DataUnderstandingAndPreparation

June 11, 2024

1 Lab 2: ML Life Cycle: Data Understanding and Data Preparation

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

In this lab, you will practice the second and third steps of the machine learning life cycle: data understanding and data preparation. You will beging preparing your data so that it can be used to train a machine learning model that solves a regression problem. Note that by the end of the lab, your data set won't be completely ready for the modeling phase, but you will gain experience using some common data preparation techniques.

You will complete the following tasks to transform your data:

1. Build your data matrix and define your ML problem:
    - Load the Airbnb "listings" data set into a DataFrame and inspect the data
    - Define the label and convert the label's data type to one that is more suitable for modeling
    - Identify features
2. Clean your data:
    - Handle outliers by building a new regression label column by winsorizing outliers
    - Handle missing data by replacing all missing values in the dataset with means
3. Perform feature transformation using one-hot encoding
4. Explore your data:
    - Identify two features with the highest correlation with label
    - Build appropriate bivariate plots to visualize the correlations between features and the label
5. Analysis:
    - Analyze the relationship between the features and the label
    - Brainstorm what else needs to be done to fully prepare the data for modeling

1.1 Part 1. Build Your Data Matrix (DataFrame) and Define Your ML Problem

Load a Data Set and Save it as a Pandas DataFrame We will 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", "airbnbData.csv")
```

Task: Load the data and save it to DataFrame `df`.
Note: You may receive a warning message. Ignore this warning.

[3]:
```
df = pd.read_csv(filename, header=0)
```

```
/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)
```

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

[4]:
```
df.shape
```

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

Task: Display the column names.

[5]:
```
df.columns
```

[5]:
```
Index(['id', 'listing_url', 'scrape_id', 'last_scraped', 'name', 'description',
       'neighborhood_overview', 'picture_url', 'host_id', 'host_url',
       'host_name', 'host_since', 'host_location', 'host_about',
       'host_response_time', 'host_response_rate', 'host_acceptance_rate',
       'host_is_superhost', 'host_thumbnail_url', 'host_picture_url',
       'host_neighbourhood', 'host_listings_count',
       'host_total_listings_count', 'host_verifications',
       'host_has_profile_pic', 'host_identity_verified', 'neighbourhood',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'latitude',
       'longitude', 'property_type', 'room_type', 'accommodates', 'bathrooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_review',
       'last_review', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
```

```
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month'],
      dtype='object')
```

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

[6]:
```
df.head()
```

[6]:
```
     id                            listing_url       scrape_id last_scraped  \
0  2595  https://www.airbnb.com/rooms/2595  20211204143024   2021-12-05
1  3831  https://www.airbnb.com/rooms/3831  20211204143024   2021-12-05
2  5121  https://www.airbnb.com/rooms/5121  20211204143024   2021-12-05
3  5136  https://www.airbnb.com/rooms/5136  20211204143024   2021-12-05
4  5178  https://www.airbnb.com/rooms/5178  20211204143024   2021-12-05

                                              name  \
0                            Skylit Midtown Castle
1  Whole flr w/private bdrm, bath & kitchen(pls r...
2                                  BlissArtsSpace!
3          Spacious Brooklyn Duplex, Patio + Garden
4                  Large Furnished Room Near B'way

                                         description  \
0  Beautiful, spacious skylit studio in the heart...
1  Enjoy 500 s.f. top floor in 1899 brownstone, w...
2  <b>The space</b><br />HELLO EVERYONE AND THANK...
3  We welcome you to stay in our lovely 2 br dupl...
4  Please dont expect the luxury here just a bas...

                             neighborhood_overview  \
0  Centrally located in the heart of Manhattan ju...
1  Just the right mix of urban center and local n...
2                                              NaN
3                                              NaN
4    Theater district, many restaurants around here.

                                         picture_url  host_id  \
0  https://a0.muscache.com/pictures/f0813a11-40b2...     2845
1  https://a0.muscache.com/pictures/e49999c2-9fd5...     4869
2  https://a0.muscache.com/pictures/2090980c-b68e...     7356
3  https://a0.muscache.com/pictures/miso/Hosting-...     7378
4  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]
```

Define the Label Assume that your goal is to train a machine learning model that predicts the price of an Airbnb. This is an example of supervised learning and is a regression problem. In our dataset, our label will be the `price` column. Let's inspect the values in the `price` column.

[7]:
```
df['price']
```

[7]:
```
0        $150.00
1         $75.00
2         $60.00
3        $275.00
4         $68.00
          ...
38272     $79.00
38273     $76.00
38274    $116.00
38275    $106.00
38276    $689.00
```

```
Name: price, Length: 38277, dtype: object
```

Notice the `price` column contains values that are listed as <currency_name><numeric_value>. For example, it contains values that look like this: `$120`.

Task: Obtain the data type of the values in this column:

[8]:
```
df['price'].dtype
```

[8]:
```
dtype('O')
```

Notice that the data type is "object," which in Pandas translates to the String data type.

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

[9]:
```
df['price'].unique()[:15]
```

[9]:
```
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 the values in the `price` column from strings to floats. We will:

- 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).
- remove the commas from all values that are in the thousands or above: for example, `$2,500`.

The code cell below accomplishes this.

[10]:
```
df['price'] = df['price'].str.replace(',', '').str.replace('$', '').
↪astype(float)
```

Task: Display the first 15 unique values of the `price` column again to make sure they have been transformed.

[11]:
```
df['price'].unique()[:15]
```

[11]:
```
array([150.,  75.,  60., 275.,  68.,  98.,  89.,  65.,  62.,  90., 199.,
        96., 299., 140., 175.])
```

Identify Features Simply by inspecting the data, let's identify some columns that should not serve as features - those that will not help us solve our predictive ML problem.

Some that stand out are columns that 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.

[12]:
```
# Identify columns that contain URLs
url_colnames = [col for col in df.columns if 'url' in col.lower()]
url_colnames
```

[12]:
```
['listing_url',
 'picture_url',
 'host_url',
```

```
 'host_thumbnail_url',
 'host_picture_url']
```

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).

[13]:
```
df.drop(columns = url_colnames, inplace = True)
```

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

[14]:
```
df.shape
```

[14]:
```
(38277, 69)
```

Task: In the code cell below, display the features that we will use to solve our ML problem.

[15]:
```
df.columns
```

[15]:
```
Index(['id', 'scrape_id', 'last_scraped', 'name', 'description',
       'neighborhood_overview', 'host_id', 'host_name', 'host_since',
       'host_location', 'host_about', 'host_response_time',
       'host_response_rate', 'host_acceptance_rate', 'host_is_superhost',
       'host_neighbourhood', 'host_listings_count',
       'host_total_listings_count', 'host_verifications',
       'host_has_profile_pic', 'host_identity_verified', 'neighbourhood',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'latitude',
       'longitude', 'property_type', 'room_type', 'accommodates', 'bathrooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_review',
       'last_review', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month'],
      dtype='object')
```

Task: Are there any other features that you think may not be well suited for our machine learning problem? Note your findings in the markdown cell below.

Yes, some features in the dataset might not be well-suited for the machine learning problem due to various reasons such as high cardinality, irrelevance, redundancy, or difficulty in quantifying their impact.

ID Features: id, scrape_id, host_id: These are unique identifiers and do not contribute mean-

ingful information for prediction.

Redundant or Less Informative: host_verifications: This field might be a list of verifications that need extensive preprocessing to be useful. license: If many listings do not have a license or if it's not a strong predictor, it might not be useful.

Free-Text: name, description, neighborhood_overview, host_name, host_location, host_about: These fields are free-text with high variability and might be difficult to use directly. amenities: This feature is a list and would require extensive preprocessing to be useful.

1.2 Part 2. Clean Your Data

Let's now handle outliers and missing data.

1.2.1 a. Handle Outliers

Let us prepare the data in our label column. Namely, we will detect and replace outliers in the data using winsorization.

Task: Create a new version of the `price` column, named `label_price`, in which you will 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:

[16]:
```
import scipy.stats as stats

# Apply winsorization to the price column
df['label_price'] = stats.mstats.winsorize(df['price'], limits=[0.01, 0.01])
```

Let's verify that the new column `label_price` was added to DataFrame `df`:

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

[17]:
```
     id       scrape_id last_scraped  \
0  2595  20211204143024   2021-12-05
1  3831  20211204143024   2021-12-05
2  5121  20211204143024   2021-12-05
3  5136  20211204143024   2021-12-05
4  5178  20211204143024   2021-12-05

                                                name  \
0                                Skylit Midtown Castle
1  Whole flr w/private bdrm, bath & kitchen(pls r...
2                                     BlissArtsSpace!
3           Spacious Brooklyn Duplex, Patio + Garden
4                   Large Furnished Room Near B'way

                                         description  \
0  Beautiful, spacious skylit studio in the heart...
1  Enjoy 500 s.f. top floor in 1899 brownstone, w...
2  <b>The space</b><br />HELLO EVERYONE AND THANK...
3  We welcome you to stay in our lovely 2 br dupl...
```

```
4  Please dont expect the luxury here just a bas...

                              neighborhood_overview  host_id    host_name  \
0  Centrally located in the heart of Manhattan ju...     2845     Jennifer
1  Just the right mix of urban center and local n...     4869  LisaRoxanne
2                                                NaN     7356        Garon
3                                                NaN     7378      Rebecca
4    Theater district, many restaurants around here.     8967     Shunichi

   host_since                      host_location  ... review_scores_location  \
0  2008-09-09  New York, New York, United States  ...                    4.86
1  2008-12-07  New York, New York, United States  ...                    4.71
2  2009-02-03  New York, New York, United States  ...                    4.47
3  2009-02-03  Brooklyn, New York, United States  ...                    4.50
4  2009-03-03  New York, New York, United States  ...                    4.87

  review_scores_value license instant_bookable calculated_host_listings_count  \
0                4.41     NaN                f                               3
1                4.64     NaN                f                               1
2                4.52     NaN                f                               2
3                5.00     NaN                f                               1
4                4.36     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 70 columns]
```

Task: Check that the values of `price` and `label_price` are *not* identical.
You will do this by subtracting the two columns and finding the resulting *unique values* of the

resulting difference. Note: If all values are identical, the difference would not contain unique values. If this is the case, outlier removal did not work.

[18]:
```
# Calculate the difference between price and label_price
price_difference = df['price'] - df['label_price']

# Find the unique values in the difference
unique_differences = price_difference.unique()
```

1.2.2 b. Handle Missing Data

Next we are going to find missing values in our entire dataset and impute the missing values by replace them with means.

Identifying missingness **Task**: Check if a given value in the data is missing, and sum up the resulting values by columns. Save this sum to variable `nan_count`. Print the results.

[19]:
```
nan_count = df.isnull().sum()
nan_count
```

[19]:
```
id                                                0
scrape_id                                         0
last_scraped                                      0
name                                             13
description                                    1192
                                               ...
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: 70, 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* missing values.

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

[20]:
```
nan_detected = df.isnull().values.any()
nan_detected
```

[20]:
```
True
```

Since replacing the missing values with the mean only makes sense for the columns that contain numerical values (and not for strings), 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 results to the variable `is_int_or_float`.

[21]:
```
is_int_or_float = df.dtypes.isin([np.int64, np.float64])
is_int_or_float
```

[21]:
```
id                                                False
scrape_id                                         False
last_scraped                                      False
name                                              False
description                                       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                                 False
label_price                                       False
Length: 70, dtype: bool
```

Task: Combine the two binary series (`nan_detected` and `is_int_or_float`) 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]:
```
to_impute = nan_detected & is_int_or_float
to_impute
```

[22]:
```
id                                                False
scrape_id                                         False
last_scraped                                      False
name                                              False
description                                       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                                 False
label_price                                       False
Length: 70, dtype: bool
```

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

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

[23]:
```
Index([], 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 you should impute the values for these specific columns: `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']
```

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

Task: For every column listed in `to_impute_selected`, create a new corresponding column called `<original-column-name>_na`. These columns should contain the a `True`or `False` value in

place of NaN.

[25]:
```
for column in to_impute_selected:
    df[column + '_na'] = df[column].isnull()
```

Check that the DataFrame contains the new variables:

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

[26]:
```
     id       scrape_id last_scraped  \
0  2595  20211204143024   2021-12-05
1  3831  20211204143024   2021-12-05
2  5121  20211204143024   2021-12-05
3  5136  20211204143024   2021-12-05
4  5178  20211204143024   2021-12-05

                                                name  \
0                                Skylit Midtown Castle
1  Whole flr w/private bdrm, bath & kitchen(pls r...
2                                     BlissArtsSpace!
3           Spacious Brooklyn Duplex, Patio + Garden
4                   Large Furnished Room Near B'way

                                           description  \
0  Beautiful, spacious skylit studio in the heart...
1  Enjoy 500 s.f. top floor in 1899 brownstone, w...
2  <b>The space</b><br />HELLO EVERYONE AND THANK...
3  We welcome you to stay in our lovely 2 br dupl...
4  Please dont expect the luxury here just a bas...

                               neighborhood_overview  host_id    host_name  \
0  Centrally located in the heart of Manhattan ju...     2845     Jennifer
1  Just the right mix of urban center and local n...     4869  LisaRoxanne
2                                                NaN     7356        Garon
3                                                NaN     7378      Rebecca
4    Theater district, many restaurants around here.     8967     Shunichi

   host_since                      host_location  ...  \
0  2008-09-09  New York, New York, United States  ...
1  2008-12-07  New York, New York, United States  ...
2  2009-02-03  New York, New York, United States  ...
3  2009-02-03  Brooklyn, New York, United States  ...
4  2009-03-03  New York, New York, United States  ...

  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

  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 75 columns]
```

Replacing the missing values with mean values of the column **Task**: For every column listed in `to_impute_selected`, fill the missing values with the corresponding mean of all values in the column (do not create new columns).

[27]:
```
for column in to_impute_selected:
    mean_value = df[column].mean()
    df[column].fillna(mean_value, 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]:
```
df['bathrooms'].unique()
```

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

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

1.3 Part 3. Perform One-Hot Encoding

Machine learning algorithms operate on numerical inputs. Therefore, we have to transform text data into some form of numerical representation to prepare our data for the model training phase. Some features that contain text data are categorical. Others are not. For example, we removed all of the features that contained URLs. These features were not categorical, but rather contained what is called unstructured text. However, not all features that contain unstructured text should be removed, as they can contain useful information for our machine learning problem. Unstructured text data is usually handled by Natural Language Processing (NLP) techniques. You will learn more about NLP later in this course.

However, for features that contain categorical values, one-hot encoding is a common feature engineering technique that transforms them into binary representations.

We will first choose one feature column to one-hot encode: `host_response_time`. Let's inspect the unique values this feature can have.

[30]:
```
df['host_response_time'].unique()
```

[30]:
```
array(['within a day', 'a few days or more', 'within an hour', nan,
       'within a few hours'], dtype=object)
```

Note that each entry can contain one of five possible values.

Task: Since one of these values is `NaN`, replace every entry in the column `host_response_time` that contains a `NaN` value with the string 'unavailable'.

[31]:
```
df['host_response_time'].fillna('unavailable', inplace=True)
```

Let's inspect the `host_response_time` column to see the new values.

[32]:
```
df['host_response_time'].unique()
```

[32]:
```
array(['within a day', 'a few days or more', 'within an hour',
       'unavailable', 'within a few hours'], dtype=object)
```

Task: Use `pd.get_dummies()` to one-hot encode the `host_response_time` column. Save the result to DataFrame `df_host_response_time`.

[33]:
```
df_host_response_time = pd.get_dummies(df['host_response_time'],␣
↪prefix='host_response_time')
df_host_response_time
```

[33]:

	host_response_time_a few days or more	host_response_time_unavailable	\
0	0	0	
1	1	0	
2	0	0	

```
3                                          0                                          0
4                                          0                                          0
...                                      ...                                        ...
38272                                      0                                          0
38273                                      0                                          0
38274                                      0                                          0
38275                                      0                                          0
38276                                      0                                          0

       host_response_time_within a day  host_response_time_within a few hours  \
0                                    1                                          0
1                                    0                                          0
2                                    0                                          0
3                                    1                                          0
4                                    1                                          0
...                                ...                                        ...
38272                                0                                          1
38273                                0                                          1
38274                                0                                          0
38275                                0                                          0
38276                                0                                          0

       host_response_time_within an hour
0                                      0
1                                      0
2                                      1
3                                      0
4                                      0
...                                  ...
38272                                  0
38273                                  0
38274                                  1
38275                                  1
38276                                  1

[38277 rows x 5 columns]
```

Task: Since the `pd.get_dummies()` function returned a new DataFrame rather than making the changes to the original DataFrame `df`, add the new DataFrame `df_host_response_time` to DataFrame `df`, and delete the original `host_response_time` column from DataFrame `df`.

[34]:
```
# Concatenate the new DataFrame with the original DataFrame
df = pd.concat([df, df_host_response_time], axis=1)

# Drop the original host_response_time column
df.drop(columns=['host_response_time'], inplace=True)
```

Let's inspect DataFrame `df` to see the changes that have been made.

[35]:
```
df.columns
```

[35]:
```
Index(['id', 'scrape_id', 'last_scraped', 'name', 'description',
       'neighborhood_overview', 'host_id', 'host_name', 'host_since',
       'host_location', 'host_about', 'host_response_rate',
       'host_acceptance_rate', 'host_is_superhost', 'host_neighbourhood',
       'host_listings_count', 'host_total_listings_count',
       'host_verifications', 'host_has_profile_pic', 'host_identity_verified',
       'neighbourhood', 'neighbourhood_cleansed',
       'neighbourhood_group_cleansed', 'latitude', 'longitude',
       'property_type', 'room_type', 'accommodates', 'bathrooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_review',
       'last_review', 'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month',
       'label_price', 'host_listings_count_na', 'host_total_listings_count_na',
       'bathrooms_na', 'bedrooms_na', 'beds_na',
       'host_response_time_a few days or more',
       'host_response_time_unavailable', 'host_response_time_within a day',
       'host_response_time_within a few hours',
       'host_response_time_within an hour'],
      dtype='object')
```

One-hot encode additional features **Task**: Use the code cell below to find columns that contain string values (the 'object' data type) and inspect the *number* of unique values each column has.

[36]:
```
# Find columns with 'object' dtype (string values)
object_columns = df.select_dtypes(include=['object']).columns

# Inspect the number of unique values in each object column
unique_value_counts = df[object_columns].nunique()
```

Task: Based on your findings, identify features that you think should be transformed using one-hot encoding.

1. Use the code cell below to inspect the unique *values* that each of these features have.

[37]:
```
features_to_inspect = ['property_type', 'room_type', 'neighbourhood_cleansed']

for feature in features_to_inspect:
    unique_values = df[feature].unique()
    print(f"Unique values for {feature}: {unique_values}")
```

```
Unique values for property_type: ['Entire rental unit' 'Entire guest suite'
'Private room in rental unit'
 'Private room in townhouse' 'Private room in condominium (condo)'
 'Private room in loft' 'Entire loft' 'Private room in residential home'
 'Entire condominium (condo)' 'Entire residential home' 'Entire townhouse'
 'Private room in bed and breakfast' 'Entire guesthouse'
 'Private room in guest suite' 'Room in boutique hotel'
 'Shared room in loft' 'Shared room in rental unit'
 'Shared room in residential home' 'Private room' 'Private room in hostel'
 'Entire place' 'Private room in guesthouse' 'Boat'
 'Entire serviced apartment' 'Room in aparthotel' 'Floor'
 'Private room in vacation home' 'Room in serviced apartment'
 'Entire cottage' 'Private room in serviced apartment' 'Room in hotel'
 'Cave' 'Tiny house' 'Private room in floor'
 'Shared room in condominium (condo)' 'Entire bungalow'
 'Private room in casa particular' 'Shared room in townhouse' 'Houseboat'
 'Private room in bungalow' 'Entire villa' 'Private room in resort'
 'Shared room in guest suite' 'Private room in castle'
 'Private room in villa' 'Shared room in floor' 'Entire bed and breakfast'
 'Entire home/apt' 'Private room in tiny house' 'Private room in tent'
 'Private room in in-law' 'Private room in barn' 'Shared room in hostel'
 'Camper/RV' 'Room in resort' 'Shared room in guesthouse' 'Bus'
 'Shared room in bed and breakfast' 'Private room in farm stay'
 'Private room in dorm' 'Room in bed and breakfast'
 'Shared room in island' 'Shared room in bungalow'
 'Shared room in serviced apartment' 'Private room in earth house'
 'Lighthouse' 'Private room in train' 'Barn' 'Private room in lighthouse'
 'Entire cabin' 'Private room in camper/rv' 'Castle' 'Tent' 'Tower'
 'Casa particular' 'Shared room in casa particular'
 'Private room in cycladic house' 'Entire vacation home']
Unique values for room_type: ['Entire home/apt' 'Private room' 'Hotel room'
'Shared room']
Unique values for neighbourhood_cleansed: ['Midtown' 'Bedford-Stuyvesant'
'Sunset Park' 'Upper West Side'
 'South Slope' 'Williamsburg' 'East Harlem' 'Fort Greene' "Hell's Kitchen"
 'East Village' 'Harlem' 'Flatbush' 'Long Island City' 'Jamaica'
 'Greenpoint' 'Nolita' 'Chelsea' 'Upper East Side' 'Prospect Heights'
 'Clinton Hill' 'Washington Heights' 'Kips Bay' 'Bushwick'
 'Carroll Gardens' 'West Village' 'Park Slope' 'Prospect-Lefferts Gardens'
 'Lower East Side' 'East Flatbush' 'Boerum Hill' 'Sunnyside' 'St. George'
```

```
'Tribeca' 'Highbridge' 'Ridgewood' 'Mott Haven' 'Morningside Heights'
'Gowanus' 'Ditmars Steinway' 'Middle Village' 'Brooklyn Heights'
'Flatiron District' 'Windsor Terrace' 'Chinatown' 'Greenwich Village'
'Clason Point' 'Crown Heights' 'Astoria' 'Kingsbridge' 'Forest Hills'
'Murray Hill' 'University Heights' 'Gravesend' 'Allerton' 'East New York'
'Stuyvesant Town' 'Sheepshead Bay' 'Emerson Hill' 'Bensonhurst'
'Shore Acres' 'Richmond Hill' 'Gramercy' 'Arrochar' 'Financial District'
'Theater District' 'Rego Park' 'Kensington' 'Woodside' 'Cypress Hills'
'SoHo' 'Little Italy' 'Elmhurst' 'Clifton' 'Bayside' 'Bay Ridge'
'Maspeth' 'Spuyten Duyvil' 'Stapleton' 'Briarwood' 'Battery Park City'
'Brighton Beach' 'Jackson Heights' 'Longwood' 'Inwood' 'Two Bridges'
'Fort Hamilton' 'Cobble Hill' 'New Springville' 'Flushing' 'Red Hook'
'Civic Center' 'Tompkinsville' 'Tottenville' 'NoHo' 'DUMBO' 'Columbia St'
'Glendale' 'Mariners Harbor' 'East Elmhurst' 'Concord'
'Downtown Brooklyn' 'Melrose' 'Kew Gardens' 'College Point' 'Mount Eden'
'Vinegar Hill' 'City Island' 'Canarsie' 'Port Morris' 'Flatlands'
'Arverne' 'Queens Village' 'Midwood' 'Brownsville' 'Williamsbridge'
'Soundview' 'Woodhaven' 'Parkchester' 'Bronxdale' 'Bay Terrace'
'Ozone Park' 'Norwood' 'Rockaway Beach' 'Hollis' 'Claremont Village'
'Fordham' 'Concourse Village' 'Borough Park' 'Fieldston'
'Springfield Gardens' 'Huguenot' 'Mount Hope' 'Wakefield' 'Navy Yard'
'Roosevelt Island' 'Lighthouse Hill' 'Unionport' 'Randall Manor'
'South Ozone Park' 'Kew Gardens Hills' 'Jamaica Estates' 'Concourse'
'Bellerose' 'Fresh Meadows' 'Eastchester' 'Morris Park' 'Far Rockaway'
'East Morrisania' 'Corona' 'Tremont' 'St. Albans' 'West Brighton'
'Manhattan Beach' 'Marble Hill' 'Dongan Hills' 'Morris Heights' 'Belmont'
'Castleton Corners' 'Laurelton' 'Hunts Point' 'Howard Beach'
'Great Kills' 'Pelham Bay' 'Silver Lake' 'Riverdale' 'Morrisania'
'Grymes Hill' 'Holliswood' 'Edgemere' 'New Brighton' 'Pelham Gardens'
'Baychester' 'Sea Gate' 'Belle Harbor' 'Bergen Beach' 'Cambria Heights'
'Richmondtown' 'Olinville' 'Dyker Heights' 'Throgs Neck' 'Coney Island'
'Rosedale' 'Howland Hook' "Prince's Bay" 'South Beach' 'Bath Beach'
'Midland Beach' 'Eltingville' 'Oakwood' 'Schuylerville' 'Edenwald'
'North Riverdale' 'Port Richmond' 'Fort Wadsworth' 'Westchester Square'
'Van Nest' 'Arden Heights' "Bull's Head" 'Woodlawn' 'New Dorp' 'Neponsit'
'Grant City' 'Bayswater' 'Douglaston' 'New Dorp Beach' 'Todt Hill'
'Mill Basin' 'West Farms' 'Little Neck' 'Whitestone' 'Rosebank'
'Co-op City' 'Jamaica Hills' 'Rossville' 'Castle Hill' 'Westerleigh'
'Country Club' 'Chelsea, Staten Island' 'Gerritsen Beach' 'Breezy Point'
'Woodrow' 'Graniteville']
```

2. List these features and explain why they would be suitable for one-hot encoding. Note your findings in the markdown cell below.

Because they represent categorical variables with a limited number of unique values. One-hot encoding is a technique used to convert categorical variables into a numerical representation that can be used by machine learning algorithms. In one-hot encoding, each unique value in a categorical variable is converted into a new binary column (0 or 1), indicating the presence

or absence of that value for each observation. This transformation allows the machine learning algorithm to understand and utilize categorical data effectively, as it requires numerical inputs.

Task: In the code cell below, one-hot encode one of the features you have identified and replace the original column in DataFrame `df` with the new one-hot encoded columns.

[38]:
```
# One-hot encode the 'property_type' column
df_property_type = pd.get_dummies(df['property_type'], prefix='property_type')

# Drop the original 'property_type' column
df.drop(columns=['property_type'], inplace=True)

# Concatenate the new one-hot encoded columns with the DataFrame
df = pd.concat([df, df_property_type], axis=1)
```

1.4 Part 4. Explore Your Data

You will now perform exploratory data analysis in preparation for selecting your features as part of feature engineering.

Identify Correlations We will focus on identifying which features in the data have the highest correlation with the label.

Let's first run the `corr()` method on DataFrame `df` and save the result to the variable `corr_matrix`. Let's round the resulting correlations to five decimal places:

[39]:
```
corr_matrix = round(df.corr(),5)
corr_matrix
```

[39]:
```
                                                 id  scrape_id  host_id  \
id                                          1.00000       -0.0  0.58617
scrape_id                                  -0.00000        1.0  0.00000
host_id                                     0.58617        0.0  1.00000
host_listings_count                         0.12986       -0.0  0.03189
host_total_listings_count                   0.12986       -0.0  0.03189
...                                             ...        ...      ...
property_type_Shared room in serviced apartment  0.00674        0.0  0.01170
property_type_Shared room in townhouse      0.00508        0.0  0.00860
property_type_Tent                          0.00604        0.0  0.00740
property_type_Tiny house                   -0.00184        0.0  0.00225
property_type_Tower                         0.00634        0.0 -0.00297

                                                 host_listings_count  \
id                                                           0.12986
scrape_id                                                   -0.00000
host_id                                                      0.03189
host_listings_count                                          1.00000
host_total_listings_count                                    1.00000
...                                                              ...
property_type_Shared room in serviced apartment             -0.00135
property_type_Shared room in townhouse                      -0.00230
```

```
property_type_Tent                                          -0.00077
property_type_Tiny house                                    -0.00253
property_type_Tower                                         -0.00065

                                                    host_total_listings_count  \
id                                                                    0.12986
scrape_id                                                            -0.00000
host_id                                                               0.03189
host_listings_count                                                   1.00000
host_total_listings_count                                             1.00000
...                                                                       ...
property_type_Shared room in serviced apartment                      -0.00135
property_type_Shared room in townhouse                               -0.00230
property_type_Tent                                                   -0.00077
property_type_Tiny house                                             -0.00253
property_type_Tower                                                  -0.00065

                                                    latitude  longitude  \
id                                                   0.01000    0.08708
scrape_id                                            0.00000   -0.00000
host_id                                              0.04148    0.11620
host_listings_count                                  0.03475   -0.08843
host_total_listings_count                            0.03475   -0.08843
...                                                      ...        ...
property_type_Shared room in serviced apartment      0.00438    0.00738
property_type_Shared room in townhouse              -0.01867    0.00084
property_type_Tent                                   0.00370   -0.00420
property_type_Tiny house                            -0.00668    0.01648
property_type_Tower                                 -0.00354    0.00342

                                                    accommodates  bathrooms  \
id                                                       0.03540        NaN
scrape_id                                                0.00000        NaN
host_id                                                  0.02723        NaN
host_listings_count                                     -0.02621        NaN
host_total_listings_count                               -0.02621        NaN
...                                                          ...        ...
property_type_Shared room in serviced apartment         -0.00375        NaN
property_type_Shared room in townhouse                  -0.01117        NaN
property_type_Tent                                       0.00877        NaN
property_type_Tiny house                                 0.01178        NaN
property_type_Tower                                      0.03611        NaN

                                                    bedrooms  ...  \
id                                                   0.04503  ...
scrape_id                                            0.00000  ...
host_id                                              0.02202  ...
```

```
host_listings_count                                -0.01710  ...
host_total_listings_count                          -0.01710  ...
...                                                     ...  ...
property_type_Shared room in serviced apartment    -0.00427  ...
property_type_Shared room in townhouse             -0.00780  ...
property_type_Tent                                  0.00516  ...
property_type_Tiny house                            0.00859  ...
property_type_Tower                                 0.03565  ...

                                                   property_type_Shared room in
hostel  \
id
0.01220
scrape_id
0.00000
host_id
0.01000
host_listings_count
-0.00254
host_total_listings_count
-0.00254
...
...
property_type_Shared room in serviced apartment
-0.00016
property_type_Shared room in townhouse
-0.00029
property_type_Tent
-0.00009
property_type_Tiny house
-0.00030
property_type_Tower
-0.00009

                                                   property_type_Shared room in
island  \
id
0.00179
scrape_id
0.00000
host_id
-0.00171
host_listings_count
-0.00057
host_total_listings_count
-0.00057
...
```

```
...
property_type_Shared room in serviced apartment
-0.00005
property_type_Shared room in townhouse
-0.00008
property_type_Tent
-0.00003
property_type_Tiny house
-0.00009
property_type_Tower
-0.00003

                                                    property_type_Shared room in
loft  \
id
-0.00248
scrape_id
0.00000
host_id
-0.01050
host_listings_count
-0.00399
host_total_listings_count
-0.00399
...
...
property_type_Shared room in serviced apartment
-0.00024
property_type_Shared room in townhouse
-0.00044
property_type_Tent
-0.00014
property_type_Tiny house
-0.00046
property_type_Tower
-0.00014

                                                    property_type_Shared room in
rental unit  \
id
0.00054
scrape_id
-0.00000
host_id
0.02761
host_listings_count
-0.01586
```

```
host_total_listings_count
-0.01586
...
...
property_type_Shared room in serviced apartment
-0.00094
property_type_Shared room in townhouse
-0.00173
property_type_Tent
-0.00055
property_type_Tiny house
-0.00181
property_type_Tower
-0.00055

                                                    property_type_Shared room in
residential home  \
id
0.01833
scrape_id
0.00000
host_id
0.03402
host_listings_count
-0.00564
host_total_listings_count
-0.00564
...
...
property_type_Shared room in serviced apartment
-0.00034
property_type_Shared room in townhouse
-0.00062
property_type_Tent
-0.00020
property_type_Tiny house
-0.00065
property_type_Tower
-0.00020

                                                    property_type_Shared room in
serviced apartment  \
id
0.00674
scrape_id
0.00000
host_id
```

```
0.01170
host_listings_count
-0.00135
host_total_listings_count
-0.00135
...
...
property_type_Shared room in serviced apartment
1.00000
property_type_Shared room in townhouse
-0.00014
property_type_Tent
-0.00005
property_type_Tiny house
-0.00015
property_type_Tower
-0.00005

                                                    property_type_Shared room in
townhouse  \
id
0.00508
scrape_id
0.00000
host_id
0.00860
host_listings_count
-0.00230
host_total_listings_count
-0.00230
...
...
property_type_Shared room in serviced apartment
-0.00014
property_type_Shared room in townhouse
1.00000
property_type_Tent
-0.00008
property_type_Tiny house
-0.00027
property_type_Tower
-0.00008

                                                    property_type_Tent  \
id                                                             0.00604
scrape_id                                                      0.00000
host_id                                                        0.00740
```

```
host_listings_count                                          -0.00077
host_total_listings_count                                    -0.00077
...                                                               ...
property_type_Shared room in serviced apartment              -0.00005
property_type_Shared room in townhouse                       -0.00008
property_type_Tent                                            1.00000
property_type_Tiny house                                     -0.00009
property_type_Tower                                          -0.00003

                                                     property_type_Tiny house  \
id                                                                  -0.00184
scrape_id                                                            0.00000
host_id                                                              0.00225
host_listings_count                                                 -0.00253
host_total_listings_count                                           -0.00253
...                                                                      ...
property_type_Shared room in serviced apartment                     -0.00015
property_type_Shared room in townhouse                              -0.00027
property_type_Tent                                                  -0.00009
property_type_Tiny house                                             1.00000
property_type_Tower                                                 -0.00009

                                                     property_type_Tower
id                                                              0.00634
scrape_id                                                       0.00000
host_id                                                        -0.00297
host_listings_count                                            -0.00065
host_total_listings_count                                      -0.00065
...                                                                 ...
property_type_Shared room in serviced apartment                -0.00005
property_type_Shared room in townhouse                         -0.00008
property_type_Tent                                             -0.00003
property_type_Tiny house                                       -0.00009
property_type_Tower                                             1.00000

[129 rows x 129 columns]
```

The result is a computed *correlation matrix*. The values on the diagonal are all equal to 1 because they represent the correlations between each column with itself. The matrix is symmetrical with respect to the diagonal.

We only need to observe correlations of all features with the column `label_price` (as opposed to every possible pairwise correlation). Se let's query the `label_price` column of this matrix:

Task: Extract the `label_price` column of the correlation matrix and save the results to the variable `corrs`.

[40]:
```
corrs = corr_matrix['label_price']
corrs
```

[40]:
```
id                                                 0.07907
scrape_id                                         -0.00000
host_id                                            0.04053
host_listings_count                                0.13104
host_total_listings_count                          0.13104
                                                     ...
property_type_Shared room in serviced apartment    0.00033
property_type_Shared room in townhouse            -0.00956
property_type_Tent                                 0.01948
property_type_Tiny house                           0.00460
property_type_Tower                                0.02772
Name: label_price, Length: 129, dtype: float64
```

Task: Sort the values of the series we just obtained in the descending order and save the results to the variable `corrs_sorted`.

[41]:
```
corrs_sorted = corrs.sort_values(ascending=False)
corrs_sorted
```

[41]:
```
label_price                                  1.00000
price                                        0.71112
accommodates                                 0.50062
bedrooms                                     0.41996
beds                                         0.37370
                                                ...
longitude                                   -0.20695
property_type_Private room in rental unit   -0.31560
bathrooms                                       NaN
calendar_updated                                NaN
bathrooms_na                                    NaN
Name: label_price, Length: 129, dtype: float64
```

Task: Use Pandas indexing to extract the column names for the top two correlation values and save the results to the Python list `top_two_corr`. Add the feature names to the list in the order in which they appear in the output above.

Note: Do not count the correlation of `label` column with itself, nor the `price` column -- which is the `label` column prior to outlier removal.

[42]:
```
top_two_corr = corrs_sorted.drop(['label_price', 'price']).head(2).index.
↪tolist()
top_two_corr
```

[42]:
```
['accommodates', 'bedrooms']
```

Bivariate Plotting: Produce Plots for the Label and Its Top Correlates Let us visualize our data.

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

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

[43]:
```
df_corrs = df[['label_price'] + top_two_corr]
df_corrs
```

[43]:

	label_price	accommodates	bedrooms
0	150.0	1	1.323567
1	75.0	3	1.000000
2	60.0	2	1.000000
3	275.0	4	2.000000
4	68.0	2	1.000000
...	...	...	...
38272	79.0	2	1.000000
38273	76.0	2	1.000000
38274	116.0	2	1.000000
38275	106.0	2	1.000000
38276	689.0	14	6.000000

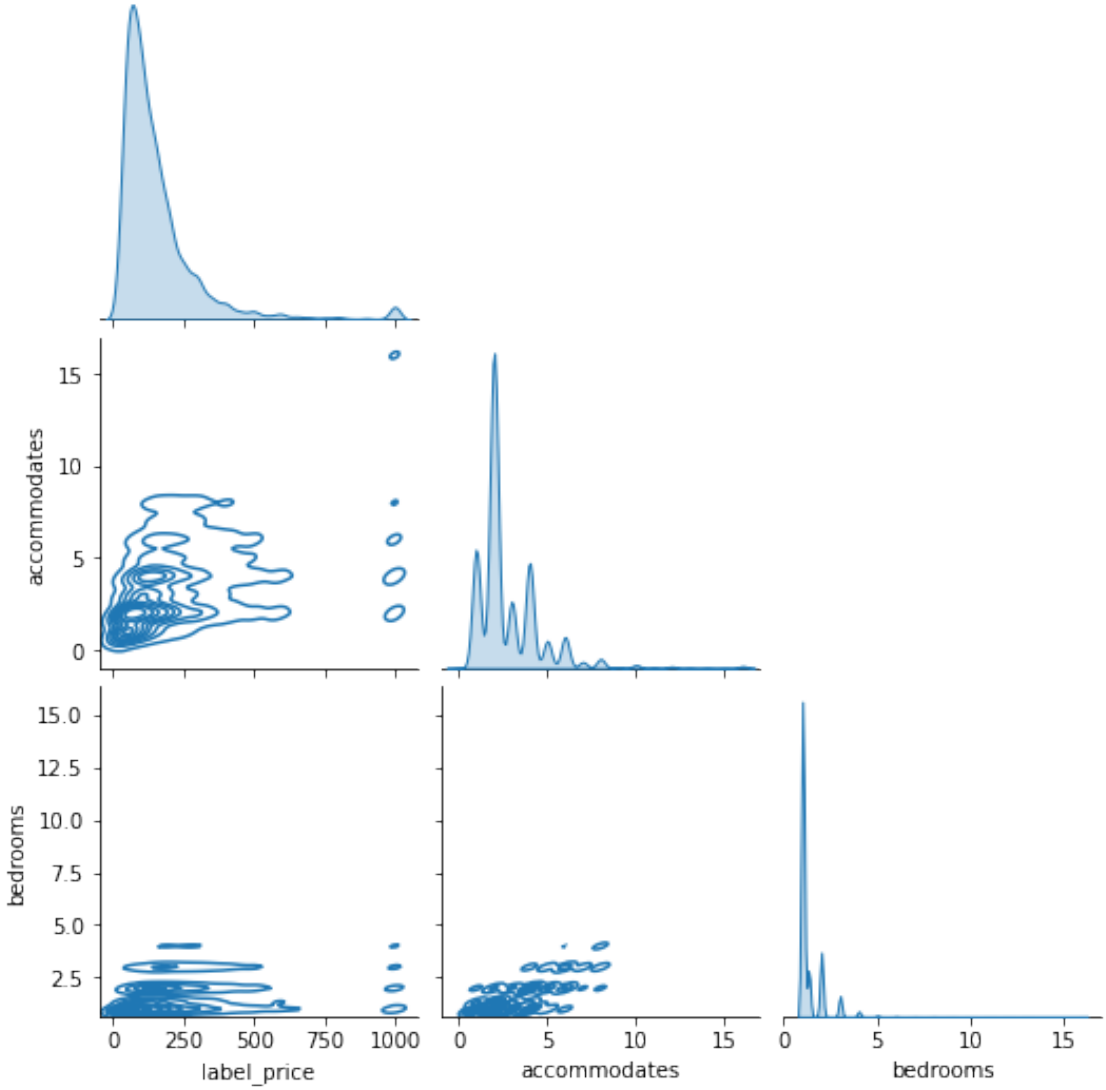
[38277 rows x 3 columns]

Task: Create a `seaborn` pairplot of the data subset you just created. Specify the *kernel density estimator* as the kind of the plot, and make sure that you don't plot redundant plots.

Note: It will take a few minutes to run and produce a plot.

[44]:
```
sns.pairplot(df_corrs, kind='kde', corner=True)
```

[44]:
```
<seaborn.axisgrid.PairGrid at 0x714a4ecb02e8>
```



1.5 Part 5: Analysis

1. Think about the possible interpretation of the plot. Recall that the label is the listing price. How would you explain the relationship between the label and the two features? Is there a slight tilt to the points cluster, as the price goes up?
2. Are the top two correlated features strongly or weakly correlated with the label? Are they features that should be used for our predictive machine learning problem?
3. Inspect your data matrix. It has a few features that contain unstructured text, meaning text data that is neither numerical nor categorical. List some features that contain unstructured text that you think are valuable for our predictive machine learning problem. Are there other remaining features that you think need to be prepared for the modeling phase? Do you have any suggestions on how to prepare these features?

Record your findings in the cell below.