lab-assignment1

June 24, 2024

1 Lab Assignment 1

Qais Youssef
QMY6CV

1.0.1 Problem 0

[8]:
```
import numpy as np
import pandas as pd
import os
import math
```

1.0.2 Problem 1

Stack Overflow Post

After reviewing the post, I found about 7 unique solutions for renaming columns in Pandas. The problem with using google and stack overflow as the first source includes: 1- Reliance on outdated methods and techniques. 2- Creating more confusion: The variety of solutions can be overwhelming and lead to incorrect implementations.

1.0.3 Problem 2

[9]:
```
print("Docstring for numpy.log:\n")
print(np.log.__doc__)
print("\nDocstring for math.log:\n")
print(math.log.__doc__)
```

```
Docstring for numpy.log:

log(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None,
subok=True[, signature, extobj])

Natural logarithm, element-wise.

The natural logarithm `log` is the inverse of the exponential function,
so that `log(exp(x)) = x`. The natural logarithm is logarithm in base
`e`.
```

```
Parameters
----------
x : array_like
    Input value.
out : ndarray, None, or tuple of ndarray and None, optional
    A location into which the result is stored. If provided, it must have
    a shape that the inputs broadcast to. If not provided or None,
    a freshly-allocated array is returned. A tuple (possible only as a
    keyword argument) must have length equal to the number of outputs.
where : array_like, optional
    This condition is broadcast over the input. At locations where the
    condition is True, the `out` array will be set to the ufunc result.
    Elsewhere, the `out` array will retain its original value.
    Note that if an uninitialized `out` array is created via the default
    ``out=None``, locations within it where the condition is False will
    remain uninitialized.
**kwargs
    For other keyword-only arguments, see the
    :ref:`ufunc docs <ufuncs.kwargs>`.

Returns
-------
y : ndarray
    The natural logarithm of `x`, element-wise.
    This is a scalar if `x` is a scalar.

See Also
--------
log10, log2, log1p, emath.log

Notes
-----
Logarithm is a multivalued function: for each `x` there is an infinite
number of `z` such that `exp(z) = x`. The convention is to return the
`z` whose imaginary part lies in `(-pi, pi]`.

For real-valued input data types, `log` always returns real output. For
each value that cannot be expressed as a real number or infinity, it
yields ``nan`` and sets the `invalid` floating point error flag.

For complex-valued input, `log` is a complex analytical function that
has a branch cut `[-inf, 0]` and is continuous from above on it. `log`
handles the floating-point negative zero as an infinitesimal negative
number, conforming to the C99 standard.

In the cases where the input has a negative real part and a very small
negative complex part (approaching 0), the result is so close to `-pi`
that it evaluates to exactly `-pi`.
```

```
References
----------
.. [1] M. Abramowitz and I.A. Stegun, "Handbook of Mathematical Functions",
       10th printing, 1964, pp. 67.
       https://personal.math.ubc.ca/~cbm/aands/page_67.htm
.. [2] Wikipedia, "Logarithm". https://en.wikipedia.org/wiki/Logarithm

Examples
--------
>>> np.log([1, np.e, np.e**2, 0])
array([  0.,   1.,   2., -Inf])

Docstring for math.log:

log(x, [base=math.e])
Return the logarithm of x to the given base.

If the base not specified, returns the natural logarithm (base e) of x.
```

The numpy.log function computes the natural logarithm (base e) of each element in an array. The math.log function computes the natural logarithm (base e) of a single number. Neither function directly supports logarithms with an arbitrary base.

Both functions compute the natural logarithm by default, we must use the change-of-base formula

[10]:
```
number = 7

log_numpy = np.log(number) / np.log(3)

# Calculate log base 3 using math
log_math = math.log(number) / math.log(3)

log_math, log_numpy
```

[10]:
```
(1.7712437491614221, 1.7712437491614221)
```

1.0.4 Problem 3

1.0.5 Problem 4

Link (https://stackoverflow.com/questions/2612802/how-do-i-clone-a-list-so-that-it-doesnt-change-unexpectedly-after-assignmentx) **Description:** In this question, the answerer shows passive toxic behavior in several comments. For example, the user "Andrew" commented, "new_list = my_list just assigns the name new_list to the object my_list refers to," which can be perceived as condescending, stating the obvious without providing a solution. The user "Bharel" also commented, "See the Python FAQ," without providing a direct answer or link, which can be seen as dismissive. These types of responses can discourage the questioner and implies that the question is trivial and the questioner should have already known the answer.

1.0.6 Problem 5

Link (https://stackoverflow.com/questions/78660102/how-to-iterate-through-a-python-list-to-get-each-additional-list-element)

Description: The user self-sabotages by asking the question in a way that frustrates the community, which is apparent by the 3 downvotes they got in a short period. The user asks a basic and commonly addressed question without showing an effort to research or try to solve the problem themselves. This can be seen as lazy or inconsiderate to the community members who are taking their time to help.

1.0.7 Problem 6

link (https://stackoverflow.com/questions/15112125/how-to-test-multiple-variables-for-equality-against-a-single-value) supplement (https://stackoverflow.com/questions/10272898/multiple-if-conditions-in-a-python-list-comprehension) **Search Terms:** python if statement multiple conditions in list.

[11]:
```
def is_avenger(name):
    og_avengers = ["Hulk", "Captain America", "Iron Man", "Black Widow",␣
 ↪"Hawkeye", "Thor"]
    if name in og_avengers:
        print(name + " is an original Avenger!")
    else:
        print(name + " is NOT an original Avenger.")

is_avenger("Black Widow")
is_avenger("Iron Man")
is_avenger("Hulk")

is_avenger("Spiderman")
is_avenger("Beyonce")
```

```
Black Widow is an original Avenger!
Iron Man is an original Avenger!
Hulk is an original Avenger!
Spiderman is NOT an original Avenger.
Beyonce is NOT an original Avenger.
```

Part B *Title for post: How to Correctly Check if a String is One of Multiple Specific Values in Python.*

Description: I'm working on a Python function to check if a given name is one of the original six Avengers: Hulk, Captain America, Iron Man, Black Widow, Hawkeye, and Thor. If the name is one of these, the function should print that the name is an original Avenger; otherwise, it should print that the name is not an original Avenger.

Part C: Minimal Working Example

[12]:
```
def is_avenger(name):
    if name=="Hulk" or "Captain America" or "Iron Man" or "Black Widow" or␣
 ↪"Hawkeye" or "Thor":
        print(name + " is an original Avenger!")
    else:
        print(name + " is NOT an original Avenger.")

# Test cases with original Avengers
is_avenger("Black Widow")
is_avenger("Iron Man")
is_avenger("Hulk")
# Test cases with non-original Avengers
is_avenger("Spiderman")
is_avenger("Beyonce")
```

```
Black Widow is an original Avenger!
Iron Man is an original Avenger!
Hulk is an original Avenger!
Spiderman is an original Avenger!
Beyonce is an original Avenger!
```

1.0.8 Problem 7

I have a dataset 'jobs_in_data.csv' with job titles and salaries. I need to filter the data for the years 2022 and 2023 and for specific job titles: Data Analyst, Data Engineer, Data Scientist, and Machine Learning Engineer. provide the code

Included dataset file and path on my device.

[13]:
```
file_path = r'C:\Users\qaism\OneDrive - University of␣
 ↪Virginia\Documents\GitHub\MSDS\DS 6001\jobs_in_data.csv'
jobs = pd.read_csv(file_path)
# Filter the dataset for the years 2022 and 2023
filtered_jobs = jobs[jobs['work_year'].isin([2022, 2023])]

# Filter the dataset for the specified job titles
job_titles = ['Data Analyst', 'Data Engineer', 'Data Scientist', 'Machine␣
 ↪Learning Engineer']
filtered_jobs = filtered_jobs[filtered_jobs['job_title'].isin(job_titles)]
filtered_jobs.head()
```

[13]:

	work_year	job_title	job_category	salary_currency	\
3	2023	Data Scientist	Data Science and Research	USD	
4	2023	Data Scientist	Data Science and Research	USD	
5	2023	Data Scientist	Data Science and Research	USD	
6	2023	Data Scientist	Data Science and Research	USD	
9	2023	Data Engineer	Data Engineering	USD	

```
   salary  salary_in_usd employee_residence experience_level employment_type  \
3  212000         212000      United States           Senior       Full-time
4   93300          93300      United States           Senior       Full-time
5  130000         130000      United States           Senior       Full-time
6  100000         100000      United States           Senior       Full-time
9  210000         210000      United States        Executive       Full-time

  work_setting company_location company_size
3    In-person    United States            M
4    In-person    United States            M
5       Remote    United States            M
6       Remote    United States            M
9       Remote    United States            M
```

Now, I need to group the filtered data by 'work_year' and 'job_title', and then calculate the average salary for each combination. I also need to pivot the table so that the years are columns and the job titles are rows. The table should show the average salaries for each job title and year.

[14]:
```
# Group by 'work_year' and 'job_title', then calculate the average salary_in_usd
average_salaries = filtered_jobs.groupby(['work_year',␣
 ↪'job_title'])['salary_in_usd'].mean().round().reset_index()

# Pivot the table to have years as columns and job titles as rows
pivot_table = average_salaries.pivot(index='job_title', columns='work_year',␣
 ↪values='salary_in_usd').reset_index()

# Rename the columns for clarity
pivot_table.columns = ['Job Title', '2022', '2023']

# Display the resulting pivot table
pivot_table
```

[14]:

	Job Title	2022	2023
0	Data Analyst	108658.0	110988.0
1	Data Engineer	139803.0	149945.0
2	Data Scientist	138529.0	163714.0
3	Machine Learning Engineer	151775.0	191026.0

Part B Using ChatGPT for data wrangling can be highly beneficial in scenarios where quick guidance and snippets are needed to handle common manipulation tasks. It excels in helping with syntax, explaining functions, and offering step-by-step instructions. This can speed up the learning process and improve productivity for anyone who is familiar with the basics or needs a refresher on correct implementation. However, relying on ChatGPT may not be practical for context-specific data wrangling that requires a deeper understanding of the nuances in the data. ChatGPT's inability to handle real-time data and potential limitations in understanding complex context highlight the need for a balanced approach, combining LLM assistance with human expertise.