Part 3.) The results from the first test, where 80% is training and 20% is test, is 95.5% accuracy.

Part 4.) The results from the second test, where 20% is training and 80% is test, is 92.9% accuracy. While both tests have good accuracy scores overall, the first test is slightlty better.

Part 5.) From testing both splits, both the 80/20 and the 20/80 training/test splits, I found that there were clear reasons why both had some wrong answers, although their respective wrong answers were slightly different. For example, the 80/20 split wrong answers were easy to see multiple numbers, usually due to a lack of a whole (small numbers like 0) where there should have been in 8 and 9. Therefore, the system got stuck and thought the numbers were different, like guessing 9 instead of 8 and 5 instead of 9. Out of the 5 incorrect examples I reviewed, 3 of them were with an 8. For the 20/80 split, there were similar problems but more egregious. Out of the 5 incorrect examples I reviewed, 3 of them were with an 2 this time, which leads me to believe that certain numbers in these algorithms are more often incorrect than others, most likely due to their structure being more complex, having more holes, or being more similar to other numbers.

Part 6.) Since the data was collected from a csv we were given, and minimal cleaning was applied to it, it is quite possible for there to be small errors or problems with the data that could lead to our tests not giving the correct answers. The tests in question only deal with numbers, so the cleaning process was quite easy: only look for rows without usable numbers. This didn't account for rows that would be usable by our testing, which is much harder to find since the line between usable and not is so small. The fact that we don't know where the data was from could also be a possible issue.

Part 8.) For the scikit-learn run, I picked k = 3 as a reasonable guess. Compared to k=1, using three neighbors reduces the chance that one odd/noisy training example dominates the prediction, but it's still "local" enough to capture the digit shapes without oversmoothing. With k=3, the accuracy on the first split (test = last 20%) was 0.966 (342/354 correct). On the swapped split (test = first 20%), accuracy was 0.963 (341/354 correct). For context, our pure 1-NN scored 0.955 and 0.969 on those splits, respectively, so k=3 performed similarly overall while being a bit less sensitive to single training points.

Part 9.) I searched k over {1, 3, 5, 7, 9} using three shuffle/validation seeds (8675309, 5551212, and 123123). On my main split (train = first 80%), the per-seed winners were k=1, k=3, and k=3, so the majority vote pointed to k=3. On the swapped split, the winners were k=1, k=7, and k=1. Since the winning k changed across seeds (and also changed when I swapped the training set), it’s clearly not perfectly stable. I chose k=3 overall because it won the majority on the main split and consistently had very strong validation accuracy, while providing a bit of smoothing over 1-NN without drifting too far from local decisions.

Part 10.)

First Split:

```
Total: 354
Correct: 342
Accuracy: 0.966
First 10 mismatches (index: true → pred):
  110: 8 → 1
  139: 9 → 5
  162: 3 → 7
  163: 3 → 8
  168: 4 → 7
  185: 4 → 9
  215: 9 → 3
  217: 4 → 8
  219: 9 → 5
  284: 3 → 2
```

Second Split:

```
Total: 354
Correct: 343
Accuracy: 0.969
First 10 mismatches (index: true → pred):
  28: 2 → 8
  30: 2 → 8
  45: 9 → 4
  53: 2 → 1
  63: 4 → 1
  71: 6 → 1
  96: 5 → 9
  99: 8 → 1
  105: 8 → 1
  186: 1 → 4
```