Mid-term

GRA4157

4th October 2024

General information: You are asked to solve the problems below programatically. Your solution to a problem should still be valid if numbers in the problem text were replaced by other numbers, or strings were replaced by other strings.

1 Dictionaries and lists

Consider the list of integers (int).

```
m = [5, 3, -10, 30, 110, -7, 45, 11, 12, -15]
```

a) Write a function that takes such a list as input argument, and returns a sorted list from lowest to highest number. (built-in sorted-function not allowed).

b) What will be the outputs of the following calls?

```
print(m[2:3]+m[3:6])
print(m[2]+m[4])
print(2*m[8])
print(2*m[7:])
```

c) Consider the dictionaries

```
temps = {'Oslo': 2.5, 'London': 12.3, 'Paris': 11.0, 'Berlin': 9.0,
         'Los Angeles': 19.5, 'Cairo': 22.5}
rainfall = {'Oslo' : 0.0, 'London': 2.2, 'Paris': 5.5, 'Berlin': 0}
```

Write a python function that takes two such dictionaries as input and returns a nested dictionary of temperature and rainfall for each city. Only include cities that are present in both dictionaries. I.e. output should be on the following format

```
results = {'Oslo': {'temperature': 2.5, 'rainfall': 0.0}} ...
```

2 Reading from and writing to file

Consider the file input.txt:

```
This is the first line of the file
This is the second line of the file
Below comes the interesting part of the file:
10 20 30
20 30 1
2.2 125 6.45
0.1 20 3.14
```

The numbers in this file could be organized as a 4x3 matrix.

a) Read the file using pure python and for each row, put all numbers found in the row in a list. If no number is found on a line, no action is required. Call the lists row1, row2 and row3.

As an example, row 1 should end up as [10,20,30]

b) Read the file using pure python and create a list that is consists of all the numbers from the last row first, then all numbers from the next to last row and so on. The list will then look like this: [0.1, 20, 3.14, 2.2, ..., 10, 20, 30]

c) Write a program that writes the new/reversed list from b) to a file "out.csv". The file should have the header "a,b,c" and then have three numbers per row. The numbers in each row of the matrix should be separated by commas. (The first row after the header in the file will thus be: 0.1,20,3.14)

3 Vectorized computations

Consider the following code:

```
x = [0.1*i for i in range(101)]
y = []
def f(x):
    return 5*x + 2

for xi in x:
    y.append(f(xi))
```

a) Rewrite the code such that y is created via vectorized computations (without the need for a for loop). You may use numpy.

b) You have been hired as a consultant to speed up computations for Price-control AS. Assume that you have already read 100 million high resolution price and supply values for a given commodity as a pandas Series named supply. This Series tells us how much a producer would produce of a good for a given price. You can also assume that you have access to the price (supply.price) and the supply volume (supply.values) as numpy arrays if you prefer to work with those. A Series with price-supply pairs may look as follows:

```
0.00        0.0
10.00      20.0
30.00      20.0
40.00      20.0
41.33      20.0
45.00      21.5
50.00      25.0
55.00     100.0
56.00     100.0
57.00     100.0
60.00     105.0
4000.00   105.0
Name: supply, dtype: float64
```

Here, the left column is price and the right column is volume. In this case, if the price is 0, the producer will not produce. If the price is 10, the producer will produce 20 of the good. In between prices, linear interpolation applies. Therefore if the price is 5, the producer will committ to produce 10 of the given good. Following this logic, there are unnecessary entries in the Series (unnecessary price-supply pairs) above. For instance, there is no need to have the price-supply pair price:56, production:100, as this would already be implied by the linear interpolation rule.

Your task here is to remove unnecessary price-supply pairs. It is important that the curve itself does not change. (E.g. if you plot the new curve with straight lines, it should appear exactly as the previous curve.). You can provide the answer as two arrays, or a pandas Series. To get full score, the cleaning of data should use vectorized computations (i.e. no for-loops), but you will get some credit if you manage to solve the problem with for-loop(s).

4 Pandas

Assume we have the csv-file city-bikes.csv:

```
Start station,Start time,End station,End time
298,2016-08-01 05:59:55 +0200,233,2016-08-01 07:47:05 +0200
248,2016-08-01 06:00:09 +0200,253,2016-08-01 06:09:05 +0200
290,2016-08-01 06:00:14 +0200,262,2016-08-01 10:02:59 +0200
277,2016-08-01 06:00:26 +0200,301,2016-08-01 06:10:01 +0200
```

a) Write a program that 1) reads the file into a pandas DataFrame and 2) extracts values from the columns into four pandas Series, st0, t0, st1, t1.

b) There is an issue with the date and end station id. When looking at the dtypes we see:

```
Start station      int64
Start time        object
End station      float64
End time          object
dtype: object
```

Fix the object types in the DataFrame such that end station is of type int and start and end-time is of type datetime64[ns, UTC]. The pd.to_datetime method is attached at the end of the document.

c) You can now assume that you have the DataFrame with correct types. Create a new column "trip_duration" that list the duration time of each trip.

d) Create a new DataFrame consisting only of the top 10% of trips sorted by trip duration. Then, compute the average length of trips included in the new DataFrame.

API reference › General functions › pandas.to_datetime

On this page
to_datetime()

Show Source

pandas.to_datetime

pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=False, format=None, exact=_NoDefault.no_default, unit=None, infer_datetime_format=_NoDefault.no_default, origin='unix', cache=True) [source]

Convert argument to datetime.

This function converts a scalar, array-like, Series or DataFrame/dict-like to a pandas datetime object.

Parameters:

arg : *int, float, str, datetime, list, tuple, 1-d array, Series, DataFrame/dict-like*

The object to convert to a datetime. If a DataFrame is provided, the method expects minimally the following columns: "year", "month", "day". The column "year" must be specified in 4-digit format.

errors : *{'ignore', 'raise', 'coerce'}, default 'raise'*

- If 'raise', then invalid parsing will raise an exception.
- If 'coerce', then invalid parsing will be set as NaT.
- If 'ignore', then invalid parsing will return the input.

dayfirst : *bool, default False*

Specify a date parse order if *arg* is str or is list-like. If True, parses dates with the day first, e.g. "10/11/12" is parsed as 2012-11-10.

Warning

dayfirst=True is not strict, but will prefer to parse with day first.

yearfirst : *bool, default False*

Specify a date parse order if *arg* is str or is list-like.

- If True parses dates with the year first, e.g. "10/11/12" is parsed as 2010-11-12.
- If both *dayfirst* and *yearfirst* are True, *yearfirst* is preceded (same as dateutil).

Warning

yearfirst=True is not strict, but will prefer to parse with year first.

utc : *bool, default False*

Control timezone-related parsing, localization and conversion.

- If True, the function *always* returns a timezone-aware UTC-localized Timestamp,

Figure 1: Attachment: pd.to_datetime