Lab1

August 8, 2023

1 Lab 1: Practice Working with NumPy and Pandas

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

In this lab you will practice working with the NumPy and Pandas packages. 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

1.1 Part 1. NumPy Arrays

1.1.1 a. Define a new Python *list*

Let us practice defining a new list in Python:

[2]:
```
my_list = [0,0,1,2,3,3,4.5,7.6]
my_list
```

[2]:
```
[0, 0, 1, 2, 3, 3, 4.5, 7.6]
```

1.1.2 b. (Two ways to) define a *range*

One type of list is a range of (e.g., integer) numbers. Ranges are useful for iterating over in a loop -- that is, to assign some variable to take on each value in this list sequentially. (For example, `i=1`, then `i=2`, then `i=3`, etc., until `i=100`.)

Let's create an evenly spaced array of integers ranging from 0 to 11:

[3]:
```
#First, the basic python way:
my_range = range(0,12)
my_range
```

[3]:
```
range(0, 12)
```

[4]:
```
#Now the numpy way:
my_range_np = np.arange(12)
my_range_np
```

[4]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

You can see that the first method returns a special 'range' object, while the latter returns an object of type 'numpy array'. If we convert both to lists, they will be the same:

[5]:
```
list(my_range)
```

[5]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

[6]:
```
list(my_range_np)
```

[6]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

1.1.3 c. List comprehension

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

[7]:
```
my_range_np = np.arange(12)

for i in range(0, len(my_range_np)):
    my_range_np[i] = my_range_np[i]**2 #square each element
print(my_range_np)
```

```
[  0   1   4   9  16  25  36  49  64  81 100 121]
```

That worked. However, there is a better, more 'Pythonic' way to do it.

List comprehension is one of the most elegant functionalities native to Python, and is beloved by programmers like you. It offers a concise way of applying a particular transformation to every element in a list.

Using list comprehension syntax, we can write a single, easily interpretable line that does the same transformation without using ranges, nor do we have to introduce an iterating index variable `i`:

[8]:
```
my_range_np = np.arange(12)

my_range_np = [x**2 for x in my_range_np]
my_range_np
```

[8]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

1.1.4 d. Creating a one-dimensional NumPy *array*.

Let's explicitly create a one-dimensional `numpy` array (as opposed to a list):

[9]:
```
arr = np.array([1, 2, 3, 4])
arr
```

[9]:
```
array([1, 2, 3, 4])
```

1.1.5 e. Retrieving the dimensions of data structure: len() and np.shape()

How would we go about creating a variable that contains the length of our array `arr` ? We could use the Python function `len()`...

[10]:
```
arr_length = len(arr)
arr_length
```

[10]:
```
4
```

... or use the numpy function `np.shape()`, saving only the first of the two values that it returns:

[13]:
```
arr_length = np.shape(arr)[0]
arr_length
```

[13]:
```
4
```

Tip: try removing the slice indicator `[0]` and see how the output changes. Notice that there appears to be an empty 'slot' for another number in the returned value pair.

1.1.6 f. Creating a uniform (same value in every position) array: np.ones()

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

[14]:
```
length = 55
np.ones(length, dtype=int)
```

[14]:
```
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 values. Let's create an array of length 13, filled with the vlue '7' in every position:

[15]:
```
7*np.ones(13, dtype=int)
```

[15]:
```
array([7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
```

1.1.7 g. Creating two-dimensional arrays

Exploring the possibilities of the `np.array()` function further, let's move on to creating not one- but two-dimensional arrays (aka matrices):

[16]:
```
M = np.array([[1,2,3], [4,5,6]])
M
```

[16]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[17]:
```
np.shape(M)
```

[17]:
```
(2, 3)
```

NumPy contains useful functions for creating identity matrices of a specified size:

1.1.8 h. Creating an identity matrix: np.eye()

[18]:
```
np.eye(5)
```

[18]:
```
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.]])
```

[19]:
```
#check your intuition: what will be the output after running this cell?
A = np.eye(3)
B = 4*np.eye(3)
A+B
```

[19]:
```
array([[5., 0., 0.],
       [0., 5., 0.],
       [0., 0., 5.]])
```

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

Using the matrix `M` below and the function `np.triu()`. Inspect the documentation by running the command `np.triu?` in the cell below. Create a matrix `N` which is identical to `M` except in the lower triangle (i.e., all the cells below the diagonal). The lower triangle should be filled with zeros.

[20]:
```
np.triu?
```

[63]:
```
M = np.round(np.random.rand(5,5),2)
print("M=\n", M)

# your code here:
N = np.triu(M)
print("N=\n", N)
```

```
M=
 [[0.84 0.48 0.33 0.63 0.45]
 [0.37 0.27 0.56 0.19 0.25]
 [0.19 0.94 0.78 0.31 0.23]
 [0.92 0.43 0.87 0.83 0.92]
 [0.52 0.19 0.93 0.24 0.2 ]]
N=
 [[0.84 0.48 0.33 0.63 0.45]
 [0.   0.27 0.56 0.19 0.25]
 [0.   0.   0.78 0.31 0.23]
 [0.   0.   0.   0.83 0.92]
 [0.   0.   0.   0.   0.2 ]]
```

Using the code provided above for generating the matrix `M`, try to figure out how to create a random matrix with 13 rows and 3 columns.

[62]:
```
# your code here:
M = np.round(np.random.rand(13,3),2)
print("M=\n", M)
```

```
M=
 [[0.77 0.66 0.62]
 [0.41 0.79 0.84]
 [0.52 0.87 0.36]
 [0.96 0.79 0.37]
 [0.02 0.44 0.44]
 [0.97 0.11 0.75]
 [0.85 0.3  0.66]
 [0.69 0.58 0.23]
 [0.69 0.08 0.29]
 [0.67 0.12 0.15]
 [0.85 0.35 0.24]
 [0.9  0.7  0.76]
 [0.92 0.74 0.42]]
```

1.1.10 j. Indexing and slicing arrays

Here is how to call an element of a 2D array by its location (i.e., its row index and column index):

[15]:
```
M[3,2]
```

[15]:
```
0.06872915146087366
```

[16]:
```
# test your intuition: what would you expect this code to return?
M[3:,2]
```

[16]:
```
array([0.06872915, 0.30423592, 0.12630904, 0.09834951, 0.15800365,
       0.85009507, 0.38711735, 0.28936366, 0.79368062, 0.26313639])
```

1.1.11 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 list, or for a matrix column (essentially, a list), or for the entire matrix. In other words, we may want to formulate a condition -- think of it as a *test* -- and run a computation that returns `True` or `False` depending on whether the test is passed or failed by a particular value in a data structure.

For example, our test may be something like "the value is greater than 0.5", and we would like to know if this is true or false for each of the values in a list. Here's how to compute a list of values that the condition test takes for each value of a given list:

[17]:
```
g = np.random.rand(1, 20) #first, create the list
print(g)
```

```
[[0.70457197 0.20457304 0.77776234 0.75383961 0.12297522 0.05138262
  0.6515948  0.50604353 0.03568077 0.9716883  0.32744666 0.35237655
  0.72616747 0.6162378  0.67676908 0.32181544 0.19082109 0.79328336
  0.34454492 0.63515499]]
```

[18]:
```
is_greater = g>0.5
print(is_greater)
```

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

[19]:
```
# Let's print the matrix M again so we can glance at it for our next exercise
print(M)
```

```
[[0.67699214 0.95386832 0.62530506]
 [0.87569665 0.85628582 0.45905964]
 [0.30314332 0.29252231 0.4159944 ]
 [0.03178337 0.34583999 0.06872915]
 [0.24519552 0.43008126 0.30423592]
 [0.4777186  0.60788409 0.12630904]
 [0.4026871  0.97812061 0.09834951]
 [0.91288559 0.3658346  0.15800365]
 [0.73199254 0.18089668 0.85009507]
 [0.22915522 0.92309107 0.38711735]
 [0.70232829 0.23209092 0.28936366]
 [0.16956249 0.82268691 0.79368062]
 [0.83408869 0.03628384 0.26313639]]
```

[20]:
```
# What would you expect to see once you run the code below?
c_is_greater = M[:,1]>0.5
c_is_greater
```

[20]:
```
array([ True,  True, False, False, False,  True,  True, False, False,
        True, False,  True, False])
```

1.1.12 L . NumPy Functions `np.any()`, `np.unique()`

We can use `np.any()` to determine if there is any entry in column 1 that is smaller than 0.1:

[21]:
```
c_is_smaller = M[:,1]<0.1
np.any(c_is_smaller)
```

[21]:
```
True
```

A small challenge: You have birthday data for a cohort of 100 people all born in 1990. Given the one-dimensional array of birthdays `random_bdays` generated below, figure out if there exists a pair of people who share a birthday.

Tip: you may find the function `np.unique()` useful. Feel free to read up on it by running the command `np.unique?` in a new cell.

[56]:
```
# do not edit this code:
random_nums = np.random.choice(365, size = 100)
random_bdays = np.datetime64('1990-01-01') + random_nums

## your code here:
duplicates_exist = len(random_bdays)!=len(np.unique(random_bdays))
print(duplicates_exist)
```

```
True
```

1.2 Part 2. 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 from scratch: from a *list of lists*, and 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.

[24]:
```
my_list = [['+1', '(929)-000-0000'], ['+34', '(917)-000-0000'], ['+7',␣
 ↪'(470)-000-0000']]
df = pd.DataFrame(my_list, columns = ['country_code', 'phone'])
df
```

[24]:
```
  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.

[25]:
```
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
```

[25]:
```
  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. In the code cell below, we add a new column of values to DataFrame `df`. Run the cell and inspect the DataFrame to see the new column that was added.

[26]:
```
df['grade']= ['A','B','A']
df
```

[26]:
```
  country_code           phone grade
0           +1  (929)-000-0000     A
1          +34  (917)-000-0000     B
2           +7  (470)-000-0000     A
```

1.2.3 c. Sorting the DataFrame by values in a specific column: `df.sort_values()`

[27]:
```
df = df.sort_values(['grade'])
df
```

[27]:
```
  country_code            phone grade
0           +1   (929)-000-0000     A
2           +7   (470)-000-0000     A
1          +34   (917)-000-0000     B
```

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

In real life settings, you will often need to combine separate sets of related data. 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.

[28]:
```
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
```

[28]:
```
  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 the second DataFrame to the first one. It saves the newly formed DataFrame to variable `df_concat`. Note that the `pd.concat()` function will not change the values in the original DataFrames.

[29]:
```
df_concat = pd.concat([df,df2])
df_concat
```

[29]:
```
  country_code            phone grade country
0           +1   (929)-000-0000     A     NaN
2           +7   (470)-000-0000     A     NaN
1          +34   (917)-000-0000     B     NaN
0          NaN   (874)-444-0000     B     +32
1          NaN   (313)-003-1000    B+     +81
2          NaN   (990)-006-0660     A     +11
```

Notice that the new DataFrame contains redundancy caused by the different spelling of the column names that contain the country code in the two original DataFrames. This can be easily fixed. The code cell below changes the name of the column in DataFrame `df2` to be consistent with the name of the column in DataFrame `df`.

[30]:
```
df2 = df2.rename(columns={'country':'country_code'})
df2
```

[30]:
```
  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_concat`. Run the cell and inpect the results.

[32]:
```
## your code here
df_concat = pd.concat([df,df2])
df_concat
```

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

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:

[33]:
```
df_concat = pd.concat([df,df2]).reset_index()
df_concat
```

[33]:
```
   index country_code           phone grade
0      0           +1  (929)-000-0000     A
1      2           +7  (470)-000-0000     A
2      1          +34  (917)-000-0000     B
3      0          +32  (874)-444-0000     B
4      1          +81  (313)-003-1000    B+
5      2          +11  (990)-006-0660     A
```

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

[34]:
```
df2
```

[34]:
```
  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`.

[35]:
```
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
```

[35]:
```
  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 next cell merges both DataFrames and applies the change to DataFrame `df2`. Feel free to consult the definition of the Pandas `merge()` method to better undertsand how `merge()` works.

[36]:
```
df2 = df2.merge(df3, on = 'phone')
df2
```

[36]:
```
  country_code_x grade_x           phone country_code_y grade_y  n_credits
0            +32       B  (874)-444-0000            +32       B         12
1            +11       A  (990)-006-0660            +11       A          9
```

1.2.5 e. Loading a dataset: `pd.read_csv()`

We are now well equipped to deal with a real dataset!

For the next few exercises, our dataset will contain information about New York City listings on the Airbnb platform. Note: the cell below may generate a warning. Ignore the warning.

[37]:
```
import os
filename = os.path.join(os.getcwd(), "labs_data", "airbnb_lab1.csv.gz") # path␣
 ↪to file and file name
data = 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)
```

[38]:
```
data.shape
```

[38]:
```
(38277, 63)
```

First, get a peek at the data:

[39]:
```
data.head()
```

[39]:
```
     id       scrape_id last_scraped  host_id  host_since  host_response_time  \
0  2595  20211204143024   2021-12-05     2845  2008-09-09          within a day
1  3831  20211204143024   2021-12-05     4869  2008-12-07  a few days or more
2  5121  20211204143024   2021-12-05     7356  2009-02-03      within an hour
3  5136  20211204143024   2021-12-05     7378  2009-02-03          within a day
4  5178  20211204143024   2021-12-05     8967  2009-03-03          within a day

  host_response_rate host_acceptance_rate host_is_superhost  \
0                80%                  17%                 f
1                 9%                  69%                 f
2               100%                 100%                 f
3               100%                  25%                 f
4               100%                 100%                 f

   host_neighbourhood  ...  review_scores_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]
```

That's a lot of columns, and the layout is a little difficult to read. Let us retrieve just the list of column names, so we can read it and get a feeling for what kind of information is presented in the dataset.

1.2.6 f. Get column names: `df.columns`

[40]:
```python
list(data.columns)
```

[40]:
```
['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.

1.2.7 g. Summary statistics of the DataFrame: `df.describe()`

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

[41]:
```
data.describe(include='all')
```

[41]:
```
                  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   2021-12-05           NaN  2019-10-29
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]
```

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.

1.2.8 h. Filtering the data: `df[ < condition > ]`

We need to filter the entries that are in Brooklyn. To do this, we need to know what is the exact way that Manhattan listings are spelled and entered in the data. Let's print all of the unique values of the 'neighborhood' column:

```
[42]: data['neighbourhood'].unique()
```

```
[42]: 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 `neighborhood` 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 data file, however, 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`:

[43]:
```
data['neighbourhood_cleansed'].unique()
```

[43]:
```
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)
```

[44]:
```
data['neighbourhood_group_cleansed'].unique()
```

[44]:
```
array(['Manhattan', 'Brooklyn', 'Queens', 'Staten Island', 'Bronx'],
      dtype=object)
```

Great, this last one is what we want! Let's filter out all data entries that pertain to Brooklyn listings:

[45]:
```
bk = data[data['neighbourhood_group_cleansed'] == 'Brooklyn']
bk.shape
```

[45]:
```
(14716, 63)
```

Tip: to better understand what happened above, you are encouraged to insert a new code cell below and copy *just the condition* of the filter that we used on the `data` object above: `data['neighbourhood_group_cleansed'] == 'Brooklyn'`. Run the new 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 `data[ < our Boolean series > ]`, i.e `data['neighbourhood_group_cleansed'] == 'Brooklyn']` , we are telling Pandas to keep the values in the DataFrame `data` only with those indices for which the condition evaluated to `True`.

1.2.9 i. 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:

[46]:
```
np.mean(bk['availability_365'])
```

[46]:
```
118.7693666757271
```

1.2.10 j. 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 `groupby` method from the Pandas package:

[47]:
```
nbhd_reviews = data.groupby('neighbourhood_cleansed')['number_of_reviews'].sum()
nbhd_reviews.head()
```

[47]:
```
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:

[48]:
```
nbhd_reviews = nbhd_reviews.sort_values(ascending = False)
nbhd_reviews.head(5)
```

[48]:
```
neighbourhood_cleansed
Bedford-Stuyvesant    88133
Williamsburg          55122
Harlem                54824
Bushwick              34776
Hell's Kitchen        31308
Name: number_of_reviews, dtype: int64
```

Success! While we re at it, what are the least reviewed neighborhoods?

[49]:
```
nbhd_reviews.tail(5)
```

[49]:
```
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:

[50]:
```
data.groupby('neighbourhood_cleansed')['number_of_reviews'].sum().
 ↪sort_values(ascending = False).head(5)
```

[50]:
```
neighbourhood_cleansed
Bedford-Stuyvesant    88133
Williamsburg          55122
Harlem                54824
Bushwick              34776
Hell's Kitchen        31308
Name: number_of_reviews, dtype: int64
```

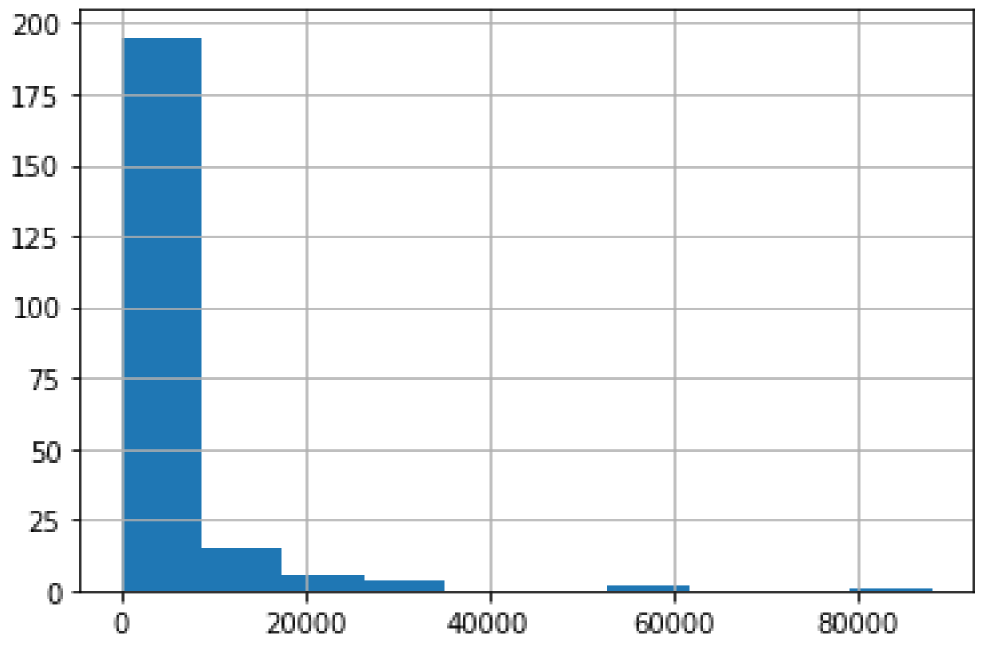
This way we don't store objects that we won't need.

1.2.11 Bonus: easy histogram plotting with Matplotlib: `plt.hist()`

As a final touch, run the cell below to instantly visualize the density of (average!) values of review numbers across all neighbourhoods:

[51]:
```
%matplotlib inline
nbhd_reviews.hist()
```

[51]:
```
<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.

[ ]: