Problem 1

https://github.com/janiceeguo/DS6600_lab1

Problem 2

Part a

The best option would be a container, since it allows each chapter to run their own copy of the portal and app, but using the same code. Since containers can package code to be deployed regardless of local infrastrucutre setup, it also complements large storage capabilities that are easily accesible over the internet. Virtual machines are not necessary since each chapter doesn't need a full, isolated operating system, it just needs a packaged container so that the applications can run consistently - this option is also usually not free. Virtual environments only really isolate programing language dependencies, which wouldn't be enough to run separate instances of the application for each chapter with their own database. Global environments are even less powerful - too many issues could arrise across chapters like dependency and version clashes.

Part b

A global environment would be sufficient enough, since this is a one-time task that only needs to be run in one instance. There is no need to package and share the environment since the final deliverable involves just the report, not the code. Thus, containers and virtual machines are unecessary because the code does not need to be run anywhere other than your computer. Virtual environments are also unecessary because the report does not have any dependency conflicts, and thus does not require a separate environement to manage.

Part c

Since this specifically requires Ubuntu Linux, a virtual machine would be the best option. Virtual machines provide an isolated operating system environment that would be compatible. They are also portable, which would allow different individuals to run the model. Containers are not sufficient because they do not guarantee that the kernel or underlying system will be compatible. Virtual environments only handel Python-level dependencies, with no guarantee about system packages. Global environments would require separate installations of Ubuntu Linux for each machine that wants to run the model, which is much more impractical.

Part d

A virtual environment would be the best option for this, since it can isolate any Python 3 dependencies even after you download Python 4 for future projects. Containers and virtual machines would be unecessary, since we only need backwards compatability, not the ability to deploy on other machines. Global environments would not be sufficient because if Python 4 is installed globally, any Python 3 projects would break because they are not isolated.

Problem 3

Part a

```
conda create -n dslab1 python=3.12 neo4j python-dotenv pandas numpy
scipy scikit-learn requests
prince ipykernel conda-forge::wquantiles
conda activate dslab1
pip install ydata_profiling
conda deactivate
```

Part b

In [1]:
```
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
```

```
c:\Users\janic\.conda\envs\dslab1\Lib\site-packages\tqdm\auto.py:21: TqdmWarning: IP
rogress not found. Please update jupyter and ipywidgets. See https://ipywidgets.read
thedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Upgrade to ydata-sdk

Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection, outlier identification, text validation, and synthetic data generation.

Problem 4

Part a

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest
RUN apt-get update and apt-get install -y python3
WORKDIR /DS6600_lab1
CMD ["python3"]
```

Part b

image is named "lab1ubuntu"

```
[+] Building 20.8s (9/9) FINISHED
docker:desktop-linux
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 178B
0.0s
 => resolve image config for docker-
image://docker.io/docker/dockerfile:1
0.3s
 => CACHED docker-
image://docker.io/docker/dockerfile:1@sha256:dabfc0969b935b2080555ace70eee
0.0s
 => => resolve
docker.io/docker/dockerfile:1@sha256:dabfc0969b935b2080555ace70ee69a5261a
0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest
0.1s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => CACHED [1/3] FROM
docker.io/library/ubuntu:latest@sha256:353675e2a41babd526e2b837d7ec780c2a
0.0s
 => => resolve
docker.io/library/ubuntu:latest@sha256:353675e2a41babd526e2b837d7ec780c2a
0.0s
 => [2/3] RUN apt-get update && apt-get install -y python3
13.1s
 => [3/3] WORKDIR /DS6600_lab1
0.1s
 => exporting to image
6.2s
 => => exporting layers
4.7s
 => => exporting manifest
sha256:d7a9afb61650ad528d8a9a421f9766f26c31f7f33f56bd290df31461071de742
0.0s
 => => exporting config
sha256:f70741c9eb5394a358e76c785008c419b472015ce6c14610526c4e1ed32766fe
0.0s
 => => exporting attestation manifest
sha256:5813b0a80d14aa6e7e9007ead89a13f517d43da8e48c5bb2cab18d7f
0.0s
 => => exporting manifest list
sha256:3e2ecd14e9f41b817f98390482caa131b63beb877f08027579504f8ebdb81c0
```

```
0.0s
 => => naming to docker.io/library/lab1ubuntu:latest
0.0s
 => => unpacking to docker.io/library/lab1ubuntu:latest
1.3s
WARNING: current commit information was not captured by the build:
git was not found in the system: exec: "git.exe": executable file
not found in %PATH%
```

Part c

```
Python 3.12.3 (main, Aug 14 2025, 17:47:21) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Problem 5

It is written in the Book of Lugh:

> After the Creation, the cruel god Moloch rebelled against the authority of Marduk the Creator. Moloch stole from Marduk the most powerful of all the artifacts of the gods, the Amulet of Yendor, and he hid it in the dark cavities of Gehennom, the Under World, where he now lurks, and bides his time.

Your god Lugh seeks to possess the Amulet, and with it to gain deserved ascendance over the other gods.

You, a newly trained Gallant, have been heralded from birth as the instrument of Lugh. You are destined to recover the Amulet for your deity, or die in the attempt. Your hour of destiny has come. For the sake of us all: Go bravely with Lugh!

Problem 6

Part a

In [2]:
```
from neo4j import GraphDatabase
import dotenv
import os
```

Part b

Default port: 7474 for HTTP and 7687 for Bolt (https://hub.docker.com/_/neo4j)
Folder: /data (https://hub.docker.com/_/neo4j)

Environmental variables: NEO4J_AUTH and NEO4J_ACCEPT_LICENSE_AGREEMENT if using the enterprise version (https://neo4j.com/docs/operations-manual/current/docker/introduction/)

Part c

```
services:
  neo4j:
    image: neo4j:latest
    env_file:
      - .env
    ports:
      - "7474:7474"
      - "7687:7687"
    volumes:
      - neo4jdata:/data

volumes:
  neo4jdata:
```

Part d

In [3]:
```
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/compatibi
    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()
```

```
Connection to Neo4j established successfully.
```