Mid-term

GRA4157

October 5th, 2023

1 Basic Python

a) Write a function named *reverse_list*. This function should accept a list as a parameter and return a new list where the elements are arranged in reverse order from the input list. Do not use built-in methods like *reverse()* or slicing methods.

b) Write a Python function named *list_to_dict*. This function should accept a list of tuples as its input parameter. Each tuple in the list will contain exactly two elements. Your task is to construct and return a dictionary using these tuples, where The first element of each tuple should be used as the key and the second element of each tuple should be assigned as the value.

c) Write a function named *max_value_key*. This function should accept a single parameter: a dictionary with keys and values. Your task is to return the key that is associated with the highest value present within the dictionary. If there are multiple keys with the highest value, return any one of them.

d) Consider the list of integers (int):

$$list\_of\_values = [1, 2, -5, 3, 12, 20, 45, 13, -10, 21]$$

What will be the outputs of the following calls?

```
print(list_of_values[1:3]+ list_of_values[4:6])
print(list_of_values[-1] + list_of_values[6])
print(list_of_values[8] * 2)
print(list_of_values[4::2])
```

2 Reading and Writing Files

Consider the input file:

```
val1,val2,val3
1,one,2,3
4,5,6,six
7,8,eight,9
```

a) Read the file using pure Python and, for each row, put all the numbers found in the row in a list. If no number is found on a line, no action is required. If something that cannot be converted to a number is found in a row, it should be skipped. The lists should be placed in a another list named *rows*.

b Read the file using pure Python and, for each column, put all the numbers found in the column in a list. If no number is found on a line, no action is required. If something that cannot be converted to a number is found in a row, it should be skipped. The lists should be placed in a another list named *columns*.

c) Write a function in pure Python that creates a 3×3 matrix with each value being ten times the corresponding value from the input file. The resulting matrix should be saved in a file named *out.csv*. The first line of this file should have the same header as the input file, and the values in each row should be separated by commas. You can use the *rows* or *columns* lists you defined earlier.

3 Numpy and Pandas

Consider the following two 3×3 matricies: $a = \begin{bmatrix} 2 & 5 & 1 \\ 11 & 6 & 8 \\ 3 & 34 & 5 \end{bmatrix}$ $b = \begin{bmatrix} 2 & 9 & 7 \\ 4 & 6 & 5 \\ -6 & 27 & 6 \end{bmatrix}$

a) Create two numpy arrays, each representing the aforementioned matrices. Assign them to the variables matrix_a and matrix_b, respectively.

b) Using the two matrices defined in the previous problem, perform an element-wise addition and element-wise subtraction of the two matrices and print the results.

c) Explain what is wrong with the following code, and modify it so that it works and allows for vectorized computations of f (e.g. uses numpy calculations instead of python for loops).

```
def f(a, b):
    if a == b:
        return a - b
    return a + b

print(f(matrix_a, matrix_b))
```

4 Web scraping

In the following, we will consider the wikipedia page with a list of the highest-grossing movies of all time (https://en.wikipedia.org/wiki/List_of_highest-grossing_films). Table number 1 (if we start counting at 1) in the Wikipedia page has the following structure:

Highest-grossing films[12]

Rank	Peak	Title	Worldwide gross	Year	Reference(s)
1	1	*Avatar*	$2,923,706,026	2009	[# 1][# 2]
2	1	*Avengers: Endgame*	$2,797,501,328	2019	[# 3][# 4]
3	3	*Avatar: The Way of Water*	$2,320,250,281	2022	[# 5][# 6]
4	1	*Titanic*	T$2,257,844,554	1997	[# 7][# 8]
5	3	*Star Wars: The Force Awakens*	$2,068,223,624	2015	[# 9][# 10]
6	4	*Avengers: Infinity War*	$2,048,359,754	2018	[# 11][# 12]
7	6	*Spider-Man: No Way Home*	$1,921,847,111	2021	[# 13][# 14]
8	3	*Jurassic World*	$1,671,537,444	2015	[# 15][# 16]
9	7	*The Lion King*	$1,656,943,394	2019	[# 17][# 4]
10	3	*The Avengers*	$1,518,815,515	2012	[# 18][# 19]

a) Read the web page (HTML) into a list of DataFrames with pandas, and extract the table above as a DataFrame in a new variable named *movies*.

b) Using the sort_values function, organize the dataset in descending order based on the 'Year' column. Extract each unique combination of 'Title' and 'Year', ensuring that each 'Year' appears only once. If multiple movies are on the list for one year, the highest grossing movie should be printed.

c) Assume you are still working with the same DataFrame from question 4a and 4b. The DataFrame you are working with contains a column labeled "Worldwide gross", which represents the total earnings of various movies. Unfortunately, this column also contains some non-numeric characters, making data analysis slightly more complex.

Write a Python script to remove all non-numeric characters from the "Worldwide Gross" column and convert it to an integer data type. Plot a scatter plot with they "Year" column on the x-axis and "Wordwide gross" on the y-axis.

pandas.DataFrame.sort_values

`DataFrame.sort_values(by, *, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last', ignore_index=False, key=None)`

Sort by the values along either axis. [source]

Parameters:

by : *str or list of str*

Name or list of names to sort by.

- if *axis* is 0 or *'index'* then *by* may contain index levels and/or column labels.
- if *axis* is 1 or *'columns'* then *by* may contain column levels and/or index labels.

axis : *"{0 or 'index', 1 or 'columns'}", default 0*

Axis to be sorted.

ascending : *bool or list of bool, default True*

Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.

inplace : *bool, default False*

If True, perform operation in-place.

kind : *{'quicksort', 'mergesort', 'heapsort', 'stable'}, default 'quicksort'*

Choice of sorting algorithm. See also `numpy.sort()` for more information. *mergesort* and *stable* are the only stable algorithms. For DataFrames, this option is only applied when sorting on a single column or label.

na_position : *{'first', 'last'}, default 'last'*

Puts NaNs at the beginning if *first*; *last* puts NaNs at the end.

ignore_index : *bool, default False*

If True, the resulting axis will be labeled 0, 1, …, n - 1.

key : *callable, optional*

Apply the key function to the values before sorting. This is similar to the *key* argument in the builtin `sorted()` function, with the notable difference that this *key* function should be *vectorized*. It should expect a `Series` and return a Series with the same shape as the input. It will be applied to each column in *by* independently.

Returns:

DataFrame or None

DataFrame with sorted values or None if `inplace=True`.