btt004_week1_lab-solution

May 1, 2024

1 Lab 1: ML Life Cycle: Business Understanding and Problem Formulation

[1]:
```
import pandas as pd
import numpy as np
```

In this lab, you will practice the first step of the machine learning life cycle: formulating a machine learning problem. But first, you will get more practice working with some of the Python machine learning packages that you will use throughout the machine learning life cycle to develop your models.

## Part 1. Practice Working with ML Python Tools

In this part of the lab, you will:

1. Work with NumPy arrays and NumPy functions
2. Create Pandas DataFrames from data
3. Use NumPy and Pandas to analyze the data
4. Visualize the data with Matplotlib

Note: In Jupyter Notebooks, you can output a variable in two different ways:

1. By writing the name of the variable
2. By using the python `print()` function

The code cells below demonstrate this. Run each cell and inspect the results.

[2]:
```
x = 5
x
```

[2]:
```
5
```

[3]:
```
x = 5
print(x)
```

```
5
```

If you want to output multiple items, you must use a `print()` statement. See the code cell below as an example.

[4]:
```
y = 4
z = 3
```

```
print(y)
print(z)
```

```
4
3
```

1.1 Practice Operating on NumPy Arrays

1.1.1 a. Define a Python list

The code cell below defines a new list in Python.

[5]:
```
python_list = [0,2,4,6,8,10,12,14]
python_list
```

[5]:
```
[0, 2, 4, 6, 8, 10, 12, 14]
```

1.1.2 b. Define a Python range

The code cell below defines a Python range in that contains the same values as those in the list above.

[6]:
```
python_range = range(0, 15, 2)
python_range
```

[6]:
```
range(0, 15, 2)
```

The above returns an object of type `range`. The code cell below coverts this object to a Python list using the Python `list()` function.

[7]:
```
list(python_range)
```

[7]:
```
[0, 2, 4, 6, 8, 10, 12, 14]
```

1.1.3 c. Define a NumPy range

Task: In the code cell below, use NumPy's `np.arange()` method to create a NumPy range that has the same output as the Python range above. Save the output to the variable `numpy_range`.

[8]:
```
# YOUR CODE HERE

#solution
numpy_range = np.arange(0,15, 2)
numpy_range
```

[8]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14])
```

The code above returns an object of type `ndarray` (i.e. an array).

Task: In the code cell below, convert the NumPy array `numpy_range` to a Python list.

[9]:
```
# YOUR CODE HERE

#solution
```

```
list(numpy_range)
```

[9]:
```
[0, 2, 4, 6, 8, 10, 12, 14]
```

1.1.4 d. List comprehension

Consider the task of replacing each value in a list with its square. The traditional way of performing a transformation on every element of a list is via a `for` loop.

Task: In the code cell below, use a for loop to replace every value in the list `my_list` with its square (e.g. 2 will become 4). In your loop, use a range.

[10]:
```
my_list = [1,2,3,4]

# YOUR CODE HERE

# solution
for i in np.arange(0, len(my_list)):
    my_list[i] = my_list[i]**2 #square each element


print(my_list)
```

```
[1, 4, 9, 16]
```

There is a different, more 'Pythonic' way to accomplish the same task.

List comprehension is one of the most elegant functionalities native to Python. It offers a concise way of applying a particular transformation to every element in a list.

By using list comprehension syntax, we can write a single, easily interpretable line of code that does the same transformation without using ranges or an iterating index variable `i`:

[11]:
```
my_list = [1,2,3,4]

my_list = [x**2 for x in my_list]

print(my_list)
```

```
[1, 4, 9, 16]
```

1.1.5 e. Create a NumPy array

Task: In the code cell below, create a `numpy` array that contains the integer values 1 through 4. Save the result to variable `arr`.

[12]:
```
# YOUR CODE HERE

#solution
arr = np.array([1, 2, 3, 4])

print(arr)
```

```
[1 2 3 4]
```

1.1.6 e. Obtain the dimensions of the NumPy array

The NumPy function `np.shape()` returns the dimensions of a `numpy` array. Because `numpy` arrays can be two dimensional (i.e. a matrix), `np.shape()` returns a tuple that contains the lengths of the array's dimensions. You can consider the result to be the number of rows and the number of columns in the NumPy array.

Task: In the code cell below, use `np.shape()` to find the 'shape' of `numpy` array `arr`.

[13]:
```
# YOUR CODE HERE

#solution
np.shape(arr)
```

[13]:
```
(4,)
```

Notice that there appears to be an empty 'slot' for another number in the tuple. Since `arr` is a one-dimensional array, we only care about the first element in the tuple that `np.shape()` returns.

Task: In the code cell below, obtain the length of `arr` by extracting the first element that is returned by `np.shape()`. Save the result to the variable `arr_length`.

[14]:
```
# YOUR CODE HERE

#solution
arr_length = np.shape(arr)[0]

print(arr_length)
```

```
4
```

1.1.7 f. Create a uniform (same value in every position) array

We will now use `np.ones()` to create an array of a specified length that contains the value '1' in each position:

[15]:
```
np.ones(55, dtype=int)
```

[15]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

We can use this method to create an array of any identical value. Let's create an array of length 13, filled with the value '7' in every position:

[16]:
```
7 * np.ones(13, dtype=int)
```

[16]:
```
array([7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
```

1.1.8 g. Create a two-dimensional NumPy array

Let us explore the possibilities of the `np.array()` function further. NumPy arrays can be of more than one dimension. The code cell below creates a two-dimensional `numpy` array (a matrix).

[17]:
```
matrix = np.array([[1,2,3], [4,5,6]])
matrix
```

[17]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

Task: In the code cell below, use `np.shape()` to find the dimensions of `matrix`.

[18]:
```
# YOUR CODE HERE

#solution
np.shape(matrix)
```

[18]:
```
(2, 3)
```

1.1.9 h. Create an identity matrix

`np.eye()` is a NumPy function that creates an identity matrix of a specified size. An identity matrix is a matrix in which all of the values of the main diagonal are one, and all other values in the matrix are zero.

[19]:
```
np.eye(5)
```

[19]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Check your intuition: What do you think will be the output after running this cell? Run the cell to see if you are correct.

[20]:
```
A = np.eye(3)
B = 4 * np.eye(3)
A+B
```

[20]:
```
array([[5., 0., 0.],
       [0., 5., 0.],
       [0., 0., 5.]])
```

1.1.10 i. A small challenge: matrix transformation and random matrix generation

The `np.triu()` function obtains the upper right triangle of a two-dimensional NumPy array (matrix). Inspect the documentation by running the command `np.triu?` in the cell below.

[1]:
```
np.triu?
```

Task: Inspect the code in the cell below, then run the code and note the resulting matrix `M`.

[22]:
```
M = np.round(np.random.rand(5,5),2)
print("M=\n", M)
```

```
M=
 [[0.75 0.7  0.41 0.73 0.8 ]
 [0.55 0.42 0.15 0.06 0.85]
 [0.68 0.63 0.6  0.11 0.87]
 [0.28 0.14 0.03 0.31 0.14]
 [0.09 0.37 0.6  0.51 0.64]]
```

Task: Use `np.triu()` to create a matrix called `new_M` which is identical to the matrix`M`, except that in the lower triangle (i.e., all the cells below the diagonal), all values will be zero.

[23]:

```
# YOUR CODE HERE

# solution
new_M = np.triu(M)

print("new_M=\n", new_M)
```

```
new_M=
 [[0.75 0.7  0.41 0.73 0.8 ]
 [0.   0.42 0.15 0.06 0.85]
 [0.   0.   0.6  0.11 0.87]
 [0.   0.   0.   0.31 0.14]
 [0.   0.   0.   0.   0.64]]
```

Task: Using the code provided above for generating the matrix `M`, try creating a matrix with 13 rows and 3 columns containing random numbers. Save the resulting matrix to the variable `random_M`.

[24]:

```
# YOUR CODE HERE

## solution:
random_M = np.random.rand(13,3)

print("random_M= \n", random_M)
```

```
random_M=
 [[0.72028107 0.02436993 0.51065193]
 [0.60973632 0.09874522 0.6686417 ]
 [0.85832115 0.13160338 0.85620202]
 [0.90788148 0.88397237 0.22805728]
 [0.61060795 0.15310251 0.40357643]
 [0.97471108 0.3530429  0.08198431]
 [0.48660625 0.79328798 0.15435985]
 [0.46283947 0.87651794 0.26846791]
 [0.948515   0.40256905 0.15315622]
 [0.01549145 0.64641918 0.61757354]
 [0.393324   0.21539402 0.31491603]
 [0.82532971 0.81649614 0.69131873]
 [0.22088186 0.50307147 0.96141081]]
```

1.1.11 j. Indexing and slicing two-dimensional NumPy arrays

The code cell below extracts an element of a two-dimensional NumPy array by indexing into the array by specifying its location. Just like Python lists, NumPy arrays use 0-based indexing.

```
[25]: random_M[3][2]
```

```
[25]: 0.2280572801565024
```

You can also use the following syntax to achieve the same result.

```
[26]: random_M[3,2]
```

```
[26]: 0.2280572801565024
```

You learned how to slice a Pandas DataFrames. You can use the same techniques to slice a NumPy array.

Task: In the code cell below, use slicing to obtain the rows with the index 3 through 5 in `random_M`.

```
[27]: # YOUR CODE HERE

      ## solution:
      random_M[3:5]
```

```
[27]: array([[0.90788148, 0.88397237, 0.22805728],
             [0.61060795, 0.15310251, 0.40357643]])
```

Task: In the code cell below, use slicing to obtain all of the rows in the second column (column has the index of 1) of `random_M`.

```
[28]: # YOUR CODE HERE

      ## solution:
      random_M[:,1]
```

```
[28]: array([0.02436993, 0.09874522, 0.13160338, 0.88397237, 0.15310251,
             0.3530429 , 0.79328798, 0.87651794, 0.40256905, 0.64641918,
             0.21539402, 0.81649614, 0.50307147])
```

Task: Use the code cell below to perform slicing on `random_M` to obtain a portion of the array of your choosing.

```
[29]: # YOUR CODE HERE

      ## solutions will vary
```

1.1.12 k. Evaluating a Boolean condition

In real-life data tasks, you will often have to compute the boolean `(True/False)` value of some statement for all entries in a given NumPy array. You will formulate a condition — think of it as a *test* — and run a computation that returns `True` or `False` depending on whether the test passed or failed by a particular value in the array.

The condition may be something like "the value is greater than 0.5". You would like to know if this is true or false for every value in the array.

The code cells below demonstrates how to perform such a task on NumPy arrays.

First, we will create the array:

```
[30]: our_array = np.random.rand(1, 20)
      print(our_array)
```

```
[[0.16413681 0.68658671 0.57274477 0.11738278 0.35342681 0.83766208
  0.76478514 0.76465352 0.2657361  0.08498243 0.09994117 0.57582751
  0.89215448 0.51822695 0.05285661 0.84455553 0.27100198 0.43418256
  0.40526204 0.8529377 ]]
```

Next, we will apply a condition to the array:

[31]:
```
is_greater = our_array > 0.5
print(is_greater)
```

```
[[False  True  True False False  True  True  True False False False  True
   True  True False  True False False False  True]]
```

Let's apply this technique to our matrix `random_M`. Let's inspect the matrix again as a refresher.

[32]:
```
print(random_M)
```

```
[[0.72028107 0.02436993 0.51065193]
 [0.60973632 0.09874522 0.6686417 ]
 [0.85832115 0.13160338 0.85620202]
 [0.90788148 0.88397237 0.22805728]
 [0.61060795 0.15310251 0.40357643]
 [0.97471108 0.3530429  0.08198431]
 [0.48660625 0.79328798 0.15435985]
 [0.46283947 0.87651794 0.26846791]
 [0.948515   0.40256905 0.15315622]
 [0.01549145 0.64641918 0.61757354]
 [0.393324   0.21539402 0.31491603]
 [0.82532971 0.81649614 0.69131873]
 [0.22088186 0.50307147 0.96141081]]
```

Task: In the code cell below, determine whether the value of every element in the second column of `random_M` is greater than 0.5. Save the result to the variable `is_greater`.

[33]:
```
# YOUR CODE HERE

## solution:
is_greater = random_M[:,1] > 0.5

print(is_greater)
```

```
[False False False  True False False  True  True False  True False  True
  True]
```

We can use the function `np.any()` to determine if there is any element in a NumPy array that is True. Let us apply this to the array `is_greater` above. Using this function we can easily determine that indeed there are values greater than 0.5 in the second row of `random_M`.

[34]:
```
np.any(is_greater)
```

[34]:
```
True
```

Let's apply `np.any()` to another condition.

Task: Use `np.any()` along with a conditional statement to determine if any value in the third row of `random_M` is less than .1.

[35]:
```
# YOUR CODE HERE

## solution:
np.any(random_M[:,2] < .1)
```

[35]:
```
True
```

1.2 Practice Working With Pandas DataFrames

1.2.1 a. Creating a DataFrame: two (of the many) ways

The code cells below demonstrate how we can create Pandas DataFrames in two ways:

1. from a *list of lists*
2. from a *dictionary*

First, the cell below creates a DataFrame from a list containing phone numbers and their country codes. The DataFrame is named `df`. Run the cell below to inspect the DataFrame `df` that was created.

[36]:
```
my_list = [['+1', '(929)-000-0000'], ['+34', '(917)-000-0000'], ['+7',␣
 ↪'(470)-000-0000']]

df = pd.DataFrame(my_list, columns = ['country_code', 'phone'])
df
```

[36]:
```
  country_code           phone
0           +1  (929)-000-0000
1          +34  (917)-000-0000
2           +7  (470)-000-0000
```

Second, the cell below creates a DataFrame from a dictionary that contains the same information as the list above. The dictionary contains phone numbers and their country codes. Run the cell below to inspect the DataFrame `df_from_dict` that was created from the dictionary. Notice that both DataFrames `df` and `df_from_dict` contain the same values.

[37]:
```
my_dict = {'country_code': ['+1', '+34', '+7'], 'phone':['(929)-000-0000',␣
 ↪'(917)-000-0000', '(470)-000-0000']}

df_from_dict = pd.DataFrame(my_dict)
df_from_dict
```

[37]:
```
  country_code           phone
0           +1  (929)-000-0000
1          +34  (917)-000-0000
2           +7  (470)-000-0000
```

1.2.2 b. Adding a column to a DataFrame object

We are going to continue working with the DataFrame `df` that was created above. The code cell below adds a new column of values to `df`. Run the cell and inspect the DataFrame to see the new column that was added.

[38]:
```
df['grade']= ['A','B','A']
df
```

[38]:
```
  country_code            phone grade
0           +1   (929)-000-0000     A
1          +34   (917)-000-0000     B
2           +7   (470)-000-0000     A
```

Task: In the cell below, create a new column in DataFrame `df` that contains the names of individuals.

- First, create a list containing three names of your choosing.
- Next, create a new column in `df` called `names` by using the list you created.

[39]:
```
# YOUR CODE HERE

## sample solution:
names = ['bobby', 'susy', 'frank']
df['name']= names



print(df)
```

```
  country_code            phone grade   name
0           +1   (929)-000-0000     A  bobby
1          +34   (917)-000-0000     B   susy
2           +7   (470)-000-0000     A  frank
```

1.2.3 c. Sorting the DataFrame by values in a specific column

The `df.sort_values()` method sorts a DataFrame by the specified column. The code cell below will use `df.sort_values()` to sort DataFrame`df` by the values contained in column `grade`. The original DataFrame `df` will not be changed, so we will assign the resulting DataFrame to variable `df` to update the values in the DataFrame.

[40]:
```
df = df.sort_values(['grade'])
df
```

[40]:
```
  country_code            phone grade   name
0           +1   (929)-000-0000     A  bobby
2           +7   (470)-000-0000     A  frank
1          +34   (917)-000-0000     B   susy
```

1.2.4 d. Combining multiple DataFrames and renaming columns with `df.rename()`

In real life settings, you will often need to combine separate sets of related data. Two functions used for this purpose are `pd.concat()` and `pd.merge()`.

To illustrate, let's create a new DataFrame. The code cell below creates a new DataFrame `df2` that also contains phone numbers, their country codes and a grade. Run the cell and inspect the new DataFrame that was created.

[41]:
```
my_dict2 = {'country': ['+32', '+81', '+11'], 'grade':['B', 'B+', 'A'], 'phone':
 ↪['(874)-444-0000', '(313)-003-1000', '(990)-006-0660']}

df2 = pd.DataFrame(my_dict2)
df2
```

[41]:
```
  country grade           phone
0     +32     B  (874)-444-0000
1     +81    B+  (313)-003-1000
2     +11     A  (990)-006-0660
```

The code cell below uses the Pandas `pd.concat()` function to append `df2` to `df`. The `pd.concat()` function will not change the values in the original DataFrames, so we will save the newly formed DataFrame to variable `df_concat`.

[42]:
```
df_concat = pd.concat([df,df2])
df_concat
```

[42]:
```
  country_code           phone grade   name country
0           +1  (929)-000-0000     A  bobby     NaN
2           +7  (470)-000-0000     A  frank     NaN
1          +34  (917)-000-0000     B   susy     NaN
0          NaN  (874)-444-0000     B    NaN     +32
1          NaN  (313)-003-1000    B+    NaN     +81
2          NaN  (990)-006-0660     A    NaN     +11
```

Notice that the new DataFrame `df_concat` contains two columns containing country codes. This is because the two original DataFrames contained different spellings for the columns.

We can easily fix this by changing the name of the column in DataFrame `df2` to be consistent with the name of the column in DataFrame `df`.

[43]:
```
df2 = df2.rename(columns={'country':'country_code'})
df2
```

[43]:
```
  country_code grade           phone
0          +32     B  (874)-444-0000
1          +81    B+  (313)-003-1000
2          +11     A  (990)-006-0660
```

Task: In the cell below, run the `pd.concat()` function again to concatenate DataFrames `df` and `df2` and save the resulting DataFrame to variable `df_concat2`. Run the cell and inspect the results.

[44]:
```
# YOUR CODE HERE

#solution
df_concat2 = pd.concat([df,df2])
```

```
print(df_concat2)
```

```
  country_code           phone grade   name
0           +1  (929)-000-0000     A  bobby
2           +7  (470)-000-0000     A  frank
1          +34  (917)-000-0000     B   susy
0          +32  (874)-444-0000     B    NaN
1          +81  (313)-003-1000    B+    NaN
2          +11  (990)-006-0660     A    NaN
```

One other problem is that the index has repeated values. This defeats the purpose of an index, and ought to be fixed. Let's try the concatenation again, this time adding `reset_index()` method to produce correct results:

[45]:
```
df_concat2 = pd.concat([df,df2]).reset_index()
df_concat2
```

[45]:
```
   index country_code           phone grade   name
0      0           +1  (929)-000-0000     A  bobby
1      2           +7  (470)-000-0000     A  frank
2      1          +34  (917)-000-0000     B   susy
3      0          +32  (874)-444-0000     B    NaN
4      1          +81  (313)-003-1000    B+    NaN
5      2          +11  (990)-006-0660     A    NaN
```

Now we have one column for `country_code`. Notice that we have missing values for the names of individuals, since names were contained in `df` but not in `df2`. In a future unit, you will learn how to deal with missing values.

What if our task were to merge `df2` with yet another dataset — one that contains additional unique columns? Let's look at DataFrame `df2` again:

[46]:
```
df2
```

[46]:
```
  country_code grade           phone
0          +32     B  (874)-444-0000
1          +81    B+  (313)-003-1000
2          +11     A  (990)-006-0660
```

The code cell below creates a new DataFrame `df3`.

[47]:
```
my_dict3 = {'country_code': ['+32', '+44', '+11'], 'phone':['(874)-444-0000',␣
 ↪'(575)-755-1000', '(990)-006-0660'], 'grade':['B', 'B+', 'A'], 'n_credits':␣
 ↪[12, 3, 9]}

df3 = pd.DataFrame(my_dict3)
df3
```

[47]:
```
  country_code           phone grade  n_credits
0          +32  (874)-444-0000     B         12
1          +44  (575)-755-1000    B+          3
```

```
2          +11  (990)-006-0660     A          9
```

The following code cell merges both DataFrames based on the values contained in the `phone` column. If one column in both DataFrames contains the same value, the rows in which the value appears are merged. Otherwise, the row will not be included in the updated DataFrame. Run the code cell below and inspect the results. Note that the values in DataFrame `df2` will be automatically changed by `merge()`.

[48]:
```
df2.merge(df3, on = 'phone')
df2
```

[48]:
```
  country_code grade           phone
0          +32     B  (874)-444-0000
1          +81    B+  (313)-003-1000
2          +11     A  (990)-006-0660
```

1.3 Practice Working With a Dataset

We are now well equipped to deal with a real dataset! Our dataset will contain information about New York City listings on the Airbnb platform.

1.3.1 a. Load the dataset: `pd.read_csv()`

The code cell below loads a dataset from a CSV file and saves it to a Pandas DataFrame.

First, we will import the `OS` module. This module enables you to interact with the operating system, allowing you access to file names, etc.

[49]:
```
import os
```

Next, we will use the `os.path.join()` method to obtain a path to our data file. This method concatenates different path components (i.e. directories and a file name, into one file system path). We will save the results of this method to the variable name `filename`.

Now that we have a path to our CSV file, we will use the `pd.read_csv()` method to load the CSV file into a Pandas DataFrame named `dataFrame`.

Examine the code in the cell below and run the cell.

Note: the cell below may generate a warning. Ignore the warning.

[50]:
```
filename = os.path.join(os.getcwd(), "data", "airbnbData.csv")
dataFrame = pd.read_csv(filename)
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2728:
DtypeWarning: Columns (56) have mixed types.Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[51]:
```
dataFrame.shape
```

[51]:
```
(38277, 63)
```

First, get a peek at the data:

[52]:
```
dataFrame.head()
```

[52]:

```
     id     scrape_id last_scraped  host_id host_since  host_response_time  \
0  2595  2.021120e+13      12/5/21     2845     9/9/08        within a day   
1  3831  2.021120e+13      12/5/21     4869    12/7/08  a few days or more   
2  5121  2.021120e+13      12/5/21     7356     2/3/09      within an hour   
3  5136  2.021120e+13      12/5/21     7378     2/3/09        within a day   
4  5178  2.021120e+13      12/5/21     8967     3/3/09        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_communication  \
0             Midtown  ...                         4.79   
1        Clinton Hill  ...                         4.80   
2  Bedford-Stuyvesant  ...                         4.91   
3   Greenwood Heights  ...                         5.00   
4      Hell's Kitchen  ...                         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 63 columns]
```

When using the `head()` method, you can specify the number of rows you would like to see by calling `head()` with an integer parameter (e.g. `head(2)`).

1.3.2 b. Get column names: `df.columns`

Let us retrieve just the list of column names.

[53]:
```python
list(dataFrame.columns)
```

[53]:
```
['id',
 'scrape_id',
 'last_scraped',
 'host_id',
 'host_since',
 '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']
```

What do the column names mean? Some of them are less intuitively interpretable than others. Careful data documentation is indispensable for business analytics. You can consult the documentation that accompanies this open source dataset for a detailed description of the key variable names, what they represent, and how they were generated.

1.3.3 c. Summary statistics of the DataFrame: `df.describe()`

Let's print some general statistics for each one of the `data` columns:

[54]:
```
dataFrame.describe(include='all')
```

[54]:
```
                  id     scrape_id last_scraped       host_id host_since  \
count   3.827700e+04  3.827700e+04        38277  3.827700e+04      38243
unique           NaN           NaN            2           NaN       4289
top              NaN           NaN      12/5/21           NaN   10/29/19
freq             NaN           NaN        31879           NaN        433
mean    2.962239e+07  2.021120e+13          NaN  1.148305e+08        NaN
std     1.742239e+07  0.000000e+00          NaN  1.299194e+08        NaN
min     2.595000e+03  2.021120e+13          NaN  2.438000e+03        NaN
25%     1.341048e+07  2.021120e+13          NaN  1.139462e+07        NaN
50%     3.081269e+07  2.021120e+13          NaN  5.005297e+07        NaN
```

```
75%     4.642855e+07  2.021120e+13          NaN  2.002395e+08        NaN
max     5.366510e+07  2.021120e+13          NaN  4.344080e+08        NaN

       host_response_time host_response_rate host_acceptance_rate  \
count               21084              21084                21791
unique                  4                 88                  101
top        within an hour               100%                 100%
freq                11151              13299                 5342
mean                  NaN                NaN                  NaN
std                   NaN                NaN                  NaN
min                   NaN                NaN                  NaN
25%                   NaN                NaN                  NaN
50%                   NaN                NaN                  NaN
75%                   NaN                NaN                  NaN
max                   NaN                NaN                  NaN

       host_is_superhost  host_neighbourhood  ...  \
count              38243               30813  ...
unique                 2                 484  ...
top                    f  Bedford-Stuyvesant  ...
freq               30865                2138  ...
mean                 NaN                 NaN  ...
std                  NaN                 NaN  ...
min                  NaN                 NaN  ...
25%                  NaN                 NaN  ...
50%                  NaN                 NaN  ...
75%                  NaN                 NaN  ...
max                  NaN                 NaN  ...

        review_scores_communication  review_scores_location  \
count                  28165.000000            28151.000000
unique                          NaN                     NaN
top                             NaN                     NaN
freq                            NaN                     NaN
mean                       4.807454                4.750307
std                        0.465544                0.416101
min                        0.000000                0.000000
25%                        4.810000                4.670000
50%                        4.970000                4.880000
75%                        5.000000                5.000000
max                        5.000000                5.000000

       review_scores_value   license instant_bookable  \
count         28150.000000         1            38277
unique                 NaN         1                2
top                    NaN  41662/AL                f
freq                   NaN         1            27851
```

```
mean              4.646892       NaN              NaN
std               0.518905       NaN              NaN
min               0.000000       NaN              NaN
25%               4.550000       NaN              NaN
50%               4.780000       NaN              NaN
75%               5.000000       NaN              NaN
max               5.000000       NaN              NaN

       calculated_host_listings_count  \
count                    38277.000000
unique                            NaN
top                               NaN
freq                              NaN
mean                        17.747655
std                         59.150451
min                          1.000000
25%                          1.000000
50%                          1.000000
75%                          3.000000
max                        421.000000

       calculated_host_listings_count_entire_homes  \
count                                 38277.000000
unique                                         NaN
top                                            NaN
freq                                           NaN
mean                                      8.042637
std                                      34.977178
min                                       0.000000
25%                                       0.000000
50%                                       1.000000
75%                                       1.000000
max                                     308.000000

       calculated_host_listings_count_private_rooms  \
count                                  38277.000000
unique                                          NaN
top                                             NaN
freq                                            NaN
mean                                       9.593934
std                                       43.310123
min                                        0.000000
25%                                        0.000000
50%                                        0.000000
75%                                        1.000000
max                                      359.000000
```

```
        calculated_host_listings_count_shared_rooms  reviews_per_month
count                                  38277.000000       28773.000000
unique                                          NaN                NaN
top                                             NaN                NaN
freq                                            NaN                NaN
mean                                       0.047966           1.721019
std                                        0.426789           4.399826
min                                        0.000000           0.010000
25%                                        0.000000           0.120000
50%                                        0.000000           0.480000
75%                                        0.000000           1.780000
max                                        8.000000         141.000000

[11 rows x 63 columns]
```

1.3.4 d. Filtering the data: `df[ < condition > ]`

Consider the following business question: What is the average availability (out of 365 days in a year) for the listings in Brooklyn?

The answer can be obtained by the use of **filters** on the dataset. We need to filter the entries that are in Brooklyn. To do this, we need to know the exact way that Manhattan listings are spelled and entered in the data. Let's print all of the unique values of the `neighbourhood` column:

[55]:
```
dataFrame['neighbourhood'].unique()
```

[55]:
```
array(['New York, United States', 'Brooklyn, New York, United States',
       nan, 'Queens, New York, United States',
       'Long Island City, New York, United States',
       'Astoria, New York, United States',
       'Bronx, New York, United States',
       'Staten Island, New York, United States',
       'Elmhurst, New York, United States',
       'Riverdale , New York, United States',
       'Briarwood, New York, United States',
       'Kips Bay, New York, United States',
       'Jackson Heights, New York, United States',
       'New York, Manhattan, United States',
       'Park Slope, Brooklyn, New York, United States',
       'Kew Gardens, New York, United States',
       'Flushing, New York, United States',
       'Astoria , New York, United States',
       'Sunnyside, New York, United States',
       'Woodside, New York, United States',
       'NY , New York, United States',
       'Bushwick, Brooklyn, New York, United States',
       'Brooklyn , New York, United States', 'United States',
       'Sunnyside , New York, United States',
       'LONG ISLAND CITY, New York, United States',
       'Astoria, Queens, New York, United States',
       'Woodhaven, New York, United States',
       'bronx, New York, United States',
       'Harlem, New York, United States',
       'brooklyn, New York, United States',
       'Middle Village, New York, United States',
       'BROOKLYN, New York, United States',
       'Brooklyn,  Ny 11221, New York, United States',
       'Staten Island , New York, United States',
       'Greenpoint, Brooklyn, New York, United States',
       'Long Island city, New York, United States',
       'astoria, New York, United States',
       'The Bronx, New York, United States',
       'ASTORIA, New York, United States',
       'Ridgewood , New York, United States',
       'Ridgewood, New York, United States',
       'Jamaica, New York, United States',
       'Bayside, New York, United States',
       'Jackson heights , New York, United States',
       'East Elmhurst, New York, United States',
       'Williamsburg, Brooklyn, New York, United States',
       'Williamsburg, New York, United States',
       'LIC, New York, United States',
       'Brooklyn. , New York, United States',
       'Manhattan, New York, United States',
       'New-York, New York, United States',
       'Far Rockaway, New York, United States',
       'Richmond Hill, New York, United States',
       'forest hills/corona, New York, United States',
       'Jackson  hights , New York, United States',
       'Clinton Hill Brooklyn, New York, United States',
       'Flushing , New York, United States',
       'Elmhurst , New York, United States',
       'Brooklyn, Northern Mariana Islands, United States',
       'queens, New York, United States',
       'Flushing /Kew Gardens Hills, New York, United States',
       'RIVERDALE, New York, United States',
       'East elmhurst, New York, United States',
       'Forest Hills, New York, United States',
       'SUNNYSIDE, New York, United States',
       'Maspeth, New York, United States',
       'Fresh Meadows , New York, United States',
       'NY, New York, United States',
       'Floral Park, New York, United States',
       'new york, New York, United States',
       'Richmond hill, New York, United States',
       'Jackson heights, New York, United States',
       'Astoria Queens, New York, United States',
       'New York city, New York, United States',
       'Queens Village, New York, United States',
       'New York , New York, United States',
       'Corona, New York, United States',
       'Gravesend Brooklyn , New York, United States',
       'MIDDLE VILLAGE, New York, United States',
       'Bronx , New York, United States',
       'Bushwick, New York, United States',
       'Queens , New York, United States',
       'Rockaway beach , New York, United States',
       'Arverne, New York, United States',
       'flushing , New York, United States',
       'Parkchester , New York, United States',
       'Fresh meadows, New York, United States',
       'flushing, New York, United States',
       'Manhattan , New York, United States',
       'Kew Gardens , New York, United States',
       'Rockaway Beach , New York, United States',
       'Rockaway Beach, New York, United States',
       'Manhattan, New York, New York, United States',
       'Jackson Heights , New York, United States',
       'Flush, New York, United States',
       'Jamaica , New York, United States',
       'Corona , New York, United States',
       ' Crown Heights,NY, New York, United States',
       'Jamaica , ny, United States',
       'ozone park queens , New York, United States',
       'Bushwick , New York, United States',
       'New York, US, New York, United States',
       'Forest hills, New York, United States',
       'Woodside , New York, United States',
       'Cambria heights , New York, United States',
       '8425 Elmhurst avenue , New York, United States',
       ', New York, United States',
       'Rego Park, New York, United States',
       'Bronx, NY, New York, United States',
       'Springfield Gardens , New York, United States',
       ', New York, United States', 'Hollis, New York, United States',
       'Springfield Gardens, New York, United States',
       'FOREST HILLS, New York, United States',
       'Brookly , New York, United States',
       'elmhurst Queens, New York, United States',
       'Ozone Park, New York, United States',
       'East elmhurst , New York, United States',
       'South Richmond Hill, New York, United States',
       'Staten island , New York, United States',
       'Glendale , New York, United States',
       'Woodhaven , New York, United States',
       'New York City , New York, United States',
       'Pomona, California, United States',
       'Williamsburg, Brooklyn , New York, United States',
       'Bronx New York, New York, United States',
       'Astoria Queens , New York, United States',
       'Fresh Meadows, New York, United States',
       'St. Albans , New York, United States',
       'New York City, New York, United States',
       'Springfield gardens, New York, United States',
       'Richmond Hill, Jamaica, Queens, New York, United States',
       'west new york , New Jersey, United States',
       'East Elmhurst , New York, United States',
       'East Elmhurst or Flushing , New York, United States',
       'Oakland Gardens , New York, United States',
       'Newyork, New York, United States',
       'Long island city , New York, United States',
       'New york, New York, United States',
       'bronx , New York, United States',
       'Flushing or east Elmhurst , New York, United States',
       'Laurelton , New York, United States',
       'Brooklyn, New York, New York, United States',
       'Lawrence, New York, United States',
       'Bushwick Brooklyn , New York, United States',
       'Richmond Hill , New York, United States',
       'Brooklyn Heights , New York, United States',
       'Rosedale , New York, United States',
       'Sunnyside, Queens, New York, United States',
       'Middle village, New York, United States',
       'BROOKLYN , New York, United States',
       'Arverne, Queens, New York, United States',
       'Saint Albans , New York, United States',
       'Fort Greene, New York, United States',
       'Saint Albans, New York, United States',
       ' Astoria, New York, United States',
       'Maspeth , New York, United States',
       'New York,Manhattan , New York, United States',
       'Williamsburg , New York, United States',
       'Long Island, New York, United States',
       'Howard Beach, New York, United States',
       'Little neck, New York, United States',
       'New York , Ny, United States',
       'New York - Sunnyside , New York, United States',
       'Glendale, New York, United States',
       'Queens Village , New York, United States',
       'forest hills, New York, United States',
       'NYC , New York, United States',
       'Rosedale, New York, United States',
       'Queens, Flushing , New York, United States',
       'Jamaica queens, New York, United States',
       'NEW YORK, New York, United States',
       'Laurelton , Queens , New York, United States',
       ' Springfield Gardens, New York, United States',
       'Queens, Astoria , New York, United States',
       'The Bronx (Riverdale), New York, United States',
       'Bushwick Brooklyn, New York, United States',
       'Laurelton, New York, United States',
       'Forest Hill, New York, United States',
       ' Forest Hills, New York, United States',
       'Long Island City, Queens, New York, United States',
       'Brooklyn , Ny, United States',
       'Queens village, New York, United States',
       'Greenpoint Brooklyn , New York, United States',
       'Elmont, New York, United States',
       'WOODSIDE , New York, United States',
       'Queens, New York , United States',
       'Broklyn , New York, United States',
       'Queens-Rego Park, New York, United States',
       'Rego Park , New York, United States',
       'North Bronx (Wakefield), New York, United States',
       'Woodside, Queens, New York, United States',
       'South Ozone Park, New York, United States',
       'woodside, New York, United States',
       'Corona queens , New York, United States',
       'Nueva York, New York, United States',
       'Forest hills , New York, United States',
       'New york, Ny, United States',
       ' East Elmhurst, New York, United States',
       'South ozone park , New York, United States',
       'Long Island city , New York, United States',
       'New York, Ny, United States',
       'Mount Vernon, New York, United States', 'New York, NY, Argentina',
       'Montbel, Lozère, France', 'Scottsdale, Arizona, United States',
       'Yonkers, New York, United States'], dtype=object)
```

You may have noticed that there is a lot of heterogeneity in the way `neighbourhood` values are specified. The values are not standardized. There are overlaps, redundancies, and inconsistencies (e.g., some entries specify `'Greenpoint, Brooklyn, New York, United States'`, some other ones list `'BROOKLYN, New York, United States',`, yet other ones say `'Williamsburg, Brooklyn, New York, United States'`, etc. In real life, you would have to clean this data and replace these values with standard, identically formated, consistent values.

For this dataset, we are lucky to already have a 'cleansed' version of the neighborhood information based on the latitude and the longitude of every listing location.

We will list the unique values of the columns titled `neighbourhood_cleansed` and neighbourhood_group_cleansed:

[56]:
```
dataFrame['neighbourhood_cleansed'].unique()
```

[56]:
```
array(['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'], dtype=object)
```

[57]:

```
dataFrame['neighbourhood_group_cleansed'].unique()
```

[57]:

```
array(['Manhattan', 'Brooklyn', 'Queens', 'Staten Island', 'Bronx'],
      dtype=object)
```

Let's filter out all data entries that pertain to Brooklyn listings:

[58]:

```
bk = dataFrame[dataFrame['neighbourhood_group_cleansed'] == 'Brooklyn']
bk.shape
```

[58]:

```
(14716, 63)
```

Tip: to better understand what happened above, in the code cell below, you are encouraged to copy *just the condition* of the filter that we used on the `data` object above: `dataFrame['neighbourhood_group_cleansed'] == 'Brooklyn'`.

Run the cell and see what that condition alone evaluates to. You should see a Pandas series containing True/False values. When we use that series as a Boolean filter by writing `dataFrame[ < our Boolean series > ]`, i.e `dataFrame['neighbourhood_group_cleansed'] == 'Brooklyn']` , we are telling Pandas to keep the values in the DataFrame `dataFrame` only with those indices for which the condition evaluated to `True`.

[59]:

```
# YOUR CODE HERE

# solution
dataFrame['neighbourhood_group_cleansed'] == 'Brooklyn'
```

[59]:

```
0        False
1         True
2         True
3         True
4        False
         ...
38272    False
38273    False
38274    False
38275    False
38276     True
Name: neighbourhood_group_cleansed, Length: 38277, dtype: bool
```

1.3.5 e. Combining values in a column: `np.mean()`

Now that we isolated only the relevant entries, it remains to average the value of a particular column that we care about:

[60]:
```
np.mean(bk['availability_365'])
```

[60]:
```
118.7693666757271
```

1.3.6 f. Group data by (categorical) column values: `df.groupby()`

The next question of interest could be:

What are the top 5 most reviewed neighborhoods in New York? (By sheer number of reviews, regardless of their quality).

We will use the Pandas `df.groupby()` method to determine this:

[61]:
```
nbhd_reviews = dataFrame.groupby('neighbourhood_cleansed')['number_of_reviews'].
 ↪sum()
nbhd_reviews.head()
```

[61]:
```
neighbourhood_cleansed
Allerton              1611
Arden Heights           86
Arrochar               867
Arverne               3091
Astoria              18207
Name: number_of_reviews, dtype: int64
```

Perform a (descending order) sorting on this series:

[62]:
```
nbhd_reviews = nbhd_reviews.sort_values(ascending = False)
nbhd_reviews.head(5)
```

[62]:
```
neighbourhood_cleansed
Bedford-Stuyvesant    88133
Williamsburg          55122
Harlem                54824
Bushwick              34776
Hell's Kitchen        31308
Name: number_of_reviews, dtype: int64
```

What are the least reviewed neighborhoods?

[63]:
```
nbhd_reviews.tail(5)
```

[63]:
```
neighbourhood_cleansed
Little Neck       11
Sea Gate           9
Graniteville       5
Country Club       1
Fort Wadsworth     0
Name: number_of_reviews, dtype: int64
```

This result makes it apparent that our dataset is somewhat messy!

Notice we could have chained the transformations above into a single command, as in:

[64]:
```
dataFrame.groupby('neighbourhood_cleansed')['number_of_reviews'].sum().
 ↪sort_values(ascending = False).head(5)
```

[64]:
```
neighbourhood_cleansed
Bedford-Stuyvesant    88133
Williamsburg          55122
Harlem                54824
Bushwick              34776
Hell's Kitchen        31308
Name: number_of_reviews, dtype: int64
```

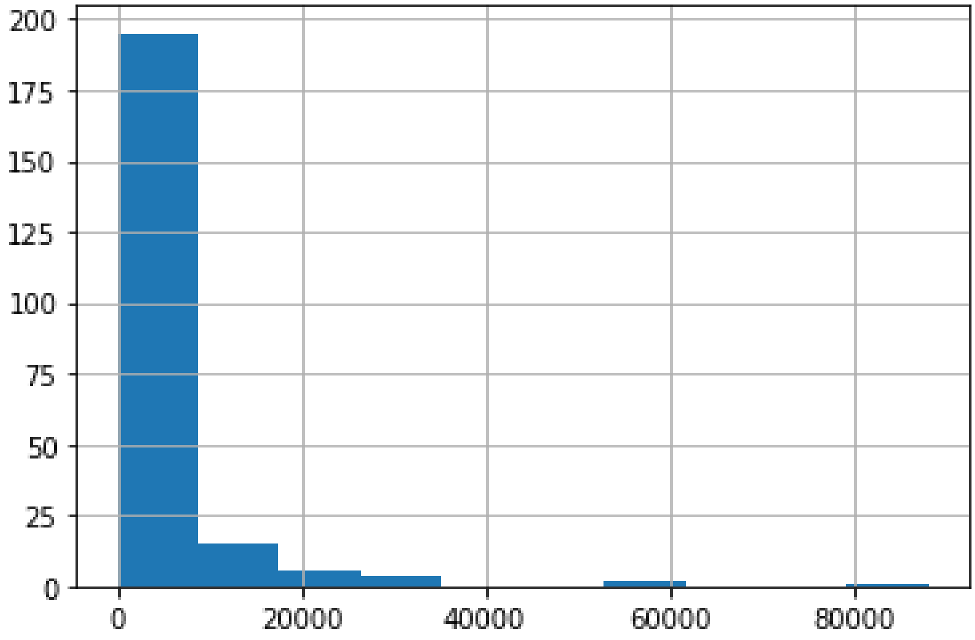
This way we don't store objects that we won't need.

1.3.7 Bonus: Histogram plotting with Matplotlib: `plt.hist()`

As a final touch, run the cell below to visualize the density of average values of review numbers across all neighborhoods. Note: The cell may take a few seconds to run.

[65]:
```
%matplotlib inline
nbhd_reviews.hist()
```

[65]:
```
<AxesSubplot:>
```



This plot suggests that the vast majority of neighborhoods have only very few reviews, with just a handful of outliers (those ranked at the top in our previous computed cell) having the number of reviews upward of 40000.

1.4 Part 2. ML Life Cycle: Business Understanding and Problem Formulation

In this part of the lab, you will practice the first step of the machine learning life cycle: business understanding and problem formulation.

Recall that the first step of the machine learning life cycle involves understanding and formulating your ML business problem, and the second step involves data understanding and preparation. In this lab however, we will first provide you with data and have you formulate a machine learning business problem based on that data.

We have provided you with four datasets that you will use to formulate a machine learning problem.

1. HousingPrices.csv: dataset that contains information about a house's characteristics (number of bedrooms, etc.) and its purchase price.

2. Top100Restaurants2020.csv: dataset that contains information about 100 top rated restaurants in 2020.

3. ZooData.csv: dataset that contains information about a variety of animals and their characteristics.

4. FlightInformation.csv: dataset that contains flight information.

The code cells below use the specified paths and names of the files to load the data into four different DataFrames.

Task #1: After you run a code cell below to load the data, use some of the techniques you have practiced to inspect the data. Do the following:

1. Inspect the first 10 rows of each DataFrame.
2. Inspect all of the column names in each DataFrame.
3. Obtain the shape of each DataFrame.

(Note: You can add more cells below to accomplish this task by going to the `Insert` Menu and clicking on `Insert Cell Below`. By default, the new code cell will be of type `Code`.)

Task #2: Once you have an idea of what is contained in a dataset, you will formulate a machine learning problem for that dataset. This will be a predictive problem. For example, the Airbnb dataset you worked with above can be used to train a machine learning model that can predict the price of a new Airbnb.

Come up with at least one machine learning problem per dataset. Specify what you would like to use the data to predict in the future. Since these will be supervised learning problems, specify whether it is a classification (binary or multiclass) or a regression problem. List the label and feature columns.

Note: Make sure you successfully ran the cell above that loads the `OS` module prior to running the cells below.

Housing Prices Dataset:

[66]:

```
filename1 = os.path.join(os.getcwd(), "data", "HousingPrices.csv")

dataFrame1 = pd.read_csv(filename1)
```

Inspect the data:

[76]:

```
# YOUR CODE HERE

# Solution:
dataFrame1.head(10)
dataFrame1.columns
dataFrame1.shape
```

[76]:

```
(545, 13)
```

Formulate ML Business Problem:

Solution:

Solutions may vary. This dataset is suited for regression and can be used to predict the price of a new home. The "price" column is the label.

Restaurants Dataset:

[77]:

```
filename2 = os.path.join(os.getcwd(), "data", "Top100Restaurants2020.csv")

dataFrame2 = pd.read_csv(filename2)
```

Inspect the data:

[78]:

```
# YOUR CODE HERE

# Solution:
dataFrame2.head(10)
dataFrame2.columns
dataFrame2.shape
```

[78]:

```
(100, 8)
```

Formulate ML Business Problem:

Solution:

Solutions may vary. This dataset is suited for regression and can be used to predict the sales of a new restaurant. The "sales" column is the label.

Zoo Dataset:

[79]:

```
filename3 = os.path.join(os.getcwd(), "data", "ZooData.csv")

dataFrame3 = pd.read_csv(filename3)
```

Inspect the data:

[80]:

```
# YOUR CODE HERE

# Solution:
dataFrame3.head(10)
dataFrame3.columns
dataFrame3.shape
```

[80]:

```
(101, 18)
```

Formulate ML Business Problem:

Solution:

Solutions may vary. This dataset is suited for multi-class classification and can be used to predict the class of an animal. The "class_type" column is the label.

Flight Dataset:

[81]:
```
filename4 = os.path.join(os.getcwd(), "data", "FlightInformation.csv")

dataFrame4 = pd.read_csv(filename4)
```

Inspect the data:

[82]:
```
# YOUR CODE HERE

# Solution:
dataFrame4.head(10)
dataFrame4.columns
dataFrame4.shape
```

[82]:
```
(65499, 9)
```

Formulate ML Business Problem:

Solution:

Solutions may vary. This dataset is suited for binary classification and can be used to predict whether a flight may be delayed. The "delay" column is the label.

Next Steps: The second step of the machine learning life cycle is data understanding and data preparation. You practiced some aspects of data understanding when using NumPy and Pandas to inspect the Airbnb dataset. You will learn more about this second step of the machine learning life cycle in the next unit.