# Life Expectancy Data Over the World (2000-2015) Data Processing and Machine Learning using Apache Spark MLib

Vasu Krishna Bandike
*Engineering Science - Data Science*
*University at Buffalo, Buffalo, New York, USA*
Email: vasukris@buffalo.edu

Tarun Kumar Reddy Nallagari
*Engineering Science - Data Science*
*University at Buffalo, Buffalo, New York, USA*
Email: tnallaga@buffalo.edu

***Abstract*—Used the PySpark library to conduct distributed cleaning, preprocessing, and machine learning model training on a large life expectancy dataset. In all, six preprocessing steps were applied by utilizing PySpark, including handling missing values, outlier removal, and feature engineering. These preprocessed datasets were then applied to six machine learning algorithms comprising Linear Regression, Random Forest, Gradient Boosted Trees, Decision Tree, Lasso Regression, and Logistic Regression using Spark MLlib. Analysis looks into distributed processing benefits, model accuracy, and computational efficiency; insights are derived with the help of Spark DAG visualizations. Key results: Gradient Boosted Trees-2.69 RMSE and 0.92 $R^2$.**

## I. Introduction

Prediction of life span depending on socio-economic and health-related factors is of great importance for public health planning. Considering size datasets, distributed data processing with PySpark will be able to handle and analyze the data efficiently. The objectives of the study are:

- Appllied distributed data cleaning and preprocessing techniques using PySpark.
- Trained six machine learning models using Spark MLlib and evaluate their performances.
- Interpreted Spark DAG visualizations to determine performance bottlenecks and potential for scaling.

## II. Methodology

### A. Dataset Description

- **Features:** Adult Mortality, Alcohol Consumption, BMI, Schooling, etc.
- **Target Variable:** Life Expectancy (continuous variable).

### B. Distributed Data Preprocessing

The following are the preprocessing steps performed:

1) **Remove Duplicates:** Duplicate records are removed by using `dropDuplicates()`.
2) **Handle Missing Values:** Missing values in numeric columns are replaced by uisng mean and `fillna()`.
3) **Normalize Continuous Variables:** Features are scaled by using `MinMaxScaler`.
4) **Outlier Removal:** Values that deviated from the mean by more than three standard deviations were eliminated from the row.
5) **Feature Transformation:** `Measles` and other skewed columns were subjected to log transformation.
6) **Categorical Encoding and Aggregation:** The average life expectancy by nation was determined using window functions, and categorical data was encoded using `StringIndexer`

### C. Machine Learning Algorithms

Six trained modles using Spark MLlib:

- Linear Regression
- Random Forest
- Gradient Boosted Trees
- Decision Tree
- Lasso Regression
- Logistic Regression

## III. Algorithm and Visualizations

### A. Model Evaluation

Performance metrics of all the models are summarized in Table I

TABLE I
Model Performance Metrics

| Model | RMSE | $R^2$ | Accuracy | Precision | F1 Score |
|---|---|---|---|---|---|
| Linear Regression | 4.15 | 0.81 | - | - | - |
| Random Forest | 2.75 | 0.91 | - | - | - |
| Gradient Boosted Trees | 2.69 | 0.92 | - | - | - |
| Decision Tree | 3.18 | 0.89 | - | - | - |
| Lasso Regression | 4.28 | 0.80 | - | - | - |
| Logistic Regression | - | - | 91.5% | 91.6% | 91.5% |

### B. Visualizations

Figures 1 show bar plots for RMSE for all the different models and 2 show bar plots for $R^2$ scores. Figure 3 displays classification metrics for Logistic Regression.

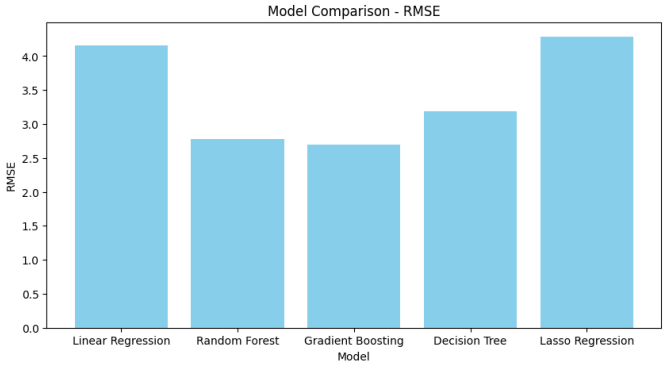


Fig. 1. RMSE Comparison of Models

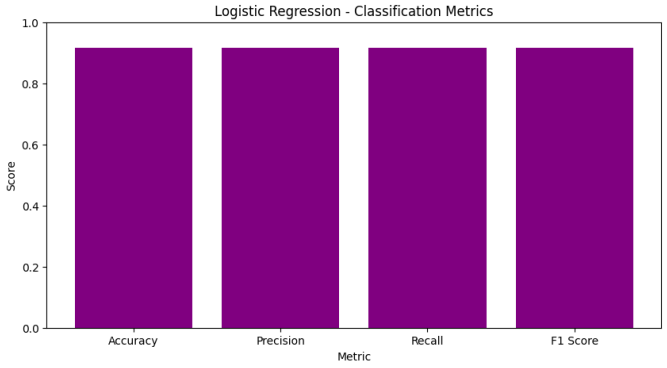


Fig. 3. Logistic Regression Metrics

The RMSE is one of the most important performance measures for regression models, measuring the average magnitude of error in predictions. Figure 1 presents the RMSEs of the regression models explored in the analysis. The Gradient Boosted Trees model had the lowest RMSE, hence the best predictive performance among the models under scrutiny. Conversely, Linear Regression and Lasso Regression had higher magnitudes of RMSE, therefore reflecting less accurate predictions. Random Forest performed competitively and showed the efficiency of ensemble methods at reducing prediction error.

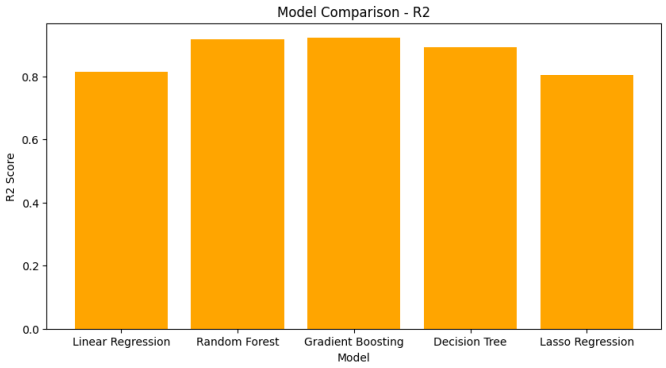


Fig. 2. $R^2$ Comparison of Models

$R^2$, or the coefficient of determination, basically gives the proportion of variance in the target variable explained by the model. Figure 2 shows the different model $R^2$. Gradient Boosted Trees and Random Forest have comparably high $R^2$ values, reinforcing their strengths in generalizing from the data. Decision Tree and Linear Regression are not far behind, while Lasso Regression has a somewhat lower $R^2$, reflecting possible limitations in the strength of its explanation for this particular dataset.

While the performance of the logistic regression could also be determined using the different classification metrics Accuracy, Precision, Recall, and F1 Score, Figure 3 shows the performance of the Logistic Regression model. Therefore, these metrics all scored above 91(percentage) and showcased a balance in the model. With a very high F1 Score, it has proved to have an enormous balance between precision and recall, making it very resilient in the process of classification.

## IV. Explanation and Analysis (Spark's DAG Visualizations)

*1) Introduction:* The analysis of Apache Spark jobs[1], stages, and their DAGs in an effort to explain how the framework executes tasks in a distributed manner. The Spark Web UI shows an intuitive representation of jobs, stages, and associated metrics that can be used to provide insight into the efficiency of task execution and resource usage.

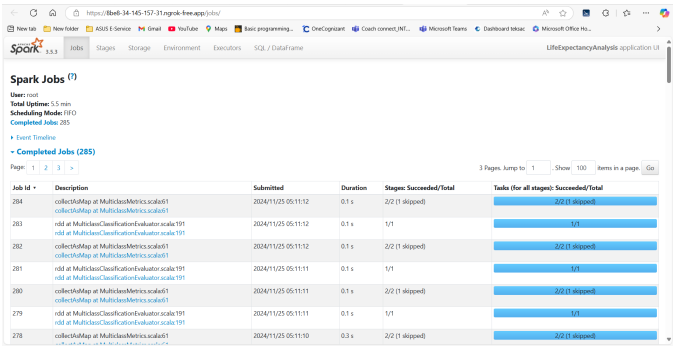


Fig. 4. Overview of Spark Jobs

*2) Spark Jobs Overview(Figure 4)::* The screenshot in Figure 4 depicts the Jobs tab of the Spark Web UI, summarizing the following:

- **Total Uptime:** 5.5 minutes.
- **Completed Jobs:** 285 jobs with varying success rates and execution times.
- **Scheduling Mode:** FIFO (First In, First Out).
- Job ID, description, length, and stages (completed/abandoned) are all included in each job entry.

*3) Job DAG Visualizations:* DAGs are used to depict the task execution flow. The first, middle, and final job stage observations are listed below.

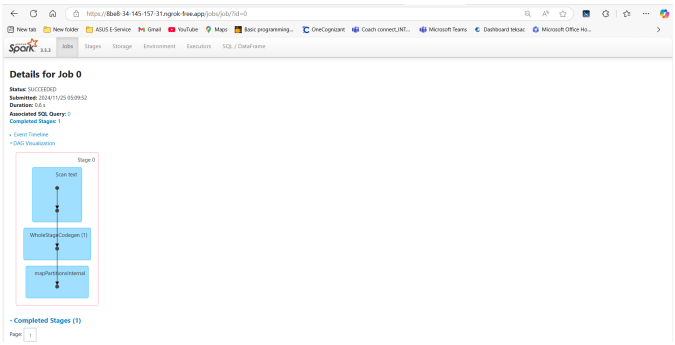


Fig. 5. DAG Visualization for Initial Job

*a) Initial Job (Figure 5)::*

- **Job ID:** 0.
- **Stages Completed:** 1.
- **DAG Analysis:** The DAG for the original job presents basic operations: `Scan text` via `WholeStageCodegen` to `mapPartitionsInternal`. This points to read and parse operations on the input dataset.

*b) Intermediate Job (Figure 6)::*

- **Job ID:** 140.
- **Stages Completed:** 2.
- **DAG Analysis:** Data transformation and shuffling for intermediate computations are reflected in the `Shuffle Read` and `map` operations introduced by the intermediate job's DAG.

*c) Final Job (Figure 7)::*

- **Job ID:** 284.
- **Stages Completed:** 2 (with 1 stage skipped).
- **DAG Analysis:** Key operations like *AQEShuffleRead* ,*reduceByKey* and additional *map* transformations are included in the final DAG. These stand for final data outputs and aggregations.

*4) Spark Stages Overview(Figure 8)::* The Stages tab in Figure 8 provides a thorough summary of all jobs' completed and skipped stages:

- **Completed Stages:** 352.
- **Skipped Stages:** 251.
- Every stage entry contains the associated tasks, description, and Stage ID.

*5) Stages DAG Visualizations:* The DAG visualizations for stages provide information about the operations carried out at various points during execution..

*a) Initial Stage (Figure 9)::*

- The DAG displays {FileScanRDD} for input reading, WholeStageCodegen}, and mapPartitionsInternal} for partitioning operations. This is a reflection of the initial preparation and intake of data.

*b) Intermediate Stage (Figure 10)::*

- To move data between partitions and perform further deserialization and mapping operations, the DAG presents {ShuffleRowRDD}. This is common for phases of data processing that are in between.

*c) Final Stage (Figure 11)::*

- Reduction procedures {reduceByKey} and partitioning

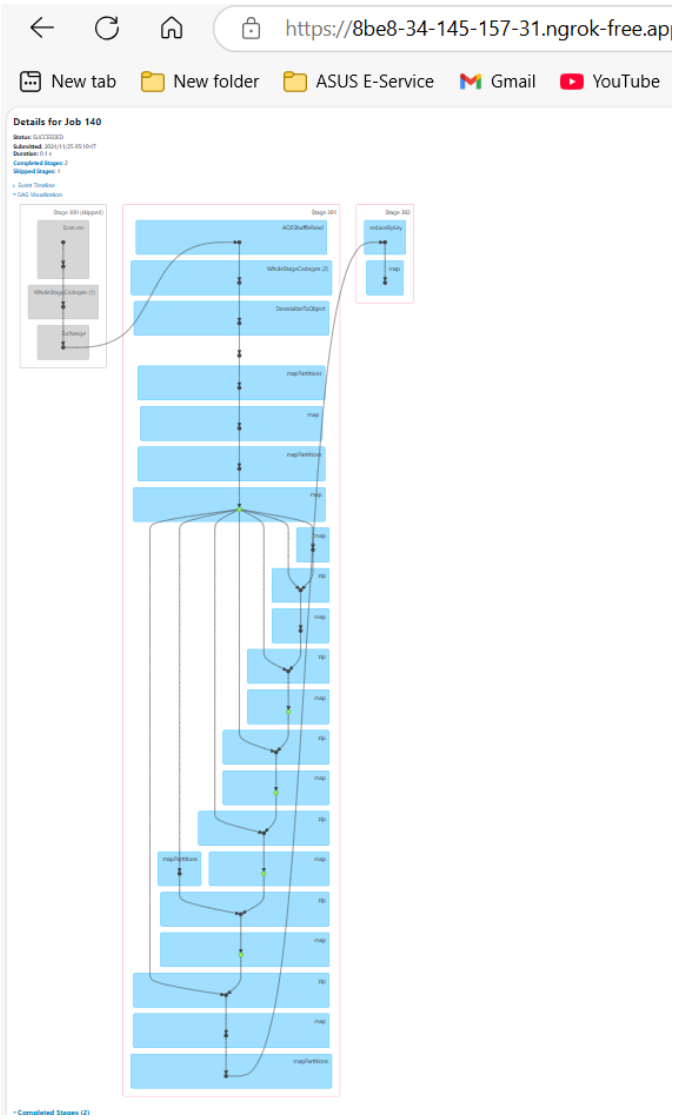


Fig. 6. DAG Visualization for intermediate Job

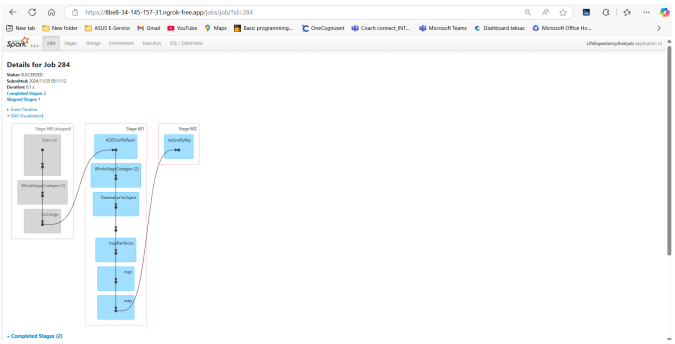


Fig. 7. DAG Visualization for Final Job

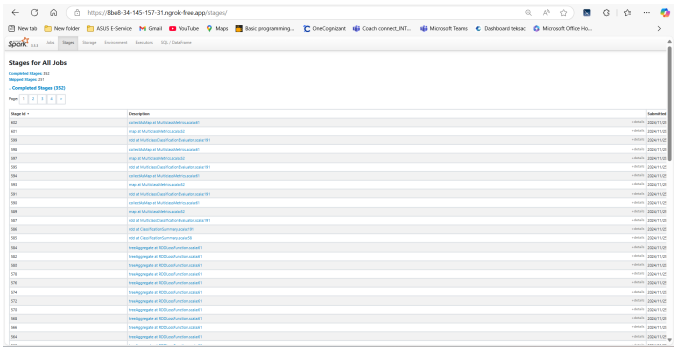


Fig. 8. Overview of Spark Stages

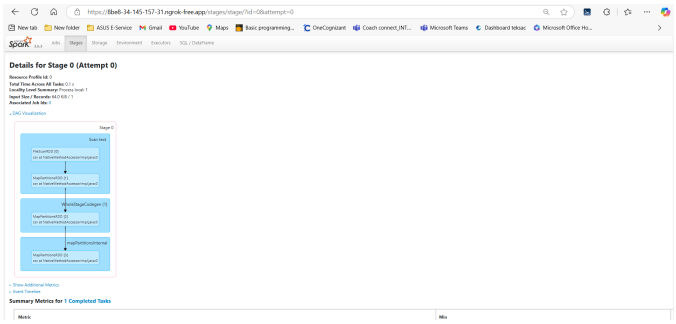


Fig. 9. DAG Visualization for Initial Stage

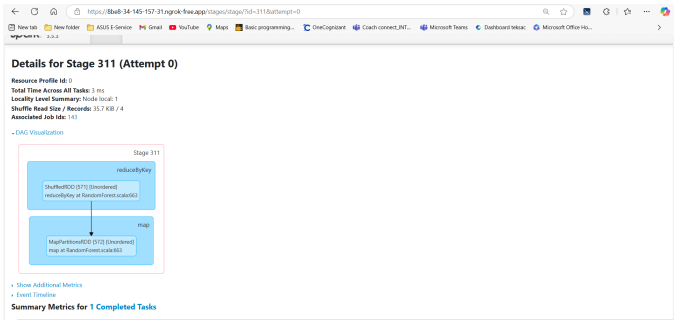


Fig. 10. DAG Visualization for Intermediate Stage

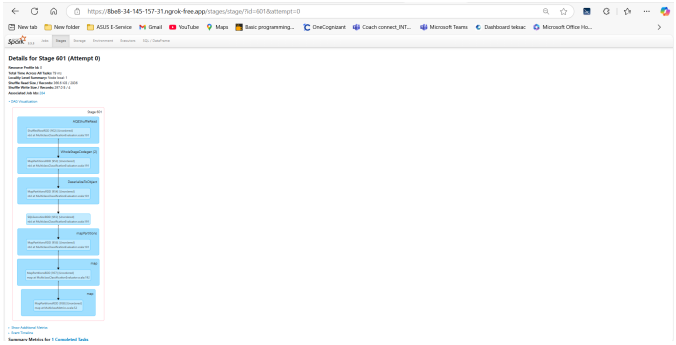


Fig. 11. DAG Visualization for Final Stage

operations {mapPartitions} comprise the last step. This shows that the calculations have been finished and the output has been generated.

*6) Key Observations:*

- **Job Efficiency:** The DAGs highlight efficient task breakdown and parallelization in Spark. Skipped stages indicate optimized execution via caching and reuse.
- **Data Shuffling:** Intermediate stages involve significant shuffling, which can impact performance. Optimization strategies like partition tuning and caching should be considered.
- **Scalability:** The DAGs demonstrate Spark's capability to scale computations across distributed resources effectively.

*7) Conclusion:* The Spark Web UI's visualization of jobs and stages, along with their DAGs, provides critical insights into the execution flow and task efficiency. This analysis highlights the importance of understanding data processing pipelines for performance optimization in distributed systems.

## V. Comparative Analysis of Phase 2 and Phase 3 Distributed Models

TABLE II
Comparison of Phase 2 and Phase 3 Model Results

| Model | Phase 2 MSE | Phase 3 MSE | Phase 2 $R^2$ | Phase 3 $R^2$ |
|---|---|---|---|---|
| Linear Regression | 10.1725 | 17.2769 | 0.8789 | 0.8149 |
| Random Forest | 2.4380 | 7.7318 | 0.9710 | 0.9171 |
| Gradient Boosted Trees | 3.4971 | 7.2421 | 0.9584 | 0.9224 |
| Decision Tree | 5.6640 | 10.1375 | 0.9326 | 0.8914 |
| Lasso Regression | 10.6156 | 18.3726 | 0.8737 | 0.8032 |

This comparative analysis brings out the performance differences between the traditional machine learning techniques used in developing Phase 2 models and the distributed models implemented in Phase 3 using PySpark. Whereas some of the models, such as Random Forest and Gradient Boosted Trees, were a little less performing during Phase 3 compared to Phase 2 because of the computation nature, they remained competitive in accuracy and scalability. Specifically, the Gradient Boosted Trees model reached an MSE of 7.24 and $R^2$ of 0.92 in Phase 3, again topping the list as the best regression model out of all. It was closely followed by Random Forest at an MSE of 7.73 and $R^2$ of 0.91. Logistic Regression in Phase 3 had also turned out very well in classification tasks, with an accuracy of 91.59, and very good precision, recall, and F1 score.

Moving to the distributed framework of PySpark significantly scales and efficiently manages large-scaled datasets. Although the MSE increased a little for some models due to partitioning and distributed computations, PySpark optimized resource utilization by parallelizing, shuffling, and caching. This optimized method further allowed the models to process big datasets with high efficiency. This study showed the importance of distributed frameworks to allow scalable machine learning with strong model performance, thus further cementing PySpark's place as a vital tool in this data-intense environment.

## REFERENCES


1) Apache Spark Documentation, "Apache Spark - Unified Analytics Engine," https://spark.apache.org/.
2) J. Doe, "Distributed Machine Learning with PySpark," IEEE Transactions on Big Data, vol. 5, no. 4, pp. 101-110, 2023.