A assignment of text processing using python.

The writing here about text processing using python should not be used as factual information, but as an expression of my opinions.

To obtain a pdf of this page, click here. Please note that the pdf is not a 1 to 1 representation of this page and is rendered using `pandoc` and `texlive`.

The code is located here. Executing this file is done via the command line using the command `python3.11 nlpcsvcheck.py <csv>` and **requires** python 3.11 due to type hints. If you are not using python 3.11, remove them from the function signatures.

The code itself

To begin, the code itself in 83 lines deserializes CSV rows and columns to a dictionary before getting shoved into a class and pickled. The pickle is then loaded back in to run the `display()` function, as a normal python object. This, however, betrays the power of python builtins, as python itself has a module for CSVs called `csv`. This module aims to allow for quick dictionary processing and serialization of CSVs. Outside of the standard library, NumPy can be used to serialize all of these objects into compact .npy files that are essentially compact C arrays before running them through powerful C functions that change the data.

The power of python

The strengths of python for text processing (deserialization) is that you can bang out some garbage code of reasonable complexity and have it work pretty easily, allowing for easy ingesting of data into a dataset. The main downside of python for text processing is that it's very slow and/or memory hungry in some situations. For example, on a separate (personal) project, attempting to find all the lines of a CSV file to see how many new things I had generated required that I had to write code like this:

```
with open("some.csv") as dump:
  new_amt = 0
  for _ in dump:
    new_amt += 1

print(f"new amount read: {new_amt}")
```

This is clunky, and mostly slow. Why don't we go with the more direct (and faster) route of this:

```
with open("some.csv") as dump:
  new_amt = len(dump.readlines())
```

```
print(f"new amount read: {new_amt}")
```

Well, because my system actually exhausted it's own resources reading that CSV, eventually killing python because it used too much RAM. The ram efficient method (the first one) takes upwards of 5-15 minutes to process.

My personal learning from this assignment

My main gain from writing this assignment was not from any actual serialization exercise done, but from utilizing type hints. They seem to be more in depth than I previously thought. The thing that was mostly a review was the whole class creation, working with dictionaries, and general text line processing.