Modelling Nutritional Dataset(Group C)

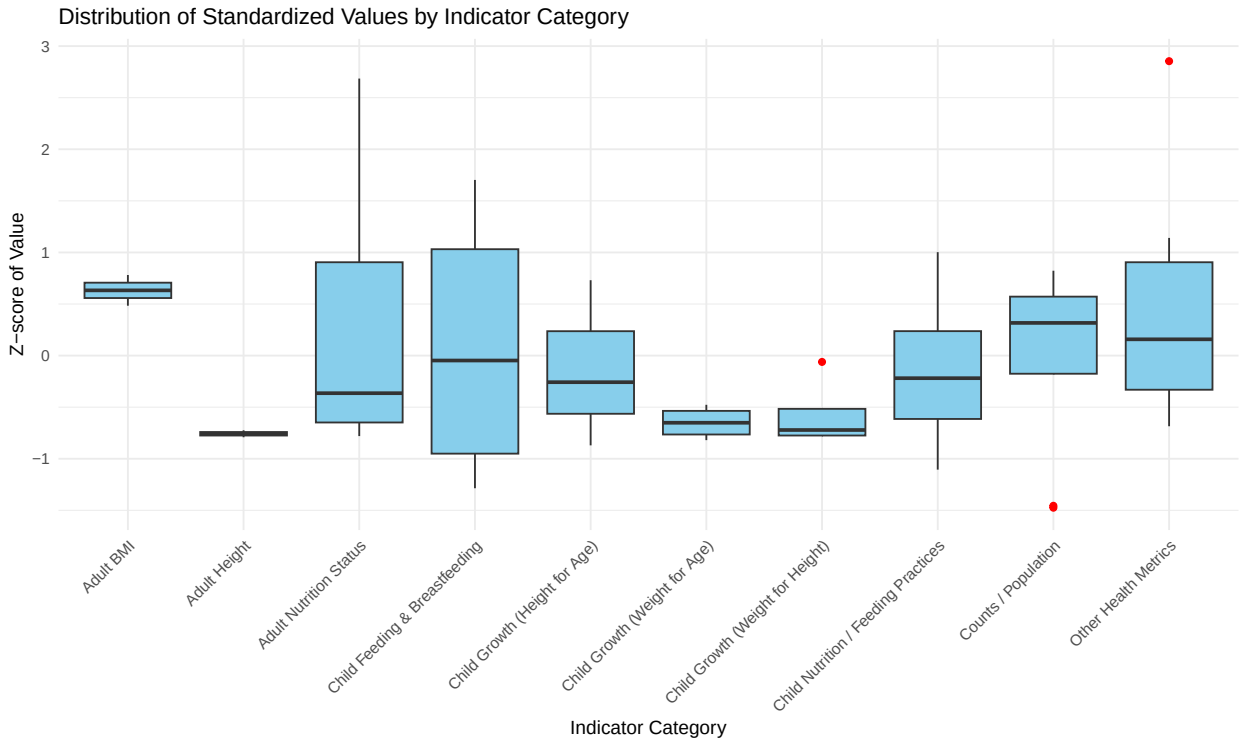
Group K

2025-10-03

```
#load the data and remove na
merged_data <- read.csv(here("cleaned_Data/Group_C_nutrition_merged.csv"))

plot_data <- merged_data %>%
  filter(!is.na(Value_zscore))
```

1. Boxplot of Value_zscore by Indicator Category

```
ggplot(plot_data, aes(x = IndicatorCategoryHigh, y = Value_zscore)) +
  geom_boxplot(fill = "skyblue", outlier.color = "red") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Distribution of Standardized Values by Indicator Category",
       x = "Indicator Category", y = "Z-score of Value")
```

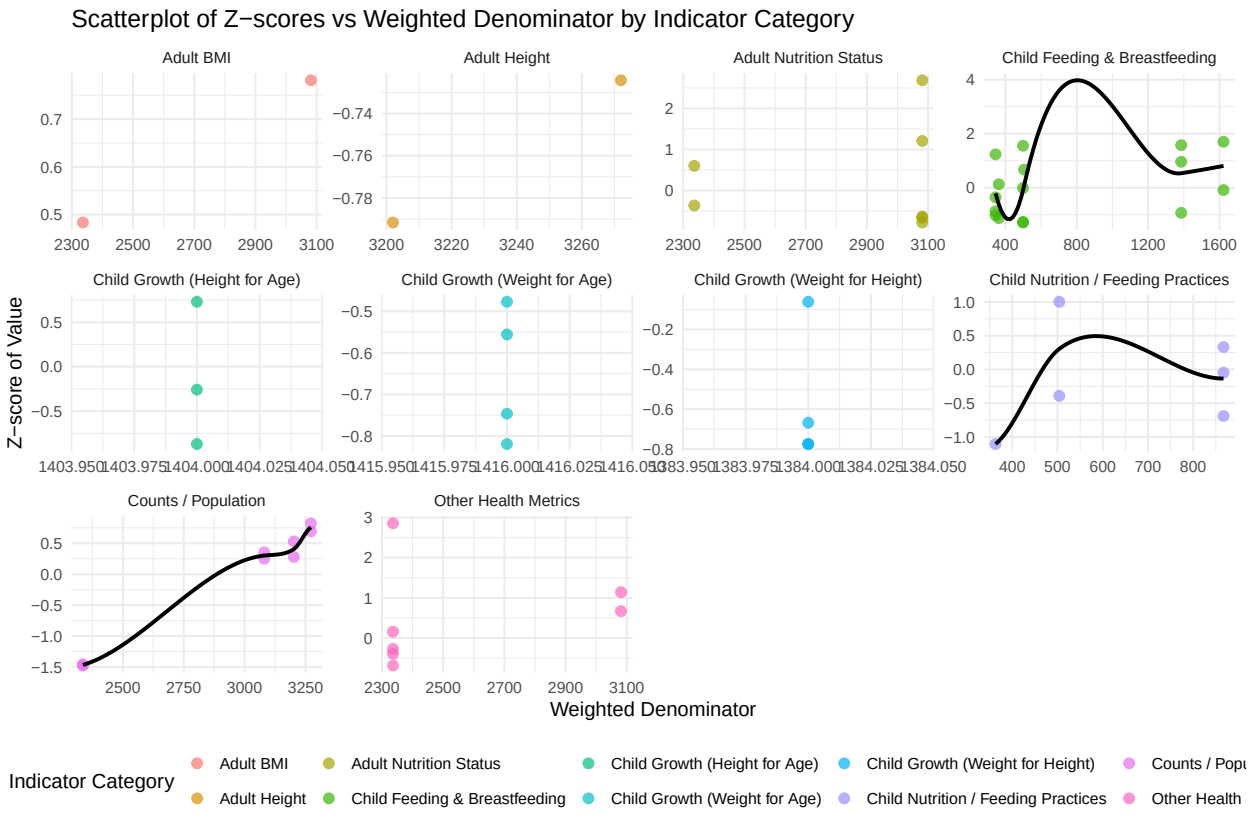


Most categories have a median close to zero, indicating that central values are similar, but the visible range and presence of outliers across categories suggest the possibility of nonlinear patterns and feature interactions that tree-based models can automatically capture.

2. Scatterplot Matrix of Selected Indicators

```
library(ggplot2)
library(dplyr)

# Remove rows with missing values in x or y
scatter_data <- plot_data %>%
  filter(!is.na(Value_zscore), !is.na(DenominatorWeighted))

# Scatterplot colored by IndicatorCategoryHigh
ggplot(scatter_data, aes(x = DenominatorWeighted,
                         y = Value_zscore, color = IndicatorCategoryHigh)) +
  geom_point(alpha = 0.7, size = 3) +   # semi-transparent points
  geom_smooth(method = "loess", se = FALSE, color = "black") + # trend line
  facet_wrap(~IndicatorCategoryHigh, scales = "free") +       # separate plots
  theme_minimal(base_size = 14) +
  labs(title = "Scatterplot of Z-scores vs Weighted Denominator by Indicator Category",
       x = "Weighted Denominator", y = "Z-score of Value",
       color = "Indicator Category") +
  theme(legend.position = "bottom")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Several indicator categories like "Child Feeding & Breastfeeding," "Child Nutrition / Feeding Practices" and "Counts / Population" show curved or complex nonlinear relationships between Z-scores and the weighted denominator. These patterns are not well captured by linear models but are naturally handled by decision trees and random forest models, which are capable of modeling complex, nonlinear trends.

A. Random Forrest

Feature selection

```
# Filter out NAs for the target variable
model_data <- plot_data %>%
  filter(!is.na(Value), !is.na(Value_zscore))

# Convert categorical variables to factors
model_data <- model_data %>%
  mutate(
    IndicatorCategoryHigh = as.factor(IndicatorCategoryHigh),
    IndicatorType = as.factor(IndicatorType),
    CharacteristicCategory = as.factor(CharacteristicCategory),
    PopulationGroup = as.factor(PopulationGroup)
  )

# Select features for modeling
features <- c("IndicatorType", "DenominatorWeighted", "DenominatorUnweighted",
              "CharacteristicCategory", "PopulationGroup", "SurveyYear")
target <- "Value"

# --- Remove columns not needed for modeling ---
cols_to_remove <- c("ChildNutritionScore", "AdultBMIScore")
model_data <- model_data %>% select(-all_of(cols_to_remove))
```

Train test split

```
# 80% training, 20% testing
train_index <- createDataPartition(model_data$Value, p = 0.8, list = FALSE)
train_data <- model_data[train_index, ]
test_data <- model_data[-train_index, ]
```

```
# Make sure the factor levels match training set
categorical_vars <- c("IndicatorType", "CharacteristicCategory", "PopulationGroup")
for (var in categorical_vars) {
  test_data[[var]] <- factor(test_data[[var]], levels = levels(train_data[[var]]))
}
```

```
# Subset to features + target for modeling
train_data_model <- train_data[, c(features, target)]
test_data_model  <- test_data[, c(features, target)]


# Train baseline
```

```r
rf_model <- randomForest(Value ~ ., data = train_data_model,
                         ntree = 500, mtry = 3, importance = TRUE)
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = Value ~ ., data = train_data_model, ntree = 500,      mtry = 3, importance =
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 150617.2
##                     % Var explained: 86.43
```

The model explains about 87.29% of the variation in the target variable (Value), and saves feature importance scores for later analysis.

Tune hyperparameters

```r
tune_grid <- expand.grid(.mtry = 2:7)
rf_formula <- as.formula(paste(target, "~", paste(features, collapse = " + ")))

# 5-fold cross-validation
train_control <- trainControl(method = "cv", number = 5)

# Train tuned random forest
rf_tuned <- train(
  rf_formula, data = train_data,
  method = "rf",
  tuneGrid = tune_grid,
  trControl = train_control,
  ntree = 1000,
  importance = TRUE
)

# Show best parameters
rf_tuned$bestTune
```

```
##   mtry
## 6    7
```

```r
# Final model
rf_model_final <- rf_tuned$finalModel
```

Predictions

```r
# Predict on test set
pred_test <- predict(rf_tuned, newdata = test_data)
```

```r
# Check predictions
head(pred_test)
```

```
##           2           3           4          13          17          29
##    6.377115    6.377115    6.810885 2992.066521   27.342930   19.609558
```

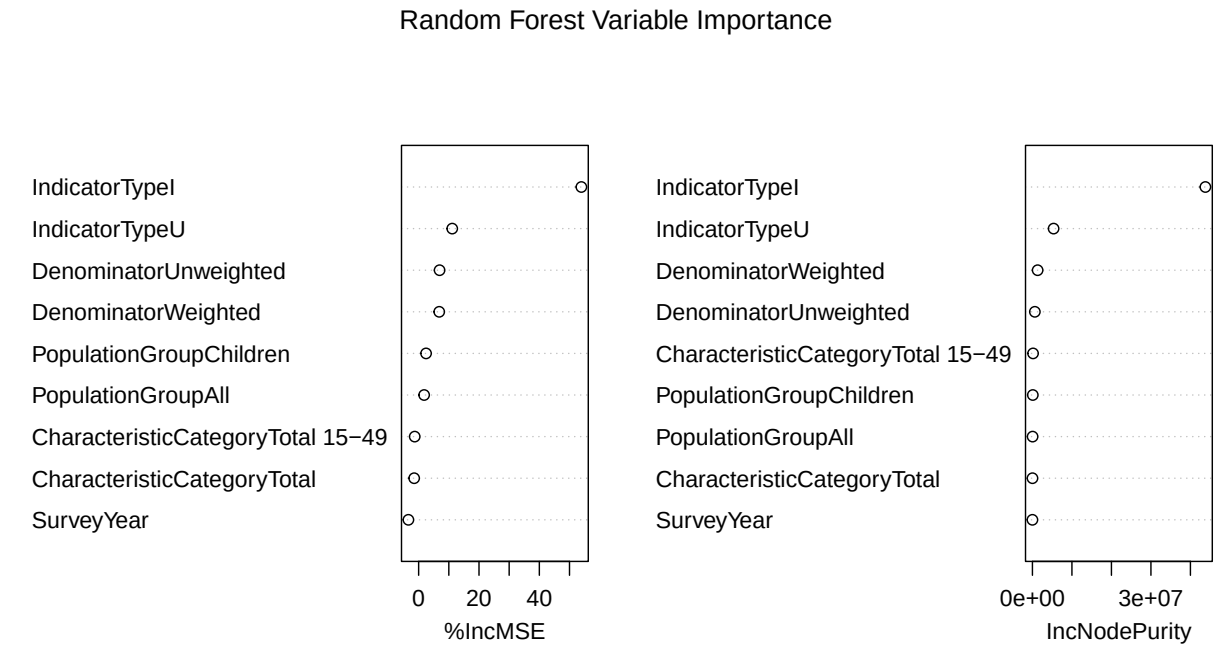
Model evaluation

```r
eval_metrics <- function(actual, predicted) {
  rmse <- sqrt(mean((actual - predicted)^2))
  mae <- mean(abs(actual - predicted))
  r2 <- cor(actual, predicted)^2
  return(data.frame(RMSE = rmse, MAE = mae, R2 = r2))
}

# evauate on test set
eval_test <- eval_metrics(test_data$Value, pred_test)
eval_test
```

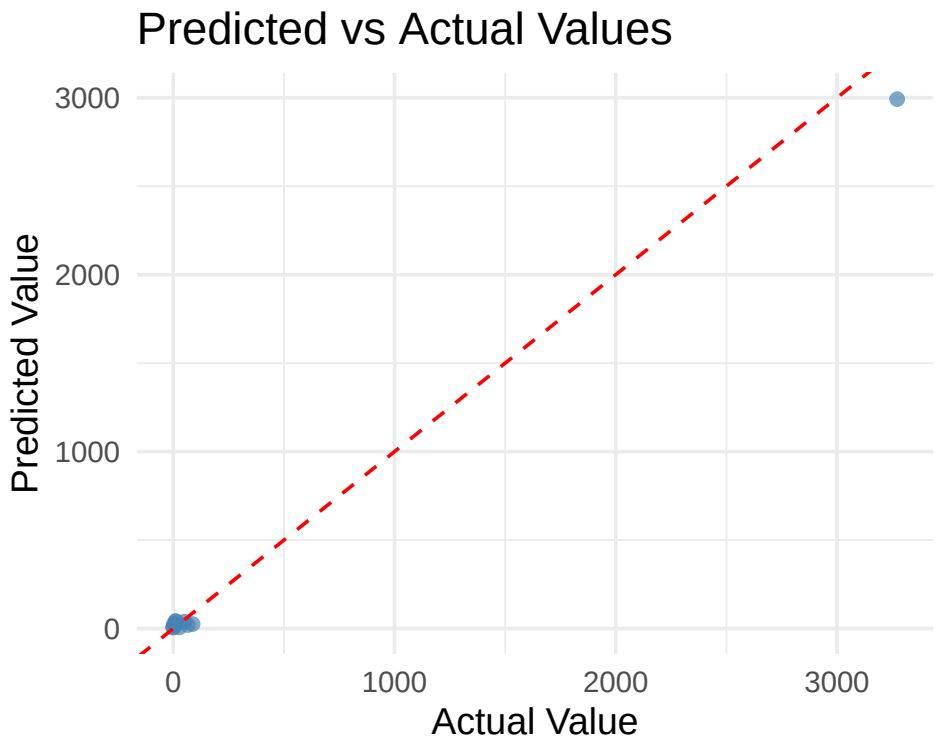
```
##       RMSE      MAE        R2
## 1 89.20276 48.22696 0.9990282
```

R^2 indicates that the Random Forest model explains about 99% of the variance in the target variable (Value). On average, predictions deviate from actual values by about 110 units. Lower RMSE is better, but it should be interpreted relative to the range of Value.A value of 63.3 indicates that, on average, predictions are off by 63 units.

```r
varImpPlot(rf_model_final, main = "Random Forest Variable Importance")
```



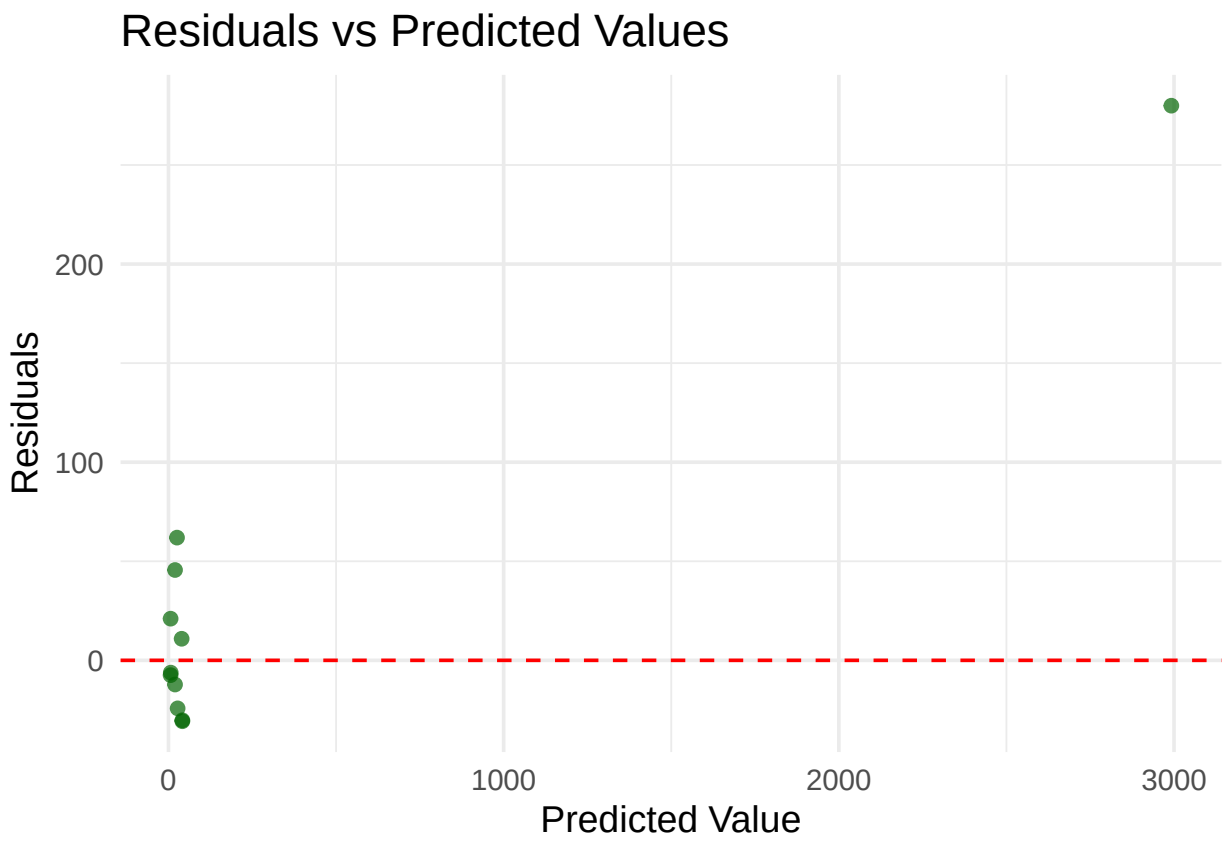

```r
ggplot(test_data, aes(x = Value, y = pred_test)) +
  geom_point(alpha = 0.7, color = "steelblue") +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  theme_minimal(base_size = 14) +
  labs(title = "Predicted vs Actual Values",
       x = "Actual Value", y = "Predicted Value")
```



Predicted versus actual plots show that the model performed consistently well across the range of observed values.

Residual analysis

```
residuals <- test_data$Value - pred_test
ggplot(test_data, aes(x = pred_test, y = residuals)) +
  geom_point(alpha = 0.7, color = "darkgreen") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  theme_minimal(base_size = 14) +
  labs(title = "Residuals vs Predicted Values",
       x = "Predicted Value", y = "Residuals")
```



The residual plots confirm that prediction errors are roughly evenly distributed around zero, suggesting minimal bias. It confirmed the lack of systematic error patterns, reinforcing confidence in the model's unbiased nature.

B. XGBoost

Feature selection

```r
categorical_vars <- c("IndicatorType", "CharacteristicCategory", "PopulationGroup")
numeric_vars <- c("DenominatorWeighted", "DenominatorUnweighted", "SurveyYear")
target_var <- "Value"

#Impute missing values
impute_mode <- function(x) {
  ux <- unique(x[!is.na(x)])
  ux[which.max(tabulate(match(x, ux)))]
}

impute_data <- function(df) {
  df %>%
    mutate(across(all_of(numeric_vars), ~ifelse(is.na(.), median(., na.rm = TRUE), .))) %>%
    mutate(across(all_of(categorical_vars), ~ifelse(is.na(.), impute_mode(.), .)))
}

train_data_clean <- impute_data(train_data)
test_data_clean <- impute_data(test_data)
```

```r
#One-hot encode categorical variables
dummies <- dummyVars(~ IndicatorType + CharacteristicCategory + PopulationGroup, data = train_data_clean
train_cat <- predict(dummies, newdata = train_data_clean)
test_cat <- predict(dummies, newdata = test_data_clean)

x_train <- cbind(train_cat, as.matrix(train_data_clean[, numeric_vars]))
y_train <- train_data_clean[[target_var]]

x_test <- cbind(test_cat, as.matrix(test_data_clean[, numeric_vars]))
y_test <- test_data_clean[[target_var]]

#DMatrix necessary for XGBoost
dtrain <- xgb.DMatrix(data = x_train, label = y_train)
dtest <- xgb.DMatrix(data = x_test, label = y_test)
```

XGBoost Parameters

```r
params <- list(
  objective = "reg:squarederror",
  eval_metric = "rmse",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8
)
set.seed(123)
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 500,
  watchlist = list(train = dtrain, test = dtest),
  early_stopping_rounds = 20,
```

```
  print_every_n = 10
)
```

```
## [1]  train-rmse:1119.247247  test-rmse:987.140874
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 20 rounds.
##
## [11] train-rmse:601.033558   test-rmse:554.830369
## [21] train-rmse:422.560595   test-rmse:401.640201
## [31] train-rmse:279.464090   test-rmse:294.705348
## [41] train-rmse:244.759190   test-rmse:243.679126
## [51] train-rmse:147.531539   test-rmse:160.620040
## [61] train-rmse:94.075973    test-rmse:106.981463
## [71] train-rmse:63.565801    test-rmse:72.001835
## [81] train-rmse:44.567390    test-rmse:52.342054
## [91] train-rmse:32.108673    test-rmse:40.301906
## [101]    train-rmse:26.807669    test-rmse:34.880595
## [111]    train-rmse:22.692100    test-rmse:32.046694
## [121]    train-rmse:19.032987    test-rmse:28.899338
## [131]    train-rmse:17.734953    test-rmse:28.397308
## [141]    train-rmse:17.046475    test-rmse:27.896452
## [151]    train-rmse:16.145042    test-rmse:27.448119
## [161]    train-rmse:15.619309    test-rmse:27.587100
## [171]    train-rmse:15.388506    test-rmse:26.573311
## [181]    train-rmse:15.268369    test-rmse:26.955966
## [191]    train-rmse:15.118967    test-rmse:26.866387
## Stopping. Best iteration:
## [171]    train-rmse:15.388506    test-rmse:26.573311
```

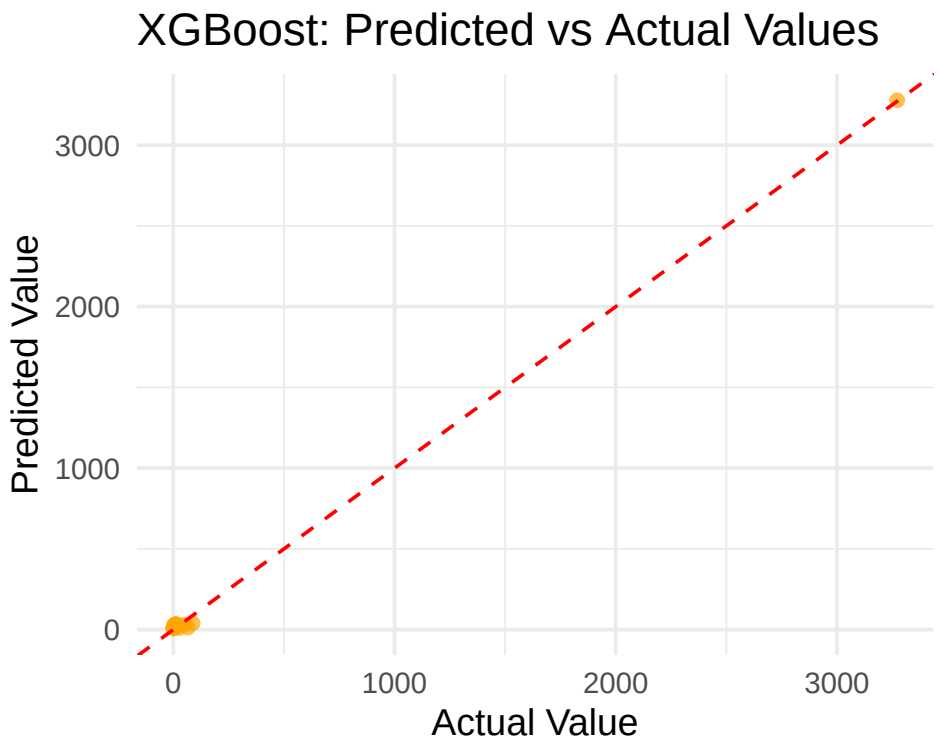
Predictions

```
pred_xgb <- predict(xgb_model, newdata = dtest)

# Evaluate performance using standard metrics
eval_metrics <- function(actual, predicted) {
  rmse <- sqrt(mean((actual - predicted)^2))
  mae <- mean(abs(actual - predicted))
  r2 <- cor(actual, predicted)^2
  data.frame(RMSE = rmse, MAE = mae, R2 = r2)
}

eval_xgb <- eval_metrics(y_test, pred_xgb)
eval_xgb
```

```
##       RMSE      MAE        R2
## 1 26.57331 21.71013 0.9992099
```

Preicted vs actual plot

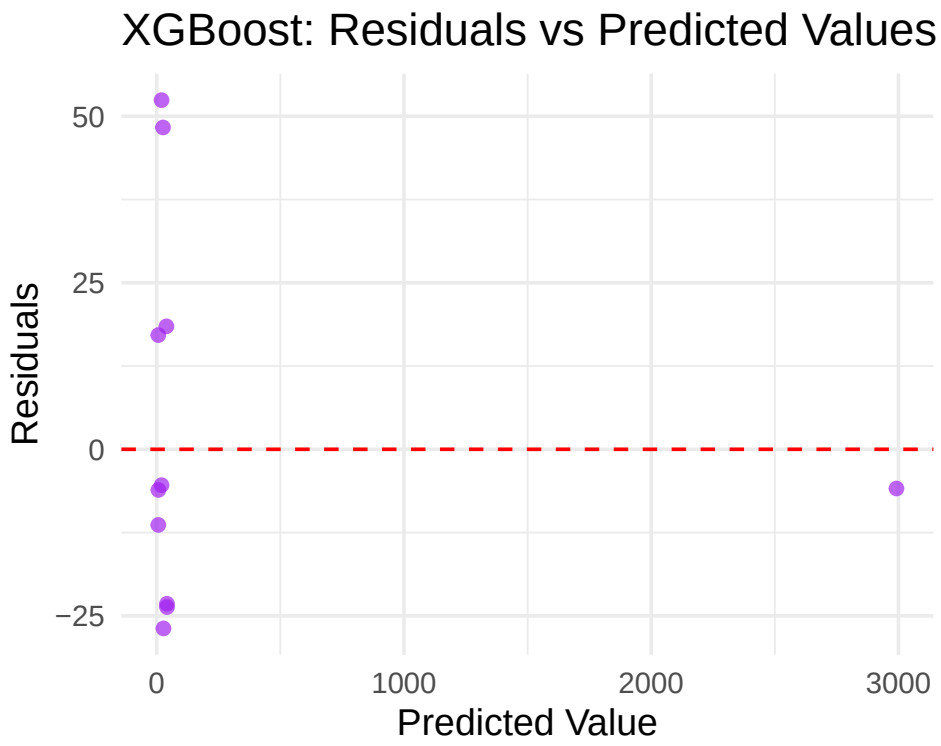
```r
ggplot(data = NULL, aes(x = y_test, y = pred_xgb)) +
  geom_point(alpha = 0.7, color = "orange") +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  theme_minimal(base_size = 14) +
  labs(title = "XGBoost: Predicted vs Actual Values",
       x = "Actual Value", y = "Predicted Value")
```



There is tight clustering around the diagonal reference line, and these plots confirm balanced error distribution across the prediction range.

Residual Analysis

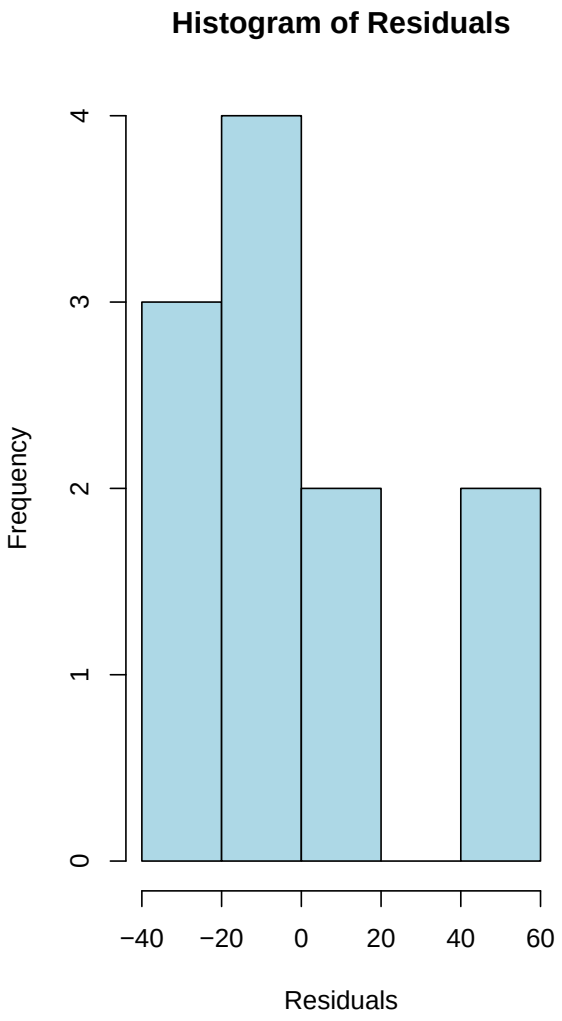
```r
residuals_xgb <- y_test - pred_xgb
ggplot(data = NULL, aes(x = pred_test, y = residuals_xgb)) +
  geom_point(alpha = 0.7, color = "purple") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  theme_minimal(base_size = 14) +
  labs(title = "XGBoost: Residuals vs Predicted Values",
       x = "Predicted Value", y = "Residuals")
```

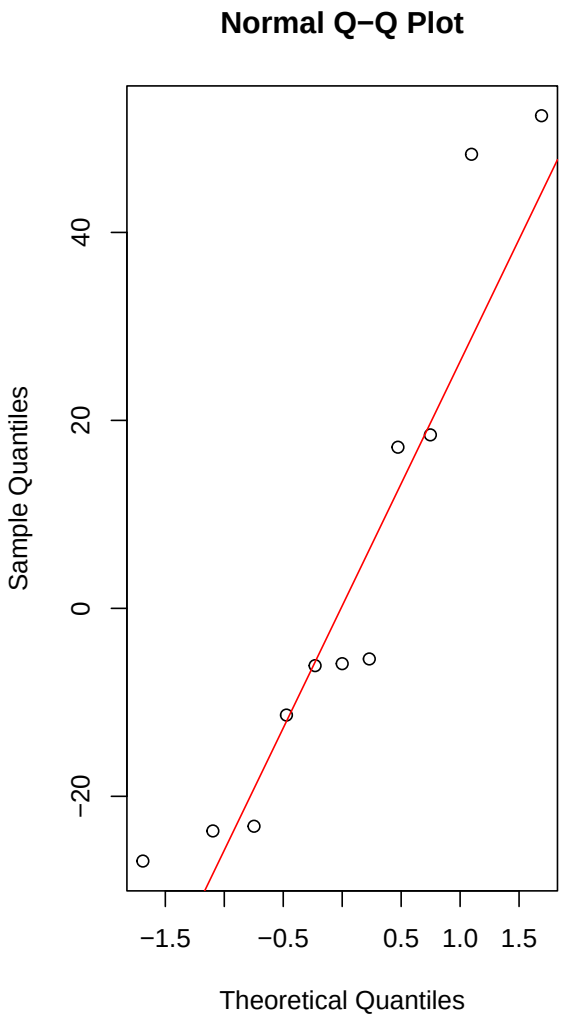


Most residuals are within ±50 units, except a few small predicted values that have slightly larger deviations.This suggests that XGBoost is producing reasonably accurate predictions, though there may be some difficulty modeling the extreme low values or sparse data points.

Diagnostic Plots

```
par(mfrow = c(1,2))

hist(residuals_xgb, main = "Histogram of Residuals", xlab = "Residuals", col = "lightblue")
qqnorm(residuals_xgb); qqline(residuals_xgb, col = "red")
```






```
par(mfrow = c(1,1))
```

The histogram is skewed slightly left (more residuals around -20), which suggests the residuals are not perfectly normal.

This could affect the validity of confidence intervals and hypothesis tests.In the qqplot we see deviations from the line, especially at the tails, that suggest non-normality.

Evaluation of each model performance is in the word document.