Stacked Model Predicting Cancer (source code of report)

Qinglan Ouyang

2025-12-12

```
# library loading
library(tidyverse)
```

```
## Warning: package 'forcats' was built under R version 4.4.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.1     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
e errors
```

```
library(caret)        # model training and cross-validation
```

```
## Warning: package 'caret' was built under R version 4.4.3
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(e1071)        # SVM and Naive Bayes
```

```
## Warning: package 'e1071' was built under R version 4.4.3
```

```
library(pROC)         # ROC curve and AUC
```

```
## Warning: package 'pROC' was built under R version 4.4.3
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(doParallel)   # parallel computing
```

```
## Warning: package 'doParallel' was built under R version 4.4.3
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 4.4.3
```

```
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loading required package: iterators
```

```
## Warning: package 'iterators' was built under R version 4.4.3
```

```
## Loading required package: parallel
```

```
library(RColorBrewer) # color
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.4.3
```

```
library(lightgbm)
```

```
## Warning: package 'lightgbm' was built under R version 4.4.3
```

```
library(reticulate)
```

```
## Warning: package 'reticulate' was built under R version 4.4.3
```

Data loading and data cleaning

```
# loaing
df <- read.csv("C:/Users/ouyan/OneDrive/文档/BIOSTAT625/The_Cancer_data_1500_V2.csv")
summary(df)
```

```
##       Age            Gender            BMI           Smoking
##  Min.   :20.00   Min.   :0.0000   Min.   :15.00   Min.   :0.0000
##  1st Qu.:35.00   1st Qu.:0.0000   1st Qu.:21.48   1st Qu.:0.0000
##  Median :51.00   Median :0.0000   Median :27.60   Median :0.0000
##  Mean   :50.32   Mean   :0.4907   Mean   :27.51   Mean   :0.2693
##  3rd Qu.:66.00   3rd Qu.:1.0000   3rd Qu.:33.85   3rd Qu.:1.0000
##  Max.   :80.00   Max.   :1.0000   Max.   :39.96   Max.   :1.0000
##   GeneticRisk     PhysicalActivity  AlcoholIntake      CancerHistory
##  Min.   :0.0000   Min.   :0.00241   Min.   :0.001215   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:2.43461   1st Qu.:1.210598   1st Qu.:0.000
##  Median :0.0000   Median :4.83432   Median :2.382971   Median :0.000
##  Mean   :0.5087   Mean   :4.89793   Mean   :2.417987   Mean   :0.144
##  3rd Qu.:1.0000   3rd Qu.:7.40990   3rd Qu.:3.585624   3rd Qu.:0.000
##  Max.   :2.0000   Max.   :9.99461   Max.   :4.987115   Max.   :1.000
##    Diagnosis
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3713
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

```
print(head(df))
```

```
##   Age Gender      BMI Smoking GeneticRisk PhysicalActivity AlcoholIntake
## 1  58      1 16.08531       0           1         8.146251      4.148219
## 2  71      0 30.82878       0           1         9.361630      3.519683
## 3  48      1 38.78508       0           2         5.135179      4.728368
## 4  34      0 30.04030       0           0         9.502792      2.044636
## 5  62      1 35.47972       0           0         5.356890      3.309849
## 6  27      0 37.10516       0           1         3.941905      2.324274
##   CancerHistory Diagnosis
## 1             1         1
## 2             0         0
## 3             0         1
## 4             0         0
## 5             0         1
## 6             0         0
```

```
colSums(is.na(df))
```

```
##              Age           Gender              BMI          Smoking
##                0                0                0                0
##      GeneticRisk PhysicalActivity    AlcoholIntake    CancerHistory
##                0                0                0                0
##        Diagnosis
##                0
```

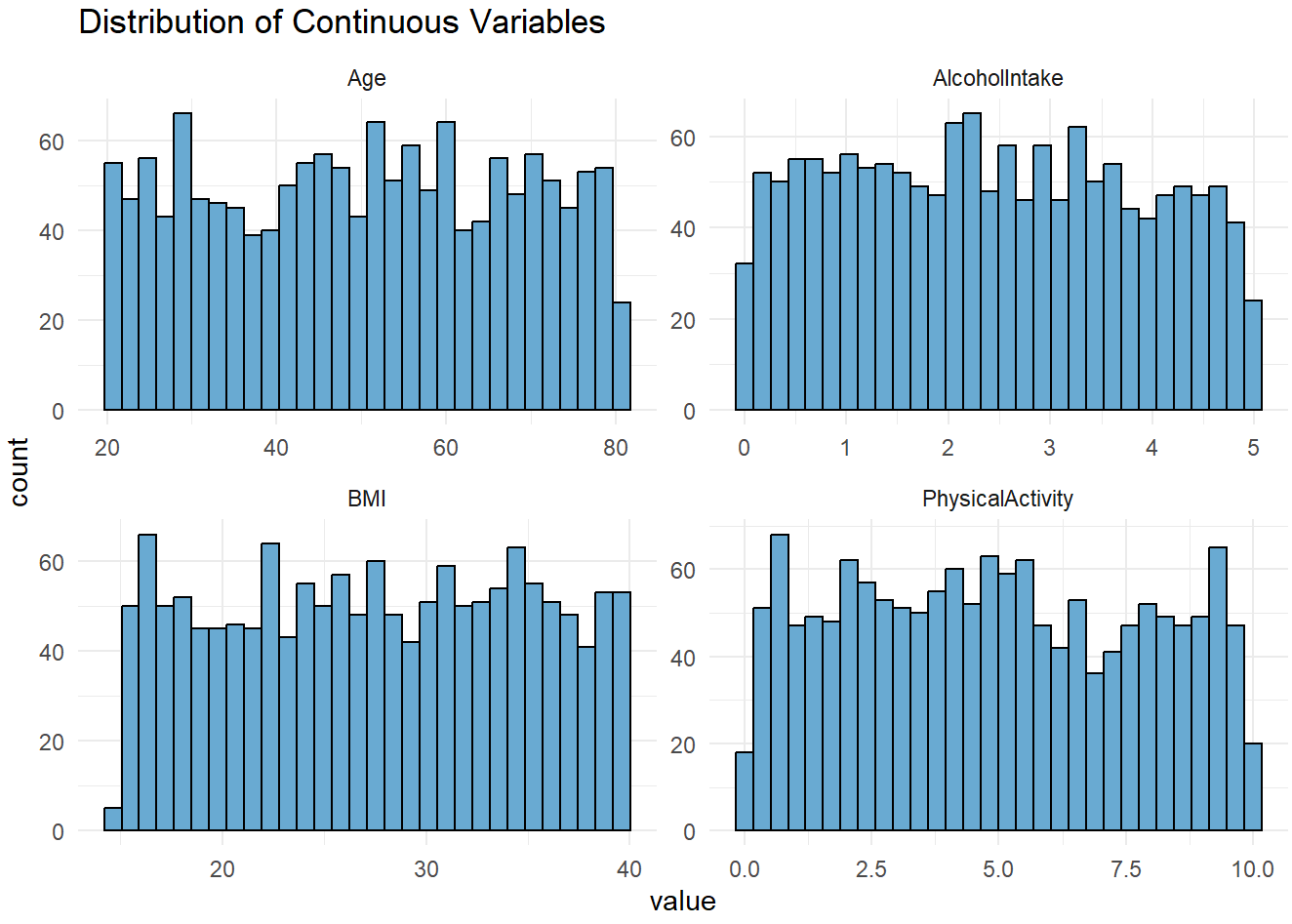
```
library(tidyverse)
df <- df %>%
  mutate(
    Gender = factor(as.character(Gender),
                    levels = c("0", "1"),
                    labels = c("Male","Female")),

    Smoking = factor(as.character(Smoking),
                     levels = c("0", "1"),
                     labels = c("No","Yes")),

    CancerHistory = factor(as.character(CancerHistory),
                           levels = c("0", "1"),
                           labels = c("No","Yes")),

    GeneticRisk = factor(as.character(GeneticRisk),
                         levels = c("0","1","2"),
                         labels = c("Low","Medium","High")),

    Diagnosis = factor(as.character(Diagnosis),
                       levels = c("0","1"),
                       labels = c("NoCancer","Cancer"))
  )
print(head(df))
```

```
##   Age Gender      BMI Smoking GeneticRisk PhysicalActivity AlcoholIntake
## 1  58 Female 16.08531      No      Medium         8.146251      4.148219
## 2  71   Male 30.82878      No      Medium         9.361630      3.519683
## 3  48 Female 38.78508      No        High         5.135179      4.728368
## 4  34   Male 30.04030      No         Low         9.502792      2.044636
## 5  62 Female 35.47972      No         Low         5.356890      3.309849
## 6  27   Male 37.10516      No      Medium         3.941905      2.324274
##   CancerHistory Diagnosis
## 1           Yes    Cancer
## 2            No  NoCancer
## 3            No    Cancer
## 4            No  NoCancer
## 5            No    Cancer
## 6            No  NoCancer
```

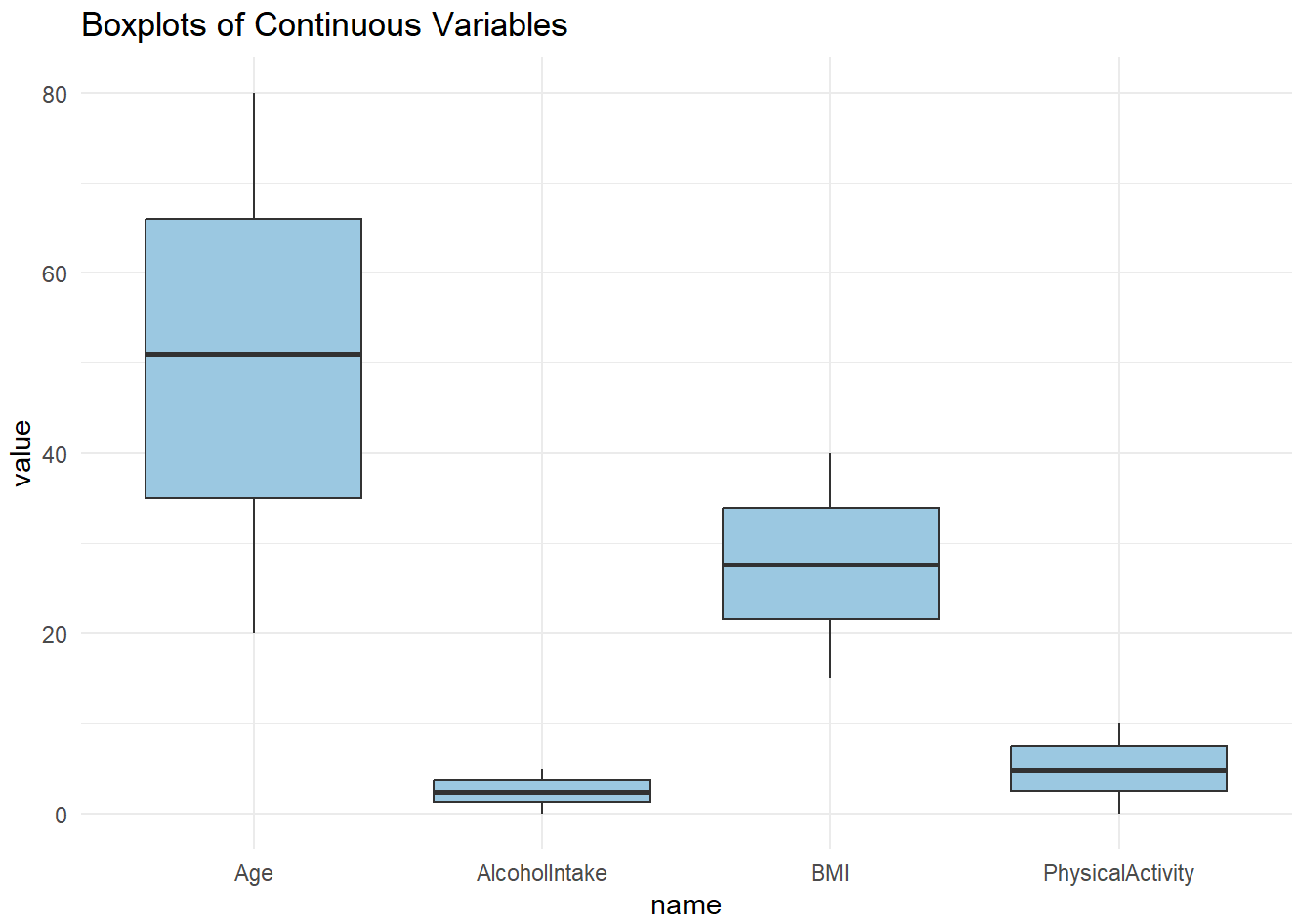
```
continuous_vars <- c("Age","BMI","PhysicalActivity","AlcoholIntake")
binary_vars <- c("Gender","Smoking","CancerHistory")
categorical_vars <- c("GeneticRisk")
target_var <- "Diagnosis"
```

Variable visualization

```
df %>%
  pivot_longer(all_of(continuous_vars)) %>%
  ggplot(aes(x = value)) +
  geom_histogram(bins = 30, fill = "#6baed6", color = "black") +
  facet_wrap(~ name, scales = "free") +
  theme_minimal() +
  labs(title = "Distribution of Continuous Variables")
```



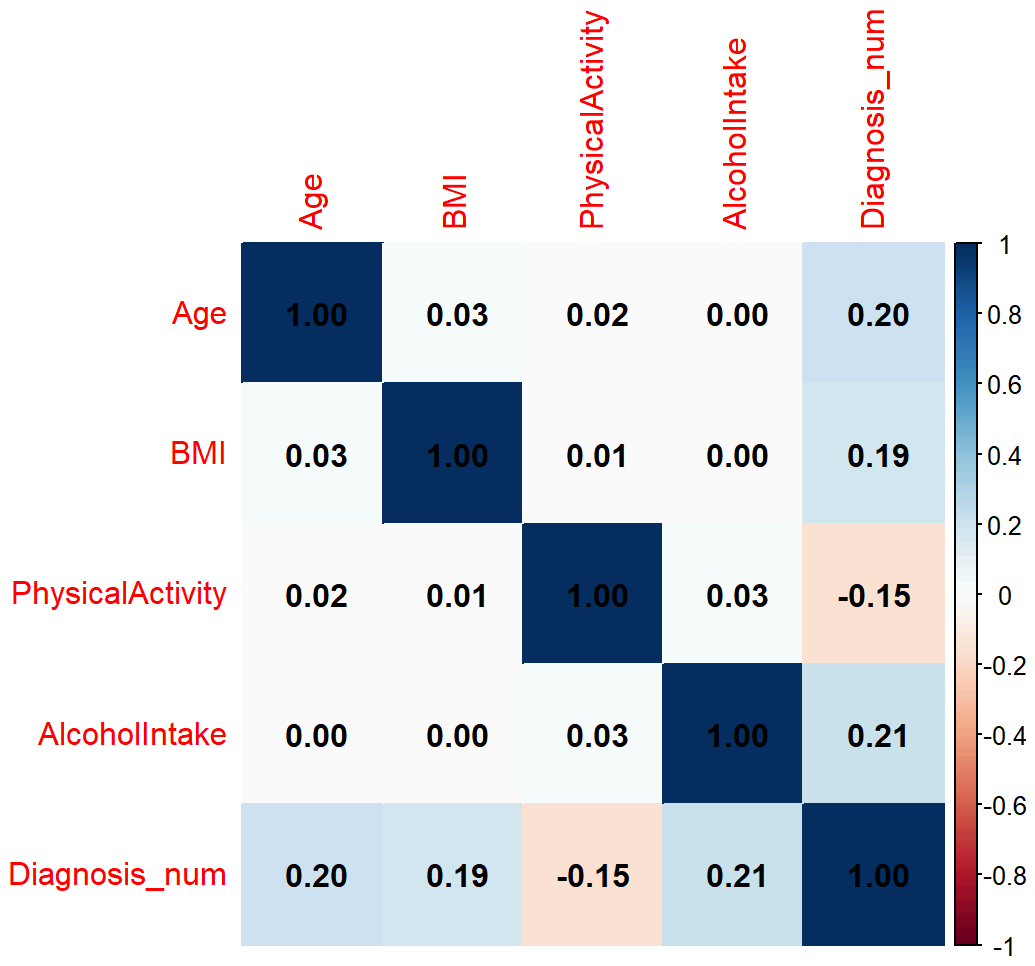

```
df %>%
  pivot_longer(all_of(continuous_vars)) %>%
  ggplot(aes(x = name, y = value)) +
  geom_boxplot(fill = "#9ecae1") +
  theme_minimal() +
  labs(title = "Boxplots of Continuous Variables")
```



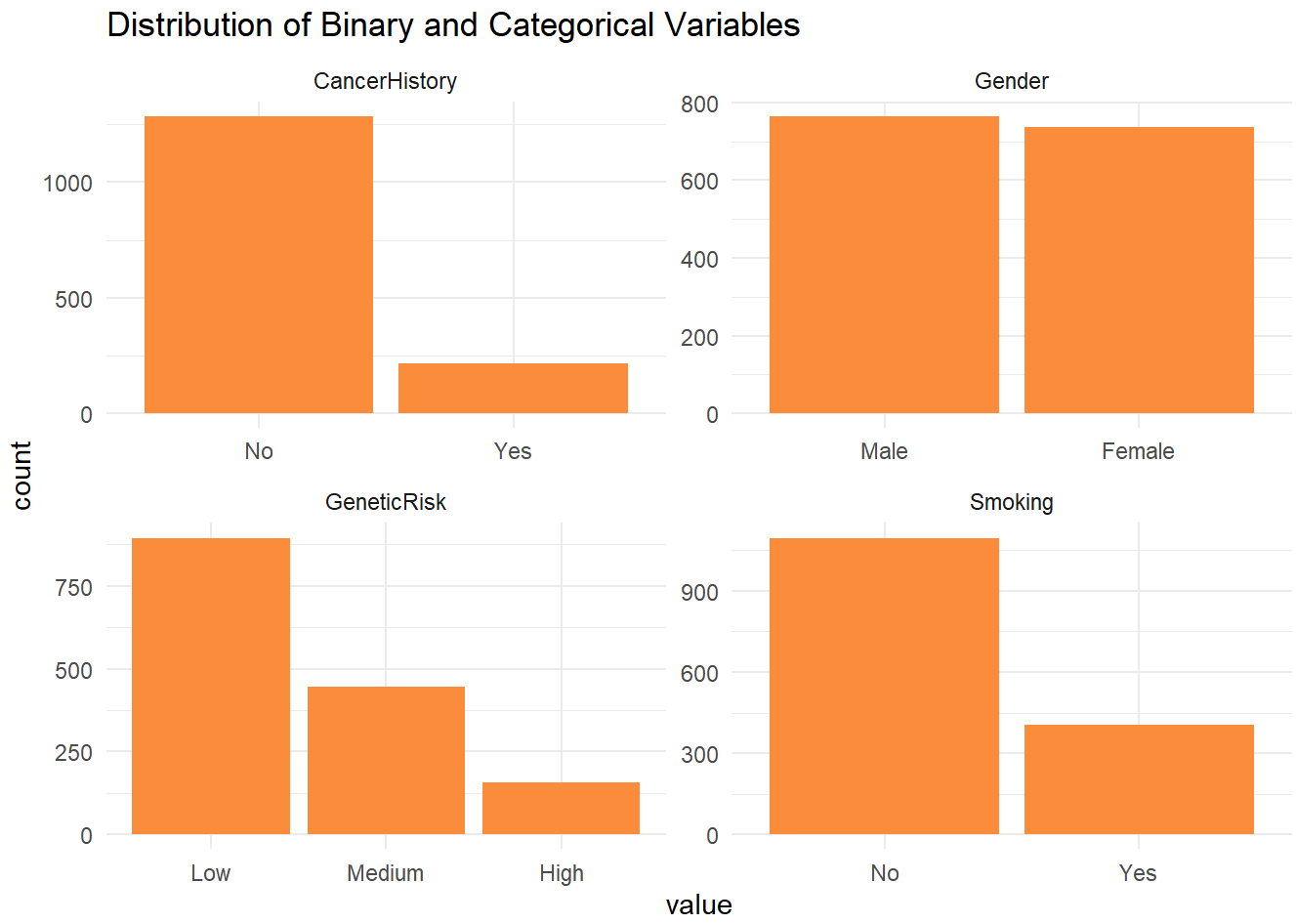

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
df_corr <- df %>%
  mutate(Diagnosis_num = ifelse(Diagnosis == "Cancer", 1, 0)) %>%
  select(all_of(continuous_vars), Diagnosis_num)

corrplot(cor(df_corr), method = "color", addCoef.col = "black")
```

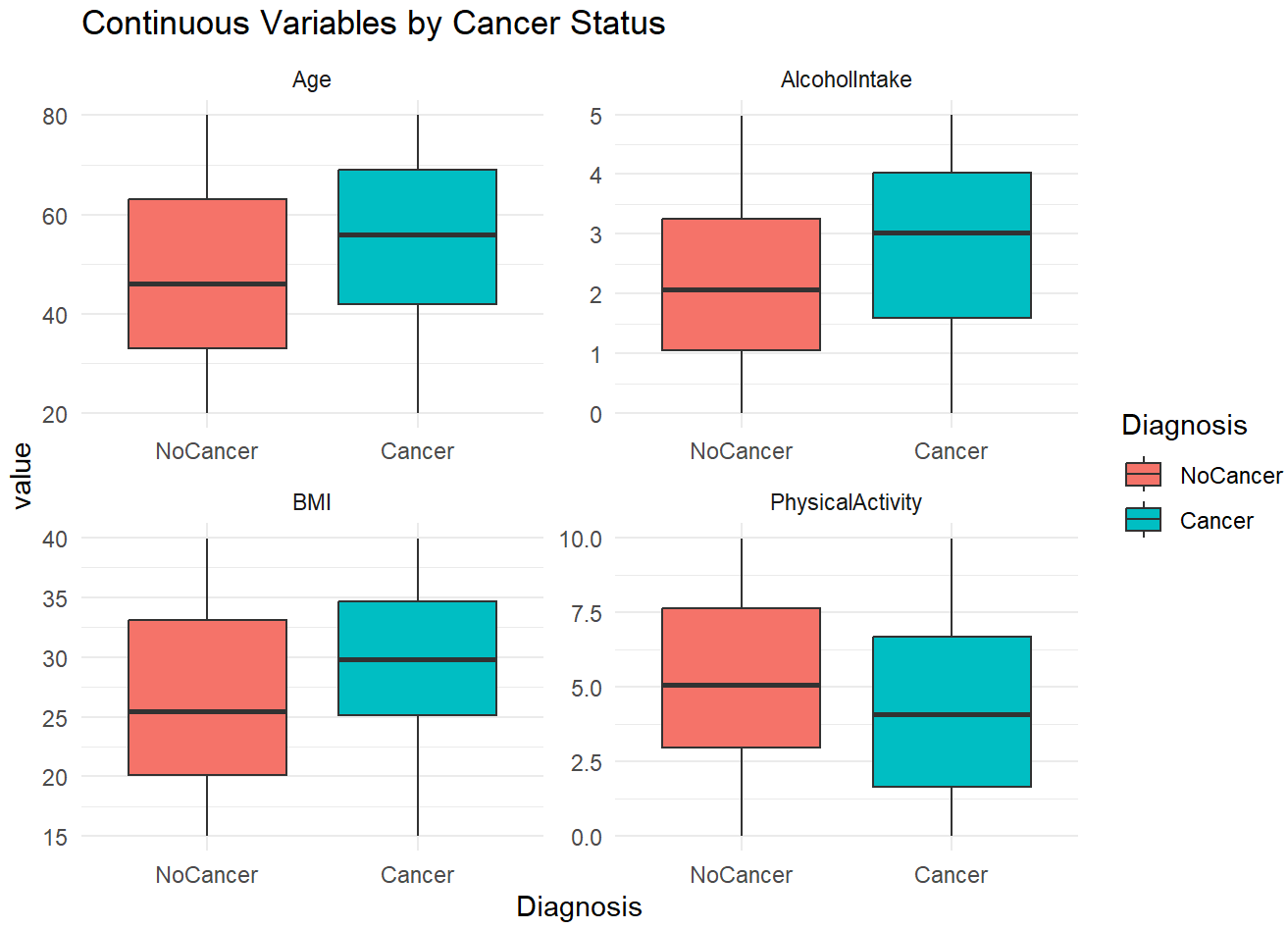



```
df %>%
  select(all_of(c(binary_vars, categorical_vars))) %>%
  pivot_longer(everything()) %>%
  ggplot(aes(x = value)) +
  geom_bar(fill = "#fd8d3c") +
  facet_wrap(~ name, scales = "free") +
  theme_minimal() +
  labs(title = "Distribution of Binary and Categorical Variables")
```

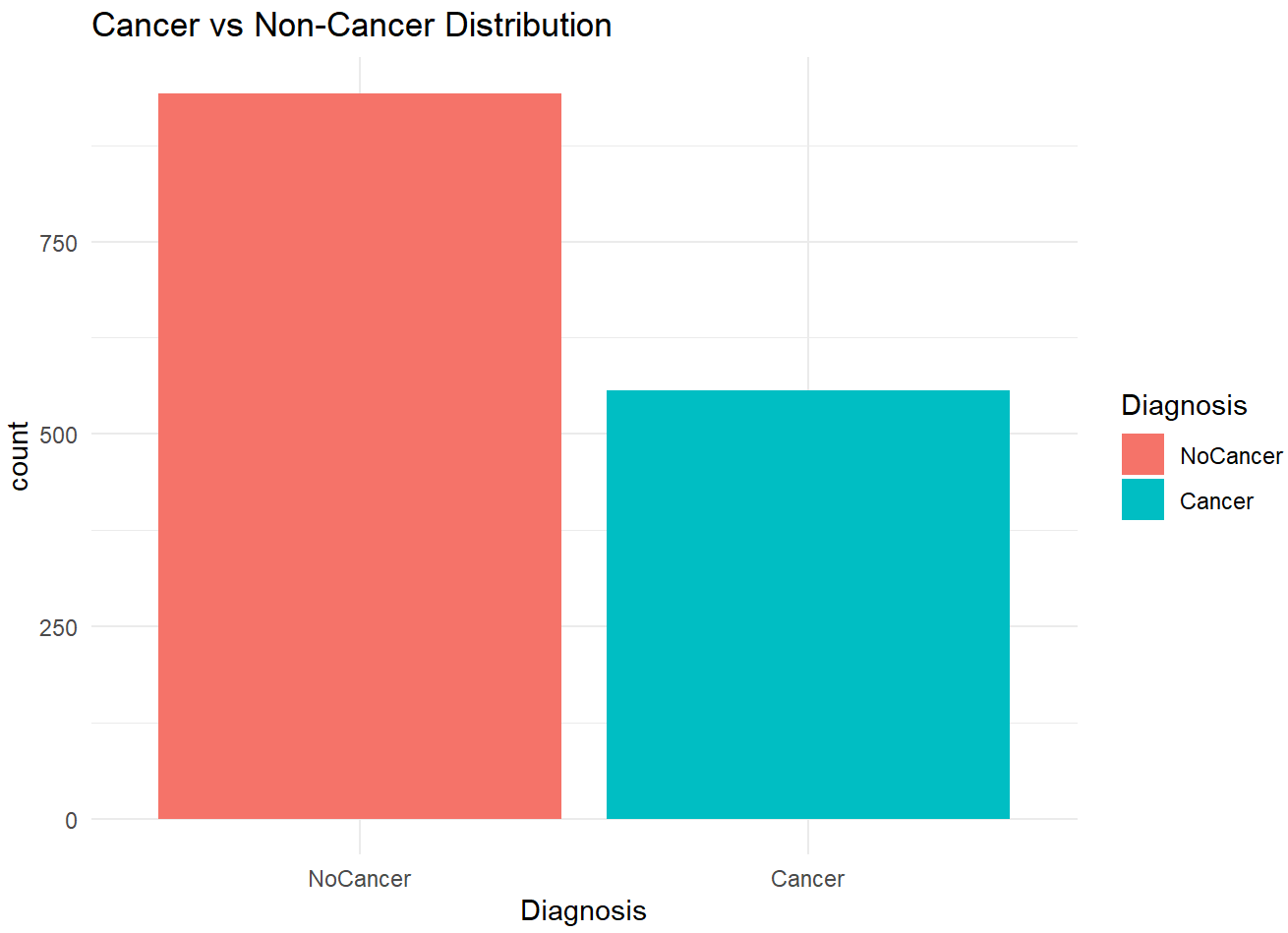



```
df %>%
  pivot_longer(continuous_vars) %>%
  ggplot(aes(x = Diagnosis, y = value, fill = Diagnosis)) +
  geom_boxplot() +
  facet_wrap(~ name, scales = "free") +
  theme_minimal() +
  labs(title = "Continuous Variables by Cancer Status")
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## ℹ Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(continuous_vars)
##
##   # Now:
##   data %>% select(all_of(continuous_vars))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```
# check whether the data balance
df %>% ggplot(aes(x = Diagnosis, fill = Diagnosis)) +
  geom_bar() +
  theme_minimal() +
  labs(title = "Cancer vs Non-Cancer Distribution")
```


Cancer vs Non-Cancer Distribution
750
500
250
0
count
NoCancer
Cancer
Diagnosis
Diagnosis
NoCancer
Cancer

Split training and testing sets

```
df_for_modeling <- df

# --- 1. Split X and y ---
X <- df_for_modeling %>% select(-all_of(target_var))
y <- df_for_modeling[[target_var]]

# --- 2. Spli training and testing sets (80/20 stratified split) ---
set.seed(42)
# Use the target variable y for stratified sampling
train_index <- createDataPartition(y, p = 0.8, list = FALSE)

X_train <- X[train_index, ]
X_test <- X[-train_index, ]
y_train <- y[train_index]
y_test <- y[-train_index]


# --- 3. Standardization (Learn parameters only on the training set) ---
# Build preprocessing steps: Fit the standardization parameters on continuous features of the tr
aining set only
preprocessor <- preProcess(X_train[, continuous_vars], method = c("center", "scale"))

# Apply preprocessing: Transform both the training and testing sets
X_train_processed <- predict(preprocessor, X_train)
X_test_processed <- predict(preprocessor, X_test)


# --- 4. 4. Create final training/testing datasets (Combine X and y) ---
# Note: X_train_processed already includes all factor variables
train_data <- cbind(Diagnosis = y_train, X_train_processed)
test_data <- cbind(Diagnosis = y_test, X_test_processed)

# Final check: Ensure the target variable is a two-level factor (already done, but this ensures
data integrity)
train_data$Diagnosis <- factor(train_data$Diagnosis, levels = c("NoCancer", "Cancer"))
test_data$Diagnosis <- factor(test_data$Diagnosis, levels = c("NoCancer", "Cancer"))

# The train_data and test_data are now ready for model training
cat("Data splitting and standardization are complete, ready to proceed to the model training sta
ge.\n")
```

```
## Data splitting and standardization are complete, ready to proceed to the model training stag
e.
```

Model fitting

```
# 1. Define 5-Fold Stratified CV function
control <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = defaultSummary,
  savePredictions = "final",
  verboseIter = FALSE
)


# 2. Model names:
# Logistic Regression -> glm
# Decision Tree -> rpart
# Random Forest -> rf
# Gradient Boosting -> gbm
# SVM -> svmRadial
# k-NN -> knn
# Naive Bayes -> nb

models_to_run <- c("glm", "rpart", "rf", "gbm", "svmRadial", "knn", "nb")
results_cv <- list()

# 3. Run cross-validation
if (requireNamespace("doParallel", quietly = TRUE)) {
  cl <- makePSOCKcluster(detectCores() - 1)
  registerDoParallel(cl)
}

# Train models
for (model_name in models_to_run) {
  cat(paste("Training model:", model_name, "...\n"))

  # Rpart's parameters
  if (model_name == "rpart") {
    tune_grid <- expand.grid(cp = seq(0.01, 0.1, 0.01))
  } else {
    tune_grid <- NULL # using default
  }

  model_fit <- train(
    Diagnosis ~ .,
    data = train_data,
    method = model_name,
    trControl = control,
    metric = "Accuracy",
    tuneGrid = tune_grid
  )
```

```
  results_cv[[model_name]] <- model_fit

  print(model_fit$results[order(model_fit$results$Accuracy, decreasing = TRUE), ][1, ])
}
```

```
## Training model: glm ...
##   parameter  Accuracy    Kappa AccuracySD    KappaSD
## 1      none 0.8792531 0.738078 0.02133139 0.04664393
## Training model: rpart ...
##     cp  Accuracy     Kappa AccuracySD   KappaSD
## 1 0.01 0.8509647 0.6745558 0.02448481 0.0489391
## Training model: rf ...
##   mtry  Accuracy     Kappa AccuracySD    KappaSD
## 2    5 0.9084094 0.8015616 0.02716388 0.05779453
## Training model: gbm ...
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2682             nan     0.1000    0.0266
##      2        1.2253             nan     0.1000    0.0220
##      3        1.1884             nan     0.1000    0.0164
##      4        1.1617             nan     0.1000    0.0135
##      5        1.1385             nan     0.1000    0.0102
##      6        1.1173             nan     0.1000    0.0086
##      7        1.0931             nan     0.1000    0.0114
##      8        1.0763             nan     0.1000    0.0059
##      9        1.0533             nan     0.1000    0.0104
##     10        1.0343             nan     0.1000    0.0093
##     20        0.8896             nan     0.1000    0.0053
##     40        0.7116             nan     0.1000    0.0024
##     60        0.6084             nan     0.1000    0.0015
##     80        0.5477             nan     0.1000    0.0007
##    100        0.5031             nan     0.1000    0.0008
##
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 5       0.1                 2             10     100 0.9492185 0.8901224
##   AccuracySD    KappaSD
## 5 0.01537361 0.03309005
## Training model: svmRadial ...
##        sigma C Accuracy     Kappa AccuracySD    KappaSD
## 3 0.08401214 1 0.870121 0.7180163 0.01173217 0.02381047
## Training model: knn ...
##   k  Accuracy     Kappa AccuracySD    KappaSD
## 2 7 0.8110028 0.5645195 0.01555807 0.03740757
## Training model: nb ...
##   usekernel fL adjust  Accuracy     Kappa AccuracySD    KappaSD
## 1     FALSE  0      1 0.7935201 0.5255632 0.01201641 0.02750503
```

```
if (exists("cl")) {
  stopCluster(cl)
  registerDoSEQ()
}

# 4. Summarize and select best models
results_summary <- data.frame(
    Model = character(),
    CV_Accuracy_Mean = numeric(),
    CV_Accuracy_Std = numeric(),
    stringsAsFactors = FALSE
)

for (name in names(results_cv)) {
    best_tune <- results_cv[[name]]$results[order(results_cv[[name]]$results$Accuracy, decreasin
g = TRUE), ][1, ]
    results_summary <- results_summary %>%
        add_row(
            Model = name,
            CV_Accuracy_Mean = best_tune$Accuracy,
            CV_Accuracy_Std = best_tune$AccuracySD
        )
}

# 3. Print
print("=== Cross-Validation Results ===")
```

```
## [1] "=== Cross-Validation Results ==="
```

```
print(results_summary)
```

```
##       Model CV_Accuracy_Mean CV_Accuracy_Std
## 1       glm        0.8792531      0.02133139
## 2     rpart        0.8509647      0.02448481
## 3        rf        0.9084094      0.02716388
## 4       gbm        0.9492185      0.01537361
## 5 svmRadial        0.8701210      0.01173217
## 6       knn        0.8110028      0.01555807
## 7        nb        0.7935201      0.01201641
```

```
# 4. Model selection
best_model_name <- results_summary$Model[which.max(results_summary$CV_Accuracy_Mean)]
best_model <- results_cv[[best_model_name]]
cat(paste("\nBest model:", best_model_name, "with mean CV Accuracy=",
          max(results_summary$CV_Accuracy_Mean), "\n"))
```

```
##
## Best model: gbm with mean CV Accuracy= 0.949218533886584
```

```r
# 1. Evaluation on testing set
test_results_list <- list()

for (model_name in names(results_cv)) {
  model_fit <- results_cv[[model_name]]

  # Prediction
  y_pred <- predict(model_fit, newdata = test_data)

  # Confusion matrix
  cm <- confusionMatrix(y_pred, test_data$Diagnosis, positive = "Cancer")

  # Metrics
  acc <- cm$overall['Accuracy']
  prec <- cm$byClass['Pos Pred Value'] # Precision
  rec <- cm$byClass['Recall']         # Recall
  f1 <- cm$byClass['F1']

  test_results_list[[model_name]] <- data.frame(
    Model = model_name,
    Test_Accuracy = acc,
    Test_Precision = prec,
    Test_Recall = rec,
    Test_F1 = f1
  )

  if (model_name == best_model_name) {
    cat(paste("\n=== Confusion Matrix -", model_name, "===\n"))
    print(cm$table)

    # Visualize the confusion matrix
    cm_df <- as.data.frame(cm$table)

    ggplot(cm_df, aes(x = Prediction, y = Reference, fill = Freq)) +
      geom_tile(color = "white") +
      scale_fill_gradient(low = "white", high = "#336699") +
      geom_text(aes(label = Freq), vjust = 1, fontface = "bold", size = 5) +
      labs(title = paste("Confusion Matrix -", model_name),
           x = "Predicted Label", y = "True Label") +
      theme_minimal() +
      theme(plot.title = element_text(face = "bold", size = 14), legend.position = "none")
  }
}
```

```
##
## === Confusion Matrix - gbm ===
##           Reference
## Prediction NoCancer Cancer
##   NoCancer      185     11
##   Cancer          3    100
```

```
# 2. Test result summary
test_results_df <- do.call(rbind, test_results_list)

print("\n=== Test Set Results ===")
```

```
## [1] "\n=== Test Set Results ==="
```

```
print(test_results_df)
```

```
##               Model Test_Accuracy Test_Precision Test_Recall   Test_F1
## glm             glm     0.8829431      0.8584906   0.8198198 0.8387097
## rpart         rpart     0.8762542      0.8557692   0.8018018 0.8279070
## rf               rf     0.9230769      0.9150943   0.8738739 0.8940092
## gbm             gbm     0.9531773      0.9708738   0.9009009 0.9345794
## svmRadial svmRadial     0.8829431      0.8800000   0.7927928 0.8341232
## knn             knn     0.8193980      0.9130435   0.5675676 0.7000000
## nb               nb     0.7892977      0.8428571   0.5315315 0.6519337
```

```
library(dplyr)

df %>%
  group_by(Diagnosis) %>%
  summarise(
    Count = n(),
    Percentage = n() / nrow(df) * 100
  ) %>%
  ungroup() %>%
  mutate(
    Percentage_Formatted = paste0(round(Percentage, 2), "%")
  )
```

```
## # A tibble: 2 × 4
##   Diagnosis Count Percentage Percentage_Formatted
##   <fct>     <int>      <dbl> <chr>
## 1 NoCancer    943       62.9 62.87%
## 2 Cancer      557       37.1 37.13%
```

Stacked model

```
set.seed(42)

K <- 5
folds <- createFolds(train_data$Diagnosis, k = K, list = TRUE)

# Store OOF stacking features
oof_rf_prob  <- rep(NA, nrow(train_data))
oof_gbm_prob <- rep(NA, nrow(train_data))

# 5-fold stacking
for (i in 1:K) {
  cat("Processing fold", i, "\n")

  idx_valid <- folds[[i]]
  idx_train <- setdiff(seq_len(nrow(train_data)), idx_valid)

  train_fold <- train_data[idx_train, ]
  valid_fold <- train_data[idx_valid, ]

  # RF
  rf_model <- train(
    Diagnosis ~ ., data = train_fold, method = "rf",
    trControl = trainControl(method = "cv", number = 3),
    tuneLength = 5
  )

  # GBM
  gbm_model <- train(
    Diagnosis ~ ., data = train_fold, method = "gbm",
    trControl = trainControl(method = "cv", number = 3),
    verbose = FALSE
  )

  # OOF predictions
  oof_rf_prob[idx_valid]  <- predict(rf_model,  valid_fold, type = "prob")[, "Cancer"]
  oof_gbm_prob[idx_valid] <- predict(gbm_model, valid_fold, type = "prob")[, "Cancer"]
}
```

```
## Processing fold 1
## Processing fold 2
## Processing fold 3
## Processing fold 4
## Processing fold 5
```

```
# Generate stacked features without data leakage
stacked_train <- data.frame(
  rf_prob  = oof_rf_prob,
  gbm_prob = oof_gbm_prob,
  y = as.numeric(train_data$Diagnosis == "Cancer")
)
```

Check 2 base models' contributions

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.4.3
```

```
## Loading required package: Matrix
```

```
## Warning: package 'Matrix' was built under R version 4.4.3
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-10
```

```
X_stack <- as.matrix(stacked_train[, c("rf_prob", "gbm_prob")])
y_stack <- stacked_train$y

stack_lr <- cv.glmnet(
  X_stack, y_stack,
  family = "binomial",
  alpha = 1,       # LASSO: prevent over fitting
  nfolds = 5
)

cat("Selected lambda:", stack_lr$lambda.min, "\n")
```

```
## Selected lambda: 0.003861134
```

```
coef(stack_lr, s = "lambda.min")
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##             lambda.min
## (Intercept) -5.1876271
## rf_prob      0.6420093
## gbm_prob    10.9185016
```

Train stacked model

```
rf_full <- train(
  Diagnosis ~ ., data = train_data, method = "rf",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(mtry = floor(sqrt(ncol(train_data) - 1))),
  ntree = 500
)

gbm_full <- train(
  Diagnosis ~ ., data = train_data, method = "gbm",
  trControl = trainControl(method = "none"),
  verbose = FALSE,
  tuneGrid = data.frame(
    n.trees = 200,
    interaction.depth = 3,
    shrinkage = 0.05,
    n.minobsinnode = 10
  )
)

rf_test_prob  <- predict(rf_full, newdata = test_data, type = "prob")[, "Cancer"]
gbm_test_prob <- predict(gbm_full, newdata = test_data, type = "prob")[, "Cancer"]

stacked_test <- as.matrix(data.frame(
  rf_prob  = rf_test_prob,
  gbm_prob = gbm_test_prob
))

final_probs <- predict(stack_lr, newx = stacked_test, s = "lambda.min", type = "response")
final_pred  <- ifelse(final_probs > 0.5, "Cancer", "NoCancer")
final_pred  <- factor(final_pred, levels = c("NoCancer","Cancer"))
```

Stacked model result

```
# Confusion Matrix
confusionMatrix(final_pred, test_data$Diagnosis)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction NoCancer Cancer
##   NoCancer      181     10
##   Cancer          7    101
##
##                Accuracy : 0.9431
##                  95% CI : (0.9105, 0.9665)
##     No Information Rate : 0.6288
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8775
##
##  Mcnemar's Test P-Value : 0.6276
##
##             Sensitivity : 0.9628
##             Specificity : 0.9099
##          Pos Pred Value : 0.9476
##          Neg Pred Value : 0.9352
##              Prevalence : 0.6288
##          Detection Rate : 0.6054
##    Detection Prevalence : 0.6388
##       Balanced Accuracy : 0.9363
##
##        'Positive' Class : NoCancer
##
```

```
train_pred <- predict(stack_lr, newx = X_stack, s = "lambda.min", type = "response")
# Convert matrix into vector
train_pred_vector <- as.numeric(train_pred[, 1])  # 提取第一列并转为 numeric 向量
final_probs_vector <- as.numeric(final_probs[, 1])

# ROC curve
roc_curve <- roc(test_data$Diagnosis, final_probs_vector)
```

```
## Setting levels: control = NoCancer, case = Cancer
```

```
## Setting direction: controls < cases
```

```
auc(roc_curve)
```

```
## Area under the curve: 0.9492
```

```
library(MLmetrics)
```

```
## Warning: package 'MLmetrics' was built under R version 4.4.3
```

```
##
## Attaching package: 'MLmetrics'
```

```
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```
## The following object is masked from 'package:base':
##
##     Recall
```

```
# Convert factor into numeric (1 = Cancer, 0 = NoCancer)
y_true <- ifelse(test_data$Diagnosis == "Cancer", 1, 0)
y_pred <- ifelse(final_pred == "Cancer", 1, 0)

Precision <- Precision(y_pred, y_true)
Recall    <- Recall(y_pred, y_true)
F1        <- F1_Score(y_pred, y_true)

cat("Precision:", Precision, "\n")
```

```
## Precision: 0.962766
```

```
cat("Recall:", Recall, "\n")
```

```
## Recall: 0.947644
```

```
cat("F1 Score:", F1, "\n")
```

```
## F1 Score: 0.9551451
```

Evaluation of stacked model

(i) Over-fitting

```
# (i) Evaluate over fitting
train_pred <- predict(stack_lr, newx = X_stack, s = "lambda.min", type = "response")

auc_train <- auc(roc(train_data$Diagnosis, train_pred_vector))
```

```
## Setting levels: control = NoCancer, case = Cancer
```

```
## Setting direction: controls < cases
```

```
auc_test  <- auc(roc(test_data$Diagnosis, final_probs_vector))
```

```
## Setting levels: control = NoCancer, case = Cancer
## Setting direction: controls < cases
```

```
cat("Train AUC:", auc_train, "\n")
```

```
## Train AUC: 0.961824
```

```
cat("Test  AUC:", auc_test, "\n")
```

```
## Test  AUC: 0.9492045
```

```
cat("Gap =", auc_train - auc_test, "\n")
```

```
## Gap = 0.01261949
```

Visualize the ROC curve

```
# 1. Get baseline logistic regression prediction probabilities
glm_model <- results_cv[["glm"]]
glm_test_probs <- predict(glm_model, newdata = test_data, type = "prob")[, "Cancer"]

# 2. Generate the baseline logistic regression's ROC curve
roc_glm <- roc(test_data$Diagnosis, glm_test_probs)
```

```
## Setting levels: control = NoCancer, case = Cancer
```

```
## Setting direction: controls < cases
```

```
# Plot
# Plot Stacked Train ROC
plot(roc(train_data$Diagnosis, train_pred_vector),
     col = "blue",
     main = "Stacked vs. Logistic Regression ROC Comparison",
     xlab = "False Positive Rate (1 - Specificity)",
     ylab = "True Positive Rate (Sensitivity)")
```

```
## Setting levels: control = NoCancer, case = Cancer
## Setting direction: controls < cases
```

```
# Add Stacked Test ROC
plot(roc(test_data$Diagnosis, final_probs_vector),
     col = "red",
     add = TRUE)
```

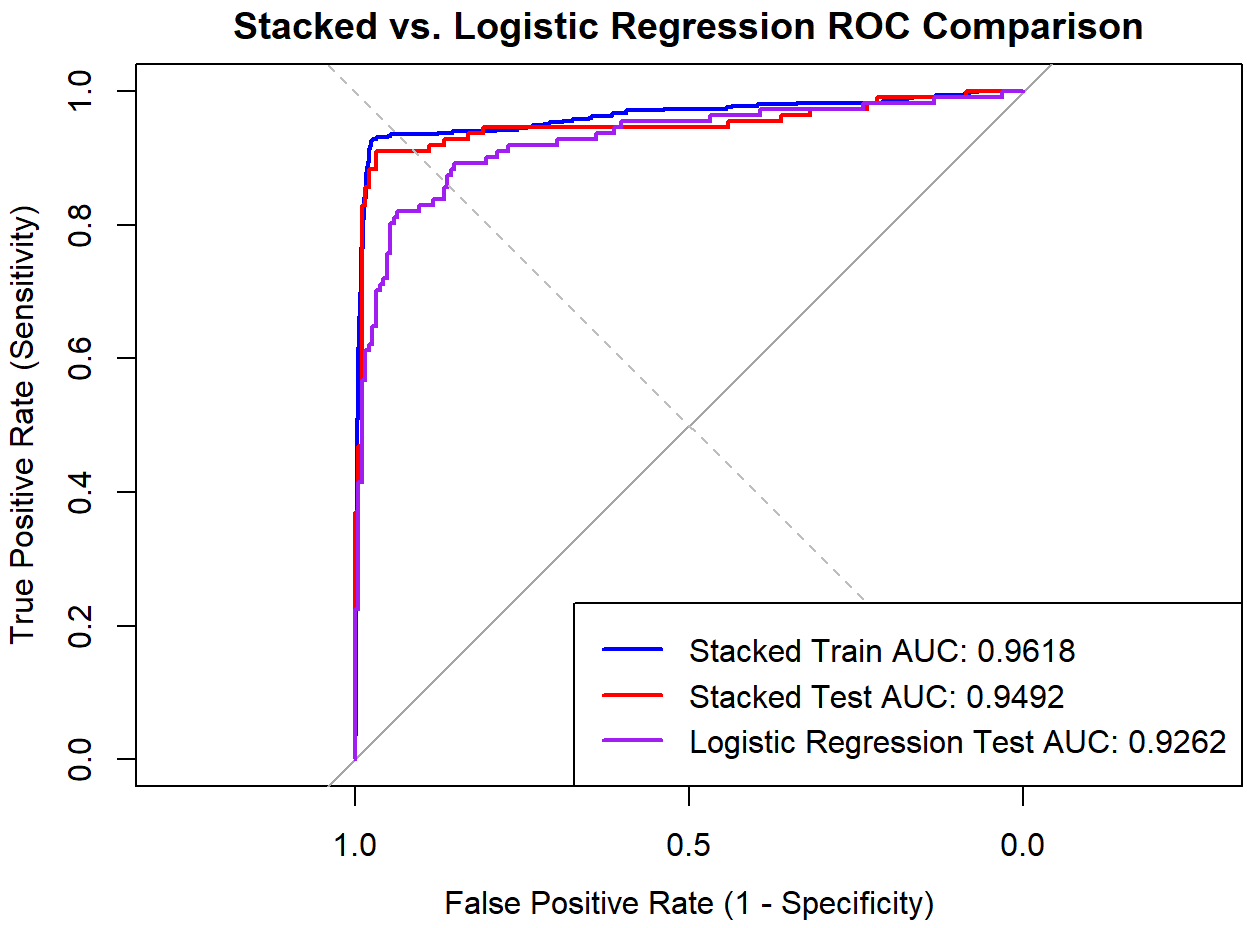
```
## Setting levels: control = NoCancer, case = Cancer
## Setting direction: controls < cases
```

```
# Add GLM Test ROC
plot(roc_glm,
     col = "purple",
     add = TRUE)

# 4. Add random prediction line
abline(a = 0, b = 1, lty = 2, col = "gray")

legend("bottomright",
       legend = c(
         paste("Stacked Train AUC:", round(auc(roc(train_data$Diagnosis, train_pred_vector)),
4)),
         paste("Stacked Test AUC:", round(auc(roc(test_data$Diagnosis, final_probs_vector)),
4)),
         paste("Logistic Regression Test AUC:", round(auc(roc_glm), 4))
       ),
       col = c("blue", "red", "purple"),
       lwd = 2)
```

```
## Setting levels: control = NoCancer, case = Cancer
## Setting direction: controls < cases
```

```
## Setting levels: control = NoCancer, case = Cancer
```

```
## Setting direction: controls < cases
```




```
cat("\nLogistic Regression Test AUC:", auc(roc_glm), "\n")
```

```
##
## Logistic Regression Test AUC: 0.9261549
```

(ii) Stratified split

```
# (ii) Evaluate the stratified split
prop.table(table(train_data$Diagnosis))
```

```
##
##  NoCancer    Cancer
## 0.6286428 0.3713572
```

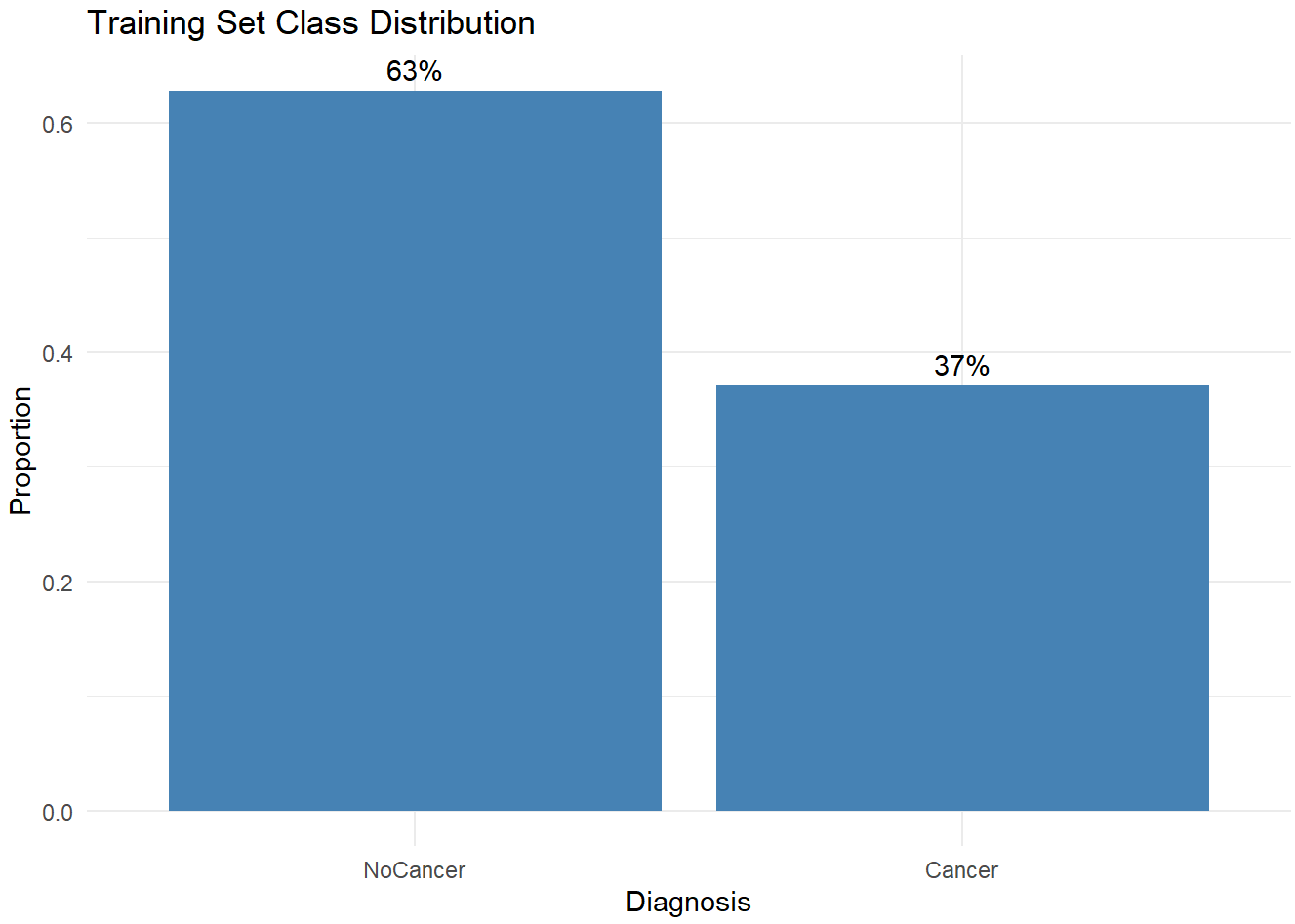
```
prop.table(table(test_data$Diagnosis))
```

```
##
##  NoCancer    Cancer
## 0.6287625 0.3712375
```

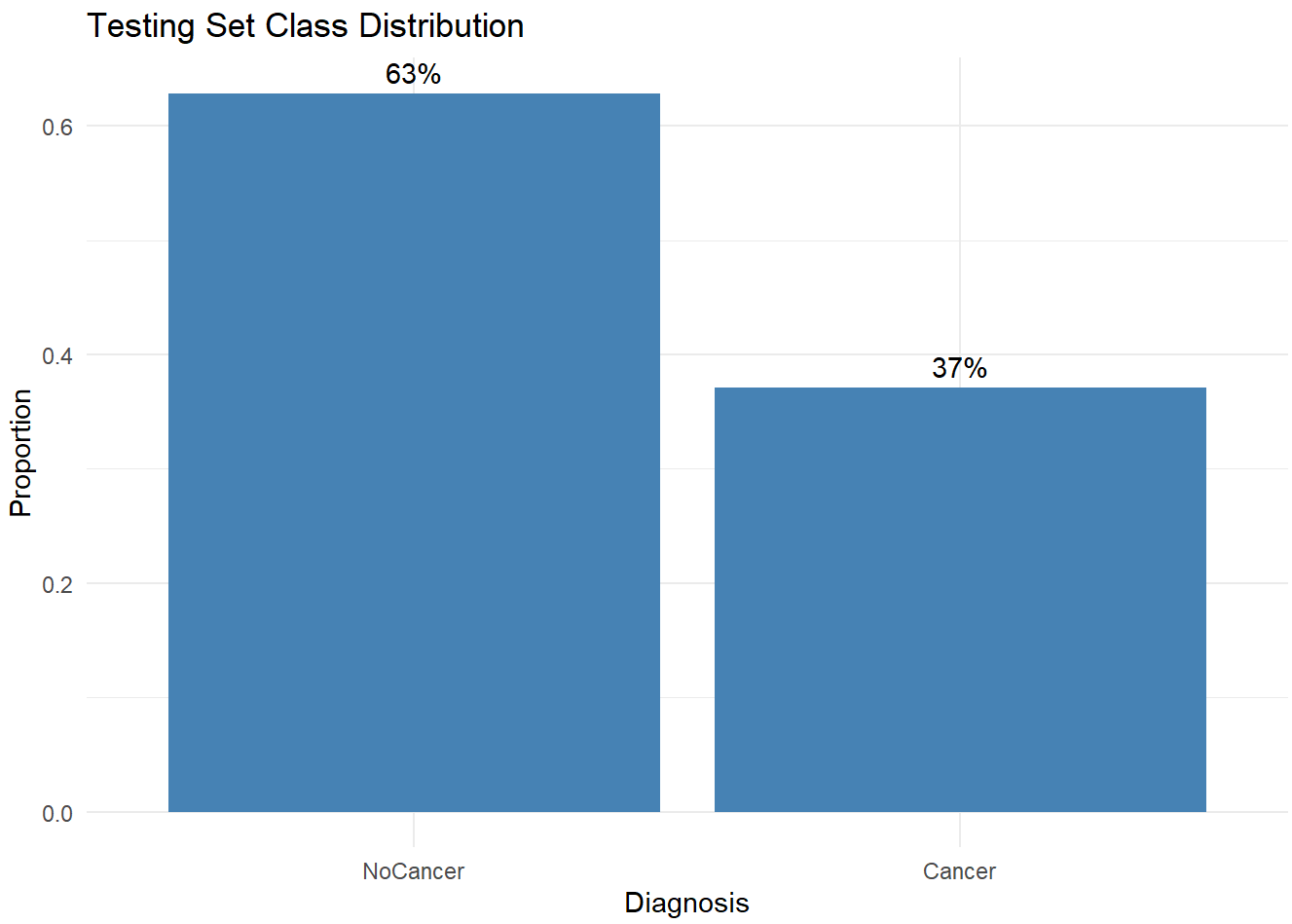
Visualize the splitting size

```
library(ggplot2)

ggplot(train_data, aes(x = Diagnosis)) +
  geom_bar(aes(y = (..count..)/sum(..count..)), fill = "steelblue") +
  geom_text(aes(y = (..count..)/sum(..count..), label = scales::percent((..count..)/sum(..coun
t..))),
            stat = "count", vjust = -0.5) +
  ylab("Proportion") +
  ggtitle("Training Set Class Distribution") +
  theme_minimal()
```

```
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
## ℹ Please use `after_stat(count)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

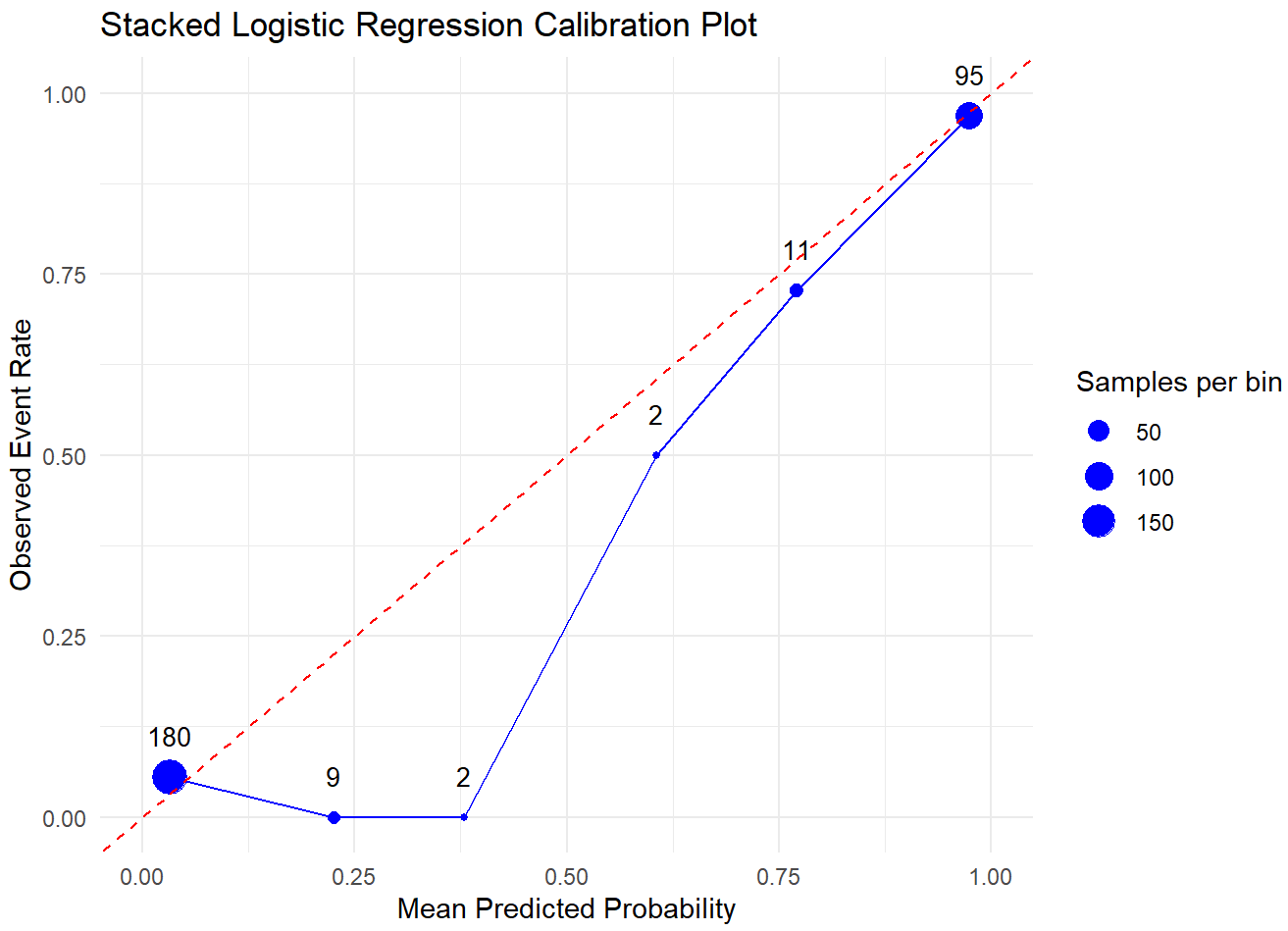


```
ggplot(test_data, aes(x = Diagnosis)) +
  geom_bar(aes(y = (..count..)/sum(..count..)), fill = "steelblue") +
  geom_text(aes(y = (..count..)/sum(..count..), label = scales::percent((..count..)/sum(..coun
t..))),
            stat = "count", vjust = -0.5) +
  ylab("Proportion") +
  ggtitle("Testing Set Class Distribution") +
  theme_minimal()
```



(iii) Calibration plot

```
# 3. (iii) Calibration Plot
# Create data frame
df_calib <- data.frame(
  prob = final_probs_vector,
  true = y_true
)

# 2. Split the prediction probabilities into different classes
n_bins <- 6
df_calib$bin <- cut(df_calib$prob, breaks = seq(0,1,length.out = n_bins+1), include.lowest = TRU
E)

calib_summary <- df_calib %>%
  group_by(bin) %>%
  summarise(
    mean_pred = mean(prob),
    obs_rate  = mean(true),
    n         = n()
  )

# 3. Plot the calibration plot
ggplot(calib_summary, aes(x = mean_pred, y = obs_rate)) +
  geom_point(aes(size = n), color = "blue") +
  geom_line(color = "blue") +
  geom_text(aes(label = n), vjust = -1.5, hjust = 0.5, size = 3.5, color = "black") +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "red") +
  xlim(0, 1) + ylim(0, 1) +
  xlab("Mean Predicted Probability") +
  ylab("Observed Event Rate") +
  ggtitle("Stacked Logistic Regression Calibration Plot") +
  theme_minimal() +
  scale_size_continuous(name = "Samples per bin")
```




```
# 4. Calculate Brier Score
brier_score <- mean((final_probs_vector - y_true)^2)
cat("Brier Score:", round(brier_score, 4), "\n")
```

```
## Brier Score: 0.0525
```

(iv) Feature importance and stacked model's interpretability

```
# (iv) Evaluate feature importance / Explainability
# rf feature importance score
rf_imp <- varImp(rf_full)$importance
rf_imp %>% arrange(desc(Overall))
```

```
##                     Overall
## CancerHistoryYes  100.00000
## BMI                83.93317
## AlcoholIntake      80.48403
## GeneticRiskHigh    79.50337
## PhysicalActivity   75.92005
## Age                74.19066
## GenderFemale       42.45197
## SmokingYes         26.34592
## GeneticRiskMedium   0.00000
```

```
# GBM feature importance score
gbm_imp <- summary(gbm_full$finalModel, n.trees = gbm_full$bestTune$n.trees, plotit = FALSE)
print(gbm_imp)
```

```
##                                 var     rel.inf
## CancerHistoryYes   CancerHistoryYes 20.90859506
## GeneticRiskHigh     GeneticRiskHigh 17.85936796
## AlcoholIntake         AlcoholIntake 11.10167064
## GenderFemale           GenderFemale 10.90965907
## BMI                             BMI 10.68898422
## PhysicalActivity   PhysicalActivity 10.60143407
## Age                             Age 10.51628965
## SmokingYes               SmokingYes  7.33820859
## GeneticRiskMedium GeneticRiskMedium  0.07579074
```

```
coef(stack_lr, s = "lambda.min")  # base models' contributions
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##             lambda.min
## (Intercept) -5.1876271
## rf_prob      0.6420093
## gbm_prob    10.9185016
```

```r
library(dplyr)
library(tidyr)

rf_data <- data.frame(
  Feature = c("CancerHistoryYes", "BMI", "AlcoholIntake", "GeneticRiskHigh", "PhysicalActivity",
              "Age", "GenderFemale", "SmokingYes", "GeneticRiskMedium"),
  RF_Importance = c(100.00, 83.93, 80.48, 79.50, 75.92, 74.19, 42.45, 26.35, 0.00)
)

gbm_data <- data.frame(
  Feature = c("CancerHistoryYes", "GeneticRiskHigh", "AlcoholIntake", "GenderFemale", "BMI",
              "PhysicalActivity", "Age", "SmokingYes", "GeneticRiskMedium"),
  GBM_Importance = c(20.91, 17.86, 11.10, 10.91, 10.69, 10.60, 10.52, 7.34, 0.08)
)

# Combine feature importance from two models into one table
combined_imp <- full_join(rf_data, gbm_data, by = "Feature") %>%
  mutate(
    Feature_Clean = case_when(
      grepl("CancerHistory", Feature) ~ "Cancer History (Yes)",
      grepl("GeneticRiskHigh", Feature) ~ "Genetic Risk (High)",
      grepl("GenderFemale", Feature) ~ "Gender (Female)",
      grepl("SmokingYes", Feature) ~ "Smoking (Yes)",
      grepl("GeneticRiskMedium", Feature) ~ "Genetic Risk (Medium)",
      TRUE ~ Feature # 保留其他变量原名
    )
  ) %>%
  select(Feature_Clean, RF_Importance, GBM_Importance) %>%
  arrange(desc(RF_Importance))

print(combined_imp)
```

```
##           Feature_Clean RF_Importance GBM_Importance
## 1  Cancer History (Yes)        100.00          20.91
## 2                   BMI         83.93          10.69
## 3         AlcoholIntake         80.48          11.10
## 4   Genetic Risk (High)         79.50          17.86
## 5      PhysicalActivity         75.92          10.60
## 6                   Age         74.19          10.52
## 7       Gender (Female)         42.45          10.91
## 8         Smoking (Yes)         26.35           7.34
## 9 Genetic Risk (Medium)          0.00           0.08
```

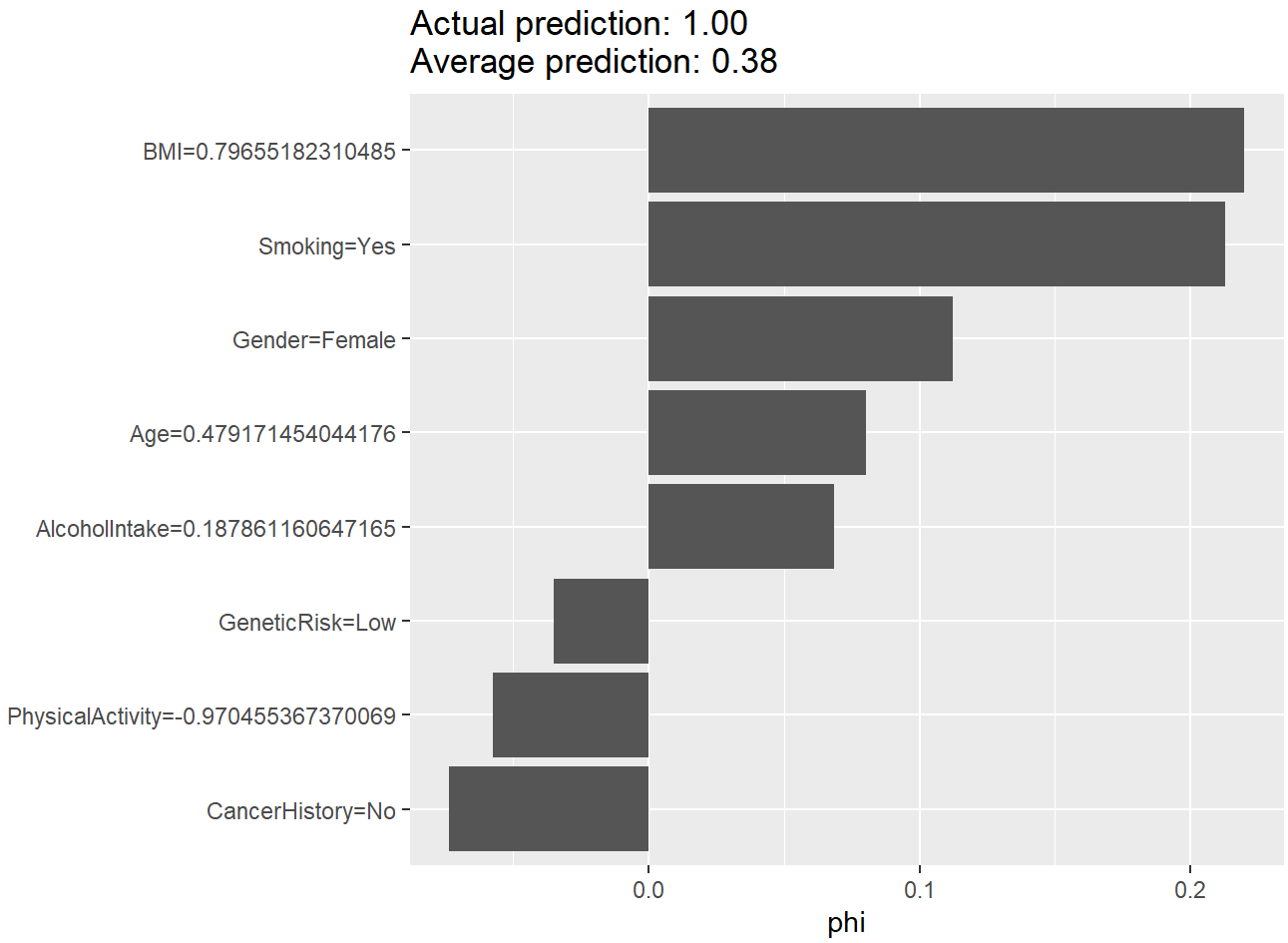
Stacked model interpretability

```
# Define a function to calculate stacked model's final prediction probability
stacked_predict_prob <- function(object, newdata) {
  # 1. Base Learner Predictions
  rf_test_prob <- predict(rf_full, newdata = newdata, type = "prob")[, "Cancer"]
  gbm_test_prob <- predict(gbm_full, newdata = newdata, type = "prob")[, "Cancer"]
# Sanity Check 1: Check base learner's prediction length
  if (length(rf_test_prob) != nrow(newdata) || length(gbm_test_prob) != nrow(newdata)) {
  warning("Sanity Check Failed: Base Learner prediction length does not match input data rows. C
heck for
  NA/missing rows due to new factor levels.")
}
  # 2. Create stacked feature matrix (Level 1)
  if (any(is.na(rf_test_prob)) || any(is.na(gbm_test_prob))) {
  warning("Sanity Check Failed: Base Learner probabilities contain NA values.")
}
  stacked_test <- as.matrix(data.frame(
    rf_prob  = rf_test_prob,
    gbm_prob = gbm_test_prob
  ))

  # 3. Final prediction using Meta learner
  final_probs <- predict(stack_lr, newx = stacked_test, s = "lambda.min", type = "response")
  # Sanity Check 2: Check final prediction probabilities
  if (any(final_probs < 0) || any(final_probs > 1) || any(is.na(final_probs))) {
  warning("Sanity Check Failed: Final probabilities contain values outside [0, 1] or contain N
A.")
  # stop("Invalid probability detected.")
}
  # Return prediction probability vector
  return(as.numeric(final_probs))
}
```

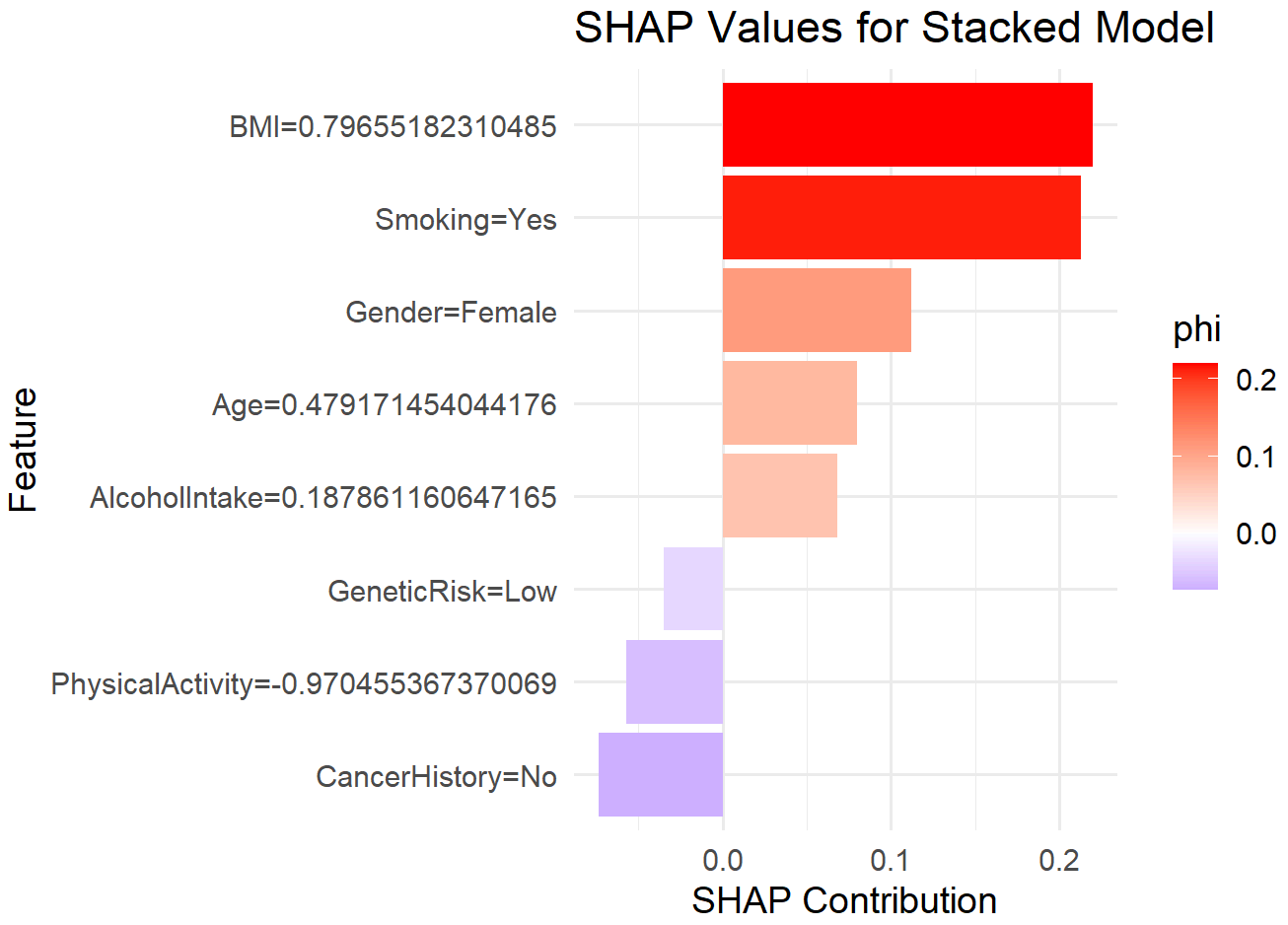
```
library(iml)
```

```
## Warning: package 'iml' was built under R version 4.4.3
```

```r
library(ggplot2)
library(dplyr)

predictor_stacked <- Predictor$new(
  # model = rf_full, # take this space
  data = train_data[,-which(names(train_data)=="Diagnosis")],
  y = train_data$Diagnosis,
  predict.function = stacked_predict_prob,
  class = "Cancer"
)

# --- 2. Select sample we want to interpret (testing set's 50th sample) ---
x_interest_test_sample <- test_data[50, -which(names(test_data)=="Diagnosis")]

# --- 3. Calculate stacked model's SHAP value ---
shapley_stacked <- Shapley$new(
  predictor = predictor_stacked,
  x.interest = x_interest_test_sample
)

# --- 4. Plot ---
shapley_stacked$plot()
```



```
shap_df <- shapley_stacked$results %>%
  arrange(abs(phi))

ggplot(shap_df, aes(x = reorder(feature.value, phi), y = phi, fill = phi)) +
  geom_col() +
  coord_flip() +
  scale_fill_gradient2(
    low = "blue",
    mid = "white",
    high = "red",
    midpoint = 0
  ) +
  labs(
    title = "SHAP Values for Stacked Model",
    x = "Feature",
    y = "SHAP Contribution"
  ) +
  theme_minimal(base_size = 14)
```



```
# Sanity check for stacked_predict_prob
# 1. Create a testing df
test_data_sample <- train_data[1:10, -which(names(train_data) == "Diagnosis")]

# 2. Run stacked_predict_prob
test_probs <- stacked_predict_prob(object = rf_full, newdata = test_data_sample)

# 3. Check max and min
max_prob <- max(test_probs, na.rm = TRUE)
min_prob <- min(test_probs, na.rm = TRUE)

print(paste("Maximum calculated probability:", max_prob))
```

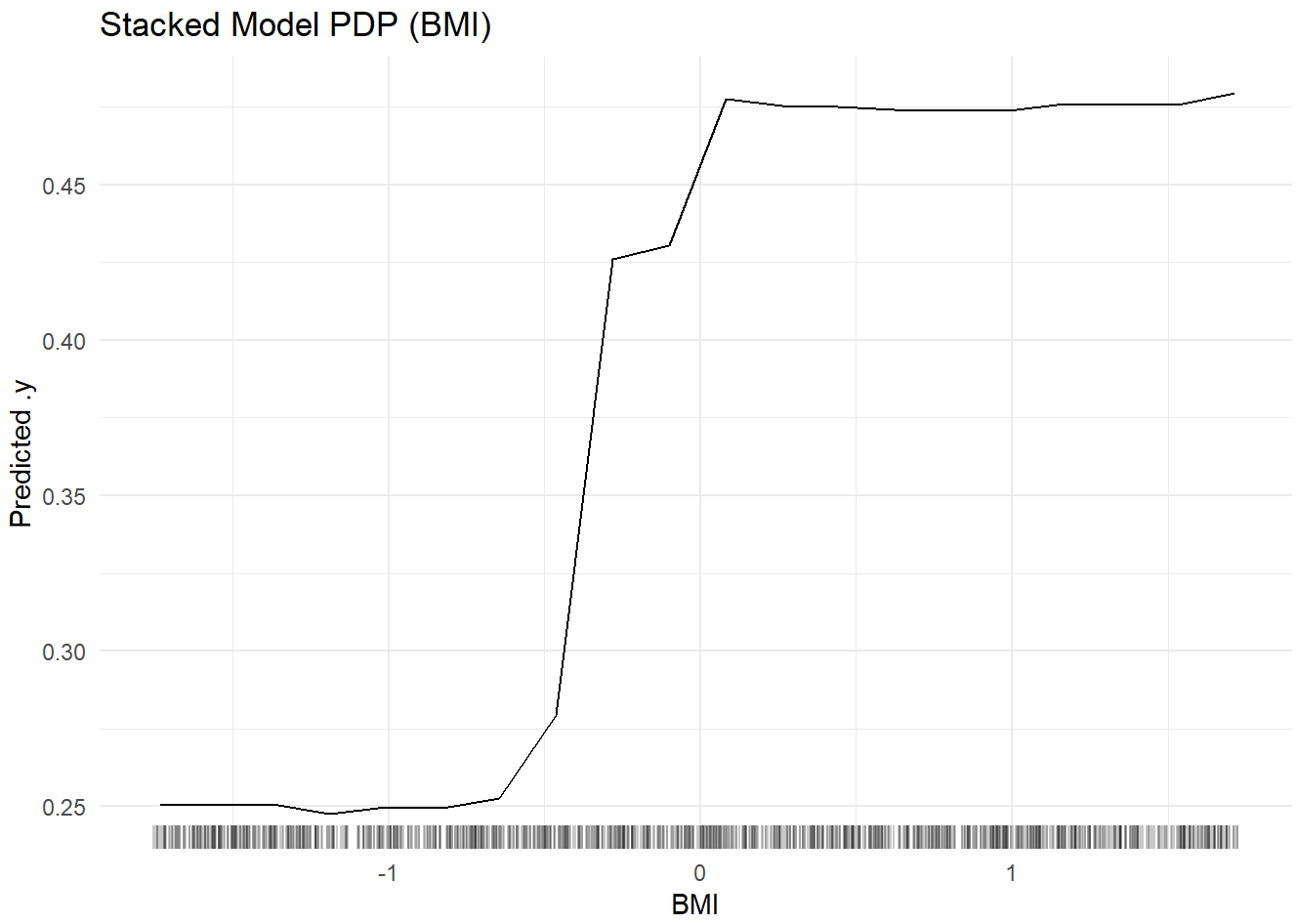
```
## [1] "Maximum calculated probability: 0.997020085345288"
```

```
print(paste("Minimum calculated probability:", min_prob))
```

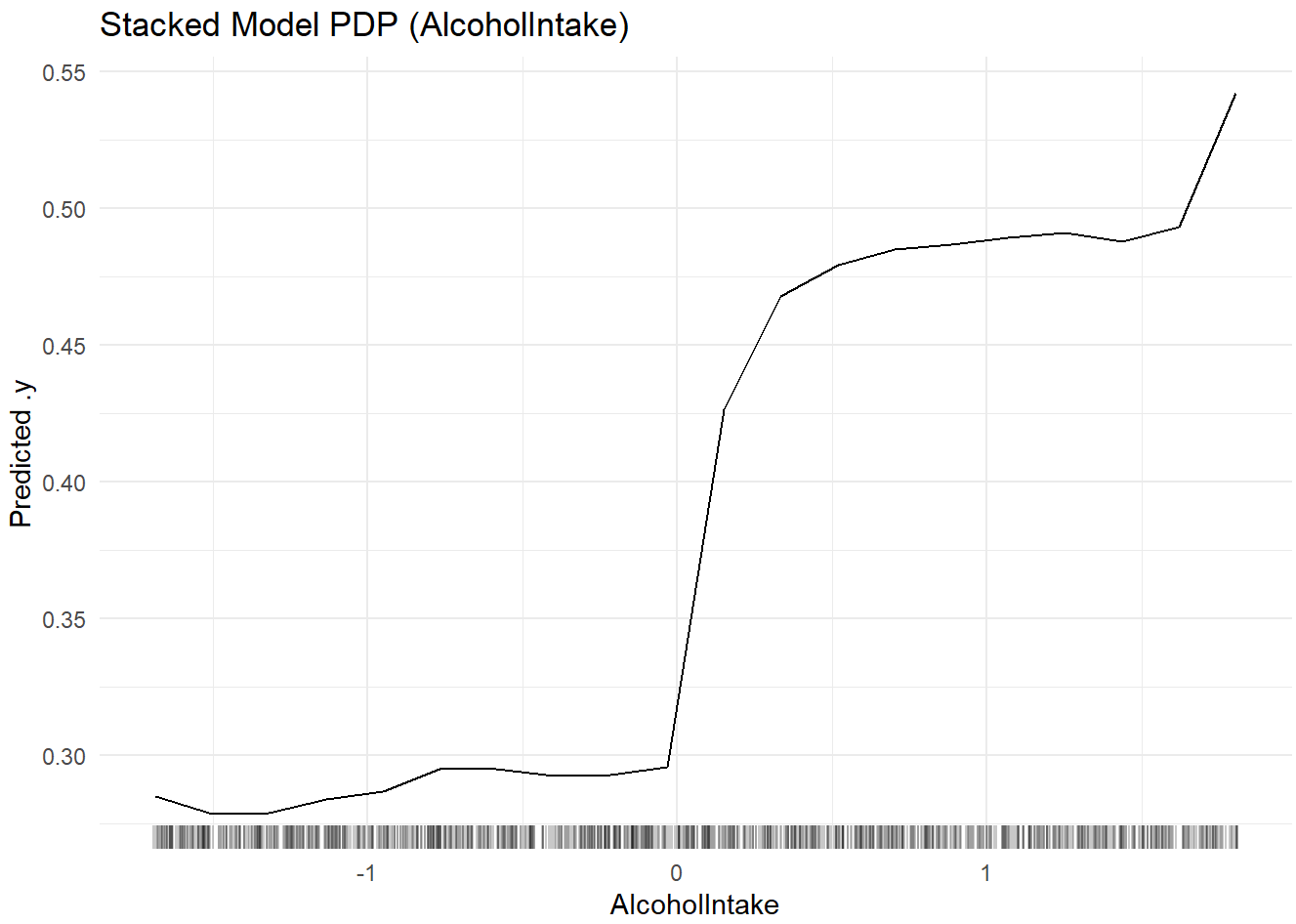
```
## [1] "Minimum calculated probability: 0.00626285133180356"
```

Visualize examples of covariate's non-linear trend

```
library(iml)
library(ggplot2)
# Validated that our stacked logistic regression has learned non-linear relationship
# BMI plot
effect_bmi <- FeatureEffect$new(
    predictor = predictor_stacked,
    feature = "BMI",
    method = "pdp"
)

plot_pdp_bmi <- effect_bmi$plot() +
    labs(title = "Stacked Model PDP (BMI)",
         y = "Predicted Cancer Probability (0-1)") +
    theme_minimal()

print(plot_pdp_bmi)
```




```
# AlcholIntake plot
effect_alch <- FeatureEffect$new(
    predictor = predictor_stacked,
    feature = "AlcoholIntake",
    method = "pdp"
)

plot_pdp_alch <- effect_alch$plot() +
    labs(title = "Stacked Model PDP (AlcoholIntake)",
         y = "Predicted Cancer Probability (0-1)") +
    theme_minimal()

print(plot_pdp_alch)
```


Stacked Model PDP (AlcoholIntake)
0.55
0.50
0.45
0.40
0.35
0.30
Predicted .y
-1
0
1
AlcoholIntake

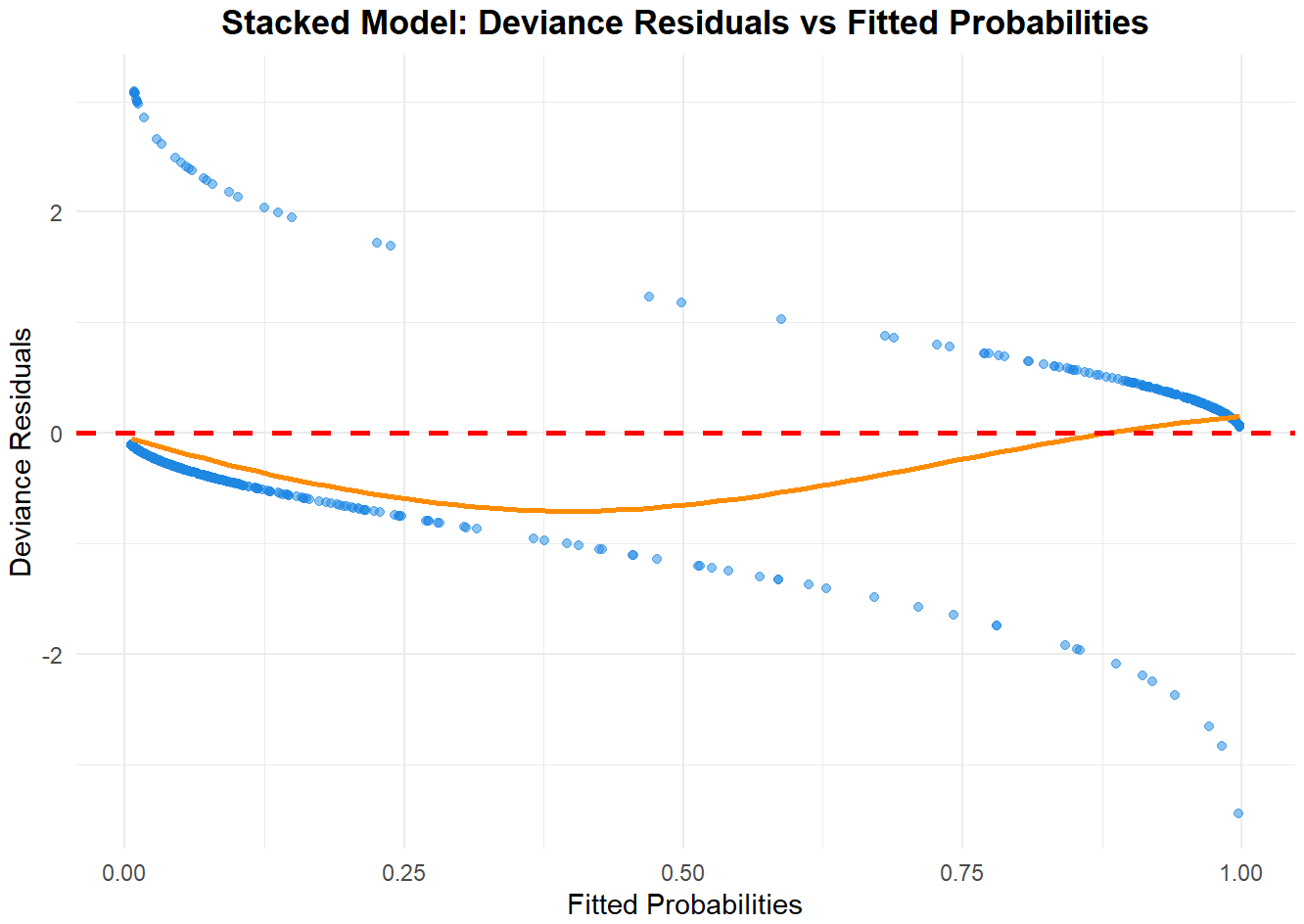
(v) Deviance and QQ-plot as diagnosis

```
# (v) Deviance and QQ-plot as diagnosis
X_train_stack <- as.matrix(data.frame(
  rf_prob  = predict(rf_full, newdata = train_data, type = "prob")[, "Cancer"],
  gbm_prob = predict(gbm_full, newdata = train_data, type = "prob")[, "Cancer"]
))
y_train <- ifelse(train_data$Diagnosis == "Cancer", 1, 0)

train_pred_prob <- as.vector(predict(stack_lr, newx = X_train_stack, s = "lambda.min", type = "r
esponse"))

# Deviance residuals
dev_resid <- ifelse(y_train == 1,
                    sqrt(-2 * log(train_pred_prob)),
                    -sqrt(-2 * log(1 - train_pred_prob)))

# Standardized residuals
std_resid <- scale(dev_resid)

# Residuals vs Fitted plot
residual_data <- data.frame(Fitted = train_pred_prob, DevianceResid = dev_resid)

plot_resid_fitted <- ggplot(residual_data, aes(x = Fitted, y = DevianceResid)) +
  geom_point(alpha = 0.5, color = "#1E88E5") +
  geom_smooth(method = "loess", color = "darkorange", se = FALSE, linetype = "solid") +
  geom_hline(yintercept = 0, color = "red", linetype = "dashed", size = 1) +
  labs(title = "Stacked Model: Deviance Residuals vs Fitted Probabilities",
       x = "Fitted Probabilities",
       y = "Deviance Residuals") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

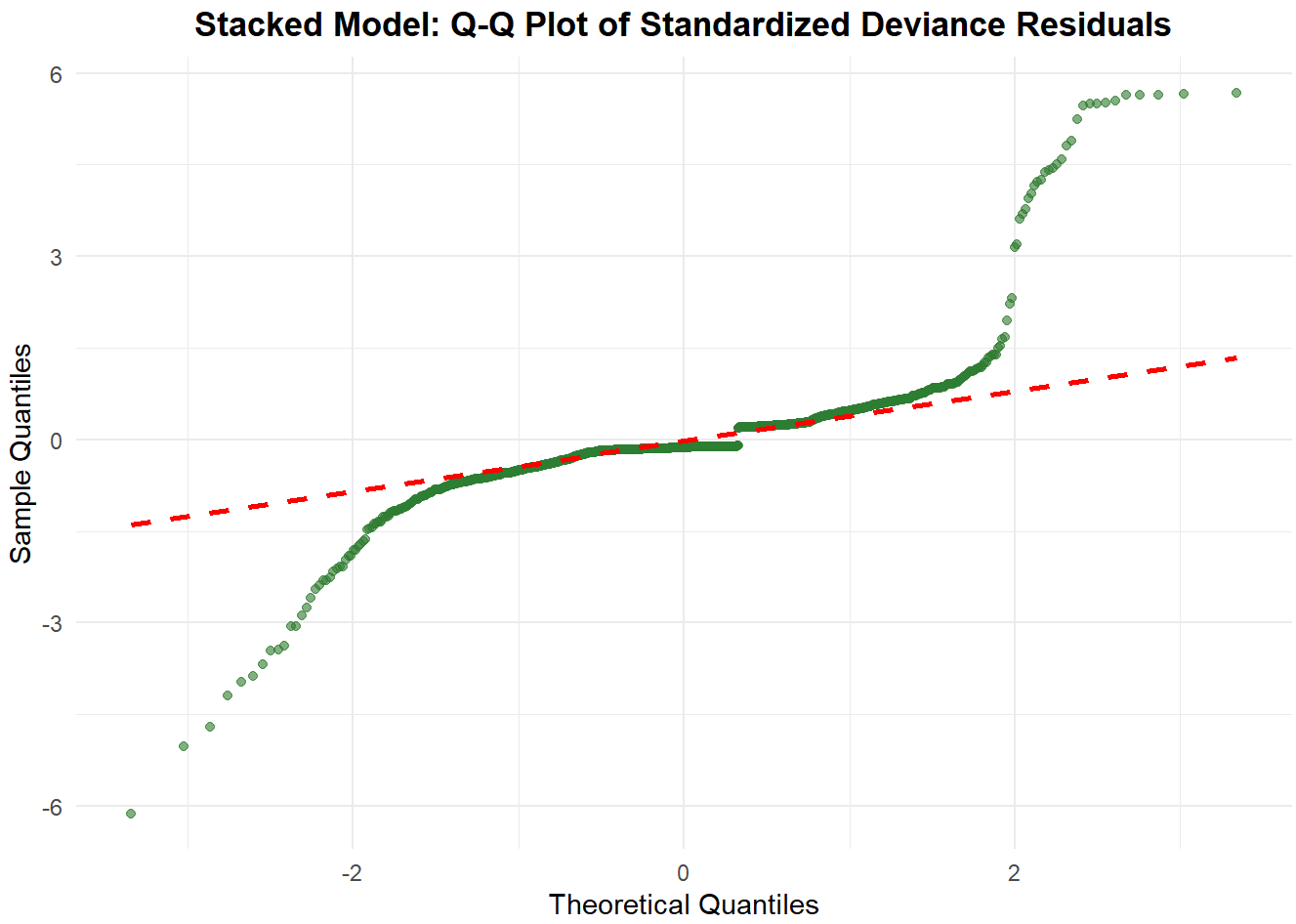
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
print(plot_resid_fitted)
```

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




```
# Q-Q plot
qq_data <- data.frame(StdResid = std_resid)

plot_qq <- ggplot(qq_data, aes(sample = StdResid)) +
  stat_qq(alpha = 0.6, color = "#2E7D32") + # 绿色散点
  stat_qq_line(color = "red", linetype = "dashed", size = 1) +
  labs(title = "Stacked Model: Q-Q Plot of Standardized Deviance Residuals",
       x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

print(plot_qq)
```



Fine tuning

```
rf_tune <- train(
  Diagnosis ~ ., data = train_data, method = "rf",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE), # 使用 5 折交叉验证
  tuneGrid = data.frame(mtry = c(2, 4, 6, 8)), # 尝试不同的 mtry 值
  ntree = 500
)
```

```
## + Fold1: mtry=2
## - Fold1: mtry=2
## + Fold1: mtry=4
## - Fold1: mtry=4
## + Fold1: mtry=6
## - Fold1: mtry=6
## + Fold1: mtry=8
## - Fold1: mtry=8
## + Fold2: mtry=2
## - Fold2: mtry=2
## + Fold2: mtry=4
## - Fold2: mtry=4
## + Fold2: mtry=6
## - Fold2: mtry=6
## + Fold2: mtry=8
## - Fold2: mtry=8
## + Fold3: mtry=2
## - Fold3: mtry=2
## + Fold3: mtry=4
## - Fold3: mtry=4
## + Fold3: mtry=6
## - Fold3: mtry=6
## + Fold3: mtry=8
## - Fold3: mtry=8
## + Fold4: mtry=2
## - Fold4: mtry=2
## + Fold4: mtry=4
## - Fold4: mtry=4
## + Fold4: mtry=6
## - Fold4: mtry=6
## + Fold4: mtry=8
## - Fold4: mtry=8
## + Fold5: mtry=2
## - Fold5: mtry=2
## + Fold5: mtry=4
## - Fold5: mtry=4
## + Fold5: mtry=6
## - Fold5: mtry=6
## + Fold5: mtry=8
## - Fold5: mtry=8
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 4 on full training set
```

```
gbm_tune <- train(
  Diagnosis ~ ., data = train_data, method = "gbm",
  trControl = trainControl(method = "cv", number = 5, verboseIter = TRUE),
  verbose = FALSE,
  tuneGrid = expand.grid( # 尝试多种参数组合
    n.trees = c(300, 500),
    interaction.depth = c(3, 5),
    shrinkage = c(0.1, 0.05),
    n.minobsinnode = 10
  )
)
```

```
## + Fold1: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold1: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold1: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold1: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold1: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold1: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold1: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold1: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold2: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold2: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold2: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold2: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold2: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold2: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold2: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold2: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold3: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold3: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold3: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold3: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold3: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold3: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold3: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold3: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold4: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold4: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold4: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold4: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold4: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold4: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold4: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold4: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold5: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold5: shrinkage=0.05, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold5: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold5: shrinkage=0.05, interaction.depth=5, n.minobsinnode=10, n.trees=500
## + Fold5: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## - Fold5: shrinkage=0.10, interaction.depth=3, n.minobsinnode=10, n.trees=500
## + Fold5: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## - Fold5: shrinkage=0.10, interaction.depth=5, n.minobsinnode=10, n.trees=500
## Aggregating results
## Selecting tuning parameters
## Fitting n.trees = 300, interaction.depth = 3, shrinkage = 0.05, n.minobsinnode = 10 on full t
raining set
```

```
# Find best hyperparameters
best_mtry <- rf_tune$bestTune$mtry
best_n.trees <- gbm_tune$bestTune$n.trees
```

```r
# --- 1. Set best hyperparameters ---
best_mtry <- 4
best_n.trees <- 300
best_depth <- 3
best_shrinkage <- 0.05

# --- 2. Train rf base learner (tuned) ---
rf_full_tuned <- train(
  Diagnosis ~ ., data = train_data, method = "rf",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(mtry = best_mtry),
  ntree = 500
)

# --- 3. Train gbm base learner (tuned) ---
gbm_full_tuned <- train(
  Diagnosis ~ ., data = train_data, method = "gbm",
  trControl = trainControl(method = "none"),
  verbose = FALSE,
  tuneGrid = data.frame(
    n.trees = best_n.trees,
    interaction.depth = best_depth,
    shrinkage = best_shrinkage,
    n.minobsinnode = 10
  )
)

X_train_tuned <- as.matrix(data.frame(
  rf_prob  = predict(rf_full_tuned, newdata = train_data, type = "prob")[, "Cancer"],
  gbm_prob = predict(gbm_full_tuned, newdata = train_data, type = "prob")[, "Cancer"]
))
y_train <- ifelse(train_data$Diagnosis == "Cancer", 1, 0)

# --- 4. Train stacked model (tuned)
stacked_lr_tuned <- cv.glmnet(
  X_train_tuned, y_train,
  family = "binomial",
  alpha = 1, # LASSO
  nfolds = 5
)
```

```r
# --- 1. Prepare Level 1's feature matrix for testing set ---
X_test_tuned <- as.matrix(data.frame(
  rf_prob  = predict(rf_full_tuned, newdata = test_data, type = "prob")[, "Cancer"],
  gbm_prob = predict(gbm_full_tuned, newdata = test_data, type = "prob")[, "Cancer"]
))
y_test <- ifelse(test_data$Diagnosis == "Cancer", 1, 0)

# --- 2. Get stacked model prediction probability from fine tuned base models ---
train_pred_tuned <- as.vector(predict(stacked_lr_tuned, newx = X_train_tuned, s = "lambda.min",
type = "response"))
final_probs_tuned <- as.vector(predict(stacked_lr_tuned, newx = X_test_tuned, s = "lambda.min",
type = "response"))

# --- 3. Calculate AUC ---
auc_train_tuned <- auc(roc(train_data$Diagnosis, train_pred_tuned, quiet = TRUE))
auc_test_tuned  <- auc(roc(test_data$Diagnosis, final_probs_tuned, quiet = TRUE))

# --- 4. Original AUC ---
auc_train_original <- auc_train
auc_test_original  <- auc_test


# AUC comparison
cat("Comparison of original stacked model vs fined tuned stacked model\n")
```

```
## Comparison of original stacked model vs fined tuned stacked model
```

```r
cat("Training set's AUC improvement: \n")
```

```
## Training set's AUC improvement:
```

```r
cat("Original stacked model's AUC:", round(auc_train_original, 4), "\n")
```

```
## Original stacked model's AUC: 0.9618
```

```r
cat("Fine tuned stacked modal's AUC:", round(auc_train_tuned, 4), "\n")
```

```
## Fine tuned stacked modal's AUC: 1
```

```r
cat("Change in AUC:", round(auc_train_tuned - auc_train_original, 4), "\n\n")
```

```
## Change in AUC: 0.0382
```

```r
cat("Testing set's AUC improvement: \n")
```

```
## Testing set's AUC improvement:
```

```
cat("Original stacked model's AUC:", round(auc_test_original, 4), "\n")
```

```
## Original stacked model's AUC: 0.9492
```

```
cat("Fine tuned stacked modal's AUC:", round(auc_test_tuned, 4), "\n")
```

```
## Fine tuned stacked modal's AUC: 0.9359
```

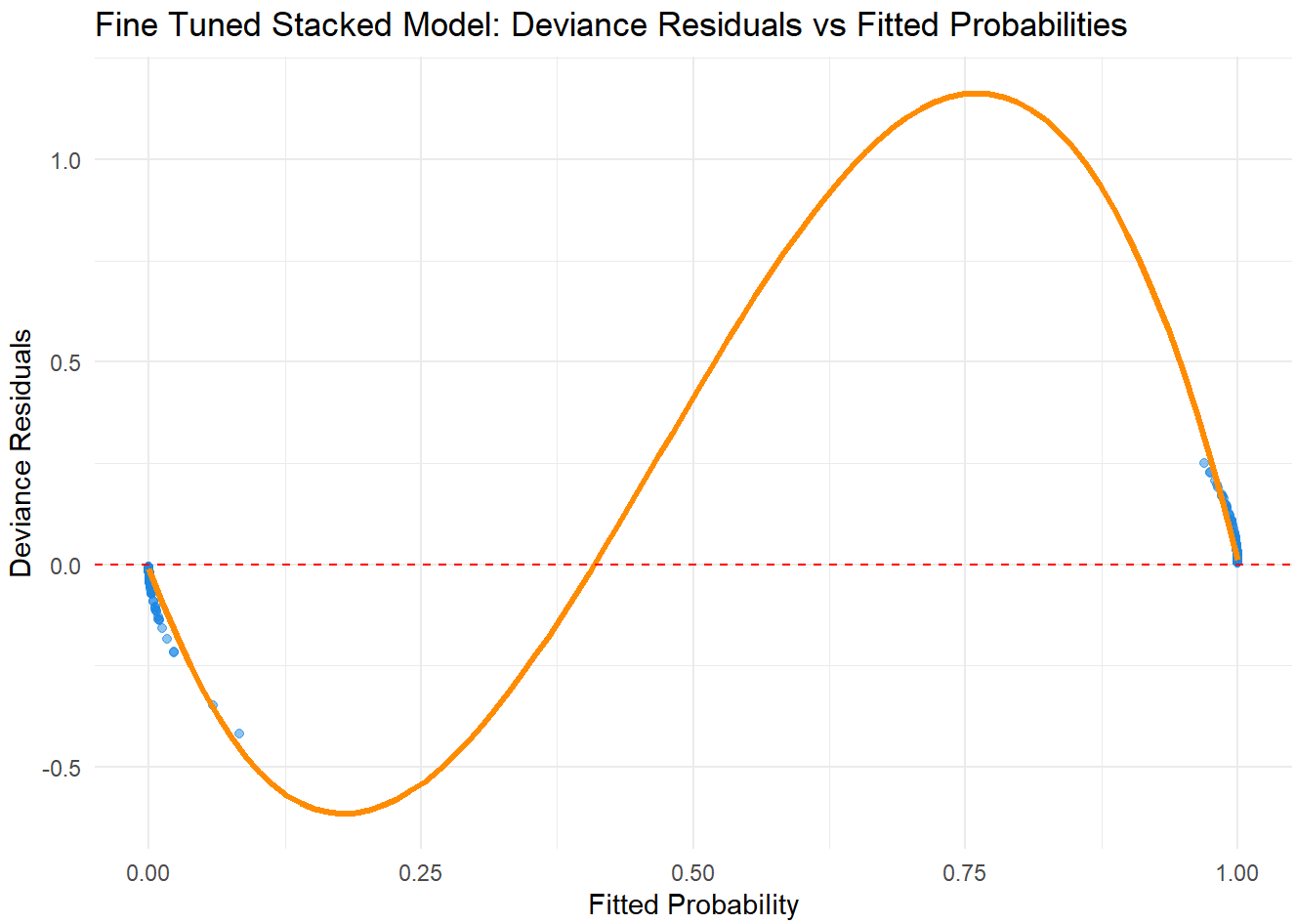
```
cat("Change in AUC:", round(auc_test_tuned - auc_test_original, 4), "\n")
```

```
## Change in AUC: -0.0133
```

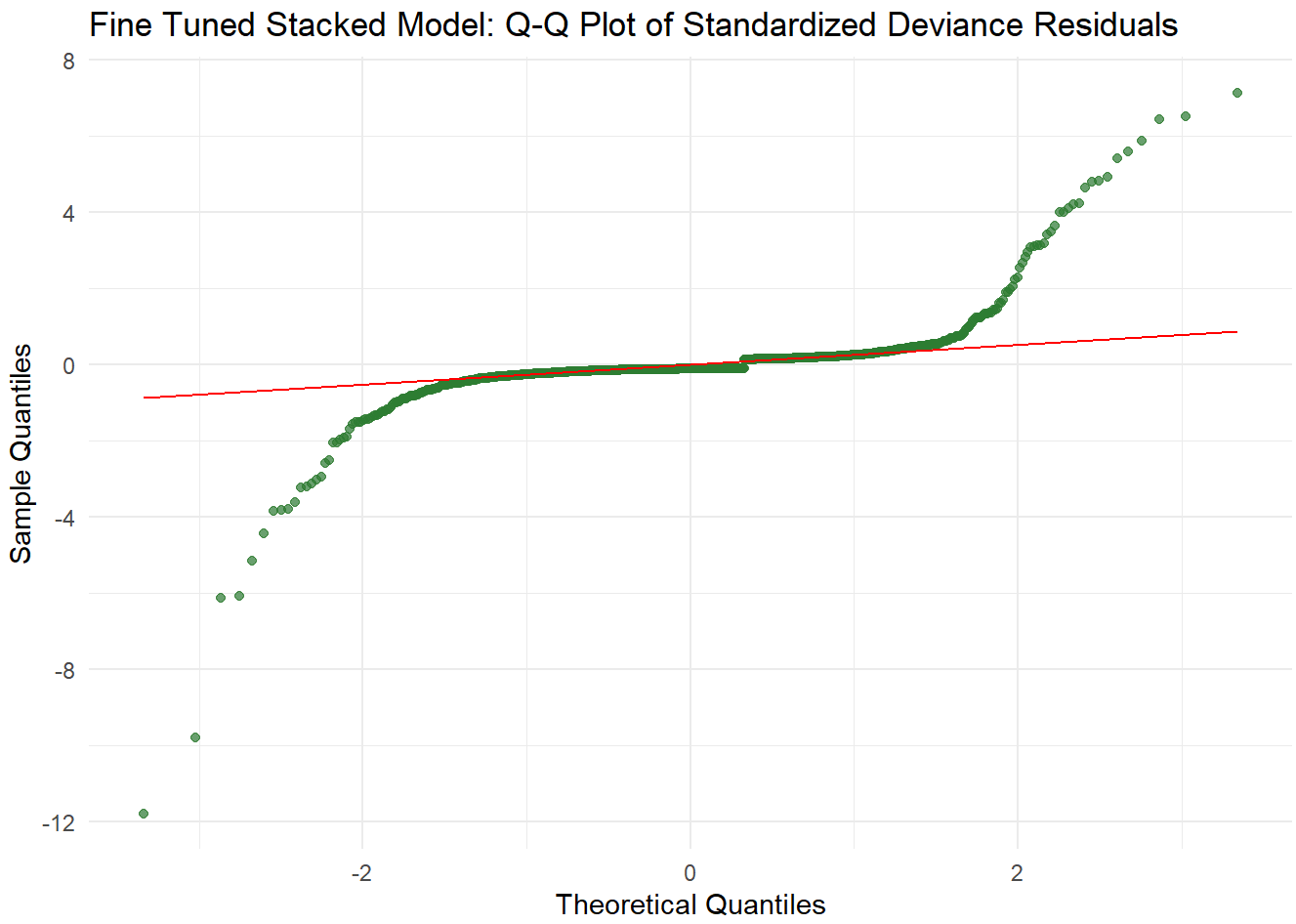
Fine tuned stacked model diagnosis

```
library(ggplot2)

# --- 1. Calculate fine tuned stacked model's prediction on training set ---
pred_prob_train_tuned <- as.vector(
    predict(stacked_lr_tuned, newx = X_train_tuned, s = "lambda.min", type = "response")
)

# --- 2. Calculate the deviance residual and standardized residual ---
y_train_binary <- y_train

dev_resid_tuned <- ifelse(y_train_binary == 1,
                          sqrt(-2 * log(pred_prob_train_tuned)),
                          -sqrt(-2 * log(1 - pred_prob_train_tuned)))
std_resid_tuned <- scale(dev_resid_tuned)

# --- 3. Plot Deviance Residuals vs Fitted Plot ---
resid_data_tuned <- data.frame(
  Fitted = pred_prob_train_tuned,
  DevianceResid = dev_resid_tuned
)

ggplot(resid_data_tuned, aes(x = Fitted, y = DevianceResid)) +
  geom_point(alpha = 0.5, color = "#1E88E5") +
  geom_smooth(method = "loess", se = FALSE, color = "darkorange", linewidth = 1.2) +
  geom_hline(yintercept = 0, color = "red", linetype = "dashed") +
  labs(
    title = "Fine Tuned Stacked Model: Deviance Residuals vs Fitted Probabilities",
    x = "Fitted Probability",
    y = "Deviance Residuals"
  ) +
  theme_minimal()
```

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




```
# --- 4. Q-Q Plot ---
ggplot(data.frame(StdResid = std_resid_tuned), aes(sample = StdResid)) +
  stat_qq(alpha = 0.7, color = "#2E7D32") +
  stat_qq_line(color = "red", linetype = "solid") +
  labs(
    title = "Fine Tuned Stacked Model: Q-Q Plot of Standardized Deviance Residuals",
    x = "Theoretical Quantiles",
    y = "Sample Quantiles"
  ) +
  theme_minimal()
```


Fine Tuned Stacked Model: Q-Q Plot of Standardized Deviance Residuals
Sample Quantiles
Theoretical Quantiles
8
4
0
-4
-8
-12
-2
0
2