GLM Investigation

Module/frame work/package	Name and a brief description of the algorithm	An example of a situation where using the provided GLM implementation provides superior performance compared to that of base R or its equivalent in Python (identify the equivalent in Python)
a) Base R (stats package)	GLM (Generalized Linear Model) uses **Iteratively Reweighted Least Squares (IWLS)** to estimate parameters. It supports various distributions (Gaussian, Binomial, Poisson, Gamma, etc.) and link functions. Used for modeling relationships between a response variable and predictors. The function returns detailed model diagnostics, including AIC, deviance, and residual analysis.	GLM in Base R is often superior to statsmodels.GLM in Python when working with small to medium-sized datasets, as it provides built-in methods for model evaluation (e.g., ANOVA, AIC calculation) and has a more optimized memory management for in-memory computations. Additionally, GLM integrates well with R's formula syntax, making model specification more intuitive.
b) Big Data Version of R (bigglm package) & H2O.ai (R)	**BigGLM (bigglm())**: A generalized linear model (GLM) implementation designed for large datasets that do not fit into memory. It processes data in chunks and uses an iterative weighted least squares method, making it efficient for handling streaming or batch-processing data. Unlike base R's glm(), bigglm() does not store the entire dataset in memory but rather updates estimates as new data is read.	When working with datasets that are too large to fit into RAM, bigglm() provides superior performance over base R's glm(). For example, if analyzing healthcare records with millions of patient entries, glm() may run out of memory, while bigglm() efficiently processes the data in chunks. However, since it does not compute standard errors by default, additional steps may be required for full statistical inference.
	Distributed and Parallelized Iterative Weighted Least Squares (IWLS) A high-performance GLM implementation optimized for distributed computing. It supports large datasets, multi-threading, and regularization techniques (LASSO and Ridge regression). Unlike bigglm(), which processes	When working with extremely large datasets that need parallel computation, h2o.glm() is a superior choice. For example, in predictive modeling for hospital readmission rates, where billions of records from multiple sources must be processed, h2o.glm() can handle distributed computing efficiently, significantly reducing processing time compared to glm() and bigglm(). Additionally, it offers built-in

	data in chunks, h2o.glm() distributes the workload across multiple CPUs or even a cluster, making it more suitable for extremely large-scale modeling tasks.	cross-validation, regularization, and feature standardization, which bigglm() does not.
c) Dask ML (dask_ml.linear_model)	Generalized Linear Models in Dask-ML, including LinearRegression, LogisticRegression, and PoissonRegression, offer scalable solutions for large datasets by distributing computation across multiple nodes, making them superior to base R's lm(), glm(), and Scikit-learn's equivalents when dealing with memory-intensive tasks such as financial modeling, fraud detection, and healthcare analytics. Optimization algorithms like **ADMM (Alternating Direction Method of Multipliers), Gradient Descent, L-BFGS, Newton's Method,** and **Proximal Gradient Method** provide efficient solutions for high-dimensional problems, with applications in portfolio optimization, deep learning, NLP, and genomics, outperforming R's optim() and Scikit-learn's solvers in handling large-scale structured data. Regularization techniques like ElasticNet, L1 (Lasso), and L2 (Ridge) help in feature selection and multicollinearity reduction, making them ideal for finance, bioinformatics, and image classification, where Dask-ML's distributed computing approach outperforms R's glmnet() and Scikit-learn's implementations by handling high-dimensional datasets efficiently.	Dask ML's GLM implementations are optimized for large-scale data and can handle distributed datasets, making them more efficient for big data problems compared to base R or Python's sklearn implementations. For example, in Python, the sklearn.linear_model.LogisticRegression() may struggle with large datasets, especially when fitting on datasets that don't fit into memory. Dask ML's distributed approach allows it to efficiently scale across a cluster, processing large datasets in parallel, which is especially beneficial when the data is too large to fit in memory on a single machine. Base R implementations like glm() also face similar limitations when scaling to large datasets, whereas Dask ML can handle them with distributed computing resources, offering a significant performance boost.

d) Spark R (spark.glm)	Generalized Linear Models (GLM) – SparkR provides a scalable implementation of generalized linear models like Logistic Regression, Poisson Regression, and Gaussian families, working directly with Spark DataFrames for distributed processing. It supports various families **(binomial, gaussian, Gamma, poisson, tweedie)** and allows tuning of hyperparameters such as regularization (L2), convergence tolerance, and number of iterations. Users can fit the model, predict new data, and save/load models using write.ml and read.ml.	SparkR outperforms base R's glm() or Python's sklearn.linear_model.LogisticRegression() when dealing with large datasets that do not fit into memory. For example, when working with a dataset in a distributed environment, SparkR can leverage Spark's distributed computing capabilities, enabling faster and more efficient training compared to the memory-limited single-machine R or Python implementations. In scenarios where data is large (e.g., millions of rows), SparkR's ability to scale on a cluster gives it a significant performance advantage over base R or sklearn, which might struggle with large data or require significant memory optimizations.
e) Spark Optimization (MLlib)	**GLM in Spark:** Distributed machine learning with GLMs using Spark's MLlib library for scalable model training. **Gradient Descent and L-BFGS Optimization** – This framework includes both Gradient Descent and L-BFGS (Limited-memory BFGS) algorithms for solving optimization problems. Gradient Descent is a first-order optimization method, useful for large-scale problems, while L-BFGS is a quasi-Newton method that approximates the second-order derivatives to speed up convergence. Both methods are designed for distributed computation and are highly suitable for machine learning tasks.	Spark MLlib outperforms base R or Python's optimization methods, such as R's optim() or Python's scikit-learn's gradient descent implementations, when dealing with large datasets in a distributed environment. For example, when optimizing a logistic regression model on a dataset too large to fit in memory, Spark's distributed optimization allows it to scale efficiently across clusters, reducing computation time compared to running gradient descent on a single machine. In contrast, base R or Python implementations would require substantial memory optimization or may not be able to handle such large-scale datasets.
f) Scikit-learn	**Linear Regression (OLS)**: Ordinary Least Squares regression used to fit a linear model by	Superior Performance: In large datasets, Scikit-learn's OLS is optimized for speed using direct solvers like QR decomposition,

	minimizing squared residuals between observed and predicted values.	whereas base R relies on more memory-intensive methods, resulting in slower performance with large datasets.
	Ridge Regression: L2 regularized linear regression that prevents overfitting by penalizing large coefficients.	Ridge regression in Scikit-learn is highly optimized with solvers like **'lbfgs'** and **'saga'**, making it faster for large-scale problems than the default **lm()** function in R, which may become slow with high-dimensional datasets.
	Lasso Regression: L1 regularized linear regression which drives some coefficients to zero, performing automatic feature selection.	Scikit-learn's Coordinate Descent algorithm for Lasso is more efficient for sparse data, which outperforms base R's glmnet implementation, especially when handling high-dimensional data, such as in text classification tasks where features are sparse.
	ElasticNet Regression: Combines L1 and L2 penalties to balance feature selection and shrinkage.	Scikit-learn's **ElasticNet** implementation is optimized for parallel processing (using **'saga'** solver), which provides faster training on large datasets compared to R's **glmnet** package, especially when the dataset contains both correlated and uncorrelated features.
	Logistic Regression: Used for binary classification, fits a model by maximizing the likelihood of the data.	Scikit-learn's **'lbfgs'** and **'saga'** solvers provide faster convergence for larger datasets compared to R's **glm()** function with logistic regression, which can be slower in optimization for large or sparse datasets.
	Poisson Regression: A GLM used for count data, typically when the response variable follows a Poisson distribution.	Scikit-learn's **Poisson Regression** implementation is well-optimized and handles larger datasets more efficiently compared to **base R's glm()** with a Poisson family, which can become memory-intensive for big data.