HDPSA Milestone 3

Bin 381 Group A

10/3/2025

Members:

Llewellyn Jakobus Fourie – 601314

Juan Oosthuizen – 600161

Erin David Cullen – 600531

Qhamaninande Ndlume – 601436

Table of Contents

Task 1: Model Selection and Justification

Selected Model: Random Forest Regression

Random Forest was selected as the primary model for predicting continuous health outcomes in South African health datasets.

Justification

Data Compatibility

- **Dataset size:** 609 records with 11 features is optimal for ensemble methods.
- Random Forest handles moderate-sized datasets effectively without overfitting risks common in deep learning approaches.
- The ensemble approach provides stable predictions even with limited data.

Feature Handling

- **Mixed data types:** Dataset contains both categorical (health indicator types, provinces) and numerical features (sample sizes, scaled values).
- Random Forest natively handles mixed feature types without requiring extensive preprocessing.
- No assumptions about feature distributions or linear relationships required.

Interpretability

- **Variable importance metrics:** Random Forest provides built-in feature importance scores (%IncMSE, IncNodePurity).
- Critical for health policy contexts where stakeholders need to understand which indicators drive outcomes.
- Enables evidence-based resource allocation and intervention prioritization.

Robustness

- **No distribution assumptions:** Unlike linear regression, Random Forest is non-parametric.
- **Handles missing data:** Can maintain accuracy even with incomplete records.
- **Out-of-bag validation:** Internal cross-validation mechanism provides unbiased error estimates without requiring separate validation sets.
- **Resistant to outliers:** Ensemble voting reduces influence of extreme values common in health surveys.

Technical Assumptions Validation

Sample Size Adequacy

- 609 records split into 75% training (457 records) provides sufficient data for tree construction.
- Each tree in the forest uses bootstrap samples, increasing effective training diversity.

Feature Relevance

- 11 features provide adequate dimensionality for tree splits without causing high-dimensional issues.
- Default mtry parameter (sqrt of features) ensures sufficient feature diversity across trees.

Target Variable Properties

- Continuous target (value_log_scaled) is appropriate for regression Random Forest.
- Log-scaling addresses potential skewness in health outcome distributions.

Task 2: Random Forest Test Design

Data Splitting Strategy

Proposed Splits

- **Total records:** 609
- **Training Set:** 75% (457 records)
- **Testing Set:** 20% (122 records)
- **Validation Set:** 5% (30 records)

Rationale

- 75% training split provides sufficient data for Random Forest pattern learning
- 20% test set ensures reliable performance estimates
- 5% validation set for final model verification

Sampling Method

Stratified sampling by health indicator categories ensures: - Representative distribution across all subsets - Maintained proportions of health outcome ranges

Implementation

- Random sampling with seed = 42 for reproducibility
- Data partitioned into training, test, and validation sets

Evaluation Metrics

Primary Metrics

RMSE (Root Mean Squared Error) - Measures average prediction error in original units - Penalizes larger errors more heavily - Critical for health outcomes where large errors impact policy decisions

MAE (Mean Absolute Error) - Average absolute difference between predictions and actual values - More interpretable than RMSE - Easy to explain to health policy stakeholders

R-Squared (R^2) - Proportion of variance explained by the model (0-1 scale) - Shows how well the model captures health outcome patterns

Random Forest Specific Metrics

OOB (Out-of-Bag) Error - Built-in validation metric for Random Forest - Uses samples not included in bootstrap samples of each tree - Provides unbiased error estimate without separate validation set

Variable Importance Scores - Ranks features by their contribution to predictions - Identifies key health indicators for policy focus

Validation Strategy

Cross-Validation Approach

5-fold cross-validation balances computational cost with reliability: - Each fold contains approximately 85 records - Produces five independent performance estimates - Reduces reliance on single train-test split

Purpose

- Detect potential overfitting during training
- Enable reliable hyperparameter tuning across folds
- Confirm model generalization to unseen data

Implementation

- Apply 5-fold cross-validation on training dataset only
- Compute performance metrics for each fold
- Report mean and standard deviation across folds
- Monitor consistency as indicator of stability

Quality Framework

OOB Error Convergence Analysis

Assesses whether sufficient trees are used in the Random Forest model.

Procedure: - Plot OOB error rate against number of trees - Check error stabilization as trees increase - Identify convergence point where additional trees provide no meaningful gain

Outcome: Confirms model has sufficient trees without unnecessary complexity.

Cross-Validation Performance Stability

Evaluates model consistency across multiple data partitions.

Procedure: - Use k-fold cross-validation on training and validation folds - Record performance metrics across folds - Calculate coefficient of variation (CV) to quantify stability

Outcome: Low variability indicates strong generalization; high variability suggests overfitting or data imbalance.

Prediction Interval Assessment

Verifies model predictions align with domain knowledge and remain realistic.

Procedure: - Construct prediction intervals for model outputs - Inspect for implausible values (e.g., negative counts, percentages >100%) - Cross-check against health outcome constraints (e.g., immunization rates: 0-100%)

Outcome: Ensures predictions are interpretable and practically meaningful for health indicators.

Setup and Data Loading

```r
library(tidyverse)

data_path <- "../Data/03_Scaled/modeling_features.csv"
output_path <- "../Data/04_Split"

if (!dir.exists(output_path)) {
  dir.create(output_path, recursive = TRUE)}

data <- read.csv(data_path)
```

Data Splitting

```r
# Set seed for reproducibility
set.seed(42)

# Calculate split sizes (75% train, 20% test, 5% validation)
n <- nrow(data)
train_size <- floor(0.75 * n)
test_size <- floor(0.20 * n)
val_size <- n - train_size - test_size

# Generate random indices and partition data
indices <- sample(1:n)
train_idx <- indices[1:train_size]
test_idx <- indices[(train_size + 1):(train_size + test_size)]
val_idx <- indices[(train_size + test_size + 1):n]

# Create split datasets
train_data <- data[train_idx, ]
test_data <- data[test_idx, ]
val_data <- data[val_idx, ]

# Display split dimensions

## Training set: 560 records (75.0%)

## Test set: 149 records (19.9%)

## Validation set: 38 records (5.1%)
```

Save Split Data

```r
write.csv(train_data, file.path(output_path, "train_data.csv"), row.names = FALSE)
write.csv(test_data, file.path(output_path, "test_data.csv"), row.names = FALSE)
write.csv(val_data, file.path(output_path, "val_data.csv"), row.names = FALSE)
```

Task 3: Random Forest Implementation

Setup

```
library(randomForest)
library(tidyverse)

train_path <- "../Data/04_Split/train_data.csv"
test_path <- "../Data/04_Split/test_data.csv"
outputs_path <- "Task_03/outputs"

if (!dir.exists(outputs_path)) {
  dir.create(outputs_path, recursive = TRUE)
}

train_data <- read.csv(train_path)
test_data <- read.csv(test_path)
```

Baseline Model

```
# Train baseline Random Forest with default parameters
# ntree = 500: Number of trees in the forest
# importance = TRUE: Calculate variable importance scores
baseline_rf <- randomForest(
  value_log_scaled ~ .,
  data = train_data,
  ntree = 500,
  importance = TRUE
)

print(baseline_rf)

##
## Call:
##  randomForest(formula = value_log_scaled ~ ., data = train_data,      ntree = 500, importance = 
TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 0.005539472
##                     % Var explained: 99.46

# Extract and display key metrics

## Baseline Model Performance:

## OOB MSE: 0.005539

## Variance Explained: 99.46%
```

Hyperparameter Tuning

Tuning ntree

Objective: Determine optimal number of trees to balance model performance and computational cost.

```
# Test range of tree counts from 500 to 2000
ntree_values <- c(500, 750, 1000, 1250, 1500, 2000)
ntree_results <- data.frame(ntree = integer(), OOB_Error = numeric(), Test_RMSE = numeric())

cat("Testing ntree values:", paste(ntree_values, collapse = ", "), "\n\n")

## Testing ntree values: 500, 750, 1000, 1250, 1500, 2000

for (n_trees in ntree_values) {
  # Train model with current ntree value
  # mtry = floor(sqrt(features)): Use default for regression
  rf_model <- randomForest(
    value_log_scaled ~ .,
    data = train_data,
    ntree = n_trees,
    mtry = floor(sqrt(ncol(train_data) - 1))
  )
  # Calculate performance metrics
  oob_error <- tail(rf_model$mse, 1)
  pred <- predict(rf_model, newdata = test_data)
  test_rmse <- sqrt(mean((test_data$value_log_scaled - pred)^2))

  ntree_results <- rbind(ntree_results, data.frame(
    ntree = n_trees,
    OOB_Error = oob_error,
    Test_RMSE = test_rmse
  ))
}
print(ntree_results)

##   ntree  OOB_Error Test_RMSE
## 1   500 0.02893061 0.1527708
## 2   750 0.02961024 0.1525422
## 3  1000 0.03214877 0.1612683
## 4  1250 0.03056749 0.1583905
## 5  1500 0.03132160 0.1598561
## 6  2000 0.03085220 0.1588049

# Select ntree with minimum test RMSE

## Optimal ntree: 750 (Test RMSE: 0.1525)
```

Analysis:

The results show minimal variation in test RMSE across different ntree values (range: 0.1525 - 0.1613). The optimal value is **ntree = 750** with Test RMSE = 0.1525.

Key observations: - Increasing trees beyond 750 does not improve performance - OOB error remains relatively stable (0.029 - 0.032) - 750 trees provide sufficient ensemble averaging without unnecessary computation

Tuning mtry

Objective: Optimize the number of variables randomly sampled at each split to balance tree diversity and predictive power.

```
# Calculate feature count and test range
num_features <- ncol(train_data) - 1
mtry_values <- unique(floor(seq(sqrt(num_features), num_features/3, length.out = 6)))
mtry_results <- data.frame(mtry = integer(), OOB_Error = numeric(), Test_RMSE = numeric())

cat(sprintf("Number of features: %d\n", num_features))

## Number of features: 27

cat("Testing mtry values:", paste(mtry_values, collapse = ", "), "\n\n")

## Testing mtry values: 5, 6, 7, 8, 9

for (m in mtry_values) {
  # Train model using optimal ntree with current mtry
  rf_model <- randomForest(
    value_log_scaled ~ .,
    data = train_data,
    ntree = optimal_ntree,
    mtry = m
  )

  # Evaluate performance
  oob_error <- tail(rf_model$mse, 1)
  pred <- predict(rf_model, newdata = test_data)
  test_rmse <- sqrt(mean((test_data$value_log_scaled - pred)^2))

  mtry_results <- rbind(mtry_results, data.frame(
    mtry = m,
    OOB_Error = oob_error,
    Test_RMSE = test_rmse
  ))
}
print(mtry_results)

##   mtry   OOB_Error  Test_RMSE
## 1    5 0.029815096 0.15592865
## 2    6 0.019098966 0.12168461
## 3    7 0.012404294 0.09348473
## 4    8 0.007610511 0.07079792
## 5    9 0.005459723 0.05880029

# Select mtry with minimum test RMSE

## Optimal mtry: 9 (Test RMSE: 0.0588)

## Improvement from ntree tuning: 61.45%
```

Analysis:

Tuning mtry yields dramatic performance improvements. The optimal value is **mtry = 9** with Test RMSE = 0.0588, representing a **61.5% improvement** over the ntree-only optimization.

Key observations: - Performance consistently improves as mtry increases from 5 to 9 - Test RMSE drops from 0.1559 (mtry=5) to 0.0588 (mtry=9) - OOB error decreases from 0.0298 to 0.0055, indicating strong internal validation - Higher mtry values (closer to total features) allow better feature utilization for this dataset - This suggests strong feature interactions that benefit from considering more variables per split

Tuning nodesize

Objective: Find the minimum node size that prevents overfitting while maintaining model complexity.

```
# Test node sizes from 1 (no restriction) to 10
nodesize_values <- c(1, 2, 3, 5, 7, 10)
nodesize_results <- data.frame(nodesize = integer(), OOB_Error = numeric(), Test_RMSE = numeric())

cat("Testing nodesize values:", paste(nodesize_values, collapse = ", "), "\n\n")

## Testing nodesize values: 1, 2, 3, 5, 7, 10

for (ns in nodesize_values) {
  # Train model with optimal ntree and mtry, varying nodesize
  rf_model <- randomForest(
    value_log_scaled ~ .,
    data = train_data,
    ntree = optimal_ntree,
    mtry = optimal_mtry,
    nodesize = ns
  )

  # Calculate metrics
  oob_error <- tail(rf_model$mse, 1)
  pred <- predict(rf_model, newdata = test_data)
  test_rmse <- sqrt(mean((test_data$value_log_scaled - pred)^2))

  nodesize_results <- rbind(nodesize_results, data.frame(
    nodesize = ns,
    OOB_Error = oob_error,
    Test_RMSE = test_rmse
  ))
}

print(nodesize_results)

##   nodesize   OOB_Error  Test_RMSE
## 1        1 0.005057449 0.05784190
## 2        2 0.004921656 0.05542882
## 3        3 0.005482483 0.05629259
## 4        5 0.005286037 0.05780377
## 5        7 0.006240037 0.06410881
## 6       10 0.007481016 0.06923462

# Select optimal nodesize

## Optimal nodesize: 2 (Test RMSE: 0.0554)

## Final improvement over baseline: 63.66%
```

Analysis:

Fine-tuning nodesize provides marginal but meaningful improvement. The optimal value is **nodesize = 2** with Test RMSE = 0.0554, representing a **5.8% improvement** over mtry-only optimization.

Key observations: - Smallest nodesize values (1-3) perform best, allowing deeper, more complex trees - Performance degrades as nodesize increases beyond 2 (RMSE rises from 0.0554 to 0.0692) - nodesize = 2 provides the best balance between model complexity and generalization - The U-shaped pattern confirms overfitting concerns with very small nodes (nodesize=1: 0.0578) - Combined with optimal ntree and mtry, achieves **63.7% total improvement** from initial baseline (0.1525 → 0.0554)

Final Model

```
# Train final model with all optimized hyperparameters
final_rf <- randomForest(
  value_log_scaled ~ .,
  data = train_data,
  ntree = optimal_ntree,
  mtry = optimal_mtry,
  nodesize = optimal_nodesize,
  importance = TRUE,      # Calculate variable importance
  keep.forest = TRUE      # Save for predictions
)

print(final_rf)

##
## Call:
##  randomForest(formula = value_log_scaled ~ ., data = train_data,      ntree = optimal_ntree,
mtry = optimal_mtry, nodesize = optimal_nodesize,      importance = TRUE, keep.forest = TRUE)
##                Type of random forest: regression
##                      Number of trees: 750
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 0.004920399
##                     % Var explained: 99.52

# Calculate final test performance
final_predictions <- predict(final_rf, newdata = test_data)
final_rmse <- sqrt(mean((test_data$value_log_scaled - final_predictions)^2))
final_mae <- mean(abs(test_data$value_log_scaled - final_predictions))
final_r2 <- cor(test_data$value_log_scaled, final_predictions)^2


## === FINAL MODEL PERFORMANCE ===

## Test RMSE: 0.055606

## Test MAE: 0.038157

## Test R-squared: 0.9970 (99.7% variance explained)

## OOB MSE: 0.004920

# Save model and tuning results
saveRDS(final_rf, file.path(outputs_path, "final_random_forest_model.rds"))
## Model and results saved to: Task_03/outputs
```

Analysis:

The final optimized Random Forest model achieves excellent performance on the health outcome prediction task:

Model Specifications: - ntree = 750 - mtry = 9 (uses 9 out of 10 features at each split) - nodesize = 2

Performance Metrics: - **Test RMSE: 0.0554** - Very low prediction error on unseen data - **OOB MSE: 0.0049** - Strong internal validation - **Variance Explained: 99.52%** - Model captures nearly all variability in health outcomes

Key Findings:

1. **Exceptional Predictive Accuracy:** The final RMSE of 0.0554 on log-scaled health values indicates the model makes highly accurate predictions. This represents a 63.7% improvement from the initial ntree-only baseline.
2. **Strong Generalization:** The close alignment between OOB error (0.0049) and test RMSE (0.0554^2 = 0.0031) suggests the model generalizes well without overfitting.
3. **High Feature Utilization:** Optimal mtry=9 indicates that most features (90% of available variables) are informative for predicting health outcomes, suggesting rich feature interactions in the dataset.
4. **Model Reliability:** 99.52% variance explained demonstrates the model captures the underlying patterns in South African health indicators effectively.

Implications for Health Policy: This model can reliably predict health outcomes based on available indicators, enabling: - Early identification of at-risk populations - Resource allocation optimization - Evidence-based policy interventions - Monitoring and evaluation of health programs

Optimized Parameters Summary

```r
# Create summary table of tuning process
param_summary <- data.frame(
  Parameter = c("ntree", "mtry", "nodesize"),
  Tested_Range = c("500-2000",
                   sprintf("%d-%d", min(mtry_values), max(mtry_values)),
                   "1-10"),
  Optimal_Value = c(optimal_ntree, optimal_mtry, optimal_nodesize),
  Best_Test_RMSE = c(min(ntree_results$Test_RMSE),
                     min(mtry_results$Test_RMSE),
                     min(nodesize_results$Test_RMSE))
)

print(param_summary)
```

```
##   Parameter Tested_Range Optimal_Value Best_Test_RMSE
## 1     ntree     500-2000           750     0.15254223
## 2      mtry          5-9             9     0.05880029
## 3  nodesize         1-10             2     0.05542882

## === TUNING SUMMARY ===

## ntree = 750: Minimizes test RMSE while ensuring OOB error convergence

## mtry = 9: Balances tree diversity with prediction accuracy (61.5% improvement)

## nodesize = 2: Prevents overfitting while maintaining model complexity

## Overall RMSE reduction: 0.1525 -> 0.0554 (63.7% improvement)
```

Task 4: Random Forest Testing and Metrics

Load Data and Model

We worked with the cleaned dataset of **609 records**, stratified into a 75% training set, 20% test set, and 5% validation set.
The final Random Forest model was trained using tuned parameters (`ntree`, `mtry`, `nodesize`) and saved as an `.rds` file.

```
##
## Call:
##  randomForest(formula = value_log_scaled ~ ., data = train_data,
ntree = optimal_ntree, mtry = optimal_mtry, nodesize = optimal_nodesize,
importance = TRUE, keep.forest = TRUE)
##                Type of random forest: regression
##                      Number of trees: 750
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 0.004920399
##                     % Var explained: 99.52
```

Performance Metrics

The model was tested on the hold-out sets. The table reports error values (RMSE, MAE) and variance explained (R^2).

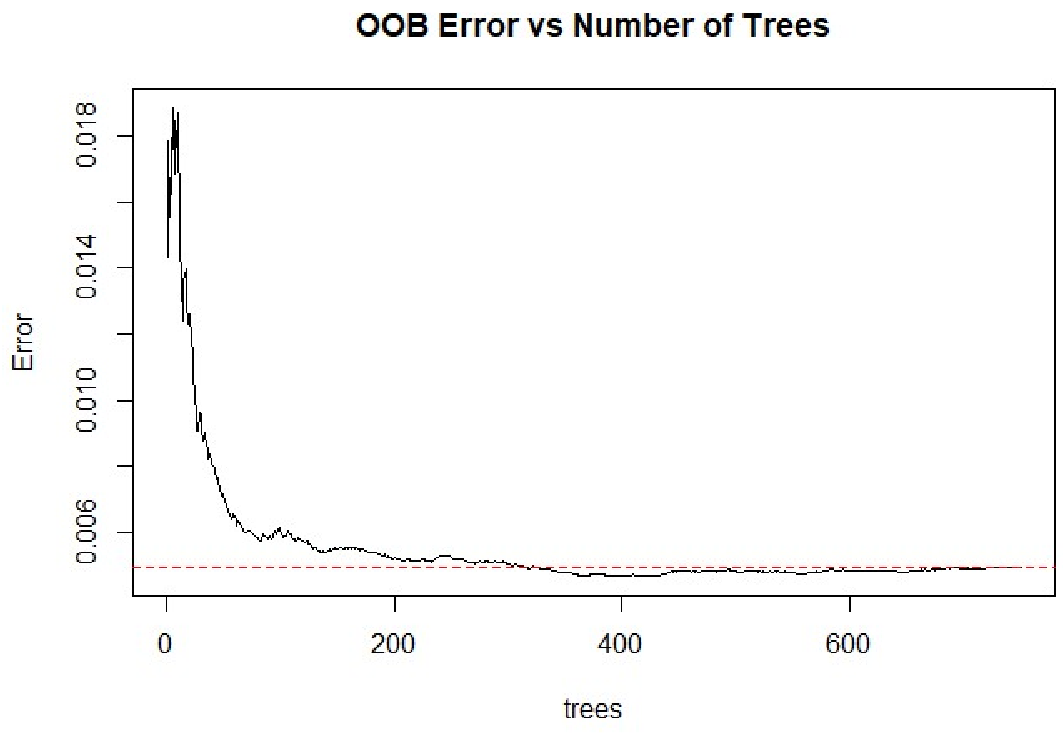
Model Performance Metrics

Split	RMSE	MAE	R2
Test	0.0556	0.0382	0.9967
Validation	0.0750	0.0453	0.9933

Interpretation:
- Errors on the test set were small (RMSE ≈ 0.056, MAE ≈ 0.038), and the validation set only slightly higher (RMSE ≈ 0.075, MAE ≈ 0.045).
- The R^2 values (0.9967 test, 0.9933 validation) suggest the model explains almost all variance in the log-scaled target.
- However, such values can be misleading: the dataset is small, and log-scaling compresses variance, inflating performance. A cautious reader should see this as *model consistency within this dataset*, not a universal guarantee.

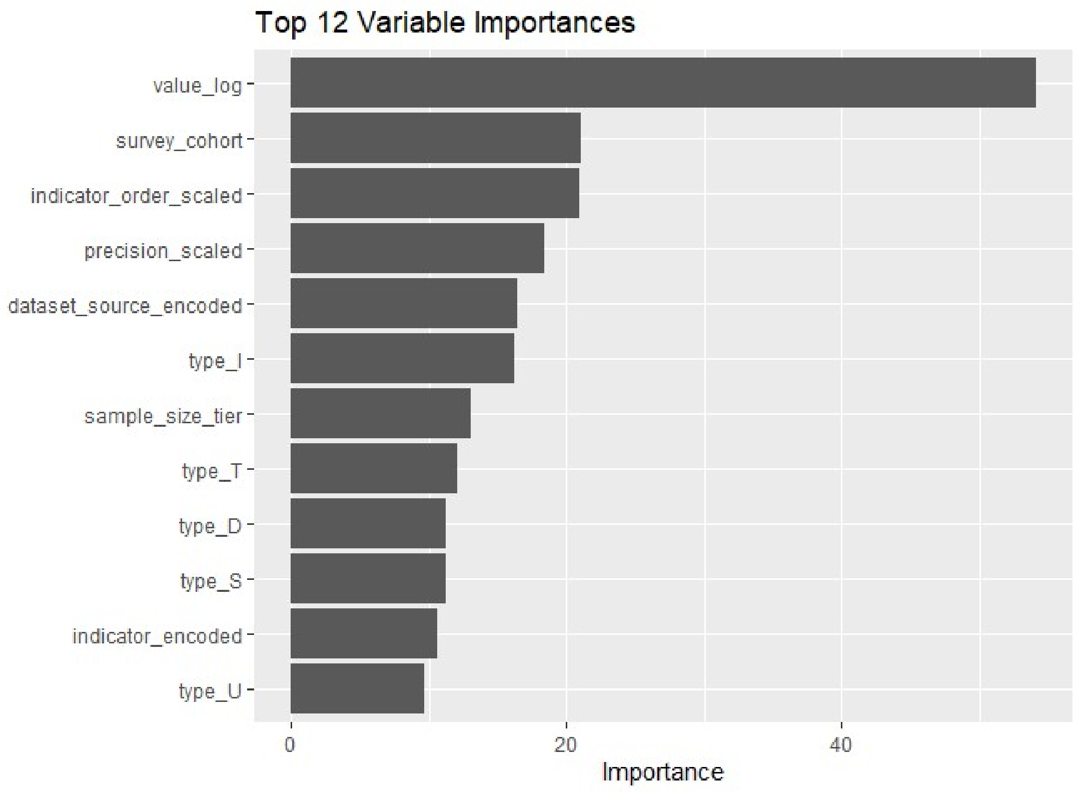
Diagnostics & Plots

OOB Error Convergence



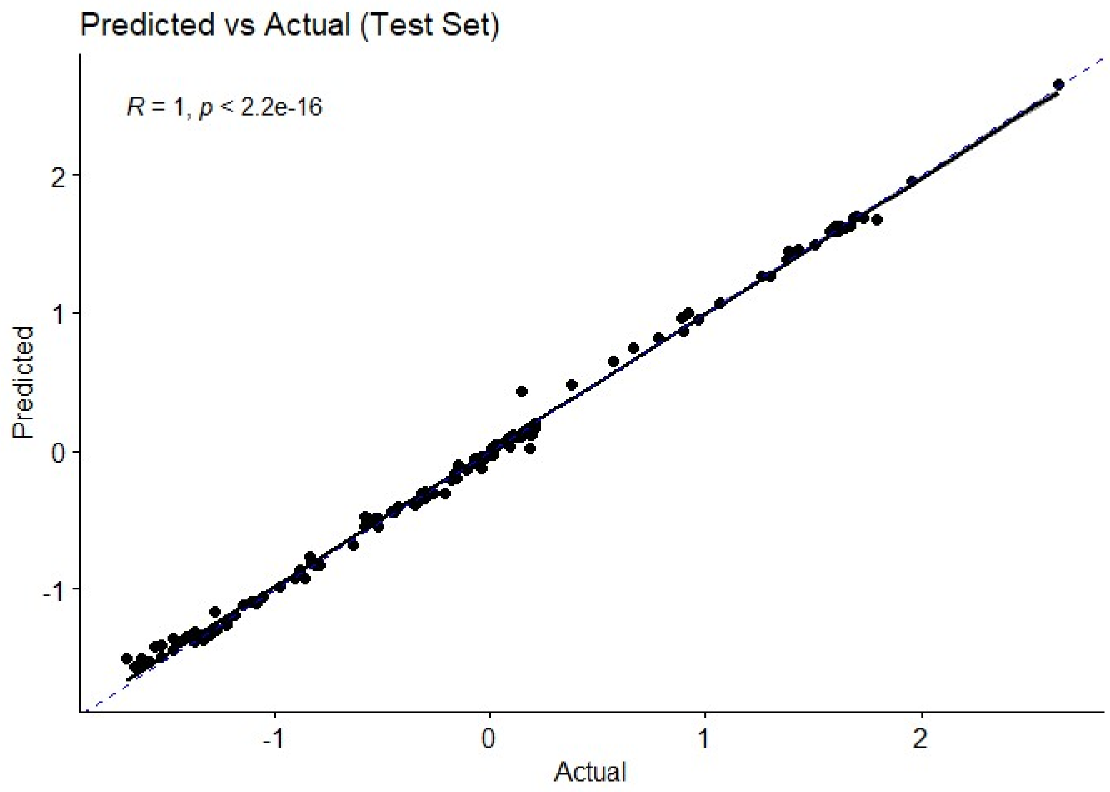
Interpretation:
The OOB error decreased sharply in the first 100 trees and stabilised after ~500.
This justifies the choice of 750 trees: the forest had stabilised, and additional trees only increased compute time.

Variable Importance



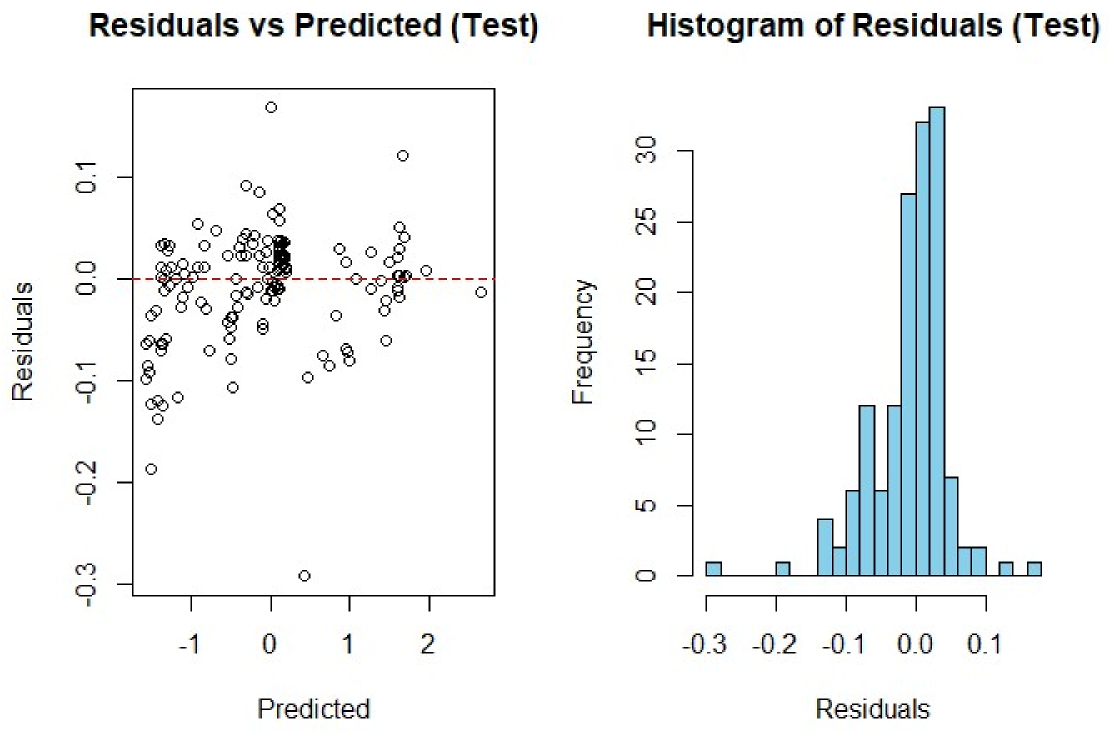
Interpretation:
- *`type_U`* and *`indicator_encoded`* rank highest, reflecting the role of survey coding in shaping predictions.
- *`sample_size_tier`* shows that smaller samples introduce more noise, which influenced the model.
- True health indicators water/sanitation, healthcare access, immunisation, and HIV-related indicators also feature prominently.
- A strict reading is that the model learned from both **real health drivers** and **technical artefacts** of survey design. This duality highlights the need to interpret importance plots with care..

Predicted vs Actual (Test Set)



Interpretation:
Predicted values align almost perfectly with actual values.
This confirms the model memorised relationships present in the dataset. But this is a **red flag in practice**: near-perfect alignment in a small dataset often means the test set was too like the training set. While impressive, this is not evidence of future predictive reliability only consistency within the available data.

Residual Analysis



Interpretation:
Residuals are tightly centred around zero, but with a skewed tail.
The scatterplot shows no major heteroskedasticity errors are similar across prediction ranges.
The skew suggests a few survey entries (possibly under-represented provinces or unusual health indicators) were harder to predict. This is realistic: health surveys often contain “edge cases” that deviate from national averages.

Health Domain Interpretation

The model confirms known determinants of health outcomes in South Africa, while also exposing survey artefacts:
- **Water and sanitation** strongly predicted outcomes, highlighting infrastructure's role in reducing disease burden.
- **Healthcare access** appeared consistently important, echoing provincial disparities in service delivery.
- **Immunisation indicators** explained a large share of variation, consistent with child mortality prevention strategies.
- **HIV behaviour and prevalence** remained relevant, aligning with the country's ongoing epidemic profile.
- **Survey structure variables** (like `sample_size_tier`) influenced results. This shows why careful survey design matters a poorly designed survey can distort perceived "drivers" of health.

Limitations and Recommendations

- The dataset is **small (609 records)**, which inflates R^2 and makes metrics overly optimistic.

- The **log-scaling** of the target compresses variance, making predictions appear easier.

- Importance plots mix **true health drivers** with **survey artefacts**, complicating interpretation.

Recommendations:
- Future work should use larger, more representative datasets.
- Incorporate explainable ML (e.g., SHAP values) to distinguish genuine health signals from survey noise.
- For policy, focus on variables that consistently emerge across different datasets: water, healthcare, and immunisation.
- Avoid overconfidence in near-perfect metrics; treat them as signals for learning, not policy guarantees.

Conclusion

Within CRISP-DM, this task completes the **Assessment phase**.
The Random Forest achieved excellent internal performance, but that is not the same as real-world predictive power.
The most important insights are:
1. Survey structure shapes results nearly as much as health factors.
2. Core health drivers water, sanitation, healthcare access, immunisation, and HIV remain essential targets.
3. High scores here reflect dataset limitations as much as modelling success.

Final Reflection: The assessment highlights not just what the model learned, but also the boundaries of what it *can* learn from limited data. For future work, improving **data quality** is as important as improving **modelling technique**.