f

   host_neighbourhood  ...  review_scores_location  review_scores_value  \
0             Midtown  ...                    4.86                 4.41
1        Clinton Hill  ...                    4.71                 4.64
2  Bedford-Stuyvesant  ...                    4.47                 4.52
3   Greenwood Heights  ...                    4.50                 5.00
4      Hell's Kitchen  ...                    4.87                 4.36

  license instant_bookable calculated_host_listings_count  \
0     NaN                f                              3
1     NaN                f                              1
2     NaN                f                              2
3     NaN                f                              1
4     NaN                f                              1

  calculated_host_listings_count_entire_homes  \
0                                           3
1                                           1
2                                           0
3                                           1
4                                           0

  calculated_host_listings_count_private_rooms  \
0                                            0
1                                            0
2                                            2
3                                            0
4                                            1

  calculated_host_listings_count_shared_rooms  reviews_per_month  label_price
0                                           0               0.33        150.0
1                                           0               4.86         75.0
2                                           0               0.52         60.0
3                                           0               0.02        275.0
4                                           0               3.68         68.0

[5 rows x 64 columns]
```

Task: Check that the values of `price` and `label_price` are *not* identical. Do this by subtracting the two columns and printing the *length* of the array (using the `len()` function) of *unique values* of the resulting difference. Note: If all values are identical, the difference would contain only one unique value -- zero. If this is the case, outlier removal did not work.

[18]:
```
# YOUR CODE HERE

```

```
### Solution:
len((df['price']-df['label_price']).unique())
```

[18]: 206

1.4 Part 4. Replace the Missing Values With Means

1.4.1 a. Identifying missingness

Task: Check if a given value in any data cell is missing, and sum up the resulting values (`True/False`) by columns. Save this sum to variable `nan_count`. Print the results.

[19]:
```
### Solution:
nan_count = np.sum(df.isnull(), axis = 0)
nan_count
```

[19]:
```
id                                                0
scrape_id                                         0
last_scraped                                      0
host_id                                           0
host_since                                       34
                                                ...
calculated_host_listings_count_entire_homes       0
calculated_host_listings_count_private_rooms      0
calculated_host_listings_count_shared_rooms       0
reviews_per_month                              9504
label_price                                       0
Length: 64, dtype: int64
```

Those are more columns than we can eyeball! For this exercise, we don't care about the number of missing values -- we just want to get a list of columns that have *any*.

Task: From variable `nan_count`, create a new series called `nan_detected` that contains `True/False` values that indicate whether the number of missing values is *not zero*:

[20]:
```
### Solution:
nan_detected = nan_count!=0
nan_detected
```

[20]:
```
id                                               False
scrape_id                                        False
last_scraped                                     False
host_id                                          False
host_since                                        True
                                                 ...
calculated_host_listings_count_entire_homes      False
calculated_host_listings_count_private_rooms     False
calculated_host_listings_count_shared_rooms      False
reviews_per_month                                 True
label_price                                      False
Length: 64, dtype: bool
```

Since replacing the missing values with the mean only makes sense for the numerically valued columns (and not for strings, for example), let us create another condition: the *type* of the column must be `int` or `float`.

Task: Create a series that contains `True` if the type of the column is either `int64` or `float64`. Save the result to variable `is_int_or_float`.

[21]:
```
### Solution:
is_int_or_float = (df.dtypes == 'int64') | (df.dtypes == 'float64')
is_int_or_float
```

[21]:
```
id                                                 True
scrape_id                                          True
last_scraped                                      False
host_id                                            True
host_since                                        False
                                                  ...
calculated_host_listings_count_entire_homes        True
calculated_host_listings_count_private_rooms       True
calculated_host_listings_count_shared_rooms        True
reviews_per_month                                  True
label_price                                        True
Length: 64, dtype: bool
```

Task: Combine the two binary series values into a new series named `to_impute`. It will contain the value `True` if a column contains missing values *and* is of type 'int' or 'float'

[22]:
```
### Solution:
to_impute = nan_detected & is_int_or_float
to_impute
```

[22]:
```
id                                                False
scrape_id                                         False
last_scraped                                      False
host_id                                           False
host_since                                        False
                                                  ...
calculated_host_listings_count_entire_homes       False
calculated_host_listings_count_private_rooms      False
calculated_host_listings_count_shared_rooms       False
reviews_per_month                                  True
label_price                                       False
Length: 64, dtype: bool
```

Finally, let's display a list that contains just the selected column names:

[23]:
```
df.columns[to_impute]
```

[23]:
```
Index(['host_listings_count', 'host_total_listings_count', 'bathrooms',
       'bedrooms', 'beds', 'minimum_minimum_nights', 'maximum_minimum_nights',
       'minimum_maximum_nights', 'maximum_maximum_nights',
       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'calendar_updated',
       'review_scores_rating', 'review_scores_accuracy',
```

```
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'reviews_per_month'],
      dtype='object')
```

We just identified and displayed the list of candidate columns for potentially replacing missing values with the column mean.

Assume that you have decided that it is safe to impute the values for `host_listings_count`, `host_total_listings_count`, `bathrooms`, `bedrooms`, and `beds`:

[24]:
```
to_impute_selected = ['host_listings_count', 'host_total_listings_count',␣
↪'bathrooms',
      'bedrooms', 'beds']
```

1.4.2 b. Keeping record of the missingness: creating dummy variables

As a first step, you will now create dummy variables indicating missingness of the values.

Task: Store the `True/False` series that indicate missingness of any value in a given column as a new variable called `<original-column-name>_na`.

[25]:
```
for colname in to_impute_selected:
    ### Solution (complete the loop):
    df[colname+'_na'] = df[colname].isnull()
```

Check that the DataFrame contains the new variables:

[26]:
```
df.head()
```

[26]:

	id	scrape_id	last_scraped	host_id	host_since	host_response_time	host_response_rate	host_acceptance_rate	host_is_superhost	host_neighbourhood	...	calculated_host_listings_count_entire_homes	calculated_host_listings_count_private_rooms
0	2595	20211204143024	2021-12-05	2845	2008-09-09	within a day	80%	17%	f	Midtown	...	3	0
1	3831	20211204143024	2021-12-05	4869	2008-12-07	a few days or more	9%	69%	f	Clinton Hill	...	1	
2	5121	20211204143024	2021-12-05	7356	2009-02-03	within an hour	100%	100%	f	Bedford-Stuyvesant	...	0	
3	5136	20211204143024	2021-12-05	7378	2009-02-03	within a day	100%	25%	f	Greenwood Heights	...	1	
4	5178	20211204143024	2021-12-05	8967	2009-03-03	within a day	100%	100%	f	Hell's Kitchen	...	0	

```
1                                                 0
2                                                 2
3                                                 0
4                                                 1

  calculated_host_listings_count_shared_rooms reviews_per_month label_price  \
0                                           0              0.33       150.0
1                                           0              4.86        75.0
2                                           0              0.52        60.0
3                                           0              0.02       275.0
4                                           0              3.68        68.0

  host_listings_count_na host_total_listings_count_na bathrooms_na  \
0                  False                         False         True
1                  False                         False         True
2                  False                         False         True
3                  False                         False         True
4                  False                         False         True

   bedrooms_na  beds_na
0         True    False
1        False    False
2        False    False
3        False    False
4        False    False

[5 rows x 69 columns]
```

1.4.3 c. Replacing the missing values with mean values of the column

Task: Fill the missing values of the selected few columns with the corresponding mean value.

[27]:
```
# YOUR CODE HERE

### Solution:
for colname in to_impute_selected:
    df[colname].fillna(np.mean(df[colname]), inplace=True)
```

Check your results below. The code displays the count of missing values for each of the selected columns.

[28]:
```
for colname in to_impute_selected:
    print("{} missing values count :{}".format(colname, np.sum(df[colname].
 ↪isnull(), axis = 0)))
```

```
host_listings_count missing values count :0
host_total_listings_count missing values count :0
bathrooms missing values count :38277
bedrooms missing values count :0
```

```
beds missing values count :0
```

Why did the `bathrooms` column retain missing values after our imputation?

Task: List the unique values of the `bathrooms` column.

[29]:
```
# YOUR CODE HERE

### Solution:
df['bathrooms'].unique()
```

[29]:
```
array([nan])
```

The column did not contain a single value (except the `NaN` indicator) to begin with.

1.5 Part 5. Identify Features With the Highest Correlation With the Label

Your next goal is to figure out which features in the data correlate most with the label.

In the next few cells, we will demonstrate how to use the Pandas `corr()` method to get a list of correlation coefficients between `label` and all other (numerical) features.

Let's first glance at what the `corr()` method does:

[30]:
```
df.corr().head()
```

[30]:
```
                                     id     scrape_id       host_id  \
id                         1.000000e+00 -4.269620e-13  5.861676e-01
scrape_id                 -4.269620e-13  1.000000e+00  2.367225e-13
host_id                    5.861676e-01  2.367225e-13  1.000000e+00
host_listings_count        1.298621e-01 -2.061273e-14  3.189206e-02
host_total_listings_count  1.298621e-01 -2.061273e-14  3.189206e-02

                           host_listings_count  host_total_listings_count  \
id                                1.298621e-01               1.298621e-01
scrape_id                        -2.061273e-14              -2.061273e-14
host_id                           3.189206e-02               3.189206e-02
host_listings_count               1.000000e+00               1.000000e+00
host_total_listings_count         1.000000e+00               1.000000e+00

                               latitude     longitude  accommodates  \
id                         1.000083e-02  8.708041e-02  3.540148e-02
scrape_id                  7.448373e-13 -1.544987e-11  1.251408e-14
host_id                    4.148254e-02  1.162017e-01  2.722884e-02
host_listings_count        3.475008e-02 -8.842627e-02 -2.620826e-02
host_total_listings_count  3.475008e-02 -8.842627e-02 -2.620826e-02

                           bathrooms      bedrooms  ...  \
id                               NaN  4.502641e-02  ...
scrape_id                        NaN  1.094740e-13  ...
host_id                          NaN  2.201904e-02  ...
host_listings_count              NaN -1.709828e-02  ...
host_total_listings_count        NaN -1.709828e-02  ...
```

```
                          calculated_host_listings_count_entire_homes  \
id                                                       1.371325e-01
scrape_id                                                2.212092e-14
host_id                                                  2.524284e-02
host_listings_count                                      5.418772e-01
host_total_listings_count                                5.418772e-01

                          calculated_host_listings_count_private_rooms  \
id                                                         2.118813e-01
scrape_id                                                 -4.315383e-14
host_id                                                    1.931984e-01
host_listings_count                                        1.491464e-01
host_total_listings_count                                  1.491464e-01

                          calculated_host_listings_count_shared_rooms  \
id                                                         4.671123e-02
scrape_id                                                 -9.152507e-15
host_id                                                    7.830736e-02
host_listings_count                                       -1.594956e-02
host_total_listings_count                                 -1.594956e-02

                          reviews_per_month   label_price  \
id                             2.316854e-01  7.906593e-02
scrape_id                      5.899236e-15 -3.138348e-14
host_id                        2.084392e-01  4.053291e-02
host_listings_count           -2.095984e-02  1.310429e-01
host_total_listings_count     -2.095984e-02  1.310429e-01

                          host_listings_count_na  \
id                                -8.301146e-03
scrape_id                         -3.883529e-15
host_id                           -3.707217e-03
host_listings_count               -1.234737e-15
host_total_listings_count         -1.234737e-15

                          host_total_listings_count_na  bathrooms_na  \
id                                      -8.301146e-03           NaN
scrape_id                               -3.883529e-15           NaN
host_id                                 -3.707217e-03           NaN
host_listings_count                     -1.234737e-15           NaN
host_total_listings_count               -1.234737e-15           NaN

                           bedrooms_na       beds_na
id                        3.342643e-02  1.363999e-01
scrape_id                 1.922869e-15 -4.177378e-14
host_id                   3.354441e-02  9.217727e-02
host_listings_count       1.296648e-02 -1.032184e-02
```

```
host_total_listings_count  1.296648e-02 -1.032184e-02

[5 rows x 46 columns]
```

The result is a computed *correlation matrix*. The values on the diagonal are all equal to 1, and the matrix is symmetrical with respect to the diagonal (note that we are only printing the first five lines of it).

We only need to observe correlations of all features with *the label* (as opposed to every possible pairwise correlation).

Task: Save the `label_price` column of the correlation matrix to the variable `corrs`:

[31]:
```
# YOUR CODE HERE

### Solution:
corrs = df.corr()['label_price']
corrs
```

[31]:
```
id                                7.906593e-02
scrape_id                        -3.138348e-14
host_id                           4.053291e-02
host_listings_count               1.310429e-01
host_total_listings_count         1.310429e-01
latitude                          4.329905e-02
longitude                        -2.069501e-01
accommodates                      5.006227e-01
bathrooms                                  NaN
bedrooms                          4.199613e-01
beds                              3.736971e-01
price                             7.111249e-01
minimum_nights                   -7.589208e-02
maximum_nights                   -9.728756e-04
minimum_minimum_nights           -3.803776e-02
maximum_minimum_nights            6.553784e-02
minimum_maximum_nights            6.581829e-02
maximum_maximum_nights            1.116868e-01
minimum_nights_avg_ntm            6.387517e-02
maximum_nights_avg_ntm            8.209898e-02
calendar_updated                           NaN
availability_30                   1.456894e-01
availability_60                   1.470082e-01
availability_90                   1.439066e-01
availability_365                  1.235559e-01
number_of_reviews                -4.197310e-02
number_of_reviews_ltm             2.757416e-02
number_of_reviews_l30d            2.158982e-02
review_scores_rating              4.319689e-02
review_scores_accuracy            5.358322e-03
review_scores_cleanliness         8.254405e-02
review_scores_checkin            -3.665125e-03
```

```
review_scores_communication                      1.206558e-04
review_scores_location                           9.724051e-02
review_scores_value                             -4.816654e-03
calculated_host_listings_count                  -1.581634e-02
calculated_host_listings_count_entire_homes      9.508782e-02
calculated_host_listings_count_private_rooms    -9.977978e-02
calculated_host_listings_count_shared_rooms     -4.333734e-02
reviews_per_month                                3.113557e-02
label_price                                      1.000000e+00
host_listings_count_na                           4.449997e-02
host_total_listings_count_na                     4.449997e-02
bathrooms_na                                              NaN
bedrooms_na                                      2.380733e-02
beds_na                                         -3.461428e-02
Name: label_price, dtype: float64
```

Task: Sort the values of the series we just obtained in the descending order.

[32]:
```
# YOUR CODE HERE

## Solution:
corrs_sorted = corrs.sort_values(ascending=False)
corrs_sorted
```

[32]:
```
label_price                                      1.000000e+00
price                                            7.111249e-01
accommodates                                     5.006227e-01
bedrooms                                         4.199613e-01
beds                                             3.736971e-01
availability_60                                  1.470082e-01
availability_30                                  1.456894e-01
availability_90                                  1.439066e-01
host_listings_count                              1.310429e-01
host_total_listings_count                        1.310429e-01
availability_365                                 1.235559e-01
maximum_maximum_nights                           1.116868e-01
review_scores_location                           9.724051e-02
calculated_host_listings_count_entire_homes      9.508782e-02
review_scores_cleanliness                        8.254405e-02
maximum_nights_avg_ntm                           8.209898e-02
id                                               7.906593e-02
minimum_maximum_nights                           6.581829e-02
maximum_minimum_nights                           6.553784e-02
minimum_nights_avg_ntm                           6.387517e-02
host_total_listings_count_na                     4.449997e-02
host_listings_count_na                           4.449997e-02
latitude                                         4.329905e-02
review_scores_rating                             4.319689e-02
host_id                                          4.053291e-02
```

```
reviews_per_month                                3.113557e-02
number_of_reviews_ltm                            2.757416e-02
bedrooms_na                                      2.380733e-02
number_of_reviews_l30d                           2.158982e-02
review_scores_accuracy                           5.358322e-03
review_scores_communication                      1.206558e-04
scrape_id                                       -3.138348e-14
maximum_nights                                  -9.728756e-04
review_scores_checkin                           -3.665125e-03
review_scores_value                             -4.816654e-03
calculated_host_listings_count                  -1.581634e-02
beds_na                                         -3.461428e-02
minimum_minimum_nights                          -3.803776e-02
number_of_reviews                               -4.197310e-02
calculated_host_listings_count_shared_rooms     -4.333734e-02
minimum_nights                                  -7.589208e-02
calculated_host_listings_count_private_rooms    -9.977978e-02
longitude                                       -2.069501e-01
bathrooms                                                 NaN
calendar_updated                                          NaN
bathrooms_na                                              NaN
Name: label_price, dtype: float64
```

Task: In the code cell below, save the *column names* for the top-2 correlation values to the list `top_two_corr` (not counting the correlation of `label` column with itself, nor the `price` column -- which is the `label` column prior to outlier removal). Add the column names to the list in the order in which they appear in the output above. Tip: `corrs_sorted` is a Pandas `Series` object, in which column names are the *index*.

[33]:
```
# YOUR CODE HERE

### Solution:
ans = corrs_sorted[2:4].index
top_two_corr = list(ans)
top_two_corr
```

[33]:
```
['accommodates', 'bedrooms']
```

1.6 Part 6. Produce Bivariate Plots for the Label and Its Top Correlates

We will use the `pairplot()` function in `seaborn` to plot the relationships between the two features we identified and the label.

Task: Create a DataFrame `df_sub` that contains only the selected three columns: the label, and the two columns which correlate with it the most.

[34]:
```
# Do not remove or edit the line below:
top_two_corr.append('label_price')

# YOUR CODE HERE
```

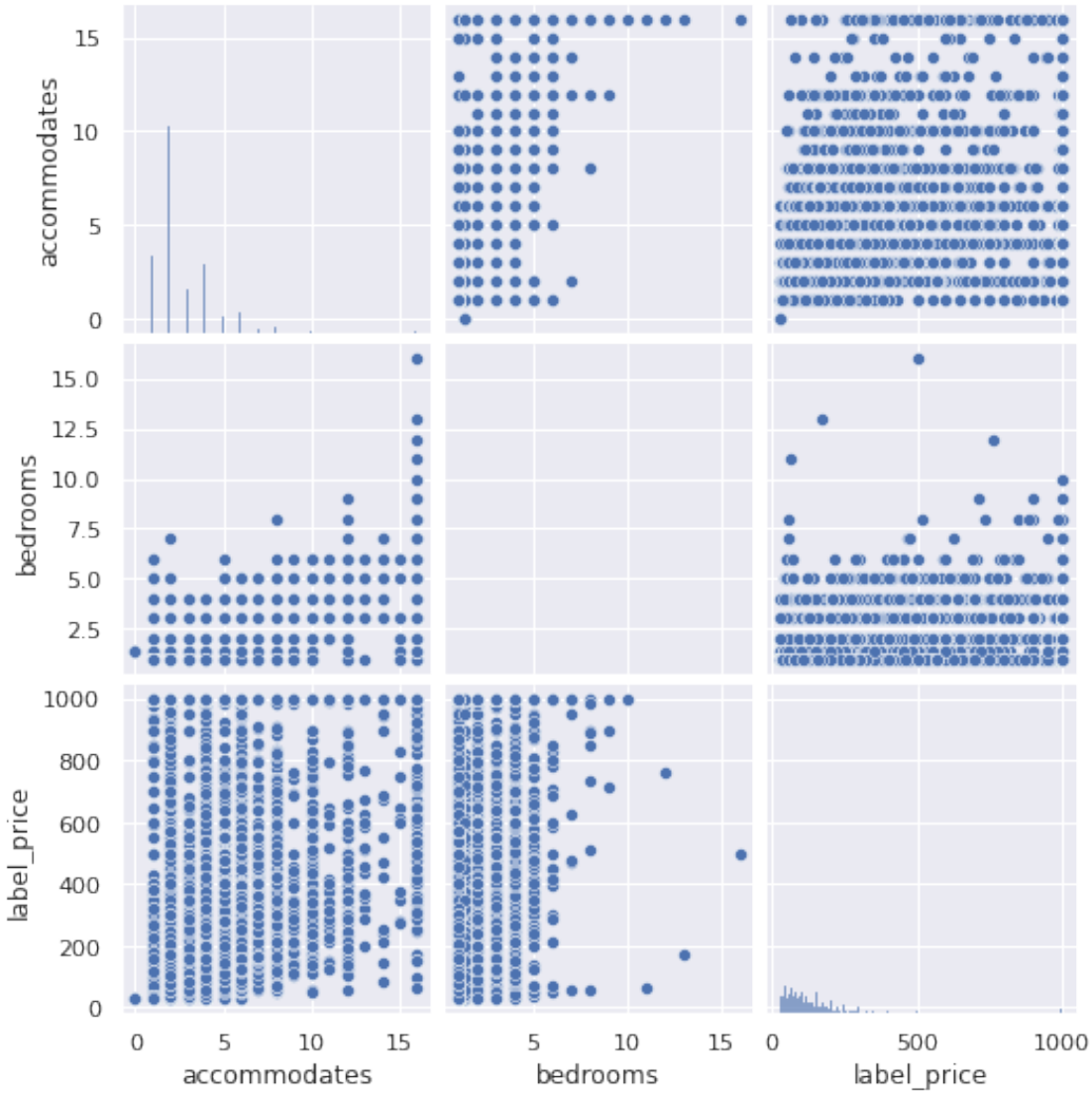
```
### Solution:
df_sub = df[top_two_corr]
```

Task: Create a `seaborn` pairplot of the data subset you just created

[35]:
```
# YOUR CODE HERE

### Solution:
sns.pairplot(data=df_sub)
```

[35]:
```
<seaborn.axisgrid.PairGrid at 0x7f0806cdf8d0>
```



This one is not very easy to make sense of: the points overlap, but we do not have visibility into how densely they are stacked together.

Task: Repeat the `pairplot` exercise, this time specifying the *kernel density estimator* as the *kind* of the plot. Tip: use `kind = 'kde'` as a parameter of the `pairplot()` function. You could also

specify `corner=True` to make sure you don't plot redundant (symmetrical) plots. Note: this one may take a while!

[ ]:
```
# YOUR CODE HERE

### Solution:
sns.pairplot(data=df_sub, kind = 'kde', corner=True)
```

Analysis: Think about the possible interpretations of these plots. (Recall that our label encodes the listing price). What kind of stories does this data seem to be telling? Is the relationship what you thought it would be? Is there anything surprising or, on the contrary, reassuring about the plots? For example, how would you explain the relationship between the label and 'accommodates'? Is there a slight tilt to the points cluster, as the price goes up? What other patterns do you observe?