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	0.930000	2.763152s
	1000	0.944000	0.378408s
	10000	0.9754000	1.117135s
	100000	0.986760	3.834116s
	1000000	0.991872	38.009035s
	10000000	0.993176	401.379248s
XGBoost in R – direct use of xgboost() with simple cross-validation	100	0.74	0.015s
	1000	0.96	0.0288s
	10000	0.9696	0.2118s
	100000	0.9855	1.4051s
	1000000	0.9885	12.9565s
	10000000	0.987	56.2556s
XGBoost in R – via caret, with 5-fold CV, simple cross-validation	100	0.919	2.103497s
	1000	0.972	4.315s
	10000	0.986	22.636s
	100000	0.984	102.629s
	1000000	0.998	506.258s
	10000000	0.992	3056.268s

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

Based on the results, I would recommend using the direct implementation of the xgboost() function with basic cross-validation for most practical applications. This approach significantly reduces computation time compared to using caret with 5-fold cross-validation. For example, running XGBoost directly on a dataset with 10 million observations takes only 56 seconds, whereas caret takes over 50 minutes (3056 seconds). Although caret slightly outperforms in predictive accuracy across all dataset sizes, with improvements ranging between 0.001 and 0.168, the marginal gain (up to 0.2% at maximum size) does not justify the enormous increase in processing time.

This recommendation is especially valuable for large datasets and environments that require frequent model retraining, where speed and efficiency are critical. The direct use of xgboost() consistently delivers excellent computational speed and stable predictive performance, achieving an accuracy of 0.989 with 10 million observations. This speed advantage supports faster model testing and iteration in real-world applications.

When dealing with smaller datasets, both R and Python deliver comparable accuracy, but Python offers faster execution, particularly when running multiple models and tests. Python generally outperforms R in terms of processing speed, making it a more productive choice for both development and deployment.

Overall, Python proves to be the most effective and scalable choice for leveraging XGBoost, particularly for large datasets. Even though R may perform marginally better on small datasets requiring maximum precision, Python’s efficiency and speed make it the preferred option for most real-world use cases.