1. **Tabulate the execution times of each of the individual approaches for computing distance in Python (i.e., run the shared code on your computer, note the times, and tabulate them).**

Approach	Execution Time (s)
For-loop	0.01698
NumPy Vectorization	0.000365
Pandas Apply	0.03223

2. **Next, replicate the for-loop based approach (the first one) and two different ways to make that version more efficient, in R. Profile these three approaches, and tabulate the results.**

Approach	Execution_Time_s
For-loop	0.05671382 secs
Vectorized (mapply)	0.01404119 secs
geosphere::distHaversine	0.01336598 secs

3. **Based on the computational efficiency of implementations in Python and R, which one would you prefer? Based on a consideration of implementation (i.e., designing and implementing the code), which approach would you prefer? Taking both of these (run time and coding time), which approach would you prefer?**
   Based on computational efficiency, Python (NumPy vectorization) is the clear winner, as it significantly outperforms R's for-loop and mapply approaches. However, in terms of ease of implementation, R's geosphere::distHaversine is the simplest, as it provides a built-in function for geospatial calculations with minimal coding effort. Balancing both execution speed and coding simplicity, Python (NumPy vectorization) is the best choice overall, offering both high performance and relatively easy implementation once familiar with NumPy operations. If the task is strictly geospatial, I would go with R's geosphere package which is a convenient option for quick implementation.
4. **Identify and describe one or two other considerations, in addition to these two, in determining which of the two environments – Python or R – is preferable to you.**
   In this project, where we explored different ways to compute distances in Python and R, the choice between the two languages really comes down to their strengths. Python's NumPy vectorization was by far the fastest, showing how well Python handles numerical computations with efficient array operations. On the other hand, R's geosphere::distHaversine() made it incredibly easy to calculate distances without much effort, highlighting R's strength in geospatial and statistical analysis.
   Another important factor is how well each language integrates with other tools. If this project were part of a larger system—like a real-time navigation app or a machine learning model that predicts distances, Python would be the better choice because it works seamlessly with databases, cloud services, and web applications. But if the goal was more focused on geospatial analysis in research or healthcare, R might be a better fit because of its specialized libraries. In the end, while R made geospatial calculations simple, Python's speed, flexibility, and ability to scale give it the edge for most real-world applications.