

DS 6600: Data Engineering 1

Lab assignment 1

Instructions: use a Jupyter notebook to complete the following questions. Use a markdown cell for text and a code cell for Python code. Once your notebook is complete, follow [these instructions](#) to save the notebook as a PDF file and submit your assignment to Gradescope on Canvas.

Problem 1

We are going to use Git and GitHub for this lab assignment. Please complete all of the following steps:

- Create a new GitHub repository for this lab assignment called "DS6600_lab1". Make it public, add a README.md file, choose a .gitignore file that is specific to Python-based projects. Also, choose a licence (using [choosealicense.com](#)) that allows other people to use the code in your repository for both commercial and non-commercial use, including full rights to distribute and modify the code, but does not allow anyone to modify and distribute your code with a more restrictive license than the one included in your repository.
- In your terminal, choose a location on your computer for the local copy of this repository and use `git clone` to download it and connect this local directory to the GitHub repository.
- In your README.md file, explain (in your own words! Copy-and-pasted answers will receive no credit) why you chose the license you did and what would have happened had you not supplied a license. In addition, examine your .gitignore file to confirm that a file named ".env" will not be pushed to GitHub. Copy and paste the relevant section of the .gitignore file in your README.md file.
- As you work on this assignment, add, commit, and push your changes to GitHub. We will look at your repository's commit history to make sure there are at least 6 commits (one for each of the questions on this lab).

For this question on your assignment, all you need to supply in this notebook is a URL for your GitHub repository. Please type it here. [8 points]

Problem 2

Describe, in words, whether a virtual machine, a container, a virtual environment, or the

global environment of a single computer is best suited for each of the following situations. Be clear about why the option you choose works best, and also about why the other options are insufficient or are overkill.

Part a

The local chapter of [Meals on Wheels](#) worked with some UVA computer science students and volunteers from [WillowTree Apps](#), who built them a [web portal for volunteers](#) to sign up for shifts and a [mobile app](#) that shows volunteers their driving routes with GPS mapping. The portal and app have been extremely useful for the local Meals on Wheels chapter, but unfortunately, the portal and app were built only for the Charlottesville chapter and is not usable by any other Meals on Wheels chapter at present. You've been asked to help every other Meals on Wheels chapter to benefit from these products by generalizing the code, and then making it available for free to any chapter that wants to use the portal and app. Every chapter has their own database of volunteers and clients, necessitating a large amount of storage that must also be accessible through an internet connection. [6 points]

Part b

The [Legal Aid Justice Center](#), a Charlottesville-based legal aid nonprofit that advocates for the rights of immigrants and the working poor, has used a Freedom of Information Act request to get Virginia's Department of Labor and Industry (DOLI) to share their data on all [official complaints of wage theft](#) in Virginia over the last five years. The DOLI stored this data in an Excel file with horrible formatting: the column headers are given vague names, text has misspelling throughout the file (you see locations such as "Rochmond" and "Norflok"), data types are inconsistent within columns (sometimes a \$ is included with the wage amounts, sometimes \$ is excluded), and so on. They've asked you to clean the data and generate a report that lists the average claim of wage theft and a frequency table of the number of claims by locality. They don't need your code or data, just the report. [6 points]

Part c

An international NGO called [Save the Children](#) provides housing, food, medical care, and other assistance to individuals who have been displaced from their homes by conflict or natural disasters. But a major problem is that these displacement events can happen suddenly, without giving Save the Children any time to mobilize. Major migration events are more likely when the infrastructure and buildings in a city or town have been destroyed. Save the Children have asked you to build a predictive model that can detect from [land-satellite images](#) the extent of infrastructure damage in an area of interest and that can be deployed by several different individuals working for Save the Children. You build this model, but in addition to Python you require several open-source image

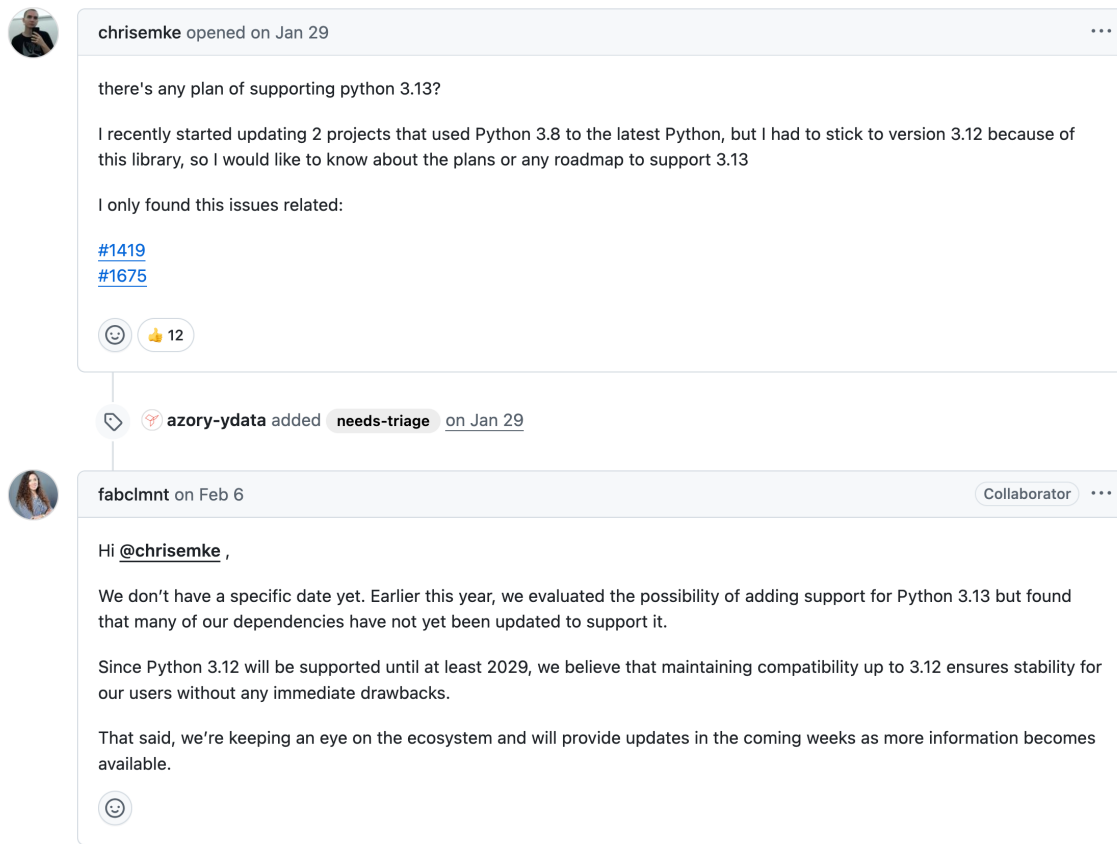
processing software packages that only work on Ubuntu Linux. [6 points]

Part d

Sometime in the near future, Python announces the release of Python version 4. In this major update, all commands are now in Swedish, and any command written with English words such as "read" will now fail to run. You are willing to relearn Python in Swedish to use this much needed and long overdue upgrade, after all Python är ett programmeringsspråk som låter dig arbeta snabbt och integrera system mer effektivt. But you don't want the code you are writing for your current project in Python 3 to stop working once Python 4 is released. [6 points]

Problem 3

One of the packages we will soon use is `ydata_profiling`, which creates an HTML dashboard from a dataframe for the purpose of exploratory data analysis. It remains one of the very best packages available for making an EDA dashboard, but the problem is this package [does not work with Python 3.13](#):



The screenshot shows a GitHub issue thread. The first comment is from user **chrisemke**, dated Jan 29. It asks if there's a plan to support Python 3.13, mentions updating projects from Python 3.8 to the latest but sticking to 3.12 due to this issue, and lists two related issues: #1419 and #1675. It has 12 likes. A label **needs-triage** is added by **azory-ydata** on Jan 29. The second comment is from **fabclmnt**, dated Feb 6, with a **Collaborator** badge. It responds to @chrisemke, stating that while they evaluated adding support for Python 3.13, many dependencies haven't been updated yet. It notes that Python 3.12 will be supported until at least 2029 and that they will provide updates as more information becomes available.

chrisemke opened on Jan 29

there's any plan of supporting python 3.13?

I recently started updating 2 projects that used Python 3.8 to the latest Python, but I had to stick to version 3.12 because of this library, so I would like to know about the plans or any roadmap to support 3.13

I only found this issues related:

[#1419](#)
[#1675](#)

12

azory-ydata added **needs-triage** on Jan 29

fabclmnt on Feb 6 Collaborator

Hi **@chrisemke**,

We don't have a specific date yet. Earlier this year, we evaluated the possibility of adding support for Python 3.13 but found that many of our dependencies have not yet been updated to support it.

Since Python 3.12 will be supported until at least 2029, we believe that maintaining compatibility up to 3.12 ensures stability for our users without any immediate drawbacks.

That said, we're keeping an eye on the ecosystem and will provide updates in the coming weeks as more information becomes available.

Tough luck. If you are working in your global environment, now you have to reinstall an older version of Python on your machine and reinstall all of your packages again. There's a good chance that some of these reinstalls will generate more version incompatibility problems. And then you are stuck in a software version purgatory. When you get your

system working again, if you can, it's fragile and may buckle and break with any additional new package. That's not a good place to be.

But you are working with virtual environments and dealing with obnoxious package version conflicts is one of the main reasons why virtual environments are so useful.

Part a

Create a new Conda environment that runs Python 3.12, *not* 3.13. Refer to the [textbook's discussion of miniconda](#) to help you with the code. Give this environment a name that is different from the one we've been using for class up to this point, and activate the environment on the terminal's command line. Use `conda` to install these packages: `neo4j python-dotenv pandas numpy scipy scikit-learn requests prince ipykernel conda-forge::wquantiles`. (We'll use `neo4j` and `python-dotenv` later in this lab.) Then use `pip` to install `ydata_profiling` (because the package is not available via `conda`). Copy and paste your command-line code into a markdown cell in your notebook.

[8 points]

Part b

Connect your notebook to this virtual environment as the kernel, and run the following code cell. If you are successful, it will run without error. (You may be prompted to "Upgrade to ydata-sdk". That's the paid version of the `ydata_profiling` package. My rule of thumb is not to pay for anything that can't be bothered to update itself to work on Python 3.13.)

[4 points]

```
In [ ]: import numpy as np
import pandas as pd
import weighted # this is a module of wquantiles
from scipy import stats
import prince
from ydata_profiling import ProfileReport
```

Problem 4

The [official Python images](#) on Docker Hub use a version of Linux called Debian. However, sometimes you might need to install additional software in a container other than Python and Python packages, and a lot of open source software only works on another version of Linux called Ubuntu.

Part a

Write a Dockerfile that builds an image from the `ubuntu:latest` image on Docker Hub. Run the following Linux commands to install Python 3 onto Ubuntu: `apt-get update` and `apt-get install -y python3`. Once Python is installed, run the following command to launch Python from the command line: `python3`. Copy and paste the Dockerfile here. [8 points]

Part b

Prove that the Dockerfile is written correctly by building the image associated with this Dockerfile. Once you are able to build the image successfully, copy and paste the output of the build in your terminal here. [8 points]

Part c

We've mostly run Docker containers by using the `-p` tag to map the operations of a container to a port on our local machine. But another way to run a container is in "interactive mode" using the `-it` tag, which immediately replaces your command line with the command line that exists inside the container. If you've correctly specified your Dockerfile to both install and launch Python inside the container, running this container in interactive mode should result in showing you a Python command line. Type `docker run -it` followed by the name of the image you created in parts (a) and (b). Confirm that the Python prompt appears. Then copy-and-paste the output from your terminal from the `docker run` command here. [4 points]

Problem 5

Docker is for more than databases and Python. It can be used to deploy any kind of software on any operating system. For example, [NetHack](#) is a classic text-based adventure video game. It was first released in the 1980s.

Search on Docker Hub for NetHack. Find the Docker image for running NetHack that was posted by the user "matsuu". Then find a way to run a container on your computer from this image. If you are successful, then after answering the first few questions you will receive a passage from a sacred book, scroll, or text. Copy the passage and paste it here. [8 points]

Problem 6

We've discussed connecting to MySQL, PostgreSQL, and Mongo using Docker, but there are many different kinds of database systems for different situations, and as a data scientist who can hang with the data engineers, you will need to be able to set up a database from a system you haven't used before. For this problem, use your Docker

skills and your navigation of available documentation to create a local Python connection to a graph database running on [Neo4j](#). See this [AWS blog[(<https://aws.amazon.com/compare/the-difference-between-graph-and-relational-database/>)] or the [textbook](#) for a deeper discussion of graph databases.

Part a

Make sure the conda environment you created in problem 3 is selected as the kernel for this notebook, then run the following import statements. [4 points]

```
In [1]: from neo4j import GraphDatabase
import dotenv
import os
```

Part b

Use the documentation listed on the [Docker Hub page for the official Neo4j Docker image](#), and the Neo4j documentation linked there, to determine:

- the default port the image runs on (there are two in this case, one called "bolt" is used for connecting to the database itself, and the other is used for an HTML dashboard with a user interface for working with the dashboard if you want to use a UI),
- the folder inside the container that stores the data (feel free to ignore examples of folders outside the container as we can use `volumes` to manage that),
- and the environmental variables required by the Neo4j image. Make sure you specify a password, and don't disable authentication with `--env=NEO4J_AUTH=None`. (You will need to use the Neo4j documentation outside of Docker Hub. Take a look at the "Getting started with Neo4j in Docker" page.)

List your answers, and how/where you found those answers. [8 points]

Part c

Create a `compose.yaml` file that launches a Neo4j container from the Neo4j official image on Docker Hub. In the `compose.yaml` file: attach the ports that are needed, load the necessary environmental variables via a `.env` file, and create a local volume named "neo4jdata" and map it to the data directory inside the container. Copy and paste your `compose.yaml` code into a Markdown cell in this notebook. [8 points]

Part d

Type `docker compose up` on the command line to launch the Neo4j container. Then

run the following Python code. If your container is launched, the following code will display the phrase "Connection to Neo4j established successfully", you've succeeded in getting Neo4j to run on your system, and you are a rockstar data engineer. [8 points]

```
In [ ]: dotenv.load_dotenv()
NEO4J_AUTH = os.getenv('NEO4J_AUTH').split("/")

URI = "bolt://localhost:7687"
USERNAME = NEO4J_AUTH[0]
PASSWORD = NEO4J_AUTH[1]

try:
    # Create a Driver instance
    # This only provides connection information, it does not establish a connection
    driver = GraphDatabase.driver(URI, auth=(USERNAME, PASSWORD))

    # Verify connectivity immediately
    # This forces the driver to create a connection and check credentials/credentials
    driver.verify_connectivity()
    print("Connection to Neo4j established successfully.")

except Exception as e:
    print(f"Failed to connect to Neo4j: {e}")

finally:
    # Close the driver to release resources
    if 'driver' in locals() and driver:
        driver.close()
```