7. *Experiments with using k-d trees for (comprehensive) proximity search.* As we discussed in class, one very bad scenario for nearest neighbor data structures is when the data are uniformly distributed over a unit sphere in $\mathbb{R}^d$; in even moderate dimension d, such data points are likely to be almost equidistant from each other.

   In this problem, we will start by seeing how k-d trees perform on data of this kind, for various values of d. We will then use k-d trees to answer nearest neighbor queries on MNIST data, and use this to get a sense of the *intrinsic dimension* of MNIST.

   (a) For dimensions $d = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50$, do the following:
   - Generate 60,000 training points uniformly at random from the unit sphere in $\mathbb{R}^d$.
   - Generate 100 test points uniformly at random from the unit sphere in $\mathbb{R}^d$.
   - Build a k-d tree on the training points using `sklearn.neighbors.KDTree`. Specify `leaf_size=2`.
   - Use this tree to compute the nearest neighbors of the 100 test points. Keep track of the total number of distance computations performed, using `reset_n_calls()` and `get_n_calls()`. In this way, obtain the average number of distance computations per query; this will be a value in the range 1 to 60,000.

   Plot the average number of distance computations per query against d.

   (b) Load in the MNIST data set. Build a k-d tree on the 60,000 training points, again using `leaf_size = 2`. Now use this tree to find the nearest neighbor of the first 100 test points. What is the average number of distance computations per query?

   (c) By comparing your answers to (a) and (b), give a rough answer to the following question: for what value of d does k-d tree performance on MNIST behave like k-d tree on uniform-random data in $\mathbb{R}^d$? We can think of this as one measure of the *intrinsic dimension* of MNIST.