HDS5230- Week11 Assignment
Wen-shan, Liu

1. Result of comparison:

Method used	Dataset size	Testing-set predictive performance(R-square)	Time taken for the model to be fit (second)
XGBoost in Python via scikit-learn and 5-fold CV	100	0.8125	0.1658
	1000	0.9425	0.6088
	10000	0.9721	1.0953
	100000	0.9857	**1.4942**
	1000000	0.9913	**4.5098**
	10000000	0.993	**43.1511**
XGBoost in R – direct use of xgboost() with simple cross-validation	100	0.85	**0.002245903**
	1000	0.965	**0.01157022**
	10000	0.9745	**0.333565**
	100000	0.9854	3.048962
	1000000	0.9925	30.67973
	10000000	0.9935	234.7912
XGBoost in R – via caret, with 5-fold CV simple cross-validation	100	0.8	0.1411471
	1000	0.8	0.08993077
	10000	0.975	1.876049
	100000	0.9854	15.89559
	1000000	0.9925	155.1312
	10000000	0.9935	1222.516

2.

Based on the results, XGBoost in Python and XGBoost in R (using direct xgboost()) are more competitive compared to the three approaches tested. Their predictive performances are quite similar; however, these two methods require less time to fit the model.

When evaluating both model performance and training time, the following strategy is suggested:
- For tiny datasets (fewer than 100,000 entries), utilize R's direct xgboost() function. This method allows for speedier model fitting while retaining high predicted accuracy.
- For big datasets (more than 1,000,000 records), XGBoost in Python using the scikit-learn interface with 5-fold cross-validation is suggested. This approach dramatically decreases training time while maintaining good predicted accuracy.

In conclusion, the direct usage of xgboost() in R is suitable for smaller datasets because to its speed and efficiency, but Python's scikit-learn version of XGBoost is better suited for large-scale data modeling, providing superior scalability without reducing accuracy.