Age Prediction Based on Gut Microbiome Profiles

Fariha Anis

15th October 2024

Introduction

The composition of the human gut microbiome evolves with age, influencing both health and disease. This project focuses on building machine learning models to predict an individual's age based on their gut microbiome profiles. The approach involves data preprocessing, feature selection, and applying multiple regression models in R to generate accurate age predictions.

Data Collection and Preprocessing

Data Sources

- Taxonomy Data (taxonomy.xlsx): Contains microbiome profiles with microbial community composition for each sample.
- Age Data (age.csv): Provides the target variable, 'Age,' associated with each sample ID.

Loading and Initial Exploration of the Dataset

```r
# Set the working directory
setwd("C:/Users/user/Downloads/Amili")

# Suppress package startup messages and Load the libraries
suppressPackageStartupMessages({
  library(caret)
  library(readxl)
  library(randomForest)
  library(dplyr)
  library(ggplot2)
  library(xgboost)
  library(e1071)
  library(reshape2)
  library(skimr)
  library(gridExtra)
  library(grid)
})

# Load the datasets
taxonomy <- read_excel("taxonomy.xlsx")
```

```
## New names:
## • `` -> `...1`

age <- read.csv("age.csv")

# Rename the first column to 'ID' for both datasets
colnames(taxonomy)[1] <- "ID"
colnames(age)[1] <- "ID"

# Merge the two datasets by the sample IDs
df <- merge(taxonomy, age, by = "ID")

# Drop the 'ID' column
df <- df %>% select(-ID)

# Check the dimensions of the dataset
cat("Dimensions of the dataset: ", dim(df))

## Dimensions of the dataset:  60 5656
```

Data Cleaning and Preparation

Removing Near-Zero Variance Predictors:

Near-zero variance predictors offer minimal variability and little information, which can lead to overfitting and unnecessary complexity. By removing these predictors, the dataset is simplified, retaining only features that provide useful variance. In this case, after filtering, the dataset contains 60 observations and 440 features.

```
# Define features and target variable
X <- df[, -which(names(df) == "Age")]
y <- df$Age

# Remove near-zero variance predictors
nzv <- nearZeroVar(X)
X_filtered <- X[, -nzv]

# Check dimensions after filtering
print(dim(X_filtered))

## [1]  60 440
```

Removing Highly Correlated Features:

Highly correlated features can cause multicollinearity, making the model less stable and harder to interpret. By eliminating features with correlations above 0.9, redundancy is reduced. The dataset is further refined, now containing 60 observations and 373 features.

```
# Find highly correlated features
correlation_matrix <- cor(X_filtered)
highly_correlated <- findCorrelation(correlation_matrix, cutoff = 0.9)

# Remove highly correlated features
X_filtered_no_corr <- X_filtered[, -highly_correlated]

# Check dimensions after removing correlated features
print(dim(X_filtered_no_corr))

## [1]  60 373
```

Scaling the Features:

Scaling ensures that all features are on the same range, preventing features with larger values from dominating the model. The scale() function standardizes each feature to have a mean of 0 and a standard deviation of 1, allowing all features to contribute equally during model training.

```
# Scale the features
X_scaled <- scale(X_filtered_no_corr)
```

Feature Selection

Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a feature selection technique that works by recursively removing the least important features based on model performance until the optimal set of features is achieved (Guyon, Weston, Barnhill, & Vapnik, 2002). In this case, RFE was applied using a Random Forest model to select the top 10 most important features. The importance of each feature is determined by its contribution to reducing model error. RFE improves model efficiency by focusing only on the most relevant features, helping reduce overfitting and enhancing interpretability. This method has been effectively used in studies, including microbiome research, where RFE with Random Forest identified key genomic features, demonstrating its value in managing high-dimensional data (Bobbo et al., 2024).

```
# Set up control parameters for RFE
control <- rfeControl(functions = rfFuncs, method = "cv", number = 5)

# Run RFE
set.seed(123)
rfe_results <- rfe(X_scaled, y, sizes = c(1:10), rfeControl = control)

# Print the results
print(rfe_results)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (5 fold)
##
## Resampling performance over subset size:
##
##  Variables  RMSE Rsquared   MAE RMSESD RsquaredSD MAESD Selected
##          1 14.77  0.07978 12.73  1.983    0.09072 1.276
##          2 13.69  0.02169 11.30  2.280    0.01759 1.969
##          3 13.18  0.01305 10.39  1.980    0.01895 1.219
##          4 13.18  0.01868 10.62  2.034    0.03838 2.040
##          5 12.84  0.03998 10.33  1.677    0.05222 1.800
##          6 13.20  0.04761 10.55  1.809    0.06713 1.963
##          7 13.15  0.04779 10.51  1.846    0.07155 1.927
##          8 12.89  0.04243 10.46  1.823    0.06744 1.795
##          9 12.91  0.03472 10.47  1.990    0.06459 1.937
##         10 12.77  0.05146 10.36  1.775    0.07685 1.770
##        373 12.70  0.09779 10.75  1.741    0.11457 1.497        *
##
## The top 5 variables (out of 373):
##    Bacteroidetes_520|Bacteroides_stercoris, Bacteroidetes_516|Bacteroides_
cellulosilyticus, Bacteroidetes_772|Bacteroides_xylanisolvens, Firmicutes_Ten
ericutes_416|Anaerotignum_propionicum_DSM_1682, Firmicutes_Tenericutes_519|Eu
bacteriales
```

```r
# Plot the RFE results
plot(rfe_results, type = c("g", "o"))
```

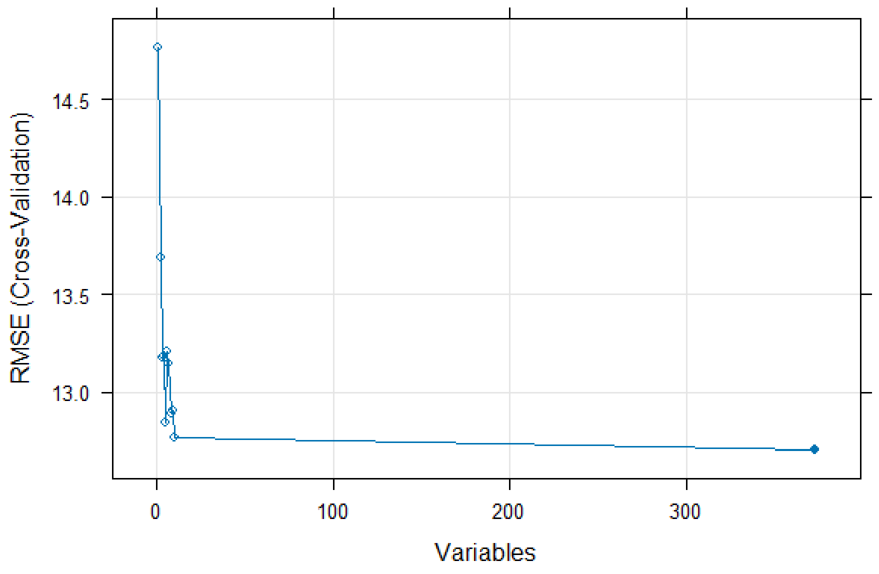

Figure 1

Figure 1 shows the RMSE from cross-validation as the number of variables selected increases during Recursive Feature Elimination (RFE) using a Random Forest model. Initially, RMSE decreases sharply, indicating that model performance improves as more features are added. However, after selecting around 10 variables, the RMSE stabilizes, suggesting that adding more features beyond this point does not significantly improve performance. This suggests that the top 10 features are sufficient to minimize error, supporting the use of RFE for selecting the most relevant features while avoiding unnecessary complexity.

```r
# Get the top 10 features
top_10_features <- head(predictors(rfe_results), 10)
```

Visualizing Top 10 Features

```r
# Create a data frame for plotting
importance_df <- data.frame(
  Feature = top_10_features,
  Importance = seq(1, 10)
)

# Plot the top 10 features
ggplot(importance_df, aes(x = reorder(Feature, Importance), y = Importance))
+
  geom_bar(stat = "identity", fill = "#BB9AB1") +
  coord_flip() +
  labs(title = "Top 10 Selected Features by RFE", x = "Features", y = "Import
ance") +
  theme_minimal() +
  theme(
    axis.title.x = element_text(size = 14),
    axis.title.y = element_text(size = 14),
    axis.text.x = element_text(size = 14),
    axis.text.y = element_text(size = 14)
  )
```

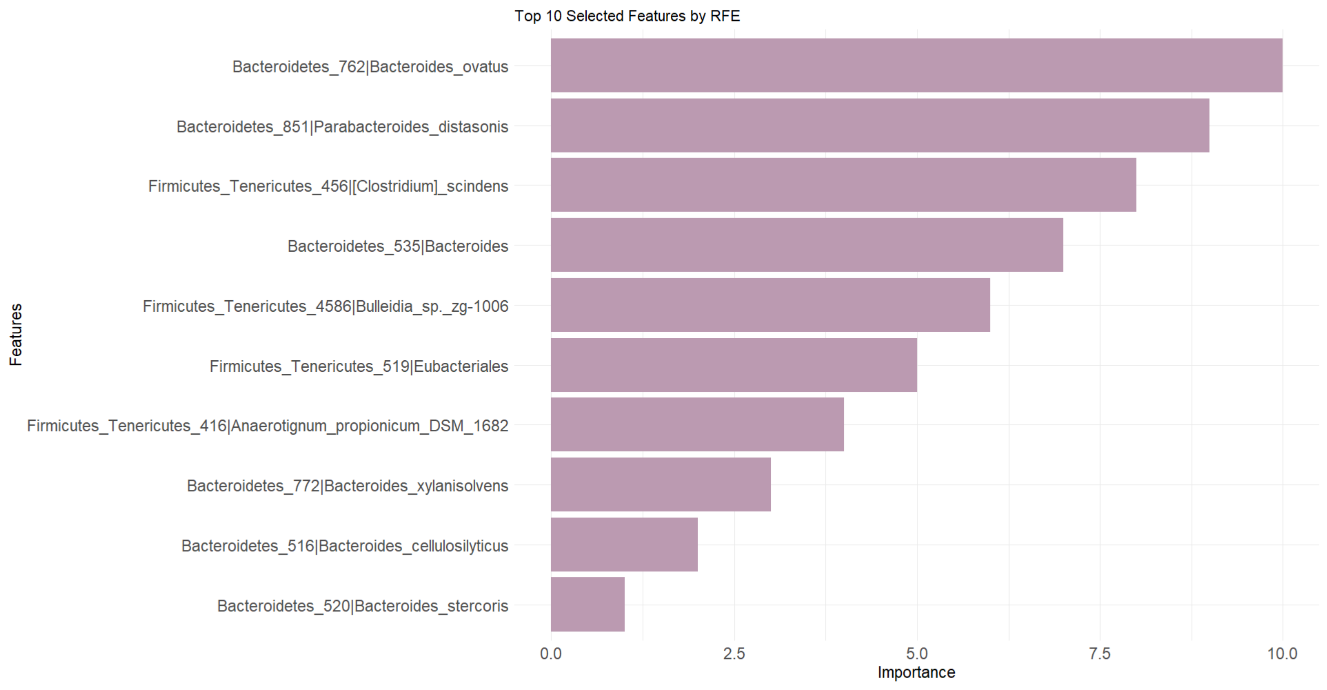

Figure 2

Figure 2 displays the top 10 features selected by RFE with a Random Forest model for predicting age based on gut microbiome profiles, ranked by importance scores. *Bacteroides ovatus* holds the highest importance score, indicating its significant influence on age prediction, followed by *Parabacteroides distasonis* and *Clostridium scindens*. These features exhibit the highest scores, highlighting their strong association with age-related microbiome changes. The importance scores reflect how much each feature contributes to the model's predictions, with the top few features playing a key role in distinguishing age-related patterns in the gut microbiome.

Exploratory Data Analysis

Exploring Top 10 Features

```
# Extract the top 10 features along with 'Age'
df_top_10 <- df[, c(top_10_features, "Age")]

# View the first few rows
head(df_top_10)

##   Bacteroidetes_520|Bacteroides_stercoris
## 1                                     51.0
## 2                                   1232.6
## 3                                    162.4
## 4                                    271.4
## 5                                      0.0
## 6                                     58.8
##   Bacteroidetes_516|Bacteroides_cellulosilyticus
## 1                                           0.00
## 2                                           0.00
## 3                                           3.25
## 4                                           0.00
## 5                                          10.00
## 6                                           6.25
##   Bacteroidetes_772|Bacteroides_xylanisolvens
## 1                                        65.0
## 2                                         0.0
## 3                                        14.6
## 4                                         0.0
## 5                                       647.6
## 6                                       163.4
##   Firmicutes_Tenericutes_416|Anaerotignum_propionicum_DSM_1682
## 1                                                    24.857143
## 2                                                    47.857143
## 3                                                     8.285714
## 4                                                   117.714286
## 5                                                    21.142857
## 6                                                   102.714286
##   Firmicutes_Tenericutes_519|Eubacteriales
## 1                                 0.000000
## 2                                 3.818182
## 3                                 2.363636
## 4                                 0.000000
## 5                                 1.818182
## 6                                 3.090909
##   Firmicutes_Tenericutes_4586|Bulleidia_sp._zg-1006
## 1                                               2.5
## 2                                               0.0
## 3                                               0.0
## 4                                               0.0
## 5                                               0.0
## 6                                              87.0
##   Bacteroidetes_535|Bacteroides
## 1                          15.0
## 2                           4.0
## 3                           2.0
## 4                           0.0
## 5                           0.0
## 6                           0.5
##   Firmicutes_Tenericutes_456|[Clostridium]_scindens
## 1                                            246.50
## 2                                            149.25
## 3                                            202.25
## 4                                             29.00
## 5                                              1.25
## 6                                            288.00
##   Bacteroidetes_851|Parabacteroides_distasonis
## 1                                     4.285714
## 2                                   152.285714
## 3                                     0.000000
## 4                                     0.000000
## 5                                     0.000000
## 6                                    36.714286
##   Bacteroidetes_762|Bacteroides_ovatus Age
## 1                                  0.0  38
## 2                                  0.0  45
## 3                                  9.6  54
## 4                                  0.0  41
## 5                                  0.0  35
## 6                                  0.0  34
```

```
# Get data types of each column
print(sapply(df_top_10, class))
```

```
##                      Bacteroidetes_520|Bacteroides_stercoris
##                                                    "numeric"
##                 Bacteroidetes_516|Bacteroides_cellulosilyticus
##                                                    "numeric"
##                   Bacteroidetes_772|Bacteroides_xylanisolvens
##                                                    "numeric"
## Firmicutes_Tenericutes_416|Anaerotignum_propionicum_DSM_1682
##                                                    "numeric"
##                     Firmicutes_Tenericutes_519|Eubacteriales
##                                                    "numeric"
##               Firmicutes_Tenericutes_4586|Bulleidia_sp._zg-1006
##                                                    "numeric"
##                               Bacteroidetes_535|Bacteroides
##                                                    "numeric"
##               Firmicutes_Tenericutes_456|[Clostridium]_scindens
##                                                    "numeric"
##                  Bacteroidetes_851|Parabacteroides_distasonis
##                                                    "numeric"
##                         Bacteroidetes_762|Bacteroides_ovatus
##                                                    "numeric"
##                                                          Age
##                                                    "integer"
```

```
# Get summary statistics
skim(df_top_10)
```

Data summary

Name	df_top_10
Number of rows	60
Number of columns	11

Column type frequency:

numeric	11
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Bacteroidetes_520|Bacteroides_stercoris	0	1	293.21	405.15	0	51.55	134.00	336.70	1952.60	▇▂▁▁▁
Bacteroidetes_516|Bacteroides_cellulosilyticus	0	1	4.20	10.97	0	0.00	0.00	3.25	63.50	▇▁▁▁▁
Bacteroidetes_772|Bacteroides_xylanisolvens	0	1	43.57	88.97	0	12.15	22.60	38.50	647.60	▇▁▁▁▁
Firmicutes_Tenericutes_416|Anaerotignum_propionicum_DSM_1682	0	1	27.34	27.96	0	8.50	19.14	36.00	126.29	▇▃▁▁▁
Firmicutes_Tenericutes_519|Eubacteriales	0	1	13.69	30.07	0	0.73	4.09	12.09	166.36	▇▁▁▁▁
Firmicutes_Tenericutes_4586|Bulleidia_sp._zg-1006	0	1	64.08	407.23	0	0.00	2.00	8.25	3152.50	▇▁▁▁▁
Bacteroidetes_535|Bacteroides	0	1	5.80	11.48	0	0.00	0.25	6.19	47.75	▇▁▁▁▁
Firmicutes_Tenericutes_456|[Clostridium]_scindens	0	1	226.64	184.06	0	110.69	185.25	310.62	1066.50	▇▆▁▁▁
Bacteroidetes_851|Parabacteroides_distasonis	0	1	21.43	44.30	0	0.00	7.64	17.82	275.43	▇▁▁▁▁
Bacteroidetes_762|Bacteroides_ovatus	0	1	10.31	23.84	0	0.00	0.00	13.00	144.40	▇▁▁▁▁

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Age	0	1	40.93	12.00	20	32.00	41.00	48.25	71.00	▄█▁

```r
# Check for missing values
colSums(is.na(df_top_10))
```

```
##                        Bacteroidetes_520|Bacteroides_stercoris
##                                                              0
##                  Bacteroidetes_516|Bacteroides_cellulosilyticus
##                                                              0
##                     Bacteroidetes_772|Bacteroides_xylanisolvens
##                                                              0
## Firmicutes_Tenericutes_416|Anaerotignum_propionicum_DSM_1682
##                                                              0
##                        Firmicutes_Tenericutes_519|Eubacteriales
##                                                              0
##              Firmicutes_Tenericutes_4586|Bulleidia_sp._zg-1006
##                                                              0
##                                  Bacteroidetes_535|Bacteroides
##                                                              0
##              Firmicutes_Tenericutes_456|[Clostridium]_scindens
##                                                              0
##                  Bacteroidetes_851|Parabacteroides_distasonis
##                                                              0
##                          Bacteroidetes_762|Bacteroides_ovatus
##                                                              0
##                                                            Age
##                                                              0
```

```r
# Check for duplicated rows
cat("Number of duplicated rows: ", sum(duplicated(df_top_10)))
```

```
## Number of duplicated rows:  0
```

Key Observations:

- **Clean Data:** No missing or duplicated values across all features, ensuring smooth model training.
- **Feature Variability:** *Bacteroides stercoris* shows high variability (mean = 293.21, SD = 405.15), likely making it a strong predictor, while features like *Parabacteroides distasonis* exhibit more consistency.
- **Distribution Insight:** Taxa like Bacteroides stercoris have wide interquartile ranges, indicating diverse abundance patterns that could help capture age-related microbiome shifts.

- **Age Stability:** The age variable has a balanced distribution (mean = 40.93, no extreme outliers), supporting reliable model training.
- **Key Taxa for Prediction:** *Bacteroidetes* and *Firmicutes* taxa are likely essential for predicting age due to their variability and biological relevance.

Visualizing Distributions

```r
# List of numeric columns
num_cols <- names(df_top_10)[sapply(df_top_10, is.numeric)]

# Create histograms with KDE
plot_list <- list()
for (col in num_cols) {
  p <- ggplot(df_top_10, aes(x = .data[[col]])) +
    geom_histogram(aes(y = ..density..), binwidth = 10, fill = "#BB9AB1", col
or = "black", alpha = 0.7) +
    geom_density(color = "darkblue", linewidth = 1) +
    labs(x = col, y = "Density") +
    theme_minimal() +
    theme(
      plot.title = element_blank(),
      axis.text.x = element_text(angle = 45, hjust = 1),
      axis.text.y = element_text(size = 8),
      plot.margin = unit(c(5, 5, 5, 5), "mm")
    )
  plot_list[[col]] <- p
}

# Arrange the plots into a grid
n_col <- 3
n_row <- ceiling(length(plot_list) / n_col)
grid.arrange(
  do.call("arrangeGrob", c(plot_list, ncol = n_col, nrow = n_row)),
  top = textGrob("Distribution of Gut Microbiome Profiles", gp = gpar(fontsiz
e = 16, fontface = "bold"))
)
```

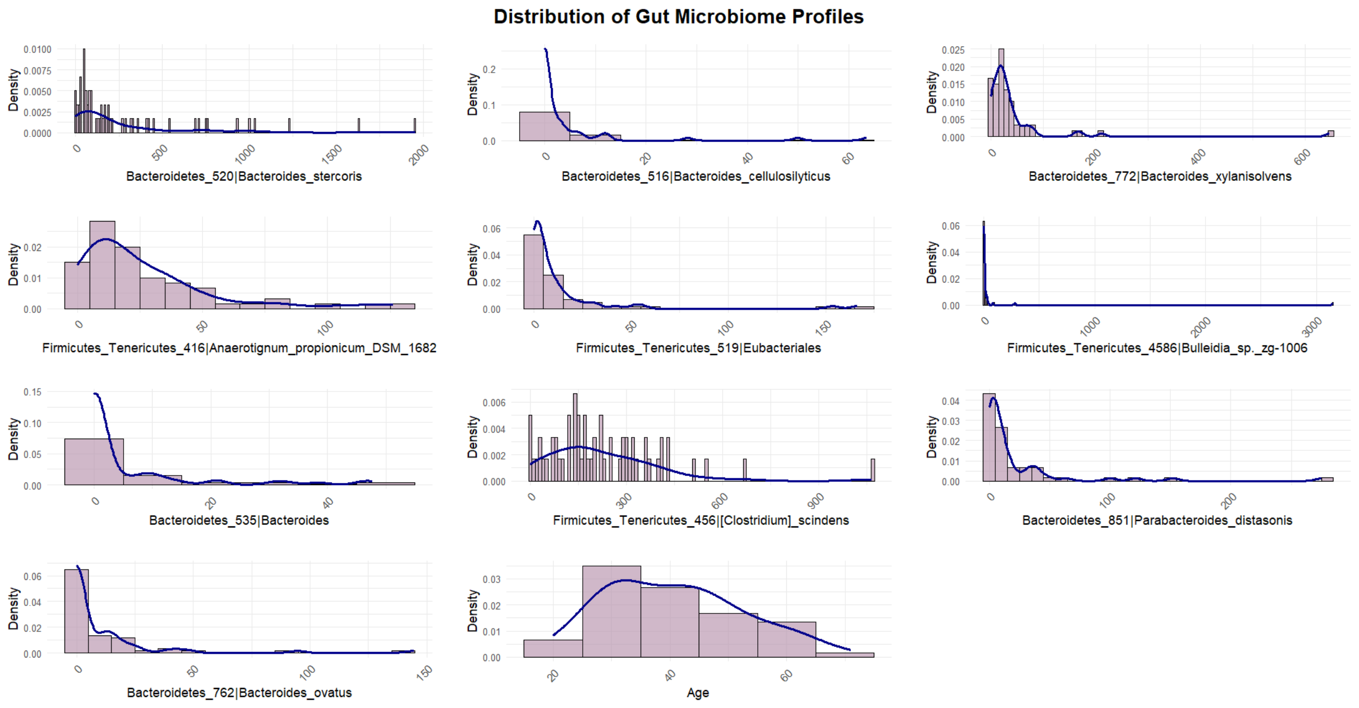

Figure 3

Key Observations:

- **Skewed Distributions:** Most microbial features are right-skewed, indicating low abundance across most samples. Transformations like logarithmic scaling may help reduce skewness and improve model performance.
- **High Variability:** Features like *Bacteroides stercoris* and *Clostridium scindens* show broader ranges, suggesting more variability among individuals. Features with higher variability could contribute significantly to distinguishing different age groups.
- **Bimodal Pattern:** *Clostridium scindens* has a bimodal distribution, possibly reflecting subgroups within the population. It might indicate different microbial profiles across age ranges, making it a potential key feature for age prediction.
- **Sparse Abundance:** Several taxa, including *Bacteroides ovatus* and *Parabacteroides distasonis*, are minimally present in most individuals. Sparse features may require specific handling, such as feature engineering or removal, to avoid noise in the model.
- **Age Distribution:** The age variable is relatively balanced, with most individuals aged between 30 and 50, providing a solid basis for training the model. This provides a good basis for training the model without significant bias towards a particular age group.

Handling Outliers

```r
# Visualize outliers using boxplots
df_numeric <- df_top_10[, sapply(df_top_10, is.numeric)]
df_melt <- melt(df_numeric)

## No id variables; using all as measure variables

ggplot(df_melt, aes(x = variable, y = value)) +
  geom_boxplot() +
  labs(title = "Boxplot for Outlier Detection", x = "Feature", y = "Value") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

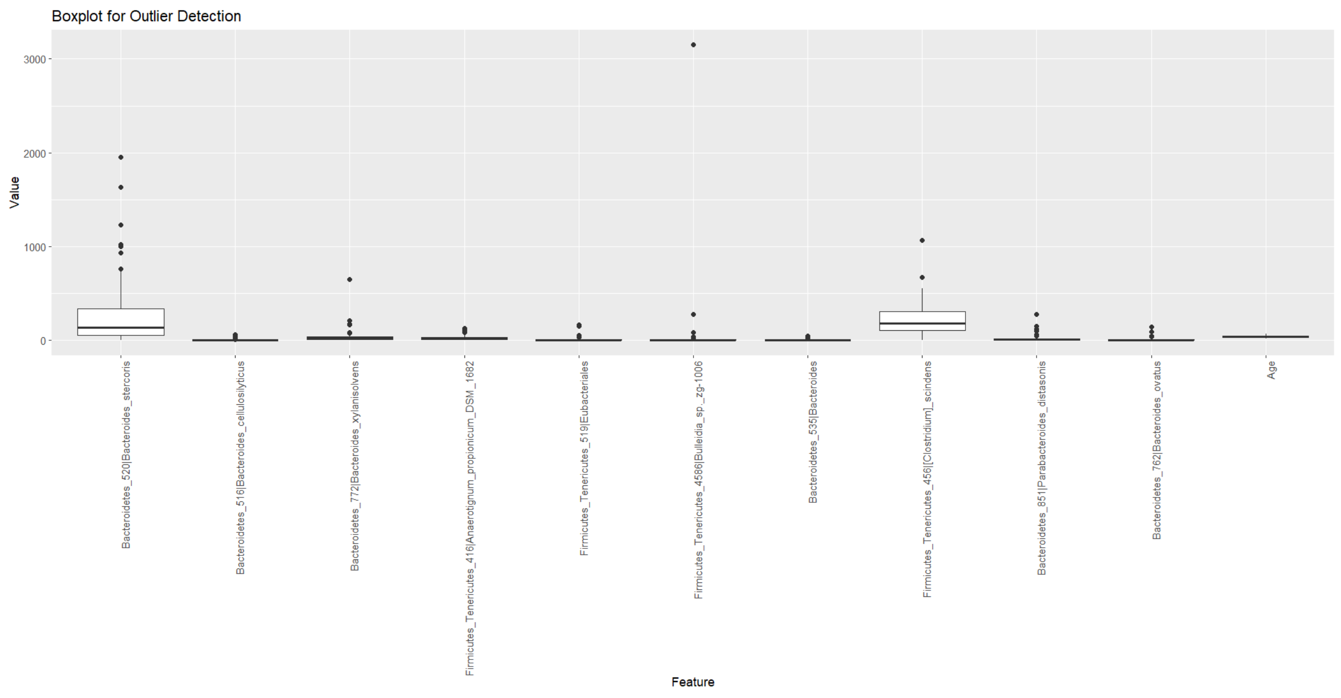

Figure 5

Figure 5 shows that certain features, like *Bacteroides stercoris* and *Clostridium scindens*, have a large number of outliers, indicating a wide variability in their values across samples. Most other features have fewer outliers and a smaller range, suggesting more consistency. The outliers in the highly variable features may impact model performance if not handled properly, so they should be considered for transformation or treatment.

Log-Transformation:

Log transformation reduces the impact of outliers by compressing large values and stabilizing variance, making data more normally distributed and easier to model. This helps to improve the performance of regression models by aligning the data with assumptions of linearity or normality, leading to more accurate and stable training. By retaining all data points and reducing skewness, the transformation ensures that the model focuses on the core patterns in the data rather than being skewed by extreme values.

```r
# Log-transform the numeric features
df_features_only <- df_numeric[, -which(names(df_numeric) == "Age")]
df_log_transformed <- log1p(df_features_only)

# Create boxplot for log-transformed data
df_melt_log <- melt(df_log_transformed)

## No id variables; using all as measure variables

ggplot(df_melt_log, aes(x = variable, y = value)) +
  geom_boxplot() +
  labs(title = "Boxplot for Outlier Detection (Log-Transformed Data)", x = "F
eature", y = "Log-Transformed Value") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

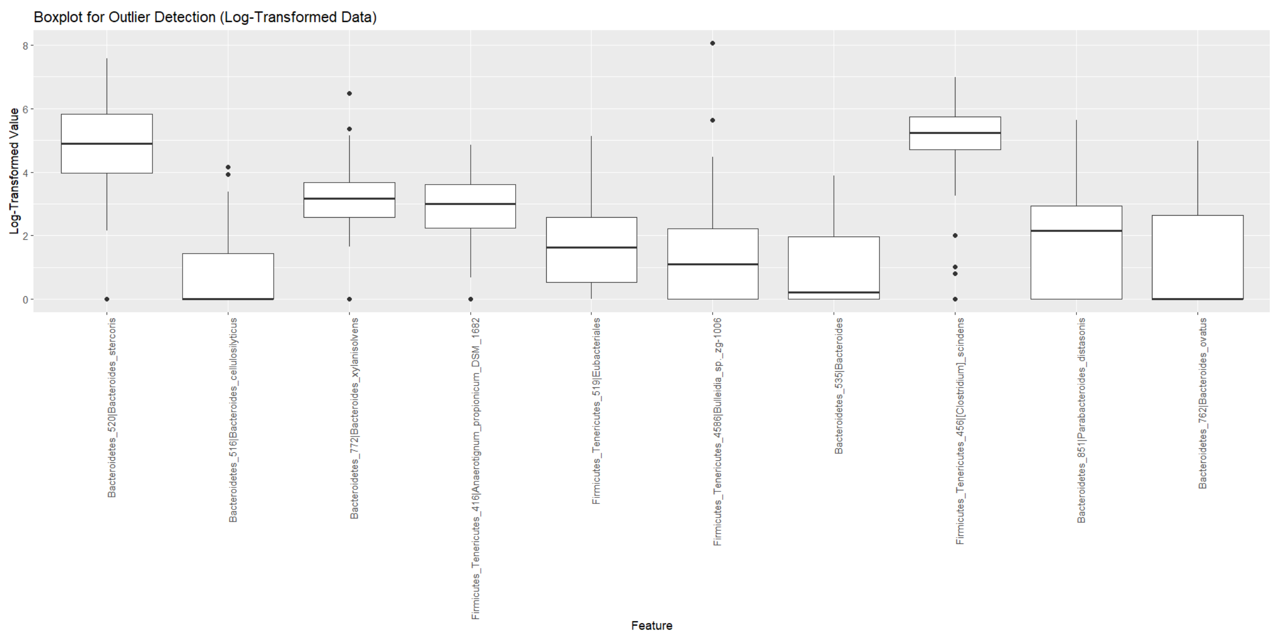

Figure 6

Figure 6 shows how log transformation has reduced the impact of outliers across the features, making the distributions more balanced. The extreme values have been compressed, bringing them closer to the rest of the data. This helps ensure that no single feature skews the model's predictions, making it easier for the machine learning model to learn consistent patterns during training. The more normalized data improves the model’s stability and overall performance.

```
# Add 'Age' back to the data
df_log_transformed <- cbind(df_log_transformed, Age = df_numeric$Age)
```

Model Development

The development of the age prediction model based on gut microbiome profiles involved several key steps. The process begins with data splitting into training and test sets, followed by preprocessing steps such as feature scaling to standardize the data. Subsequently, various machine learning models, including Random Forest, XGBoost, and Support Vector Regression (SVR), are trained and evaluated using metrics like RMSE, MAE, and R-squared to measure their predictive performance. Hyperparameter tuning is then performed for each model to optimize performance, leading to a comparison of their effectiveness. This structured methodology underscores the importance of model selection and tuning in achieving reliable predictions in this research area.

Data Splitting and Preprocessing

```
# Split the data into training and test sets
set.seed(123)
train_index <- createDataPartition(df_log_transformed$Age, p = 0.7, list = FA
LSE)
train_data <- df_log_transformed[train_index, ]
test_data <- df_log_transformed[-train_index, ]

# Define features and target variable
X_train <- train_data[, -which(names(train_data) == "Age")]
y_train <- train_data$Age
X_test <- test_data[, -which(names(test_data) == "Age")]
y_test <- test_data$Age

# Scale the features
preprocess_params <- preProcess(X_train, method = c("center", "scale"))
X_train <- predict(preprocess_params, X_train)
X_test <- predict(preprocess_params, X_test)
```

Random Forest Model

For Random Forest, the tuning process resulted in a slight increase in RMSE (from 10.75 to 10.79) and MAE (from 8.80 to 8.85), alongside a minor decrease in R-squared (from 0.17 to 0.17). This suggests that hyperparameter tuning did not substantially enhance the model's predictive capabilities, indicating that the default settings may already be well-optimized for this specific dataset.

Training the Model

```
# Train the Random Forest model
set.seed(123)
```

```
rf_model <- randomForest(x = X_train, y = y_train, ntree = 500, importance =
TRUE)

# Print the model summary
print(rf_model)

##
## Call:
##  randomForest(x = X_train, y = y_train, ntree = 500, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 130.3384
##                     % Var explained: 8.4
```

Model Evaluation (Default Settings)

```
# Evaluate the model using the test data
y_pred <- predict(rf_model, X_test)

# Calculate evaluation metrics for Random Forest
rmse_rf <- sqrt(mean((y_pred - y_test)^2))
mae_rf <- mean(abs(y_pred - y_test))
r_squared_rf <- 1 - sum((y_test - y_pred)^2) / sum((y_test - mean(y_test))^2)

# Print Random Forest evaluation results
cat("Random Forest RMSE:", rmse_rf, "\n")

## Random Forest RMSE: 10.74899

cat("Random Forest MAE:", mae_rf, "\n")

## Random Forest MAE: 8.802081

cat("Random Forest R-squared:", r_squared_rf, "\n")

## Random Forest R-squared: 0.1735782
```

Hyperparameter Tuning

```
# Set up the grid for tuning
rf_grid <- expand.grid(
  mtry = c(2, 3, 4, 5)
)

# Set up cross-validation control
control <- trainControl(
  method = "cv",
  number = 5,
  verboseIter = FALSE
)
```

```
# Train the model with tuning
set.seed(123)
rf_tune <- train(
  x = X_train,
  y = y_train,
  method = "rf",
  trControl = control,
  tuneGrid = rf_grid,
  ntree = 500,
  metric = "RMSE"
)

# Print the best hyperparameters
print(rf_tune$bestTune)

##   mtry
## 1    2
```

Model Evaluation

```
# Predict on the test data
y_pred_rf_tuned <- predict(rf_tune, newdata = X_test)

# Calculate evaluation metrics
rmse_rf_tuned <- sqrt(mean((y_pred_rf_tuned - y_test)^2))
mae_rf_tuned <- mean(abs(y_pred_rf_tuned - y_test))
r_squared_rf_tuned <- 1 - sum((y_test - y_pred_rf_tuned)^2) / sum((y_test - m
ean(y_test))^2)

# Print evaluation results
cat("Tuned Random Forest RMSE:", rmse_rf_tuned, "\n")

## Tuned Random Forest RMSE: 10.79268

cat("Tuned Random Forest MAE:", mae_rf_tuned, "\n")

## Tuned Random Forest MAE: 8.850235

cat("Tuned Random Forest R-squared:", r_squared_rf_tuned, "\n")

## Tuned Random Forest R-squared: 0.1668466
```

XGBoost Model

In contrast, XGBoost showed significant improvement following tuning. The RMSE decreased from 11.81 to 10.32, while the MAE improved from 10.16 to 8.57. Notably, the R-squared value rose from a low 0.0018 to a substantial 0.2377, indicating that the tuned model captures the variance in age predictions more effectively. This emphasizes the effectiveness of hyperparameter tuning for XGBoost, reinforcing its status as a leading choice for this prediction task.

Training the Model

```r
# Convert data to matrix format
X_train_matrix <- as.matrix(X_train)
X_test_matrix <- as.matrix(X_test)
y_train_matrix <- as.numeric(y_train)
y_test_matrix <- as.numeric(y_test)

# Set up parameters
params <- list(
  objective = "reg:squarederror",
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8
)

# Train the XGBoost model
set.seed(123)
xgb_model <- xgboost(
  data = X_train_matrix,
  label = y_train_matrix,
  params = params,
  nrounds = 500,
  verbose = 0
)
```

Model Evaluation (Default Settings)

```r
# Evaluate the XGBoost model using the test data
y_pred_xgb <- predict(xgb_model, X_test_matrix)

# Calculate evaluation metrics for XGBoost
rmse_xgb <- sqrt(mean((y_pred_xgb - y_test_matrix)^2))
mae_xgb <- mean(abs(y_pred_xgb - y_test_matrix))
r_squared_xgb <- 1 - sum((y_test_matrix - y_pred_xgb)^2) / sum((y_test_matrix
- mean(y_test_matrix))^2)

# Print XGBoost evaluation results
cat("XGBoost RMSE:", rmse_xgb, "\n")

## XGBoost RMSE: 11.81334

cat("XGBoost MAE:", mae_xgb, "\n")

## XGBoost MAE: 10.16224

cat("XGBoost R-squared:", r_squared_xgb, "\n")
```

```
## XGBoost R-squared: 0.001814585
```

Hyperparameter Tuning

```
suppressWarnings({
  # Define the grid for tuning
  xgb_grid <- expand.grid(
    nrounds = c(100, 200, 300),
    eta = c(0.01, 0.1, 0.3),
    max_depth = c(3, 6, 9),
    colsample_bytree = c(0.5, 0.7, 0.9),
    subsample = c(0.5, 0.7, 0.9),
    gamma = 0,
    min_child_weight = 1
  )

  # Set up cross-validation control
  control <- trainControl(
    method = "cv",
    number = 5,
    verboseIter = FALSE
  )

  # Train the model with tuning
  set.seed(123)
  xgb_tune <- train(
    x = X_train_matrix,
    y = y_train_matrix,
    method = "xgbTree",
    trControl = control,
    tuneGrid = xgb_grid,
    metric = "RMSE"
  )

  # Print the best hyperparameters
  print(xgb_tune$bestTune)
})
## 111     300         6 0.1     0              0.5                 1       0.
5
```

Model Evaluation

```
# Predict on the test data
y_pred_xgb_tuned <- predict(xgb_tune, newdata = X_test_matrix)

# Calculate evaluation metrics
rmse_xgb_tuned <- sqrt(mean((y_pred_xgb_tuned - y_test_matrix)^2))
mae_xgb_tuned <- mean(abs(y_pred_xgb_tuned - y_test_matrix))
r_squared_xgb_tuned <- 1 - sum((y_test_matrix - y_pred_xgb_tuned)^2) / sum((y
_test_matrix - mean(y_test_matrix))^2)
```

```
# Print evaluation results
cat("Tuned XGBoost RMSE:", rmse_xgb_tuned, "\n")

## Tuned XGBoost RMSE: 10.32355

cat("Tuned XGBoost MAE:", mae_xgb_tuned, "\n")

## Tuned XGBoost MAE: 8.571157

cat("Tuned XGBoost R-squared:", r_squared_xgb_tuned, "\n")

## Tuned XGBoost R-squared: 0.2377022
```

Support Vector Regression (SVR)

However, the SVR model demonstrated a decline in performance after tuning, with RMSE increasing from 10.42 to 11.27 and MAE rising from 8.67 to 9.29. The R-squared value also dropped from 0.22 to 0.09, suggesting that the tuning process adversely affected the model's accuracy.

Training the Model

```
# Train the SVR model
set.seed(123)
svr_model <- svm(x = X_train, y = y_train, type = "eps-regression", kernel =
"radial")

# Print the model summary
print(summary(svr_model))

##
## Call:
## svm.default(x = X_train, y = y_train, type = "eps-regression", kernel = "r
adial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  40
```

Model Evaluation (Default Settings)

```
# Predict the target variable on the test set
y_pred_svr <- predict(svr_model, X_test)
```

```r
# Calculate evaluation metrics for SVR
rmse_svr <- sqrt(mean((y_pred_svr - y_test)^2))  # Root Mean Squared Error
mae_svr <- mean(abs(y_pred_svr - y_test))  # Mean Absolute Error
r_squared_svr <- 1 - sum((y_test - y_pred_svr)^2) / sum((y_test - mean(y_test
))^2)  # R-squared

# Print evaluation results for SVR
cat("SVR RMSE:", rmse_svr, "\n")

## SVR RMSE: 10.42428

cat("SVR MAE:", mae_svr, "\n")

## SVR MAE: 8.671634

cat("SVR R-squared:", r_squared_svr, "\n")

## SVR R-squared: 0.2227539
```

Hyperparameter Tuning

```r
# Set up the grid for tuning
svr_grid <- expand.grid(
  C = c(0.1, 1, 10),
  sigma = c(0.01, 0.1, 1)
)

# Set up cross-validation control
control <- trainControl(
  method = "cv",
  number = 5,
  verboseIter = FALSE
)

# Train the model with tuning
set.seed(123)
svr_tune <- train(
  x = X_train,
  y = y_train,
  method = "svmRadial",
  trControl = control,
  tuneGrid = svr_grid,
  metric = "RMSE"
)

# Print the best hyperparameters
print(svr_tune$bestTune)
```

```
##   sigma  C
## 7  0.01 10
```

Model Evaluation

```
# Predict on the test data
y_pred_svr_tuned <- predict(svr_tune, newdata = X_test)

# Calculate evaluation metrics
rmse_svr_tuned <- sqrt(mean((y_pred_svr_tuned - y_test)^2))
mae_svr_tuned <- mean(abs(y_pred_svr_tuned - y_test))
r_squared_svr_tuned <- 1 - sum((y_test - y_pred_svr_tuned)^2) / sum((y_test -
mean(y_test))^2)

# Print evaluation results
cat("Tuned SVR RMSE:", rmse_svr_tuned, "\n")

## Tuned SVR RMSE: 11.27377

cat("Tuned SVR MAE:", mae_svr_tuned, "\n")

## Tuned SVR MAE: 9.294454

cat("Tuned SVR R-squared:", r_squared_svr_tuned, "\n")

## Tuned SVR R-squared: 0.09091534
```

Results

Residual Plots

```
# Create residuals
residuals_rf <- y_test - y_pred_rf_tuned
residuals_xgb <- y_test - y_pred_xgb_tuned
residuals_svr <- y_test - y_pred_svr_tuned

# Combine residuals into a data frame
residuals_df <- data.frame(
  Model = rep(c("Random Forest", "XGBoost", "SVR"), each = length(y_test)),
  Predicted = c(y_pred_rf_tuned, y_pred_xgb_tuned, y_pred_svr_tuned),
  Residuals = c(residuals_rf, residuals_xgb, residuals_svr)
)

# Plot residuals
ggplot(residuals_df, aes(x = Predicted, y = Residuals, color = Model)) +
  geom_point(alpha = 0.6) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
  facet_wrap(~ Model) +
  labs(title = "Residual Plots for Each Model", x = "Predicted Values", y = "
Residuals") +
  theme_minimal() +
  theme(legend.position = "none")
```

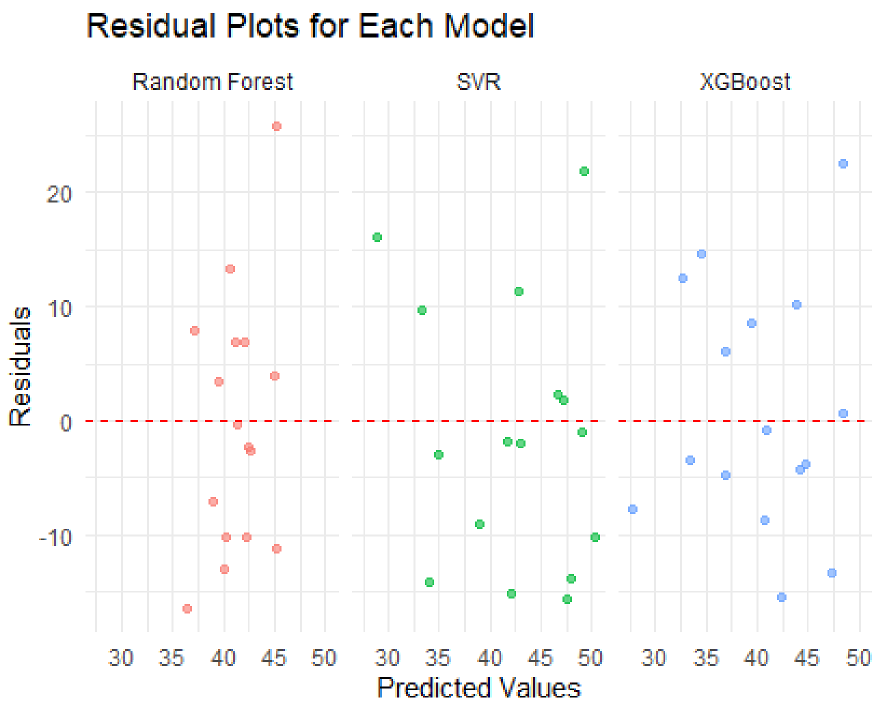

Figure 7

The residual plots as shown in Figure 7 reveal how errors behave across the predicted values:

- **Random Forest:** The residuals show a clear pattern of heteroscedasticity, with errors increasing as predicted values increase. This suggests that Random Forest struggles more with higher predicted ages, leading to larger errors. The spread of residuals between -10 and +10 shows that the model does reasonably well for mid-range predictions but struggles with extreme values.
- **XGBoost:** The residuals are more dispersed, especially for lower predicted values, showing that the model struggles with lower age predictions. While XGBoost reduces overall error, there is still significant variability, which suggests some difficulties in capturing the full relationship between features and age.
- **SVR:** The residuals for SVR appear more consistent, with fewer large outliers. However, this consistency doesn't translate to accurate predictions, as seen in the actual vs. predicted plot. SVR seems to produce more stable errors but fails to predict age accurately across the full range.

Actual vs. Predicted Plots

Random Forest:

```
ggplot(data.frame(Actual = y_test, Predicted = y_pred_rf_tuned), aes(x = Actu
al, y = Predicted)) +
  geom_point(color = "#987D9A") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(title = "Actual vs. Predicted - Random Forest", x = "Actual Values", y
= "Predicted Values") +
  theme_minimal()
```

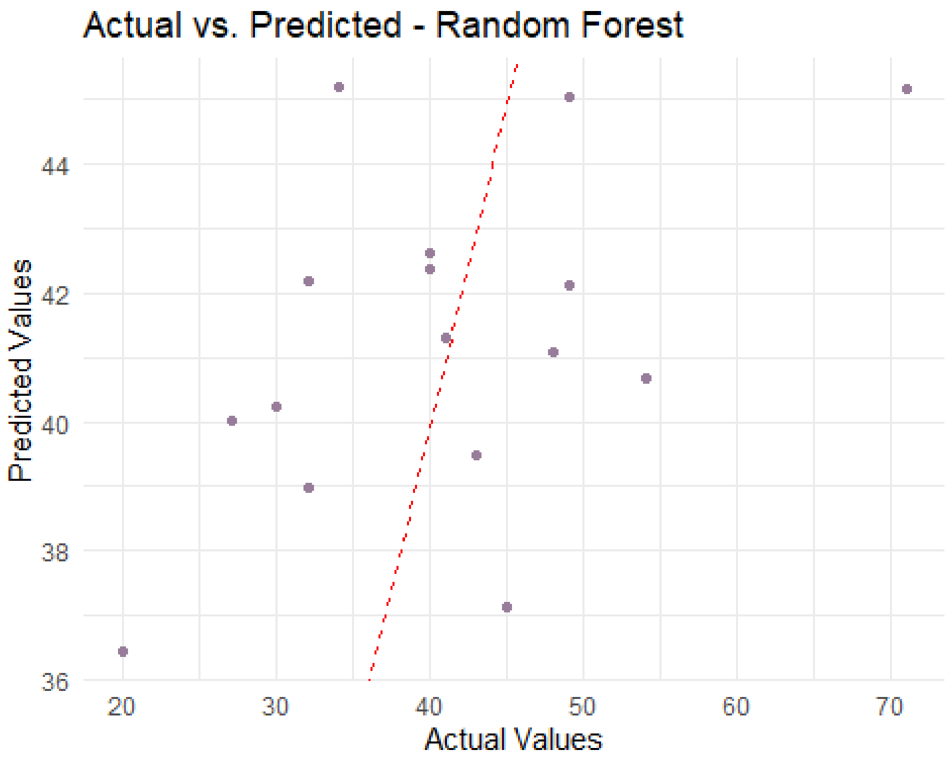

Figure 8

As shown in Figure 8, the actual vs. predicted plot confirms the issues observed in the residual plot. Random Forest predictions deviate significantly from the actual values, particularly for higher ages. This suggests that while it handles mid-range age predictions relatively well, it struggles with more extreme values, leading to a wider spread in predictions.

XGBoost:

```
ggplot(data.frame(Actual = y_test, Predicted = y_pred_xgb_tuned), aes(x = Act
ual, y = Predicted)) +
  geom_point(color = "#FFEEAD") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(title = "Actual vs. Predicted - XGBoost", x = "Actual Values", y = "Pr
edicted Values") +
  theme_minimal()
```

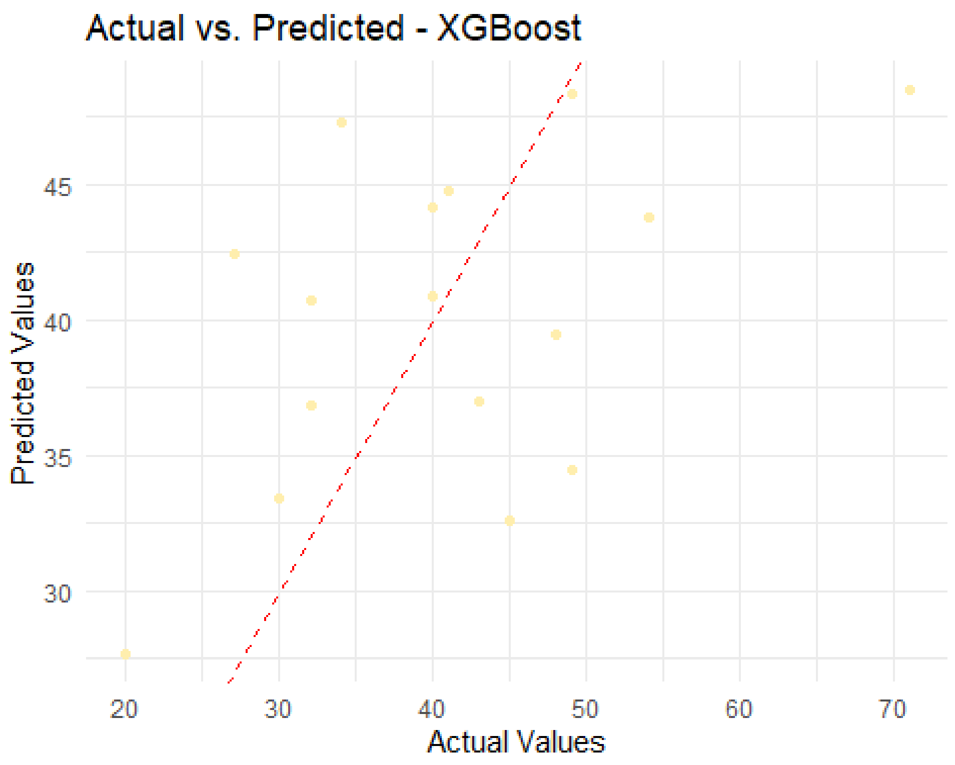

Figure 9

Figure 9 shows that the XGBoost model produces predictions that are generally closer to the actual values, particularly in the middle age ranges. However, there is some variance in the lower predicted ages, which is consistent with the residual plot, indicating that XGBoost struggles slightly with these predictions. Overall, XGBoost aligns better with the actual values compared to Random Forest or SVR.

SVR:

```
ggplot(data.frame(Actual = y_test, Predicted = y_pred_svr_tuned), aes(x = Act
ual, y = Predicted)) +
  geom_point(color = "#9BC4E2") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(title = "Actual vs. Predicted - SVR", x = "Actual Values", y = "Predic
ted Values") +
  theme_minimal()
```

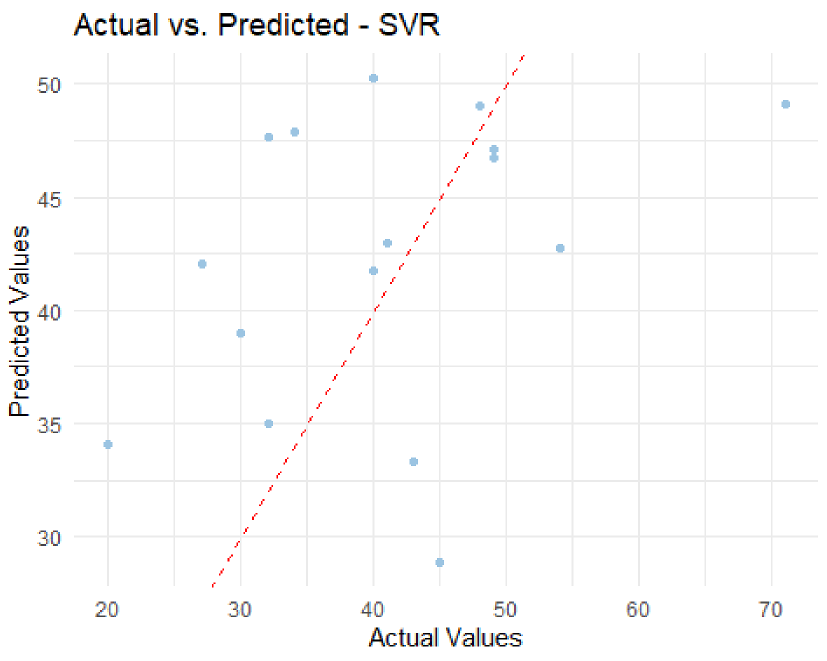

Figure 10

The actual vs. predicted plot for SVR in Figure 10 shows that many predictions are far from the diagonal (ideal prediction line), indicating that SVR struggles to capture the relationship between the features and age effectively. Although the residuals are more stable, the overall accuracy of the model is lower.

Model Comparison

```r
# Create a data frame with evaluation metrics
model_results <- data.frame(
  Model = c("Random Forest", "XGBoost", "SVR"),
  RMSE = c(rmse_rf_tuned, rmse_xgb_tuned, rmse_svr_tuned),
  MAE = c(mae_rf_tuned, mae_xgb_tuned, mae_svr_tuned),
  R_squared = c(r_squared_rf_tuned, r_squared_xgb_tuned, r_squared_svr_tuned)
)

# Melt the data for plotting
model_results_long <- melt(model_results, id.vars = "Model", variable.name =
"Metric", value.name = "Value")

# Plot RMSE
ggplot(subset(model_results_long, Metric == "RMSE"), aes(x = Model, y = Value
, fill = Model)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Value, 2)), vjust = -0.5) +
  labs(title = "RMSE Comparison by Model", x = "Model", y = "RMSE") +
  theme_minimal() +
  scale_fill_manual(values = c("Random Forest" = "#FFEEAD", "XGBoost" = "#987
D9A", "SVR" = "#9BC4E2")) +
  theme(legend.position = "none")
```

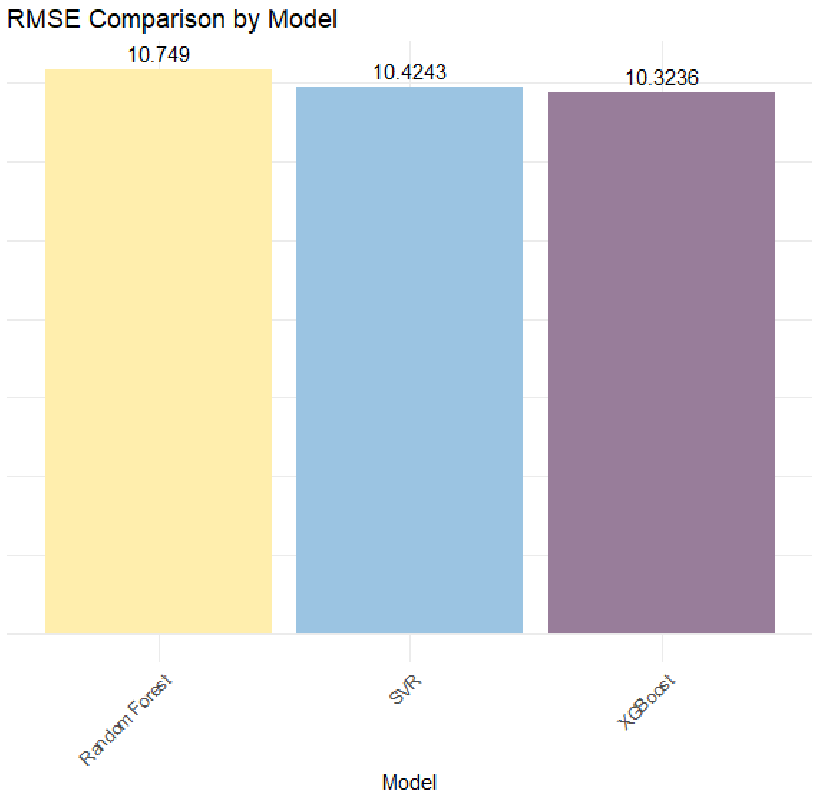

Figure 11

RMSE measures the average magnitude of errors between predicted and actual values, giving higher weight to larger errors. A lower RMSE indicates better predictive accuracy. In this analysis, XGBoost has the lowest RMSE at 10.32, suggesting it consistently makes smaller prediction errors compared to Random Forest and SVR.

```
# Plot MAE
ggplot(subset(model_results_long, Metric == "MAE"), aes(x = Model, y = Value,
fill = Model)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Value, 2)), vjust = -0.5) +
  labs(title = "MAE Comparison by Model", x = "Model", y = "MAE") +
  theme_minimal() +
  scale_fill_manual(values = c("Random Forest" = "#FFEEAD", "XGBoost" = "#987
D9A", "SVR" = "#9BC4E2")) +
  theme(legend.position = "none")
```

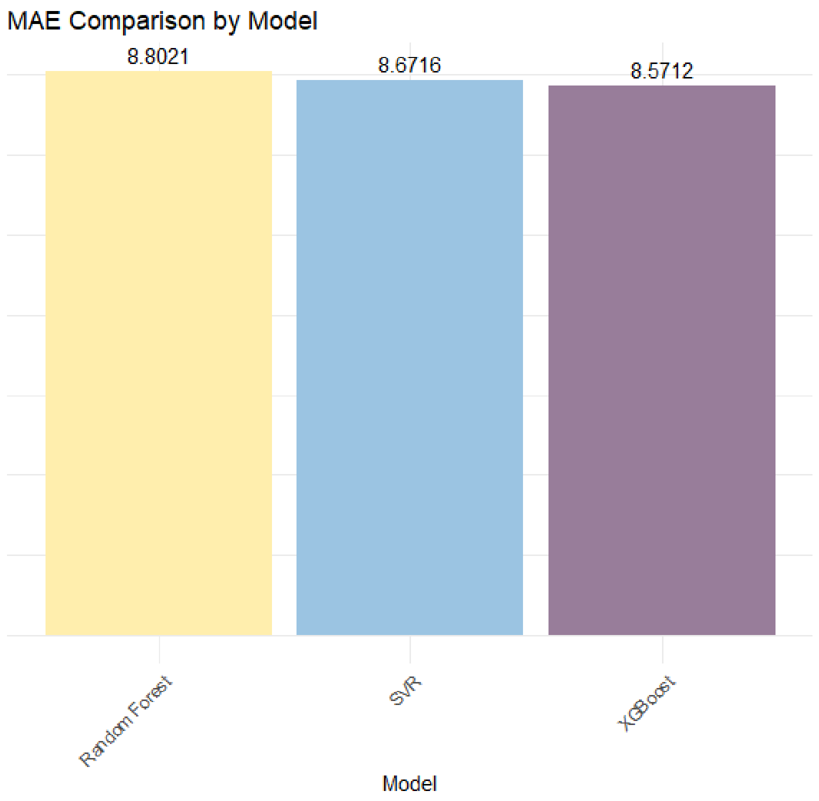

Figure 12

MAE represents the average absolute difference between predicted and actual values. Unlike RMSE, it treats all errors equally, making it useful for understanding the average prediction error in a more straightforward manner. XGBoost again excels with an MAE of 8.57, indicating it reliably predicts age across different samples.

```r
# Plot R-squared
ggplot(subset(model_results_long, Metric == "R_squared"), aes(x = Model, y =
Value, fill = Model)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Value, 2)), vjust = -0.5) +
  labs(title = "R-squared Comparison by Model", x = "Model", y = "R-squared")
+
  theme_minimal() +
  scale_fill_manual(values = c("Random Forest" = "#FFEEAD", "XGBoost" = "#987
D9A", "SVR" = "#9BC4E2")) +
  theme(legend.position = "none")
```

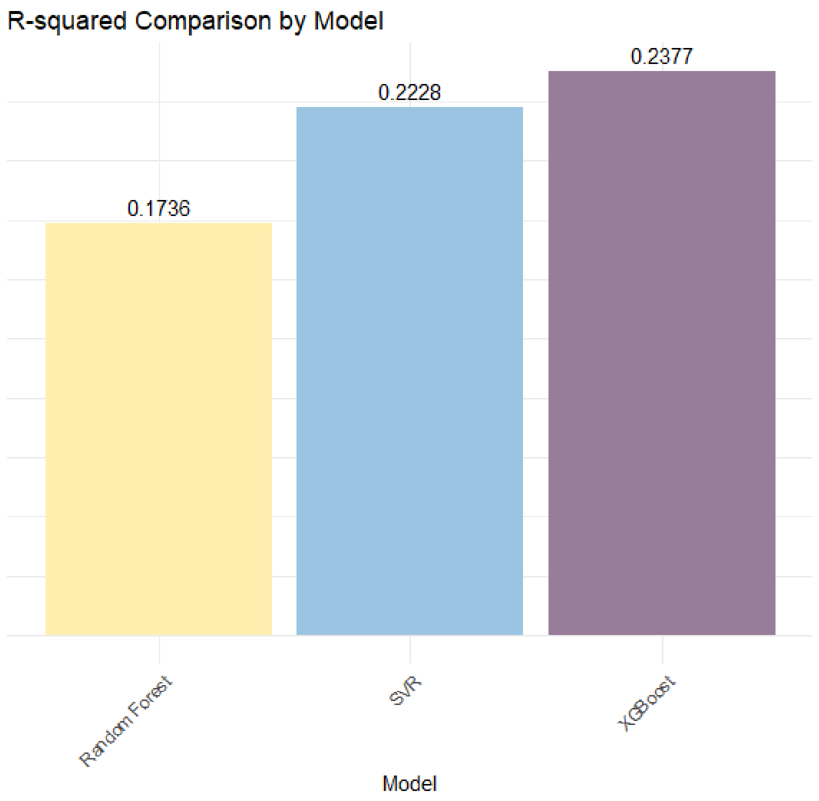

Figure 13

R-squared quantifies the proportion of variance in the dependent variable (age) explained by the independent variables (microbiome features). A higher R-squared value indicates a better fit of the model to the data. XGBoost's R-squared of 0.24 indicates it explains more variance in age than Random Forest and SVR, suggesting it captures the underlying relationships in the data more effectively.

Conclusion

In conclusion, this project successfully developed machine learning models to predict age based on gut microbiome profiles, highlighting the significant influence of microbial composition on age estimation. Through thorough data preprocessing and feature selection, the XGBoost model outperformed both Random Forest and SVR, as evidenced by its lower RMSE and MAE values and higher R-squared. These results indicate its effectiveness in capturing the complex relationships within the data. The findings emphasize the potential of using gut microbiome data for predicting age, which could provide valuable insights for age-related health assessments and interventions. Although XGBoost demonstrates the best performance among the models, with an RMSE of 10.32, an MAE of 8.57, and an R-squared of 0.24, these metrics suggest that there is still considerable room for improvement in age prediction. Future efforts should focus on refining modeling techniques and feature selection to enhance the model's overall performance.

References

Bobbo, T., Biscarini, F., Yaddehige, S. K., Alberghini, L., Rigoni, D., Bianchi, N., & Taccioli, C. (2024). Machine learning classification of archaea and bacteria identifies novel predictive genomic features. *BMC Genomics, 25*(1), 955. doi:10.1186/s12864-024-10832-y

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning, 46*(1), 389-422. doi:10.1023/A:1012487302797