1. Compare the accuracy values of XGBoost models fit on the newly created data, for the following sizes of datasets. Along with accuracy, report the time taken for computing the results. Report your results in a table with the following schema.

Method used	Dataset size	Testing -set predictive performance	Time taken for the model to be fit
XGBoost in Python via scikit-learn and 5-fold CV	100	0.890	0.344
	1000	0.969	0.280
	10000	0.974	0.203
	100000	0.980	1.606
	1000000	0.982	14.266
	10000000	0.982	132.34
XGBoost in R – direct use of xgboost() with simple cross-validation	100	0.889	0.10
	1000	0.942	0.26
	10000	0.969	0.82
	100000	0.979	2.71
	1000000	0.983	15.17
	10000000	0.983	135.71
XGBoost in R – via caret, with 5-fold CV simple cross-validation	100	0.919	1.16
	1000	0.924	1.08
	10000	0.940	1.90
	100000	0.944	8.43
	1000000	0.944	71.94
	10000000	0.943	960.62

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. In the results, **scikit-learn** consistently achieves **very high predictive performance** across all dataset sizes, maintaining values around **0.98** for larger datasets, which is comparable to or slightly better than the performance achieved by direct XGBoost usage in R and XGBoost through caret in R. Importantly, the time taken to fit the model using scikit-learn is significantly lower than the caret-based R implementation, particularly as the data size increases. For instance, for 10 million rows, scikit-learn takes around 132 seconds, while caret in R takes an enormous 960 seconds, showing that scikit-learn scales much better with data size.

Compared to the direct use of XGBoost in R (without caret), the performance is close but training times are still slightly longer than Python scikit-learn for the same dataset sizes. The direct R method does offer faster times than the R-caret pipeline, but not enough to surpass the efficiency and smooth scaling provided by scikit-learn. Also, scikit-learn's integration of cross-validation with model training is highly optimized internally, giving it an advantage in terms of code simplicity, runtime efficiency, and flexibility when adjusting hyperparameters or running repeated experiments. The R-caret approach, while functional and structured, clearly introduces a lot of overhead. The performance starts relatively lower for small datasets and doesn't scale significantly better with data size; moreover, the training time becomes a serious bottleneck with larger datasets, making it impractical for real-world large-scale machine learning tasks where time and computational resources are critical factors.

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.