Lab Assignment 1: How to Get Yourself Unstuck

DS 6001: Practice and Application of Data Science

Instructions

Please answer the following questions as completely as possible using text, code, and the results of code as needed. Format your answers in a Jupyter notebook. To receive full credit, make sure you address every part of the problem, and make sure your document is formatted in a clean and professional way.

Problem 0

Import the following libraries:

In [1]:
```
import numpy as np
import pandas as pd
import os
import math
```

Problem 1

Python is open-source, and that's beautiful: it means that Python is maintained by a world-wide community of volunteers, that Python develops at the same rate as advancements in science, and that Python is completely free of charge. But one downside of being open-source is that different people design many alternative ways to perform the same task in Python.

Read the following Stack Overflow post: https://stackoverflow.com/questions/11346283/renaming-columns-in-pandas/46912050. The question is simply how to rename the columns of a dataframe using Pandas. Count how many unique different solutions were proposed, and write this number in your lab report. (Hint: the number of solutions is not the number of answers to the posted question.)

Remember: your goal as a data scientist needs to be to process/clean/wrangle/manage data as quickly as possible while still doing it correctly. A big part of that job is knowing how to seek help to find the right answer quickly. Given the number of proposed solutions on this Stack Overflow page, what's the problem with developing a habit of using Google and Stack Overflow as your first source for seeking help? (2 points)

The number of unique different solutions in the Stack Overflow post is roughly 40.

This high number of potential solutions means that looking at Stack Overflow or Google first makes it difficult to find the best solution - or even a correct one - without first sifting through all of the answers.

Problem 2

There are several functions implemented in Python to calculate a logarithm. Both the `numpy` and `math` libraries have a `log()` function. Your task in this problem is to calculate log$_3(7)$ directly (without using the change-of-base formula). Note that this particular log has a base of 3, which is unusual. For this problem:

- Write code to display the docstrings for each function.
- Read the docstrings and explain, in words in your lab report, whether it is possible to use each function to calculate log$_3(7)$ or not. Why did you come to this conclusion?

According to the docstrings shown below, np.log() can only calculate the natural (base *e*) log and so cannot calculate log$_3(7)$, while math.log() has base as parameter and therefore can calculate log$_3(7)$.

If possible, use one or both functions to calculate log$_3(7)$ and display the output. (2 points)

In [2]:
```
?np.log
```

```
Call signature:  np.log(*args, **kwargs)
Type:            ufunc
String form:     <ufunc 'log'>
File:            /opt/miniconda3/lib/python3.12/site-packages/numpy/__init_
_.py
Docstring:
log(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=Non
e, subok=True[, signature])

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])
Class docstring:
Functions that operate element by element on whole arrays.

To see the documentation for a specific ufunc, use `info`.  For
example, ``np.info(np.sin)``.  Because ufuncs are written in C
(for speed) and linked into Python with NumPy's ufunc facility,
Python's help() function finds this page whenever help() is called
on a ufunc.

A detailed explanation of ufuncs can be found in the docs for :ref:`ufuncs`.

**Calling ufuncs:** ``op(*x[, out], where=True, **kwargs)``

Apply `op` to the arguments `*x` elementwise, broadcasting the arguments.

The broadcasting rules are:

* Dimensions of length 1 may be prepended to either array.
* Arrays may be repeated along dimensions of length 1.

Parameters
----------
*x : array_like
    Input arrays.
out : ndarray, None, or tuple of ndarray and None, optional
    Alternate array object(s) in which to put the result; if provided, it
    must have a shape that the inputs broadcast to. A tuple of arrays
    (possible only as a keyword argument) must have length equal to the
    number of outputs; use None for uninitialized outputs to be
    allocated by the ufunc.
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.kwarg
s>`.

Returns
-------
r : ndarray or tuple of ndarray
    `r` will have the shape that the arrays in `x` broadcast to; if `out` is
    provided, it will be returned. If not, `r` will be allocated and
    may contain uninitialized values. If the function has more than one
    output, then the result will be a tuple of arrays.
```

In [3]:
```
?math.log
```

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

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

In [4]:
```
print(math.log(7, 3))
```

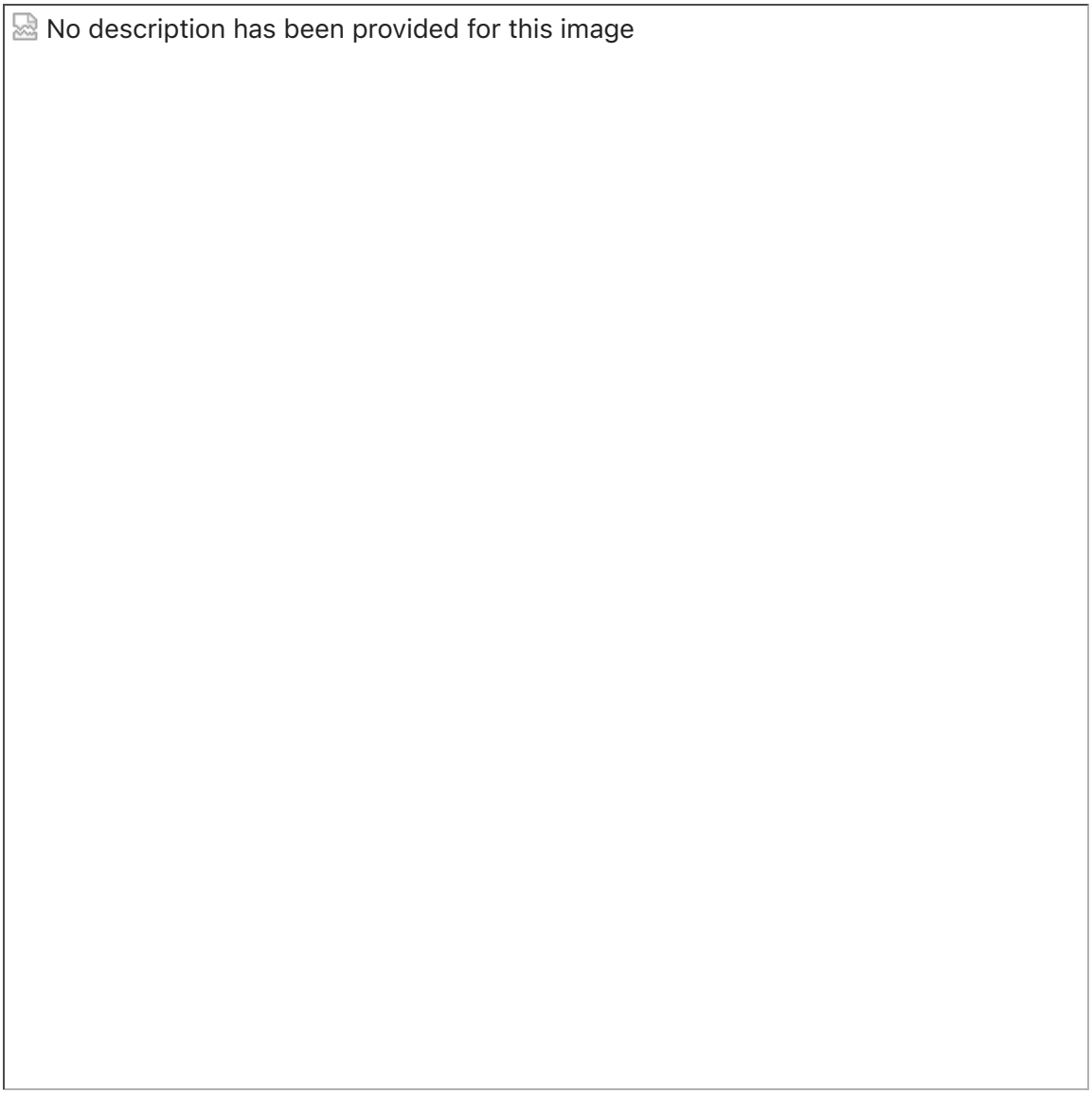
```
1.7712437491614221
```

Problem 3

Open a console window and place it next to your notebook in Jupyter labs. Load the kernel from the notebook into the console, then call up the docstring for the `pd.DataFrame` function. Take a screenshot and include it in your lab report. (To include a locally saved image named `screenshot.jpg`, for example, create a Markdown cell and paste

```
<img src="screenshot.jpg" width=600>
```

(2 points)



Note: for some reason, the pdf version of this notebook does not load this image, despite me copying the example text exactly (and it showing up fine in the notebook itself). However, I have fixed this by appending the image to the end of the file manually.

Problem 4

Search through the questions on Stack Overflow tagged as Python questions: https://stackoverflow.com/questions/tagged/python. Find a question in which an answerer exhibits passive toxic behavior as defined in this module's notebook. Provide a link, and describe what specific behavior leads you to identify this answer as toxic. (2 points)

In the question https://stackoverflow.com/questions/23186128/python-why-is-this-wrong, the answerer is unnecessarily sarcastic and passive aggressive in their response, especially the first paragraph. Their comments shame the question asker to the point where they feel they must apologize "for not being clear".

Problem 5

Search through the questions on Stack Overflow tagged as Python questions: https://stackoverflow.com/questions/tagged/python. Find a question in which a questioner self-sabotages by asking the question in a way that the community does not appreciate. Provide a link, and describe what the questioner did specifically to annoy the community of answerers. (2 points)

In the question https://stackoverflow.com/questions/52119760/what-is-wrong-with-my-code-which-deals-with-creating-python-dictionaries, in addition to asking a very basic question (which Stack Overflow users do not like answering), the questioner uses non-ASCII quotes in their code, which seems to confuse and frustrate the responders.

Problem 6

These days there are so many Marvel superheros, but only six superheros count as original Avengers: Hulk, Captain America, Iron Man, Black Widow, Hawkeye, and Thor. I wrote a function, `is_avenger()`, that takes a string as an input. The function looks to see if this string is the name of one of the original six Avengers. If so, it prints that the string is an original Avenger, and if not, it prints that the string is not an original Avenger. Here's the code for the function:

In [5]:
```
def is_avenger(name):
    if name=="Hulk" or name=="Captain America" or name=="Iron Man" or name==
        print(name  + "'s an original Avenger!")
    else:
        print(name + " is NOT an original Avenger.")
```

To test whether this function is working, I pass the names of some original Avengers to the function:

In [6]:
```
is_avenger("Black Widow")
```

```
Black Widow's an original Avenger!
```

In [7]:
```
is_avenger("Iron Man")
```

```
Iron Man's an original Avenger!
```

In [8]:
```
is_avenger("Hulk")
```

```
Hulk's an original Avenger!
```

Looks good! But next, I pass some other strings to the function

In [9]:
```
is_avenger("Spider-Man")
```

```
Spider-Man is NOT an original Avenger.
```

In [10]:
```
is_avenger("Beyonce")
```

```
Beyonce is NOT an original Avenger.
```

Beyonce is a hero, but she was too busy going on tour to be in the Avengers movie. Also, Spiderman definitely was NOT an original Avenger. It turns out that this function will display that any string we write here is an original Avenger, which is incorrect. To fix this function, let's turn to Stack Overflow.

Part a

The first step to solving a problem using Stack Overflow is to do a comprehensive search of available resources to try to solve the problem. There is a post on Stack Overflow that very specifically solves our problem. Do a Google search and find this post. In your lab report, write the link to this Stack Overflow page, and the search terms you entered into Google to find this page.

Then apply the solution on this Stack Overflow page to fix the `is_avenger()` function, and test the function to confirm that it works as we expect. (2 points)

Despite making numerous attempts at finding the referenced post with both general and specific search terms, including 'avengers only returns true site:stackoverflow.com', 'superhero names if statement error python site:stackoverflow.com', and '"is_avenger" python site:stackoverflow.com', and even searching in Stack Overflow directly, I was unable to actually find it. I have nevertheless applied a solution to the above function.

Part b

Suppose that no Stack Overflow posts yet existed to help us solve this problem. It would be time to consider writing a post ourselves. In your lab report, write a good title for this post. Do NOT copy the title to the posts you found for part a. (Hint: for details on how to write a good title see the slides or https://stackoverflow.com/help/how-to-ask) (2 points)

Why is my Python function's if statement only returning true?

Part c

One characteristic of a Stack Overflow post that is likely to get good responses is a minimal working example. A minimal working example is code with the following properties:

1. It can be executed on anyone's local machine without needing a data file or a hard-to-get package or module
2. It always produces the problematic output
3. It using as few lines of code as possible, and is written in the simplest way to write

that code

Write a minimal working example for this problem. (2 points)

In [11]:
```
def is_name(name):
    if name=="A" or "B":
        print(name  + " is here")
    else:
        print(name + " is not here")

is_name("C")
```

```
C is here
```

Problem 7

This problem will test your ability to use a chatbot based on a large-language model, such as ChatGPT, to do data wrangling. Please sign-up for a free account to use ChatGPT if you have not already done so, and log on to the chat interface website for ChatGPT.

Part a

The following data comes from a Kaggle project on Jobs and Salaries in Data Science, compiled by Lucas Galanti (though I don't see an attribution to the original data source, so please take these numbers with a grain of salt). Load the data by running this cell:

In [12]:
```
jobs = pd.read_csv('jobs_in_data.csv')
jobs
```

Out[12]:

	work_year	job_title	job_category	salary_currency	salary	salary_in_usd	en
0	2023	Data DevOps Engineer	Data Engineering	EUR	88000	95012	
1	2023	Data Architect	Data Architecture and Modeling	USD	186000	186000	
2	2023	Data Architect	Data Architecture and Modeling	USD	81800	81800	
3	2023	Data Scientist	Data Science and Research	USD	212000	212000	
4	2023	Data Scientist	Data Science and Research	USD	93300	93300	
...	...	...	...	...	...	...	
9350	2021	Data Specialist	Data Management and Strategy	USD	165000	165000	
9351	2020	Data Scientist	Data Science and Research	USD	412000	412000	
9352	2021	Principal Data Scientist	Data Science and Research	USD	151000	151000	
9353	2020	Data Scientist	Data Science and Research	USD	105000	105000	
9354	2020	Business Data Analyst	Data Analysis	USD	100000	100000	

9355 rows × 12 columns

Our goal is to manipulate the `jobs` dataframe to create a table with four rows: one for each of the job titles Data Analyst, Data Engineer, Data Scientist, and Machine Learning Engineer; and two columns: one for the year 2022 and one for 2023. Inside each cell should be the average salary (`salary_in_usd`) for that job title and year, rounded to the nearest dollar. The resulting table should look something like this:

	2022	2023
Data Analyst	108658	110988
Data Engineer	139803	149945
Data Scientist	138529	163714
Machine Learning Engineer	151775	191026

Your task is to use ChatGPT -- and ONLY chatGPT -- to generate Python code that uses `pandas` that can generate a dataframe that looks like the above table. For this problem, use markdown cells in your notebook to display both your prompts and ChatGPT's responses. You will almost certainly need to issue several follow-up prompts to get to an answer, and you should list all of your prompts and the responses in your answer.

A few points to keep in mind:

- You will receive better responses by following the guidelines listed here: https://jkropko.github.io/surfing-the-data-pipeline/ch1.html#method-5-using-a-large-language-model-a-chatbot-to-generate-and-debug-code
- You are more likely to get an answer that works by chunking the task into discrete steps. Some of the steps that are needed here are:
  - Keep only the rows from 2022 and 2023
  - Keep only the rows with job titles Data Analyst, Data Engineer, Data Scientist, and Machine Learning Engineer
  - Collapse the data by taking the average `salary_in_usd` within each remaining year and job title combination
  - Reshape the data by moving the years to the columns
  - Round the average salaries to the nearest dollar

Whether you state these specific steps in your prompts or not, and the order in which you state them if you do, is up to you.

One last note: remember you are trying to generate code to generate the average salary table, not the table itself. If the code that ChatGPT generates yields using a small sample of the data calculates incorrect averages, that's OK as long as the code works properly for the full dataframe. (3 points)

Prompt: Using the pandas library, load in the data found in the csv file on https://www.kaggle.com/code/lucasgalanti/jobs-in-data/input.

ChatGPT:

To load the data from the CSV file found on Kaggle into a pandas DataFrame, you can use the following steps:

1. Download the dataset: Visit the dataset page on Kaggle and download the CSV file. For example, you can find relevant datasets here, here, or here.

2. Load the CSV file using pandas: Use the read_csv function from pandas to load the CSV file into a DataFrame.

Here's a sample code to achieve this:

```
import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'path_to_your_downloaded_file.csv'

# Load the CSV file into a pandas DataFrame
data = pd.read_csv(file_path)

# Display the first few rows of the DataFrame
print(data.head())
```

Make sure to replace 'path_to_your_downloaded_file.csv' with the actual path to the downloaded CSV file on your machine. This code snippet will read the CSV file and print the first few rows of the dataset, allowing you to inspect the data structure.

Prompt: Perform a transform function on this dataframe that keeps only the rows with Data Analyst, Data Engineer, Data Scientist, and Machine Learning Engineer from the job_title column, and from those keeps only the rows with 2022 and 2023 from the work_year column.

ChatGPT:

To perform the required transformations on the DataFrame, you need to filter the rows based on the job_title and work_year columns. Here's how you can do it using pandas:

1. Filter by job titles: Keep only the rows where the job_title is one of "Data Analyst", "Data Engineer", "Data Scientist", or "Machine Learning Engineer".
2. Filter by work years: Keep only the rows where the work_year is 2022 or 2023.

Here's a sample code to achieve this:

```
{

import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'path_to_your_downloaded_file.csv'

# Load the CSV file into a pandas DataFrame
data = pd.read_csv(file_path)

# Filter rows based on job_title and work_year
filtered_data = data[
    (data['job_title'].isin(['Data Analyst', 'Data Engineer',
'Data Scientist', 'Machine Learning Engineer'])) &
    (data['work_year'].isin([2022, 2023]))
]
```

```
# Display the first few rows of the filtered DataFrame
print(filtered_data.head())
}
```

Make sure to replace 'path_to_your_downloaded_file.csv' with the actual path to your downloaded CSV file. This code snippet will filter the DataFrame and display the first few rows of the filtered data.

Prompt: Using this filtered data, create a table with 2022 and 2023 as columns, the job_title variables as rows, and the cells containing the average salary_in_usd for each year and job title combination.

ChatGPT:

To create a table with 2022 and 2023 as columns, the job titles as rows, and the cells containing the average salary_in_usd for each year and job title combination, you can use the pandas library to group and pivot the filtered DataFrame. Here's how you can do it:

1. Group the filtered data by job_title and work_year, and then calculate the average salary_in_usd.
2. Pivot the table to have work_year as columns and job_title as rows.

Here's the code to achieve this:

```
import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'path_to_your_downloaded_file.csv'

# Load the CSV file into a pandas DataFrame
data = pd.read_csv(file_path)

# Filter rows based on job_title and work_year
filtered_data = data[
    (data['job_title'].isin(['Data Analyst', 'Data Engineer',
'Data Scientist', 'Machine Learning Engineer'])) &
    (data['work_year'].isin([2022, 2023]))
]

# Group by job_title and work_year, then calculate the
average salary_in_usd
grouped_data = filtered_data.groupby(['job_title',
'work_year'])['salary_in_usd'].mean().reset_index()

# Pivot the table to have work_year as columns and job_title
as rows
```

```
pivot_table = grouped_data.pivot(index='job_title',
columns='work_year', values='salary_in_usd')

# Display the resulting pivot table
print(pivot_table)
```

This code snippet will produce a table where the rows are the job titles and the columns are the years 2022 and 2023. The cells in the table will contain the average salary_in_usd for each job title and year combination.

Here's a sample output of what the pivot table might look like:

```
work_year                    2022          2023
job_title
Data Analyst           70000.0      72000.0
Data Engineer          90000.0      95000.0
Data Scientist        110000.0     115000.0
Machine Learning Engineer 120000.0    125000.0
```

This table shows the average salaries for each job title in 2022 and 2023.

Prompt: Now create this table with "work_year" and "job_title" removed, the years as column headers, and the cells rounded to the nearest dollar.

ChatGPT:

Here's the updated code to create a table with "work_year" and "job_title" removed, the years as column headers, and the cells rounded to the nearest dollar:

```
import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'path_to_your_downloaded_file.csv'

# Load the CSV file into a pandas DataFrame
data = pd.read_csv(file_path)

# Filter rows based on job_title and work_year
filtered_data = data[
    (data['job_title'].isin(['Data Analyst', 'Data Engineer',
'Data Scientist', 'Machine Learning Engineer'])) &
    (data['work_year'].isin([2022, 2023]))
]

# Group by job_title and work_year, then calculate the
average salary_in_usd
grouped_data = filtered_data.groupby(['job_title',
'work_year'])['salary_in_usd'].mean().reset_index()
```

```
# Pivot the table to have work_year as columns and job_title
as rows
pivot_table = grouped_data.pivot(index='job_title',
columns='work_year', values='salary_in_usd')

# Round the values to the nearest dollar
pivot_table = pivot_table.round()

# Reset the index to remove "work_year" and "job_title" from
the table
pivot_table.reset_index(inplace=True)

# Display the resulting pivot table
print(pivot_table)
```

This code will produce a table where the rows are indexed by job titles, and the columns are the years 2022 and 2023, with the salary values rounded to the nearest dollar. Here's an example of what the resulting table might look like:

```
work_year              job_title     2022     2023
0                Data Analyst   70000   72000
1                Data Engineer  90000   95000
2               Data Scientist 110000 115000
3  Machine Learning Engineer 120000 125000
```

This table shows the average salaries for each job title in 2022 and 2023, rounded to the nearest dollar.

Prompt: Now create this table with "work_year" and "job_title" removed, and the job titles as row headers.

ChatGPT:

To create the desired table with "work_year" and "job_title" removed, and the job titles as row headers, you can use the following updated code:

```
import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'path_to_your_downloaded_file.csv'

# Load the CSV file into a pandas DataFrame
data = pd.read_csv(file_path)

# Filter rows based on job_title and work_year
filtered_data = data[
    (data['job_title'].isin(['Data Analyst', 'Data Engineer',
'Data Scientist', 'Machine Learning Engineer'])) &
    (data['work_year'].isin([2022, 2023]))
```

```
]

# Group by job_title and work_year, then calculate the
average salary_in_usd
grouped_data = filtered_data.groupby(['job_title',
'work_year'])['salary_in_usd'].mean().reset_index()

# Pivot the table to have work_year as columns and job_title
as rows
pivot_table = grouped_data.pivot(index='job_title',
columns='work_year', values='salary_in_usd')

# Round the values to the nearest dollar
pivot_table = pivot_table.round()

# Display the resulting pivot table
print(pivot_table)
```

This will produce a DataFrame where the job titles are row headers and the columns are the years 2022 and 2023, with salary values rounded to the nearest dollar. The resulting table will look like this:

```
work_year                    2022      2023
job_title
Data Analyst              70000.0  72000.0
Data Engineer             90000.0  95000.0
Data Scientist           110000.0 115000.0
Machine Learning Engineer 120000.0 125000.0
```

To format the table with job titles as row headers and without displaying "work_year" and "job_title" as headers, you can directly print the DataFrame:

```
import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'path_to_your_downloaded_file.csv'

# Load the CSV file into a pandas DataFrame
data = pd.read_csv(file_path)

# Filter rows based on job_title and work_year
filtered_data = data[
    (data['job_title'].isin(['Data Analyst', 'Data Engineer',
'Data Scientist', 'Machine Learning Engineer'])) &
    (data['work_year'].isin([2022, 2023]))
]

# Group by job_title and work_year, then calculate the
average salary_in_usd
```

```
grouped_data = filtered_data.groupby(['job_title',
'work_year'])['salary_in_usd'].mean().reset_index()

# Pivot the table to have work_year as columns and job_title
as rows
pivot_table = grouped_data.pivot(index='job_title',
columns='work_year', values='salary_in_usd')

# Round the values to the nearest dollar
pivot_table = pivot_table.round()

# Print the resulting pivot table
print(pivot_table)
```

This will produce the desired table without the "work_year" and "job_title" as headers and the job titles as row headers.

Prompt: Remove the decimal point from the cell values, also the labels "work_year" and "job_title" should not be present in the final table.

ChatGPT:

To produce a final table where the decimal points are removed from the cell values and the labels "work_year" and "job_title" are not present, you can use the following code:

```
import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'path_to_your_downloaded_file.csv'

# Load the CSV file into a pandas DataFrame
data = pd.read_csv(file_path)

# Filter rows based on job_title and work_year
filtered_data = data[
    (data['job_title'].isin(['Data Analyst', 'Data Engineer',
'Data Scientist', 'Machine Learning Engineer'])) &
    (data['work_year'].isin([2022, 2023]))
]

# Group by job_title and work_year, then calculate the
average salary_in_usd
grouped_data = filtered_data.groupby(['job_title',
'work_year'])['salary_in_usd'].mean().reset_index()

# Pivot the table to have work_year as columns and job_title
as rows
pivot_table = grouped_data.pivot(index='job_title',
columns='work_year', values='salary_in_usd')
```

```
# Round the values to the nearest dollar and convert to
integers
pivot_table = pivot_table.round().astype(int)

# Print the resulting pivot table without the default pandas
index name
pivot_table.columns.name = None
pivot_table.index.name = None
print(pivot_table)
```

This code does the following:

1. Loads the CSV data into a pandas DataFrame.
2. Filters the DataFrame for the specified job titles and work years.
3. Groups the data by job_title and work_year, then calculates the average salary_in_usd.
4. Pivots the table so that work_year becomes the columns and job_title becomes the rows.
5. Rounds the values to the nearest dollar and converts them to integers to remove the decimal points.
6. Removes the column and index names for a cleaner table output.

Here is an example of what the resulting table might look like:

```
                           2022    2023
Data Analyst              70000   72000
Data Engineer             90000   95000
Data Scientist           110000  115000
Machine Learning Engineer 120000 125000
```

This table has job titles as row headers and the years 2022 and 2023 as column headers, with the average salary values rounded to the nearest dollar and displayed as integers.

Prompt: Now print this code with the file_path variable being 'jobs_in_data.csv' and the data variable being renamed to "jobs".

ChatGPT:

Here is the code with the file_path variable set to 'jobs_in_data.csv' and the DataFrame variable renamed to jobs:

```
import pandas as pd

# Assuming the CSV file is named 'jobs_in_data.csv' and
located in the current directory
file_path = 'jobs_in_data.csv'
```

```
# Load the CSV file into a pandas DataFrame
jobs = pd.read_csv(file_path)

# Filter rows based on job_title and work_year
filtered_jobs = jobs[
    (jobs['job_title'].isin(['Data Analyst', 'Data Engineer',
'Data Scientist', 'Machine Learning Engineer'])) &
    (jobs['work_year'].isin([2022, 2023]))
]

# Group by job_title and work_year, then calculate the
average salary_in_usd
grouped_jobs = filtered_jobs.groupby(['job_title',
'work_year'])['salary_in_usd'].mean().reset_index()

# Pivot the table to have work_year as columns and job_title
as rows
pivot_table = grouped_jobs.pivot(index='job_title',
columns='work_year', values='salary_in_usd')

# Round the values to the nearest dollar and convert to
integers
pivot_table = pivot_table.round().astype(int)

# Print the resulting pivot table without the default pandas
index name
pivot_table.columns.name = None
pivot_table.index.name = None
print(pivot_table)
```

This code will read the data from the jobs_in_data.csv file, filter and process it to create a pivot table with job titles as row headers and years as column headers, and then print the table with the salary values rounded to the nearest dollar and displayed as integers.

In [13]:
```
# Filter rows based on job_title and work_year
filtered_jobs = jobs[
    (jobs['job_title'].isin(['Data Analyst', 'Data Engineer', 'Data Scientis
    (jobs['work_year'].isin([2022, 2023]))
]

# Group by job_title and work_year, then calculate the average salary_in_usd
grouped_jobs = filtered_jobs.groupby(['job_title', 'work_year'])['salary_in_

# Pivot the table to have work_year as columns and job_title as rows
pivot_table = grouped_jobs.pivot(index='job_title', columns='work_year', val

# Round the values to the nearest dollar and convert to integers
pivot_table = pivot_table.round().astype(int)

# Print the resulting pivot table without the default pandas index name
pivot_table.columns.name = None
pivot_table.index.name = None
```

```
print(pivot_table)
```

```
                            2022    2023
Data Analyst                108658  110988
Data Engineer               139803  149945
Data Scientist              138529  163714
Machine Learning Engineer   151775  191026
```

Part b

Having worked through using ChatGPT for data wrangling, take a moment to reflect on when it makes sense and when it doesn't make sense to use ChatGPT for working with data prior to an analysis. Write a short paragraph here summarizing your thoughts. (1 point)

I am decently impressed by ChatGPT's ability to write clear and effective chunks of data transformation code. However, there were some details in my process that leave me apprehensive. For starters, the cell values in the example table seem to be completely made up, which suggests that ChatGPT did not test the actual dataset. As such, it's probably best to use ChatGPT only with datasets that are verified to be clean, or that you have cleaned yourself. Additionally, ChatGPT struggled most with my request to remove the "work_year" and "job_title", only successfully doing so once I had instructed it in the clearest possible terms. Therefore, ChatGPT is best used with a specific goal in mind, so that you know exactly how to express what you want it to do.

In [ ]:

```
Python 3.12.2 | packaged by conda-forge | (main, Feb 16 2024, 20:54:21) [Clang 16.0.6 ]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.25.0 -- An enhanced Interactive Python. Type '?' for help.
```

[20]:
```
?pd.DataFrame
```

```
Init signature:
pd.DataFrame(
    data=None,
    index: 'Axes | None' = None,
    columns: 'Axes | None' = None,
    dtype: 'Dtype | None' = None,
    copy: 'bool | None' = None,
) -> 'None'
Docstring:
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure.

Parameters
----------
data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
    Dict can contain Series, arrays, constants, dataclass or list-like objects. If
    data is a dict, column order follows insertion-order. If a dict contains Series
    which have an index defined, it is aligned by its index. This alignment also
    occurs if data is a Series or a DataFrame itself. Alignment is done on
```

[ ]:

Problem 3

Open a console window and place it next to your notebook in Jupyter labs. Load the kernel from the notebook into the console, then call up the docstring for the `pd.DataFrame` function. Take a screenshot and include it in your lab report. (To include a locally saved image named `screenshot.jpg`, for example, create a Markdown cell and paste

```
<img
src="screenshot.jpg"
width=600>
```

(2 points)

[ ]:

Problem 4