# Comparison of R/Python for XGBoost

This project evaluated and compared the performance of XGBoost models across three implementations: Python's scikit-learn, R's direct xgboost function, and R's caret package using 5-fold cross-validation.
Using bootstrapped datasets generated from a real-world diabetes dataset, models were assessed on predictive accuracy and computational efficiency across various dataset sizes.
Python scikit-learn consistently achieved perfect accuracy and faster training times, establishing it as the most efficient approach for large-scale modeling tasks.

**Methodology:**

The analysis involved the creation of synthetic datasets through bootstrapping predictor variables individually with replacement from an original medical dataset (Pima Indians Diabetes).
A logistic regression model was first fitted on the original dataset to predict the binary outcome (presence or absence of diabetes).
Using the predicted probabilities from this model, new binary outcomes were assigned to each bootstrapped observation based on a probability threshold of 0.5.

Datasets were generated for sizes:

- 100
- 1,000
- 10,000
- 100,000
- 1,000,000 rows.

Subsequently, XGBoost models were trained and evaluated using three different approaches:

- Python's scikit-learn XGBoost (XGBClassifier with 5-fold cross-validation),
- R's direct xgboost function (xgb.cv with simple cross-validation),
- R's caret-based model training (caret::train using 5-fold cross-validation).

The models' predictive performance (mean accuracy) and the time taken to fit the models were recorded and compared.

**Final Results Table:**

| **Method Used** | **Dataset Size** | **Testing-set predictive performance** | **Time taken for the model to be fit** |
|---|---|---|---|
| **Python (scikit-learn, 5-fold CV)** | 100 | 0.95 | 0.16 |

| | | | |
|---|---|---|---|
| | 1,000 | 1.00 | 0.20 |
| | 10,000 | 1.00 | 0.44 |
| | 100,000 | 1.00 | 4.12 |
| | 1,000,000 | 1.00 | 27.79 |
| **R (direct xgboost(), simple CV)** | 100 | 0.94 | 0.13 |
| | 1,000 | 0.95 | 0.58 |
| | 10,000 | 0.97 | 1.27 |
| | 100,000 | 0.98 | 4.71 |
| | 1,000,000 | 0.99 | 44.34 |
| **R (caret XGBoost, 5-fold CV)** | 100 | 0.92 | 0.82 |
| | 1,000 | 0.95 | 0.86 |
| | 10,000 | 0.95 | 1.83 |
| | 100,000 | 0.95 | 11.11 |
| | 1,000,000 | 0.95 | 103.23 |

**Analysis and Interpretation:**

The results highlight distinct performance differences among the three XGBoost implementation approaches:

- **Python's scikit-learn XGBoost** consistently achieved perfect accuracy (1.00) from datasets of size 1,000 onwards.
  It also exhibited the fastest training times, taking only about 28 seconds even for 1 million rows.
- **R's direct xgboost() function** demonstrated slightly slower performance but still achieved high accuracy, reaching 0.99 at 1 million rows.
  However, training times were roughly double compared to Python, taking approximately 49 seconds for the largest dataset.
- **R's caret XGBoost implementation** showed consistently lower accuracy (~0.95) across all dataset sizes and exhibited the slowest training times.
  Training on 1 million rows using caret took almost 100 seconds, demonstrating significant computational overhead introduced by caret's internal processes.

In summary, while all three approaches successfully scaled with increasing data size, Python offered superior efficiency and faster convergence to optimal model performance.

**Recommendation:**

Based on the experimental results:

- **Python's scikit-learn XGBoost is strongly recommended** for large-scale model training tasks.
  It provides the best combination of high predictive accuracy, minimal computational time, and scalability across increasing dataset sizes.
- **R's direct xgboost** remains a viable option when Python is not available, offering decent accuracy and acceptable training times, but it is relatively slower compared to Python.
- **R's caret XGBoost**, while useful for hyperparameter tuning, is **not recommended for very large datasets** in production environments due to its significant training time overhead without offering proportional gains in model performance.