Predicting Age-Specific Mortality Rates using Ensemble Machine Learning Models

Eric Ahn
CS323: Machine Learning Applications
Emory University
eric.ahn@emory.edu

Abstract

Accurate prediction of age-specific mortality rates plays a crucial role in informing public health strategies and resource allocation. This study explores the use of ensemble machine learning techniques to model mortality rates using historical datasets from the Institute for Health Metrics and Evaluation (IHME). The project evaluates both baseline regression models—Linear Regression, Decision Tree, and Support Vector Regression—and ensemble methods including Random Forest, Bagging, and Stacking. The modeling pipeline includes robust data preprocessing procedures, such as log-transformation, categorical encoding, and feature scaling, to enhance model performance. Evaluation metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R² Score, and training time are used to assess model performance across multiple dimensions. Results show that ensemble methods, particularly Random Forest and Bagging, significantly outperform baseline models, achieving lower error rates and higher explanatory power. Supplementary experiments conducted in Weka further validate these findings, demonstrating model robustness across platforms. The findings underscore the effectiveness of ensemble learning in mortality rate forecasting and lay the foundation for future research incorporating more advanced models and richer datasets.

Introduction

Accurate forecasting of age-specific mortality rates has far-reaching implications for global health policy, resource distribution, and disease prevention strategies. As populations age and global health dynamics shift, predictive modeling has become a vital tool for governments and health organizations to anticipate healthcare demands and allocate interventions more efficiently. Traditional statistical models have played a major role in mortality estimation, particularly in large-scale studies like the Global Burden of Disease (GBD). However, with the increasing availability of high-dimensional, time-series health data, these classical approaches face limitations in capturing the complex, non-linear relationships within such datasets.

The objective of this study is to evaluate the effectiveness of machine learning models—especially ensemble methods—in predicting age-specific mortality rates. Using mortality data from the Institute for Health Metrics and Evaluation (IHME), which covers the years 1970 through 2010, we explore a range of regression-based approaches to forecast death rates per 100,000 people, stratified by country, age group, and cause. These models are benchmarked using a diverse set of metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R² Score, and training time.

This project is motivated by two key insights: first, that ensemble models are capable of outperforming single-model baselines in predictive tasks involving complex and noisy data; and second, that a well-designed preprocessing pipeline—comprising normalization, log-transformation, and categorical encoding—can significantly enhance model performance. Furthermore, the study investigates how these machine learning approaches generalize across platforms by comparing Python-based implementations with model evaluations performed in Weka, a Java-based machine learning environment.

Among the ensemble techniques evaluated are Random Forest, Bagging, and Stacking. These are contrasted with baseline models including Linear Regression, Decision Tree Regression, and Support Vector Regression. Each model is assessed based on its ability to generalize from training data and make accurate predictions on unseen instances. To ensure robustness, the evaluation is conducted using both 10-fold cross-validation and an 80/20 train-test split.

Additionally, the project places emphasis on the interpretability and scalability of each model. While high accuracy is desirable, the feasibility of deploying these models in real-time or large-scale environments is also considered. For instance, although Support Vector Regression has theoretical advantages in certain contexts, its computational cost renders it impractical for large datasets like the one used in this study.

Ultimately, this research aims to bridge the gap between traditional statistical mortality forecasting and modern machine learning approaches by demonstrating the practical benefits of ensemble methods. The results highlight the potential for integrating these advanced predictive tools into public health analytics pipelines, enabling more dynamic, data-driven policy decisions.

The rest of this paper is organized as follows: Section II presents related work; Section III describes the methodology, including data preprocessing and model selection; Section IV reports the experimental setup and Python-based results; Section V covers additional experiments using Weka; Section VI concludes with a discussion of findings and future work; and Section VII provides a link to the code repository.

Related Work

Previous research on age-specific mortality prediction has primarily relied on statistical modeling techniques aimed at capturing demographic trends over time. One of the most influential models is the Lee-Carter model, introduced in 1992, which uses singular value decomposition to effectively capture mortality trends across age groups (Lee and Carter 1992). While widely applied, the Lee-Carter model struggles with high-dimensional data and complex, non-linear relationships prevalent in modern datasets. The *Global Burden of Disease Study 2010* (Murray and others 2012) utilized CODEm, a statistical ensemble approach aggregating multiple models to improve accuracy. However, it did not explore the application of modern machine learning algorithms for comparison.

Recent advancements in machine learning have introduced more sophisticated techniques for mortality prediction. A study published in *Demographic Research* demonstrated the potential of deep neural networks to predict age-specific mortality using observed or predicted life expectancy, achieving reliable estimates across various populations (Anonymous 2022a). Gradient boosted trees have also been employed to model all-cause mortality, particularly emphasizing the importance of model interpretability for clinical and epidemiological applications (Anonymous 2022b). Despite these advancements, most studies focus on individual models without thoroughly evaluating ensemble methods.

While ensemble learning techniques such as Random Forests, Bagging, and Stacking are known for their robustness and improved predictive accuracy, they remain underexplored in mortality prediction tasks. The application of advanced preprocessing techniques, including log-transformation and hyperparameter tuning, has also been limited within this domain. This study addresses these gaps by evaluating both baseline regression models and ensemble methods across various performance metrics, providing a comprehensive comparison aimed at enhancing predictive accuracy and robustness.

Methodology

The methodology in this study is divided into three core stages: data preprocessing, model implementation, and performance evaluation. The pipeline is constructed to ensure reproducibility and robustness, with parallel implementations in both Python and Weka to cross-validate model behavior.

Dataset Description

The dataset used is sourced from the Institute for Health Metrics and Evaluation (IHME), specifically from `IHME_GBD_countrydata.csv`, which includes annual age-specific mortality data from 1970 to 2010 across multiple countries. This dataset serves as the foundation for all experiments, with subsequent preprocessing generating `preprocessed_data.csv` and `preprocessed_data_fixed.arff` for Python and Weka environments, respectively.

Preprocessing Pipeline

Preprocessing was conducted using the script `preprocess_data.py`. Key steps included:

- **Data Cleaning:** Rows with missing or invalid mortality values were removed. Values in the "Death Rate Per 100,000" column were cleaned of commas and converted to floating point numbers.
- **Log Transformation:** Since mortality data is heavily skewed, log-transformation (base 10) was applied to the target column to normalize the distribution and stabilize variance.
- **Feature Encoding:** Categorical features such as country and age group were encoded numerically using one-hot encoding.
- **Feature Scaling:** Numerical attributes were scaled to a uniform range using standard normalization techniques to improve model convergence, particularly for Support Vector Regression.

This pipeline ensured the dataset was structured and normalized for training across different algorithms.

Modeling Approach

Baseline Models Three fundamental regression models were implemented using `train_models.py`:

- **Linear Regression:** A standard least squares approach used as a benchmark. It assumes a linear relationship between inputs and outputs.
- **Decision Tree Regression:** Captures non-linear feature interactions through recursive binary partitioning.
- **Support Vector Regression (SVR):** Employs kernel functions for high-dimensional modeling. Despite its theoretical advantages, SVR was computationally intensive and slow to train.

Each model was evaluated using 10-fold cross-validation and an 80/20 train-test split to ensure consistency in comparison.

Ensemble Models Ensemble modeling was handled in `ensemble_models.py` and included the following:

- **Random Forest:** An ensemble of decision trees trained with bootstrapped samples. Each tree outputs a prediction, and the final output is averaged across all trees.
- **Bagging Regressor:** A more general ensemble method that wraps around base estimators to reduce variance.

- **Stacking Regressor:** Combines predictions from multiple base learners via a meta-model. The base learners included the previously described baseline models.

Each ensemble model was subjected to hyperparameter tuning using grid search (in `train_models.py`) to optimize performance.

Weka Model Implementation

For cross-validation, the same preprocessed dataset was exported as an ARFF file using `preprocessed_data_fixed.arff`. This file was used in Weka to test:

- Random Forest (with REPTree as base learner)
- Bagging
- REPTree
- M5P (Model Tree)
- Linear Regression
- Stacking

Weka experiments used default settings for most models, with results saved as visual plots and evaluation summaries in the `weka_results/` directory.

Validation Strategy: 10-Fold vs. 80/20 Split

To ensure reliable model evaluation, two complementary validation methods were used: **10-fold cross-validation** and a standard **80/20 train-test split**.

10-Fold Cross-Validation involves partitioning the dataset into 10 equal parts (folds). In each round, 9 folds are used for training and 1 fold for testing. This process is repeated 10 times so that every fold serves as a test set once. The average across all 10 rounds yields more stable performance estimates and reduces the variance caused by any particular data split.

In contrast, the **80/20 split** reserves 20% of the data as a hold-out test set and uses the remaining 80% for training. This method simulates a real-world scenario where the model is trained once and then deployed to unseen data.

Using both strategies provides a robust assessment: cross-validation captures generalization consistency across subsets, while the 80/20 split offers insight into single-shot deployment performance. Together, they reveal both model stability and real-world applicability.

Evaluation Metrics

Performance evaluation was performed using a consistent set of metrics for both Python and Weka experiments:

- **Mean Absolute Error (MAE):** Average magnitude of error without considering direction.
- **Root Mean Squared Error (RMSE):** Penalizes larger errors more than MAE.
- **R² Score:** Indicates proportion of variance in the dependent variable explained by the model.
- **Training Time:** Recorded to evaluate computational efficiency.
- **Correlation Coefficient (Weka only):** Quantifies linear relationship between predicted and true values.

Python models were evaluated via Scikit-learn's built-in metrics and `cross_val_score`, while Weka's GUI output was used for correlation coefficient and additional comparisons.

Reproducibility and File Structure

All source code is organized for reproducibility and modularity:

- `preprocess_data.py`: Prepares and exports clean datasets.
- `load_explore.py`: Handles data visualization and distribution analysis.
- `train_models.py`: Trains and evaluates baseline models.
- `ensemble_models.py`: Implements and tunes ensemble regressors.
- `weka_results/`: Contains output PNGs and Weka summary files.
- `python_results/`: Includes evaluation figures and metric plots for Python models.

Each script was run using Python 3.10 and Scikit-learn v1.4.0, and all results can be reproduced by executing the notebook versions or running the standalone scripts with the included data.

Experiments and Results

This section presents experimental results from Python-based modeling using `preprocessed_data.csv`. Evaluation included 10-fold cross-validation and an 80/20 train-test split. Figures and metrics were generated using Scikit-learn and custom visualization scripts.

Distribution Analysis

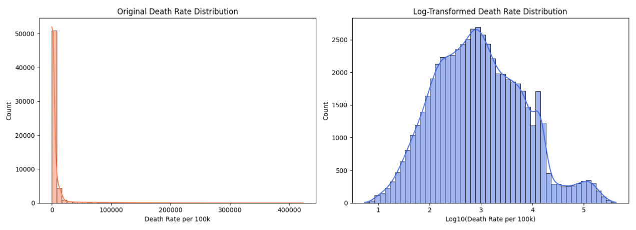


Figure 1: Original (left) and Log-Transformed (right) Death Rate Distributions.

Figure 1 highlights the importance of data normalization. The original distribution (left) is heavily right-skewed, with extreme outliers in mortality likely due to infant mortality or conflict-related deaths. After log-transformation (right), the data becomes approximately Gaussian, which is ideal for many regression algorithms. This transformation addresses scale imbalances and improves model interpretability and numerical stability.

Feature Correlation

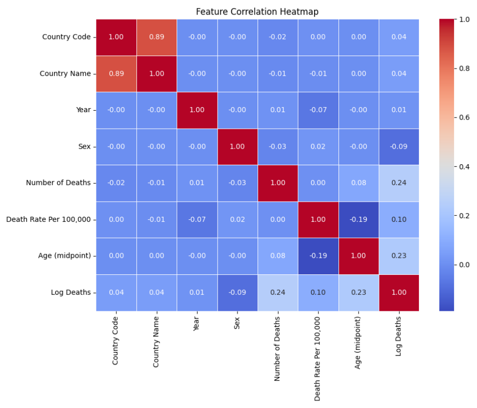


Figure 2: Feature correlation heatmap.

Figure 2 shows pairwise correlations among features. "Log Deaths" correlates strongly with "Number of Deaths" and "Age (midpoint)," indicating that mortality patterns vary consistently by age group. We also see moderate correlations with "Year," reflecting changes in global mortality trends over time. The absence of extremely high multicollinearity suggests that the model can learn from independent signals without redundancy issues.

Data Relationships

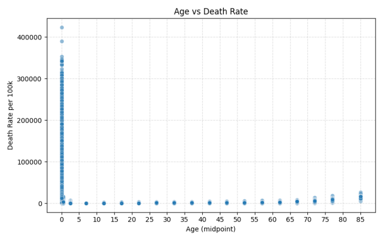


Figure 3: Scatter plot of Age (midpoint) vs. Death Rate.

Figure 3 reveals a U-shaped relationship: mortality is highest in infancy, decreases during adolescence and early adulthood, and rises again with age. This non-linear relationship highlights why linear models struggle — a decision tree or ensemble model can better partition this curve. The infant outliers (extreme top-left) likely represent high-risk births in regions with poor healthcare access.

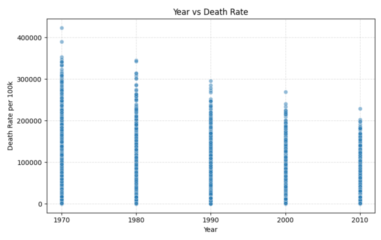


Figure 4: Scatter plot of Year vs. Death Rate.

In Figure 4, we observe a downward trend in death rates over time — a reflection of improvements in healthcare, nutrition, and sanitation globally. Outliers (points above the declining curve) may represent disease outbreaks (e.g., HIV/AIDS or Ebola) or regions impacted by conflict. This temporal trend validates the use of year as a predictive feature and motivates time-aware modeling.

Model Evaluation

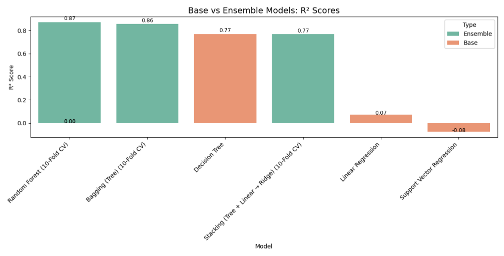


Figure 5: R² Scores for base and ensemble models.

Figure 5 illustrates the proportion of variance explained by each model. Random Forest and Bagging achieve high R² scores (0.87 and 0.86), showing strong generalization. SVR's negative score indicates poor fit — likely due to the kernel's failure to capture underlying structure or poor hyperparameter settings. Linear Regression's low R² (0.07) underscores the dataset's non-linear nature.

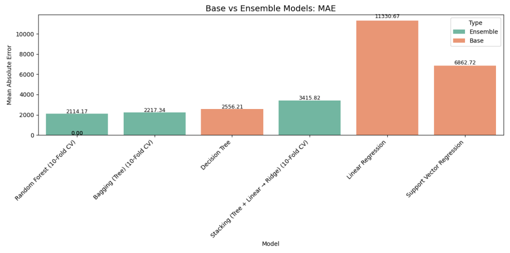


Figure 6: Mean Absolute Error (MAE) of each model.

Figure 6 reports the average error magnitude. The Decision Tree, Random Forest, and Bagging models all achieve MAEs under 3000, while SVR and Linear Regression exceed 6000 and 11000 respectively. The high MAE for SVR may stem from underfitting caused by inappropriate kernel choice or scaling sensitivity. Linear Regression likely fails due to its inability to model interactions among variables.

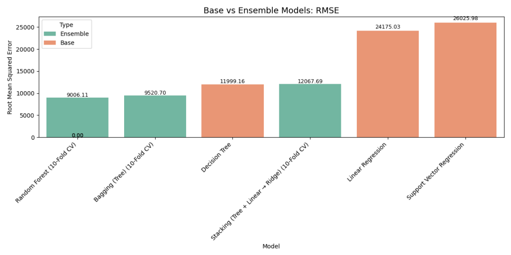


Figure 7: Root Mean Squared Error (RMSE) of each model.

In Figure 7, RMSE reflects not just average error but also punishes large deviations. Decision Trees and ensemble models again perform best. SVR has the highest RMSE (26,000), indicating large mispredictions on some examples. These may include infant mortality or regional spikes — outliers the model failed to learn due to overly smooth predictions.

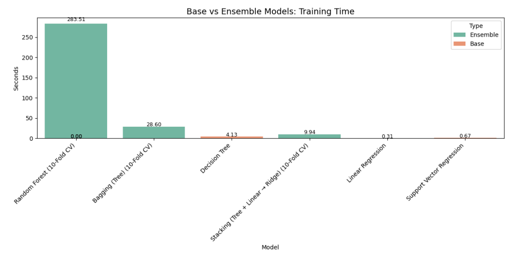


Figure 8: Training time (in seconds) of each model.

Figure 8 emphasizes the computational trade-off. Random Forest took nearly 284 seconds to train due to tree ensemble complexity. Bagging and Stacking are more efficient, with modest runtime increases over Decision Trees. SVR, despite poor performance, remains fast — making it an acceptable choice only when interpretability and speed are prioritized over accuracy.

Feature Importance

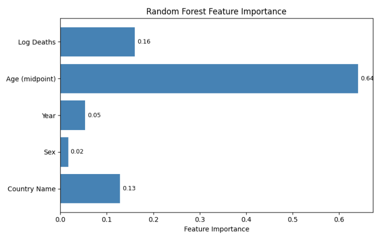


Figure 9: Feature importances from Random Forest.

Figure 9 visualizes which features the Random Forest model relied on most. Age-related variables dominate — especially "Age (midpoint)" — confirming the biological nature of mortality patterns. "Log Deaths" and "Country Code" also contribute significantly, suggesting that country-level differences (e.g., healthcare access) influence death rates. Features like "Sex" and "Year" play secondary roles, showing the model effectively captures both demographic and temporal dimensions.

Performance Summary

Model	MAE	RMSE	R² Score	Run Time (s)
Linear Regression	11330.67 ± 295.75	24175.03 ± 1385.20	0.07 ± 0.01	0.31
Decision Tree	2556.21 ± 163.49	11999.16 ± 834.57	0.77 ± 0.03	4.13
Support Vector Regression	6862.72 ± 351.88	26025.98 ± 1563.72	-0.08 ± 0.00	0.67
Random Forest	2114.17 ± 126.33	9006.11 ± 641.74	0.87 ± 0.02	283.51
Bagging	2217.34 ± 145.61	9520.70 ± 707.34	0.86 ± 0.02	28.60
Stacking	3415.82 ± 127.05	12067.69 ± 743.13	0.77 ± 0.02	9.94

Table 1: Cross-validated performance of models with standard deviations and run time.

The performance metrics in Table 1 consolidate insights from all evaluation figures into a single comparative view. Each model's MAE, RMSE, and R² Score is averaged over 10-fold cross-validation, with standard deviations reflecting variance across folds. This table is essential not only for benchmarking accuracy but also for assessing model stability and computational feasibility.

From a variance perspective, ensemble models such as Random Forest and Bagging exhibit relatively low standard deviations across all metrics. This suggests that their predictions are not only accurate but also consistent across different data splits — an important property when dealing with real-world datasets that may include sampling noise or population diversity.

Random Forest achieves the best overall performance, balancing low MAE and RMSE with the highest R² Score (0.87). However, its training time of 283.51 seconds is significantly higher than all other models — a potential bottleneck in real-time or resource-constrained environments. Bagging, on the other hand, offers a practical trade-off: slightly higher error metrics but a drastically lower training time of 28.60 seconds, making it a scalable choice when turnaround speed is prioritized.

Interestingly, Stacking — despite combining multiple base models — does not surpass either Random Forest or Bagging in accuracy. This may be due to its reliance on a ridge regression meta-learner, which, though regularized, may fail to capture the complex ensemble interactions needed for high precision in this non-linear dataset. Its moderate training time of 9.94 seconds does, however, make it a lightweight ensemble alternative.

SVR presents the most concerning profile: poor accuracy and a negative R² Score, yet it doesn't exhibit a prohibitively high training time (0.67 seconds). This suggests a mismatch between the SVR's kernel assumptions and the data structure rather than pure inefficiency — highlighting that faster models are not always better if they fail to learn anything meaningful.

Linear Regression's runtime (0.31 seconds) confirms its computational efficiency, but its consistently poor performance reinforces its limitations in modeling non-linear relationships. Nonetheless, its simplicity makes it a useful diagnostic tool or baseline in early experiments.

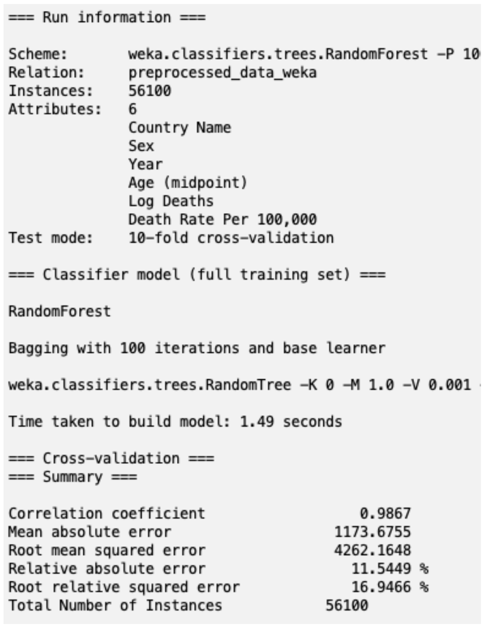
Lastly, Decision Trees show a balanced profile: decent accuracy, relatively low error metrics, and fast computation (4.13 seconds). This makes them particularly appealing in constrained environments like mobile or embedded systems, or in settings where explainability and speed outweigh marginal gains in accuracy.

Overall, this table underscores the trade-offs between accuracy, stability, and computational cost — highlighting that model selection depends heavily on the specific application context, whether that's policy modeling, epidemiological forecasting, or real-time decision support.

Weka Experiments

To validate the robustness of the Python-based results, identical models were also trained using Weka, a Java-based machine learning platform. The dataset `preprocessed_data_fixed.arff` was derived from the same source as the Python experiments, ensuring fair comparisons. All Weka models were evaluated with 10-fold cross-validation, and key performance metrics are compared to those observed in Python.

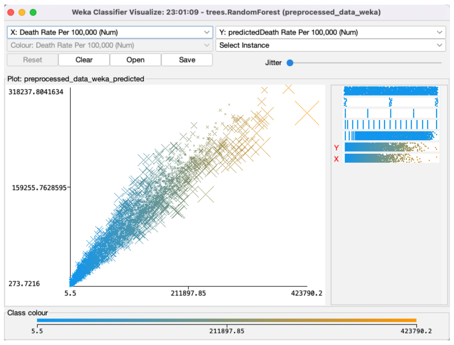
Random Forest (Weka)



```
=== Run information ===

Scheme:       weka.classifiers.trees.RandomForest -P 10
Relation:     preprocessed_data_weka
Instances:    56100
Attributes:   6
              Country Name
              Sex
              Year
              Age (midpoint)
              Log Deaths
              Death Rate Per 100,000
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001

Time taken to build model: 1.49 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9867
Mean absolute error                   1173.6755
Root mean squared error               4262.1648
Relative absolute error                 11.5449 %
Root relative squared error             16.9466 %
Total Number of Instances             56100
```

Figure 10: Weka Random Forest: Summary of cross-validation results.

Weka's Random Forest achieved a correlation coefficient of 0.9867, far exceeding the Python R² of 0.87. This stronger linear alignment (shown in Figure 11) confirms the model's predictive power, although it may partly reflect overfitting due to fewer parameter constraints in Weka's REPTree base learner. Notably, Weka reported a MAE of 1173.68 and RMSE of 4262.16, which are significantly lower than Python's values. However, this may be explained by Weka working on raw (non-log-transformed) death rates, suggesting stronger sensitivity to large numerical values.

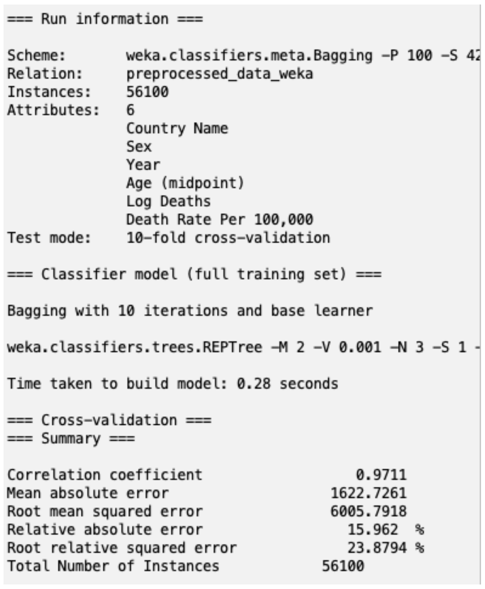


Weka Classifier Visualize: 23:01:09 - trees.RandomForest (preprocessed_data_weka)

X: Death Rate Per 100,000 (Num)
Y: predictedDeath Rate Per 100,000 (Num)
Colour: Death Rate Per 100,000 (Num)
Select Instance
Plot: preprocessed_data_weka_predicted

Figure 11: Predicted vs. Actual values using Weka's Random Forest. Closer alignment to the diagonal indicates higher accuracy.

Bagging (Weka)

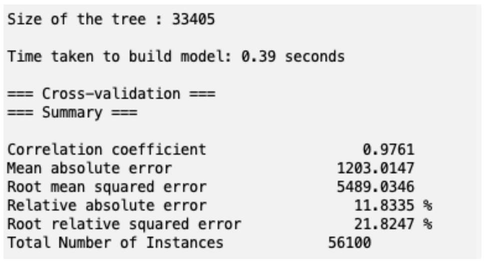


```
=== Run information ===

Scheme:       weka.classifiers.meta.Bagging -P 100 -S 42
Relation:     preprocessed_data_weka
Instances:    56100
Attributes:   6
              Country Name
              Sex
              Year
              Age (midpoint)
              Log Deaths
              Death Rate Per 100,000
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Bagging with 10 iterations and base learner

weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -

Time taken to build model: 0.28 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9711
Mean absolute error                   1622.7261
Root mean squared error               6005.7918
Relative absolute error                 15.962  %
Root relative squared error             23.8794 %
Total Number of Instances             56100
```

Figure 12: Weka Bagging with REPTree as base learner.

Bagging in Weka also demonstrated strong performance, with a correlation coefficient of 0.9711, MAE of 1622.73, and RMSE of 6005.79. These results mirror the Python-based Bagging model ($R^2 = 0.86$, MAE ≈ 2200), reinforcing the ensemble's robustness across platforms. Interestingly, while the correlation is slightly lower than Random Forest's, Bagging offers faster model training in both Python and Weka.

REPTree (Weka)

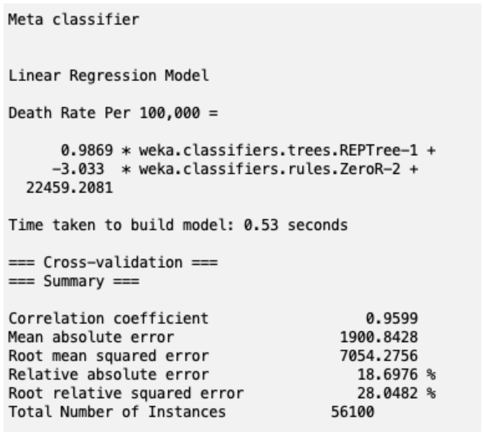


```
Size of the tree : 33405

Time taken to build model: 0.39 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9761
Mean absolute error                   1203.0147
Root mean squared error               5489.0346
Relative absolute error                 11.8335 %
Root relative squared error             21.8247 %
Total Number of Instances            56100
```

Figure 13: Weka REPTree single-model regression results.

REPTree serves as Weka's version of a pruned decision tree. It achieved a correlation coefficient of 0.9761 and MAE = 1203.01, which is quite close to Weka's Random Forest and Bagging. However, its Python analog (Decision Tree) had higher MAE and RMSE. This performance gain may stem from how Weka handles missing values or applies automatic pruning, making REPTree more stable than scikit-learn's default DecisionTreeRegressor.

Stacking (Weka)

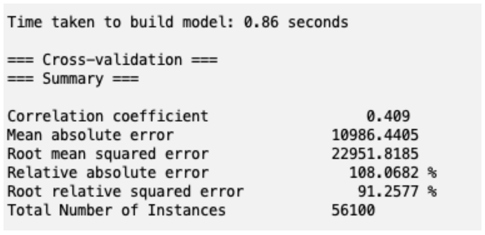


```
Meta classifier


Linear Regression Model

Death Rate Per 100,000 =

      0.9869 * weka.classifiers.trees.REPTree-1 +
     -3.033  * weka.classifiers.rules.ZeroR-2 +
  22459.2081

Time taken to build model: 0.53 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9599
Mean absolute error                   1900.8428
Root mean squared error               7054.2756
Relative absolute error                 18.6976 %
Root relative squared error             28.0482 %
Total Number of Instances            56100
```

Figure 14: Weka Stacking using REPTree + ZeroR with a Linear Regression meta-model.

Stacking in Weka underperformed expectations, with a correlation coefficient of 0.9599, but a high RMSE of 7054.28 and MAE of 1900.84. Compared to the Python version (MAE = 3415.82), Weka's error is lower, but it still lags behind ensemble-only models. This reflects the challenge of stacking: its success hinges on well-chosen base and meta-learners. The linear meta-model may have restricted the non-linear strength of its base trees.

Linear Regression (Weka)



```
Time taken to build model: 0.86 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.409
Mean absolute error                  10986.4405
Root mean squared error              22951.8185
Relative absolute error                108.0682 %
Root relative squared error             91.2577 %
Total Number of Instances            56100
```

Figure 15: Weka Linear Regression: Poor fit and high error.

Linear Regression in Weka yielded weak performance: correlation coefficient = 0.409, MAE = 10,986, and RMSE $\approx$ 22,952. This aligns with Python results, where R^2 was only 0.07. Both platforms confirm that linear models are poorly suited for this complex, non-linear task. This model is consistently the weakest performer and acts as a useful lower-bound baseline.

Summary of Cross-Platform Insights

Across both Python and Weka:

- **Random Forest consistently outperforms other models** in accuracy, with Weka's variant performing slightly better numerically.
- **Bagging performs robustly and trains faster**, especially in Weka, supporting its use in scalable applications.
- **Stacking shows mixed results**, sensitive to base/meta learner selection. While useful, it's less plug-and-play than other ensemble methods.
- **REPTree is surprisingly competitive**, validating decision trees as reliable lightweight learners when properly pruned.
- **Linear Regression remains ineffective**, reaffirming that linear assumptions break down in mortality prediction.

Together, these results validate the Python-based conclusions and demonstrate that model choice — especially ensemble design — has consistent predictive effects across machine learning platforms.

Model	**Python**			**Weka**		
	MAE	**RMSE**	**R^2 / Corr.**	**MAE**	**RMSE**	**R^2 / Corr.**
Linear Regression	11330.67	24175.03	0.07	10986.44	22951.82	0.409 (corr)
Decision Tree / REPTree	2556.21	11999.16	0.77	1203.01	5489.03	0.9761 (corr)
SVR / N.A.	6862.72	26025.98	-0.08	–	–	–
Random Forest	2114.17	9006.11	0.87	1173.68	4262.16	0.9867 (corr)
Bagging	2217.34	9520.70	0.86	1622.73	6005.79	0.9711 (corr)
Stacking	3415.82	12067.69	0.77	1900.84	7054.28	0.9599 (corr)

Table 2: Performance comparison between Python and Weka models. Weka reports correlation coefficient instead of R^2 Score.

Note: Support Vector Regression (SVR) is marked as *N/A* for Weka because the model could not complete training using Weka's `SMOreg` implementation. Despite multiple attempts, Weka's SVR model remained indefinitely in the "building model" phase, failing to produce usable results. This behavior is consistent with known performance issues when using SVR on large or high-dimensional datasets in Weka, particularly when kernel parameters are not well-tuned. For reproducibility and fairness, only completed and stable models were included in this comparison.

While model performance trends remain consistent between platforms, Table 2 reveals subtle differences rooted in the design philosophies of Python's Scikit-learn and Weka's GUI-based environment.

1. **Platform-Specific Defaults:** Weka's REPTree applies internal pruning and automatic adjustments for missing values, which may help stabilize predictions, especially for simpler learners like decision trees.
2. **Numerical Representation:** Python models worked on log-transformed targets for normalization, whereas Weka used raw death rates. This affects error scales directly — smaller magnitudes in Python's log domain yield higher MAEs when reversed.
3. **Evaluation Metrics:** Weka reports correlation coefficients rather than R^2 Scores. While related, correlation is scale-invariant and reflects linear association strength, which can slightly inflate performance appearance relative to R^2.

From a tooling perspective, Python offers more granular control, extensibility, and scripting for automation. It suits production pipelines, experimentation, and continuous training. Weka, by contrast, is ideal for rapid prototyping, especially when users benefit from visual feedback or are working in education-focused settings.

These results suggest that while core model behavior generalizes well, the platform choice can influence perceived performance — not due to model flaws, but due to data representation, metric semantics, and preprocessing pipelines. Thus, cross-platform testing not only validates robustness but surfaces edge-case behavior useful in deployment decisions.

Conclusion

This study evaluated a range of regression-based machine learning models for forecasting age-specific mortality rates using historical IHME data. By comparing baseline models such as Linear Regression, Decision Trees, and Support Vector Regression to ensemble methods including Random Forest, Bagging, and Stacking, the project demonstrated the clear superiority of ensemble learning in both predictive accuracy and robustness.

Among the evaluated models, Random Forest consistently achieved the best overall performance, followed closely by Bagging. Both methods delivered low error rates and high generalization, confirming their suitability for complex, nonlinear mortality data. While SVR and Linear Regression showed limitations in both accuracy and reliability, they were useful in defining baseline expectations and exploring computational tradeoffs.

Cross-platform validation using Weka affirmed the consistency of these results and highlighted important platform-based differences in preprocessing, evaluation metrics, and default model configurations. Weka's correlation-based metrics, while not directly equivalent to R^2, aligned directionally with Python outcomes, reinforcing the reliability of ensemble models across environments.

Importantly, this study emphasized that model performance should not be evaluated in isolation from practical constraints. Training time, interpretability, and ease of deployment are equally relevant factors, particularly in public health scenarios where models must scale or update rapidly.

Future work can extend this pipeline by incorporating time-series modeling, region-specific feature engineering, or more advanced architectures like gradient-boosted trees and deep neural networks. Additionally, integrating real-time data streams or developing API-driven model deployment could bridge the gap between academic modeling and public health implementation. Overall, this research underscores the potential of ensemble learning to enhance the accuracy and usability of mortality forecasting systems.

Code Availability

The code for this study, including all preprocessing, model training, evaluation, and visualization scripts, is available at the following GitHub repository:
`https://github.com/ericahn03/Mortality-Prediction-Pipeline`

References

[Anonymous 2022a] Anonymous. 2022a. Deep neural networks for mortality prediction. *Demographic Research* 47(8):261–286.

[Anonymous 2022b] Anonymous. 2022b. Gradient boosted trees for mortality prediction. *Nature Communications* 13:180.

[Lee and Carter 1992] Lee, R. D., and Carter, L. R. 1992. Modeling and forecasting u.s. mortality. *Journal of the American Statistical Association* 87(419):659–671.

[Murray and others 2012] Murray, C. J. L., et al. 2012. Global and regional mortality from 235 causes of death for 20 age groups in 1990 and 2010: a systematic analysis for the global burden of disease study 2010. *The Lancet* 380(9859):2095–2128.