2. Based on the results, which approach to leveraging XGBoost would you recommend? Explain the rationale for your recommendation.

When evaluating the different approaches to leveraging XGBoost, direct use of **xgboost() in R**, using **caret** with XGBoost in R, and using scikit-learn with **XGBoost in Python**, it's important to consider both model performance (accuracy) and computational efficiency (time taken), especially as the dataset size grows. Starting with direct use of **xgboost() in R**, the model achieved decent accuracy (around **91.6%**) fairly consistently across different dataset sizes. It was also quite fast, particularly on small and medium-sized datasets. However, the slight downside here is that the accuracy plateaued around **91-92%**, and did not improve much even as the dataset grew from 1,000 to 10 million observations. This suggests that while the method is computationally efficient, it might not fully exploit the data to maximize predictive performance. In the second method, using **XGBoost via caret in R**, the accuracy improved slightly for small datasets (e.g., **92.9%** for size 100), but as the dataset grew larger, the performance stabilized around **91.7%**, very similar to the direct xgboost use. However, a major drawback of the caret method was dramatically **higher computation time**. For example, at 10 million observations, the model fitting time was nearly 400 seconds (~6.5 minutes), compared to about 160 seconds when using direct xgboost() in R. caret adds overhead because it wraps models inside a general framework for hyperparameter tuning and resampling, even when explicit tuning is not needed. This makes it less efficient for large datasets unless tuning complexity is needed.

On the other hand, the third method, **XGBoost via scikit-learn in Python**, outperformed both R approaches significantly in terms of predictive performance. The scikit-learn approach achieved about **96-98% accuracy** as the dataset size increased, a massive improvement compared to **91-92%** with R. This suggests that the scikit-learn implementation was able to better leverage the large volume of training data, perhaps because of more stable internal optimizations or default handling of cross-validation and model fitting. In terms of **computation time**, while **scikit-learn also showed increasing time** with dataset size (as expected), it remained comparable or even better than the caret approach for larger datasets. For example, fitting the 10 million record dataset took around 132 seconds in scikit-learn, compared to 396 seconds in caret. Finally, I recommend using XGBoost via scikit-learn in Python. It achieves significantly higher predictive performance, scales better with large datasets, and integrates well into modern machine learning workflows for tuning, evaluation, and deployment. It offers the best tradeoff between speed, accuracy, and flexibility.