Code Appendix

```
#|Warning: FALSE
library(png)
library(jpeg)
library(dplyr)
library(tidyr)
library(tidyverse)
library(caret)
library(ggcorrplot)
library(ggplot2)
library(gridExtra)
library(kableExtra)
library(knitr)
library(plotly)
library(purrr)
library(kernlab)
library(BiocManager)
library(readr)
library(doParallel)
library(xgboost)
library(EBImage)
library(pROC)

if (!requireNamespace("EBImage", quietly = TRUE)) {
  BiocManager::install("EBImage")
}
library(EBImage)

set.seed(100)

#load dataests
train_drowsy_dir <- "Drowsy_datset/train/DROWSY"
train_natural_dir <- "Drowsy_datset/train/NATURAL"
test_drowsy_dir <- "Drowsy_datset/test/DROWSY"
test_natural_dir <- "Drowsy_datset/test/NATURAL"

# Check files in each directory
drowsy_files_train <- list.files(train_drowsy_dir,
                                 pattern = "\\.png$",
                                 full.names = TRUE)
```

```r
natural_files_train <- list.files(train_natural_dir,
                                  pattern = "\\.png$",
                                  full.names = TRUE)
drowsy_files_test <- list.files(test_drowsy_dir,
                                pattern = "\\.png$",
                                full.names = TRUE)
natural_files_test <- list.files(test_natural_dir,
                                 pattern = "\\.png$",
                                 full.names = TRUE)

# Some of the files are png, some jpeg. We need to data clean.
check_png <- function(filepath) {
  tryCatch({
    img <- png::readPNG(filepath)
    return(TRUE)
  }, error = function(e) {
    return(FALSE)
  })
}

check_jpeg <- function(filepath) {
  tryCatch({
    jpeg::readJPEG(filepath)
    return(TRUE)
  }, error = function(e) {
    return(FALSE)
  })
}


# Define a wrapper function to read an image
read_image <- function(filepath) {
  if (check_png(filepath)) {
    img <- png::readPNG(filepath)
  } else if (check_jpeg(filepath)) {
    img <- jpeg::readJPEG(filepath)
  } else {
    stop("Unrecognized image format for file: ", filepath)
  }
  return(img)
}
```

```r
# Use functions to load all the images as vectors
loaded_images_drowsy_train <- lapply(drowsy_files_train, read_image)
loaded_images_natural_train <- lapply(natural_files_train, read_image)
loaded_images_drowsy_test <- lapply(drowsy_files_test, read_image)
loaded_images_natural_test <- lapply(natural_files_test, read_image)

# Create a base data frame using data.frame()
image_dataset_train <- data.frame(
  image = I(c(loaded_images_drowsy_train, loaded_images_natural_train)),
  label = c(rep("Drowsy",
                length(loaded_images_drowsy_train)),
            rep("Natural",
                length(loaded_images_natural_train))),
  stringsAsFactors = FALSE
)

image_dataset_test <- data.frame(
  image = I(c(loaded_images_drowsy_test, loaded_images_natural_test)),
  label = c(rep("Drowsy",
                length(loaded_images_drowsy_test)),
            rep("Natural",
                length(loaded_images_natural_test))),
  stringsAsFactors = FALSE
)

# Shuffle Datasets
set.seed(100)
image_dataset_train <-
  image_dataset_train[sample(nrow(image_dataset_train)), ]
image_dataset_test <-
  image_dataset_test[sample(nrow(image_dataset_test)), ]

# split for stacking and ablation
mid <- floor(nrow(image_dataset_test) / 2)
image_dataset_test_ablation <-
  image_dataset_test[1:mid, ]
image_dataset_test_stacking <-
  image_dataset_test[(mid + 1):nrow(image_dataset_test), ]

# Define a function that applies CLAHE
applyFeatureEngineering <- function(df) {
```

```r
# Helper function to apply CLAHE using EBImage.
apply_CLAHE <- function(img_matrix) {
  # Convert the 48x48 matrix into an EBImage object.
  img <- EBImage::Image(img_matrix)

  # Apply CLAHE.
  # Note: Your version of EBImage must have the clahe() function available.
  img_clahe <- EBImage::clahe(img,
                              nx = 8,
                              ny = 8,
                              bins = 256,
                              limit = 2,
                              keep.range = FALSE)


  # Convert back to a matrix.
  return(as.matrix(img_clahe))
}

# Helper function for High Frequency Extraction (HFE).
high_frequency_extraction <- function(img_matrix) {
  # Convert the 48x48 matrix to an EBImage object.
  img <- EBImage::Image(img_matrix)

  # Apply a Gaussian blur to capture low frequency detail.
  img_blurred <- EBImage::gblur(img, sigma = 1)

  # Subtract the blurred image
  img_hfe <- img - img_blurred

  # Convert the result back to a matrix.
  return(as.matrix(img_hfe))
}

# Apply the feature engineering functions to each image in the dataset.
# Creates two new columns:
#   - image_clahe: with CLAHE-enhanced images.
#   - image_hfe: with high frequency extracted images.
to_ret <- df %>%
  mutate(
    image = map(image, ~ apply_CLAHE(.))
  ) %>%
```

```
    mutate(
      image = map(image, ~ high_frequency_extraction(.))
    )

  return(to_ret)
}

# apply feature engineeering to the related datasets
processedtrain <- applyFeatureEngineering(image_dataset_train)
processedTestAblation <- applyFeatureEngineering(image_dataset_test_ablation)
processedTestStacking <- applyFeatureEngineering(image_dataset_test_stacking)


# perform flattening before seperating with PCA
trainNoFeatureEngineering <- image_dataset_train %>%
  mutate(flat_image = map(image, ~ as.vector(t(.)))) %>%
  select(-image) %>%
  unnest_wider(flat_image, names_sep = "_")

testNoFeatureEngineeringAblation <- image_dataset_test_ablation %>%
  mutate(flat_image = map(image, ~ as.vector(t(.)))) %>%
  select(-image) %>%
  unnest_wider(flat_image, names_sep = "_")

testNoFeatureEngineeringStacking <- image_dataset_test_stacking %>%
  mutate(flat_image = map(image, ~ as.vector(t(.)))) %>%
  select(-image) %>%
  unnest_wider(flat_image, names_sep = "_")
trainWithFeatureEngineering <- processedtrain %>%
  mutate(flat_image = map(image, ~ as.vector(t(.)))) %>%
  select(-image) %>%
  unnest_wider(flat_image, names_sep = "_")

testWithFeatureEngineeringAblation <- processedTestAblation %>%
  mutate(flat_image = map(image, ~ as.vector(t(.)))) %>%
  select(-image) %>%
  unnest_wider(flat_image, names_sep = "_")

testWithFeatureEngineeringStacking <- processedTestStacking %>%
  mutate(flat_image = map(image, ~ as.vector(t(.)))) %>%
  select(-image) %>%
  unnest_wider(flat_image, names_sep = "_")
```

```r
# Function used to unflatten an image for printing out
reconstruct_image <- function(df_row, label_col = "label") {

  label_value <- df_row[[label_col]]
  pixel_columns <- grep("^flat_image_", names(df_row), value = TRUE)
  pixel_values  <- as.numeric(df_row[pixel_columns])
  mat <- matrix(pixel_values, nrow = 48, ncol = 48, byrow = TRUE)

  image(
    x      = 1:48,
    y      = 1:48,
    z      = t(apply(mat, 2, rev)),
    col    = gray(seq(0, 1, length.out = 256)),
    main   = paste("Label:", label_value),
    xlab   = "",
    ylab   = "",
    axes   = FALSE,
    asp    = 1
  )

  return(mat)
}


# Function that plots the feature flattened images for
plot_image_ggplot <- function(df_row, label_col = "label") {
  label_value <- df_row[[label_col]]

  pixel_columns <- grep("^flat_image_", names(df_row), value = TRUE)
  pixel_values  <- as.numeric(df_row[pixel_columns])

  image_matrix <- matrix(pixel_values, nrow = 48, ncol = 48, byrow = TRUE)

  df_image <- expand.grid(x = 1:48, y = 1:48)

  df_image$fill <- as.vector(t(image_matrix))

  p <- ggplot(df_image, aes(x = x, y = y, fill = fill)) +
    geom_raster() +
    scale_fill_gradient(low = "black", high = "white") +
    scale_y_reverse() +
    labs(title = paste("Label:", label_value)) +
```

```
    theme_minimal() +
    theme(
      axis.title = element_blank(),
      axis.ticks = element_blank(),
      axis.text = element_blank(),
      plot.title = element_text(hjust = 0.5)
    )

  return(p)
}



# Create list of plots for the first 9 images
plot_list <- lapply(1:9, function(i) {
  plot_image_ggplot(trainNoFeatureEngineering[i, ])
})

# Arrange and display the plots in a 3x3 grid
grid_arrange_result <- grid.arrange(grobs = plot_list, ncol = 3, nrow = 3)

ggsave(
  filename = "PlotsAndPictures/Faces/trainNoFeatureEngineering.png",
  plot     = grid_arrange_result,
  width    = 8,
  height   = 8,
  units    = "in",
  dpi      = 300
)

# Create list of plots for the first 9 images
plot_list <- lapply(1:9, function(i) {
  plot_image_ggplot(trainWithFeatureEngineering[i, ])
})

# Arrange and display the plots in a 3x3 grid
grid_arrange_result <- grid.arrange(grobs = plot_list, ncol = 3, nrow = 3)

ggsave(
  filename = "PlotsAndPictures/Faces/trainWtihFeatureEngineeringUnscaled.png",
  plot     = grid_arrange_result,
  width    = 8,
```

```
  height   = 8,
  units    = "in",
  dpi      = 300
)
# standardize features back to [0,1] scale after feature processing
trainWithFeatureEngineering <- trainWithFeatureEngineering %>%
  mutate(across(where(is.numeric), ~ (. - min(.)) / (max(.) - min(.))))
testWithFeatureEngineeringAblation <- testWithFeatureEngineeringAblation %>%
  mutate(across(where(is.numeric), ~ (. - min(.)) / (max(.) - min(.))))
testWithFeatureEngineeringStacking <- testWithFeatureEngineeringStacking %>%
  mutate(across(where(is.numeric), ~ (. - min(.)) / (max(.) - min(.))))
# Create list of plots for the first 9 images
plot_list <- lapply(1:9, function(i) {
  plot_image_ggplot(trainWithFeatureEngineering[i, ])
})

# Arrange and display the plots in a 3x3 grid
grid_arrange_result <- grid.arrange(grobs = plot_list, ncol = 3, nrow = 3)

ggsave(
  filename = "PlotsAndPictures/Faces/trainWithFeatureEngineering.png",
  plot     = grid_arrange_result,
  width    = 8,
  height   = 8,
  units    = "in",
  dpi      = 300
)

#Save our datasets here.

write.csv(trainNoFeatureEngineering,
          file ="Prepped_Data/trainNoFeatureEngineering.csv",
          row.names = FALSE)
write.csv(testNoFeatureEngineeringAblation,
          file="Prepped_Data/testNoFeatureEngineeringAblation.csv",
          row.names = FALSE)
write.csv(testNoFeatureEngineeringStacking,
          file = "Prepped_Data/testNoFeatureEngineeringStacking.csv",
          row.names = FALSE)

write.csv(trainWithFeatureEngineering,
          file = "Prepped_Data/trainWithFeatureEngineering.csv",
```

```
          row.names = FALSE)
write.csv(testWithFeatureEngineeringAblation,
          file = "Prepped_Data/testWithFeatureEngineeringAblation.csv",
          row.names = FALSE)
write.csv(testWithFeatureEngineeringStacking,
          file = "Prepped_Data/testWithFeatureEngineeringStacking.csv",
          row.names = FALSE)

# perfomrm pca on the non-engineered set
numeric_data <- trainNoFeatureEngineering %>%
  select(where(is.numeric))

# Perform PCA, scaling the variables since they may be on different scales
pca_result <- prcomp(numeric_data, scale. = TRUE)

# Calculate the proportion of variance explained for each principal component.
# pca_result$sdev holds the standard deviations for each PC.
variance_explained <- pca_result$sdev^2 / sum(pca_result$sdev^2)
cumulative_variance <- cumsum(variance_explained)

# Create a data frame for plotting
pca_df <- data.frame(
  PC = 1:length(variance_explained),
  VarianceExplained = variance_explained,
  CumulativeVariance = cumulative_variance
)

# find the threshold where 95% of the variance is explained,
#and use that many components.
label_threshold <- 0.95
n_components <- which(cumulative_variance >= label_threshold)[1]
cat(sprintf("Number of components to reach at least %.0f%% variance explained: %d", label_thr


# construct a pca plot
scree_plot <- ggplot(pca_df, aes(x = PC)) +
  geom_bar(aes(y = VarianceExplained),
           stat = "identity",
           fill = "steelblue",
           alpha = 0.7) +
  geom_line(aes(y = CumulativeVariance),
            color = "red",
```

```
            size = 1) +
  geom_point(aes(y = CumulativeVariance),
             color = "red",
             size = 2) +
  geom_vline(xintercept = n_components,
             linetype = "dashed",
             color = "darkgreen") +
  labs(title = "Scree Plot for PCA",
       subtitle = sprintf("Vertical dashed line indicates %d components (>= %.0f%% variance e
                          n_components,
                          label_threshold*100),
       x = "Principal Component",
       y = "Proportion of Variance Explained") +
  theme_minimal()

ggsave(
  filename = "PlotsAndPictures/PCA_ScreePlots/NoPreprocessingScree.png",
  plot = scree_plot,
  width = 8,
  height = 6
)


# perfomrm pca on the non-engineered set
test_numeric_ablation <- testNoFeatureEngineeringAblation %>%
  select(-label) %>%
  select(where(is.numeric))

test_numeric_stacking <- testNoFeatureEngineeringStacking %>%
  select(-label) %>%
  select(where(is.numeric))

# For the training set
pca_train <- as.data.frame(pca_result$x[, 1:183])
pca_train$label <- trainNoFeatureEngineering$label
pca_train$label <- factor(pca_train$label, levels = c("Drowsy", "Natural"))

# For the test sets:
scaled_test_ablation <-
  scale(test_numeric_ablation,
        center = pca_result$center,
        scale = pca_result$scale)
```

```r
pca_test_full_ablation <-
  as.data.frame(as.matrix(scaled_test_ablation) %*% pca_result$rotation)
pca_test_ablation <- pca_test_full_ablation[, 1:183]
pca_test_ablation$label <-
  testNoFeatureEngineeringAblation$label
pca_test_ablation$label <-
  factor(pca_test_ablation$label,
         levels = c("Drowsy", "Natural"))

scaled_test_stacking <-
  scale(test_numeric_stacking,
        center = pca_result$center,
        scale = pca_result$scale)
pca_test_full_stacking <-
  as.data.frame(as.matrix(scaled_test_stacking) %*% pca_result$rotation)
pca_test_stacking <- pca_test_full_stacking[, 1:183]
pca_test_stacking$label <-
  testNoFeatureEngineeringStacking$label
pca_test_stacking$label <-
  factor(pca_test_stacking$label,
         levels = c("Drowsy", "Natural"))
# save csvs
write.csv(pca_train,
          file = "Prepped_Data/trainNoFeatureEngineeringPCA.csv",
          row.names = FALSE)
write.csv(pca_test_ablation,
          file = "Prepped_Data/testNoFeatureEngineeringAblationPCA.csv",
          row.names = FALSE)
write.csv(pca_test_stacking,
          file = "Prepped_Data/testNoFeatureEngineeringStackingPCA.csv",
          row.names = FALSE)
# Perform the exact same feature but on the non-engineered dataset.
numeric_data <- trainWithFeatureEngineering %>%
  select(where(is.numeric))

pca_result <- prcomp(numeric_data, scale. = TRUE)

variance_explained <- pca_result$sdev^2 / sum(pca_result$sdev^2)
cumulative_variance <- cumsum(variance_explained)

pca_df <- data.frame(
  PC = 1:length(variance_explained),
```

```
  VarianceExplained = variance_explained,
  CumulativeVariance = cumulative_variance
)

target_threshold <- 0.95
n_components <- which(cumulative_variance >= target_threshold)[1]
cat(sprintf("Number of components to reach at least %.0f%% variance explained: %d", target_t

# Create plot
scree_plot <- ggplot(pca_df, aes(x = PC)) +
  geom_bar(aes(y = VarianceExplained),
           stat = "identity",
           fill = "steelblue",
           alpha = 0.7) +
  geom_line(aes(y = CumulativeVariance),
            color = "red",
            size = 1) +
  geom_point(aes(y = CumulativeVariance),
             color = "red",
             size = 2) +
  geom_vline(xintercept = n_components,
             linetype = "dashed",
             color = "darkgreen") +
  labs(title = "Scree Plot for PCA",
       subtitle = sprintf("Vertical dashed line indicates %d components (>= %.0f%% variance
                          n_components,
                          target_threshold*100),
       x = "Principal Component",
       y = "Proportion of Variance Explained") +
  theme_minimal()

ggsave(
  filename = "PlotsAndPictures/PCA_ScreePlots/PreprocessingScree.png",
  plot = scree_plot,
  width = 8,
  height = 6
)

test_numeric_ablation <- testWithFeatureEngineeringAblation %>%
  select(-label) %>%
  select(where(is.numeric))
```

```r
test_numeric_stacking <- testWithFeatureEngineeringStacking %>%
  select(-label) %>%
  select(where(is.numeric))

pca_train <- as.data.frame(pca_result$x[, 1:1313])
pca_train$label <- trainWithFeatureEngineering$label
pca_train$label <-
  factor(pca_train$label, levels = c("Drowsy", "Natural"))

scaled_test_ablation <-
  scale(test_numeric_ablation,
        center = pca_result$center,
        scale = pca_result$scale)
pca_test_full_ablation <-
  as.data.frame(as.matrix(scaled_test_ablation) %*% pca_result$rotation)
pca_test_ablation <- pca_test_full_ablation[, 1:1313]
pca_test_ablation$label <- testWithFeatureEngineeringAblation$label
pca_test_ablation$label <-
  factor(pca_test_ablation$label,
         levels = c("Drowsy", "Natural"))

scaled_test_stacking <-
  scale(test_numeric_stacking,
        center = pca_result$center,
        scale = pca_result$scale)
pca_test_full_stacking <-
  as.data.frame(as.matrix(scaled_test_stacking) %*% pca_result$rotation)
pca_test_stacking <- pca_test_full_stacking[, 1:1313]
pca_test_stacking$label <- testWithFeatureEngineeringStacking$label
pca_test_stacking$label <-
  factor(pca_test_stacking$label,
         levels = c("Drowsy", "Natural"))

#Save results
write.csv(pca_train,
          file = "Prepped_Data/trainWithFeatureEngineeringPCA.csv",
          row.names = FALSE)
write.csv(pca_test_ablation,
          file = "Prepped_Data/testWithFeatureEngineeringAblationPCA.csv",
          row.names = FALSE)
write.csv(pca_test_stacking,
          file = "Prepped_Data/testWithFeatureEngineeringStackingPCA.csv",
```

```
          row.names = FALSE)
# function to perform model training for all 12 in our ablation study
# folders are set up so they can be combined later
prepare_classification <- function(modelType,
                                   featureEng = FALSE,
                                   isPCA      = FALSE,
                                   seed       = 100) {
  # derive filename tags
  dataTag <- if (featureEng) "WithFeatureEngineering" else "NoFeatureEngineering"
  pcaTag  <- if (isPCA) "PCA" else ""

  # build input paths
  train_path <- file.path("Prepped_Data",
                          sprintf("train%s%s.csv", dataTag, pcaTag))
  test_path  <- file.path("Prepped_Data",
                          sprintf("test%sAblation%s.csv", dataTag, pcaTag))

  # load & factor labels
  train <- readr::read_csv(train_path, show_col_types = FALSE)
  test  <- readr::read_csv(test_path,  show_col_types = FALSE)
  train$label <- factor(train$label, levels = c("Drowsy", "Natural"))
  test$label  <- factor(test$label,  levels = c("Drowsy", "Natural"))

  # pick caret method and output dirs
  method_str <- switch(modelType,
    RandomForest = "rf",
    XGBoost      = "xgbTree",
    gaussianSVM  = "svmRadial",
    stop("Unknown modelType:", modelType)
  )
  file_tag <- paste0(
    switch(modelType, RandomForest="RandomForest",
           XGBoost="XGBoost",
           gaussianSVM="SVM"),
    dataTag, pcaTag
  )
  base_out      <- file.path("Base_Models_Data", tolower(modelType))
  model_out_dir <- "Base_Models"
  dir.create(base_out,      recursive=TRUE, showWarnings=FALSE)
  dir.create(model_out_dir, recursive=TRUE, showWarnings=FALSE)

  # set up parallel and trainControl
```

```
  # for faster processing
  set.seed(seed)
  cl <- parallel::makeCluster(parallel::detectCores() - 1)
  doParallel::registerDoParallel(cl)
  train_control <- caret::trainControl(
    method          = "cv",
    number          = 5,
    allowParallel   = TRUE,
    classProbs      = TRUE,
    summaryFunction = caret::defaultSummary
  )
  parallel::stopCluster(cl)

  list(
    train         = train,
    test          = test,
    method        = method_str,
    file_tag      = file_tag,
    base_out      = base_out,
    model_out_dir = model_out_dir,
    train_control = train_control
  )
}

# function 2: produce all the classification reports
# plots, charts, etc.
run_classification <- function(prepped) {
  with(prepped, {
    # train
    model_fit <- caret::train(
      label ~ ., data   = train,
      method         = method,
      trControl      = train_control,
      metric         = "Accuracy"
    )
    saveRDS(model_fit, file = file.path(model_out_dir,
                                        paste0(file_tag, ".rds")))
    best_df <- cbind(model = file_tag, model_fit$bestTune)
    write.csv(best_df, file = file.path(base_out,
                                        paste0("BestHyperparams_",
                                               file_tag, ".csv")),
              row.names=FALSE)
```

```r
# predict + confusion matrix
preds <- predict(model_fit, test)
probs <- predict(model_fit, test, type="prob")
cm    <- caret::confusionMatrix(preds, test$label)
pos   <- levels(test$label)[1]; neg <- levels(test$label)[2]
tbl   <- cm$table
cm_df <- data.frame(
  model = file_tag,
  TP    = tbl[pos, pos],
  FP    = tbl[pos, neg],
  FN    = tbl[neg, pos],
  TN    = tbl[neg, neg]
)
write.csv(cm_df, file = file.path(base_out,
                                  paste0("ConfusionMatrix_",
                                         file_tag, ".csv")),
          row.names=FALSE)

# metrics & ROC
accuracy  <- cm$overall["Accuracy"]
recall    <- cm$byClass["Sensitivity"]
precision <- cm$byClass["Pos Pred Value"]
f1_score  <- 2*(precision*recall)/(precision+recall)
roc_obj   <- pROC::roc(response = test$label, predictor = probs[[pos]])
auc_val   <- as.numeric(pROC::auc(roc_obj))

metrics_df <- data.frame(
  model     = file_tag,
  Accuracy  = accuracy,
  Precision = precision,
  Recall    = recall,
  FOne      = f1_score,
  AUC       = auc_val
)
write.csv(metrics_df, file = file.path(base_out,
                                       paste0("Metrics_",
                                              file_tag, ".csv")),
          row.names=FALSE)

# ROC data for future plot concatentation
roc_data <-
  data.frame(fpr = 1 - roc_obj$specificities,
```

```
                tpr = roc_obj$sensitivities)
    write.csv(roc_data,
              file = file.path(base_out,
                               paste0("ROCData_", file_tag, ".csv")),
              row.names=FALSE)
    roc_plot <-
      ggplot2::ggplot(roc_data, ggplot2::aes(x=fpr, y=tpr)) +
      ggplot2::geom_line() +
      ggplot2::geom_abline(slope=1, intercept=0, linetype="dotted") +
      ggplot2::labs(x="FPR", y="TPR", title=paste("ROC:", file_tag)) +
      ggplot2::theme_minimal() +
      ggplot2::annotate("text", x=0.75, y=0.95,
                        label=paste("AUC =",
                                    format(round(auc_val,3),
                                           nsmall=3)),
                        size=5)
    ggplot2::ggsave(filename = file.path(base_out,
                                         paste0("ROC_", file_tag, ".png")),
                    plot= roc_plot,
                    width=7,
                    height=7,
                    dpi=300)

    message("Pipeline completed for: ", file_tag)
  })
}

#######################################
# RUN ALL 12 MODELs
#######################################
# 1. gaussianSVM, no FE, PCA
prep <- prepare_classification(modelType = "gaussianSVM",
                               featureEng = F,
                               isPCA = T,
                               seed = 100)
run_classification(prep)
# 2. XGBoost, no FE, PCA
prep <- prepare_classification(modelType = "XGBoost",
                               featureEng = F,
                               isPCA = T,
                               seed = 100)
run_classification(prep)
```

```
# 3. RandomForest, no FE, PCA
prep <- prepare_classification(modelType = "RandomForest",
                               featureEng = F,
                               isPCA = T,
                               seed = 100)
run_classification(prep)
# 4. gaussianSVM, FE, PCA
prep <- prepare_classification(modelType = "gaussianSVM",
                               featureEng = T,
                               isPCA = T,
                               seed = 100)
run_classification(prep)
# 5. XGBoost, FE, PCA
prep <- prepare_classification(modelType = "XGBoost",
                               featureEng = T,
                               isPCA = T,
                               seed = 100)
run_classification(prep)
# 6. RandomForest, FE, PCA
prep <- prepare_classification(modelType = "RandomForest",
                               featureEng = T,
                               isPCA = T,
                               seed = 100)
run_classification(prep)
# 7. gaussianSVM, no FE, no PCA
prep <- prepare_classification(modelType = "gaussianSVM",
                               featureEng = F,
                               isPCA = F,
                               seed = 100)
run_classification(prep)
# 8. XGBoost, no FE, no PCA
prep <- prepare_classification(modelType = "XGBoost",
                               featureEng = F,
                               isPCA = F,
                               seed = 100)
run_classification(prep)
# 9. RandomForest, no FE, no PCA
prep <- prepare_classification(modelType = "RandomForest",
                               featureEng = F,
                               isPCA = F,
                               seed = 100)
run_classification(prep)
```

```r
# 10. gaussianSVM, FE, no PCA
prep <- prepare_classification(modelType = "gaussianSVM",
                               featureEng = T,
                               isPCA = F,
                               seed = 100)
run_classification(prep)
# 11. XGBoost, FE, no PCA
prep <- prepare_classification(modelType = "XGBoost",
                               featureEng = T,
                               isPCA = F,
                               seed = 100)
run_classification(prep)
# 12. RandomForest, FE, no PCA
prep <- prepare_classification(modelType = "RandomForest",
                               featureEng = T,
                               isPCA = F,
                               seed = 100)
run_classification(prep)
```

```r
# the following combines all the files to make final reports for the bast
# models
combine_model_reports <- function(model) {
  # map the logical model name to the actual folder name
  dir_name <- switch(model,
                     XGBoost      = "XGBoost",
                     SVM          = "SVM",
                     RandomForest = "Random_Forest",
                     stop("Unknown model: ", model))

  base_path <- file.path("Base_Models_Data", dir_name)

  # find all the files
  metrics_files      <- list.files(base_path,
                                   pattern = "^Metrics_.*\\.csv$",
                                   full.names = TRUE)
  confusion_files    <- list.files(base_path,
                                   pattern = "^ConfusionMatrix_.*\\.csv$",
                                   full.names = TRUE)
  hyperparam_files   <- list.files(base_path,
                                   pattern = "^BestHyperparams_.*\\.csv$",
                                   full.names = TRUE)

  # helper to strip prefix and .csv
  extract_label <- function(path, prefix) {
    basename(path) %>%
      str_remove(paste0("^", prefix)) %>%
      str_remove("\\.csv$")
  }

  # map labels to confusion files
  confusion_map <- set_names(confusion_files,
                             map_chr(confusion_files,
                                     extract_label,
                                     prefix = "ConfusionMatrix_"))

  classification_list <- map(metrics_files, function(metrics_file) {
    label <- extract_label(metrics_file, "Metrics_")
    if (!label %in% names(confusion_map)) {
      warning("No ConfusionMatrix for ", label); return(NULL)
    }
    conf_file <- confusion_map[[label]]
```

```
    metrics_df  <- read_csv(metrics_file,  show_col_types = FALSE)
    confusion_df<- read_csv(conf_file,   show_col_types = FALSE)

    inner_join(metrics_df, confusion_df, by = "model")
  }) %>% compact()

  combined_classification <- bind_rows(classification_list)

  combined_hyperparameters <- map_dfr(hyperparam_files,
                                      ~ read_csv(.x, show_col_types = FALSE))

  # write out
  write_csv(combined_classification,
            file.path(base_path,
                      paste0(model, "_CombinedClassificationReport.csv")))
  write_csv(combined_hyperparameters,
            file.path(base_path,
                      paste0(model, "_Hyperparameters.csv")))

  message("Done for model: ", model,
          "\n • Classification report: ",
          file.path(base_path,
                    paste0(model,"_CombinedClassificationReport.csv")),
          "\n • Hyperparameters file:  ",
          file.path(base_path,
                    paste0(model, "_Hyperparameters.csv")))

  # return invisibly if you want to inspect
  invisible(list(classification  = combined_classification,
                 hyperparameters = combined_hyperparameters))
}
```

```
combine_model_reports("XGBoost")
combine_model_reports("SVM")
combine_model_reports("RandomForest")
```

```
# Define file paths for each combined classification report
rf_report <- "Base_Models_Data/Random_Forest/RandomForestCombinedClassificationReport.csv"
xgb_report <- "Base_Models_Data/XGBoost/XGBoostCombinedClassificationReport.csv"
svm_report <- "Base_Models_Data/SVM/SVMCombinedClassificationReport.csv"
```

```r
# Read each CSV into a data frame
rf_df <- read_csv(rf_report, show_col_types = FALSE)
xgb_df <- read_csv(xgb_report, show_col_types = FALSE)
svm_df <- read_csv(svm_report, show_col_types = FALSE)

# Rbind (stack) the three data frames into one and then
# sort by accuracy then ROC (both in descending order)
final_report <- bind_rows(rf_df, xgb_df, svm_df) %>%
  arrange(desc(Accuracy), desc(AUC))

# Define the output path and write the final combined data frame as CSV
output_path <- "Base_Models_Data/FinalBaseModelReport.csv"
write_csv(final_report, output_path)
```

```r
# Define file paths for each combined set of hyperparams
rf_report <- "Base_Models_Data/Random_Forest/RandomForestHyperparameters.csv"
xgb_report <- "Base_Models_Data/XGBoost/XGBoostHyperparameters.csv"
svm_report <- "Base_Models_Data/SVM/SVMHyperparameters.csv"

# Read each CSV into a data frame
rf_df <- read_csv(rf_report, show_col_types = FALSE)
xgb_df <- read_csv(xgb_report, show_col_types = FALSE)
svm_df <- read_csv(svm_report, show_col_types = FALSE)


# Define the output path and write the final combined data frame as CSV
write_csv(rf_df, "Base_Models_Data/RandomForestHyperparameters.csv")
write_csv(xgb_df, "Base_Models_Data/XGBoostHyperparameters.csv")
write_csv(svm_df, "Base_Models_Data/SVMHyperparameters.csv")
```

```r
# ROC-plotting function for all base models
plot_model_roc <- function(model_folder,
                           file_prefix,
                           plot_title,
                           output_filename,
                           base_dir     = "Base_Models_Data",
                           out_dir      = "PlotsAndPictures/Combined_AUCROCPlots",
                           curve_labels = c(
                             "No pre-processing + No feature engineering",
                             "No pre-processing + Feature engineering",
                             "Pre-processing + No feature engineering",
                             "Pre-processing + Feature engineering"
```

```
                          )) {
# Used to plot the AUCs
calc_auc <- function(x, y) {
  sum(diff(x) * (head(y, -1) + tail(y, -1)) / 2)
}

# 2) locate and sort the four ROC CSVs
model_path <- file.path(base_dir, model_folder)
roc_files  <- list.files(
  model_path,
  pattern    = paste0("^ROCData_", file_prefix, ".*\\.csv$"),
  full.names = TRUE
) %>% sort()

if (length(roc_files) != length(curve_labels)) {
  stop("Expected ", length(curve_labels),
       " files, but found ", length(roc_files), ".")
}

# 3) read, calc AUC, tag each
roc_list <- map2(
  roc_files, curve_labels,
  ~ read_csv(.x, show_col_types = FALSE) %>%
      arrange(fpr) %>%
      mutate(
        AUC   = calc_auc(fpr, tpr),
        Curve = .y
      )
)

# 4) stack them
roc_data <- bind_rows(roc_list)

# 5) order factor by desc AUC
auc_summary <- roc_data %>%
  group_by(Curve) %>%
  summarize(AUC = unique(AUC), .groups = "drop") %>%
  arrange(desc(AUC))
roc_data$Curve <- factor(roc_data$Curve,
                         levels = auc_summary$Curve,
                         ordered = TRUE)
```

```r
# 6) colors: highest -> green, blue, purple, red
color_vec <- c("green", "blue", "purple", "red")

# 7) annotation in bottom-right
ann_df <- auc_summary %>%
  mutate(
    x     = 0.98,
    y     = seq(0.15, by = -0.05, length.out = n()),
    label = paste0("AUC = ", round(AUC, 3))
  )

# 8) build the ggplot
p <- ggplot(roc_data, aes(fpr, tpr, color = Curve, group = Curve)) +
  geom_line(size = 1, key_glyph = "path") +
  geom_abline(intercept = 0, slope = 1, linetype = "dotted") +
  labs(title = plot_title,
       x     = "False Positive Rate",
       y     = "True Positive Rate",
       color = "Model") +
  theme_minimal() +
  scale_color_manual(values = color_vec) +
  scale_shape_discrete(guide = "none") +
  guides(color = guide_legend(override.aes = list(
    shape     = NA,
    linetype  = 1,
    key_glyph = "path"
  ))) +
  geom_text(
    data   = ann_df,
    aes(x = x, y = y, label = label, color = Curve),
    hjust  = 1,
    size   = 4
  )

# 9) save it (create dir if needed)
if (!dir.exists(out_dir)) dir.create(out_dir, recursive = TRUE)
ggsave(
  filename = file.path(out_dir, output_filename),
  plot     = p,
  width    = 8,
  height   = 6
)
```

```
  message("  Saved: ", file.path(out_dir, output_filename))
  invisible(p)
}


# Model Wrappers
plot_svm_roc <- function() {
  plot_model_roc(
    model_folder  = "SVM",
    file_prefix   = "SVM",
    plot_title    = "SVM model AUC ROC Plot",
    output_filename = "SVMPlot.png"
  )
}

plot_xgboost_roc <- function() {
  plot_model_roc(
    model_folder  = "XGBoost",
    file_prefix   = "XGBoost",
    plot_title    = "XGBoost model AUC ROC Plot",
    output_filename = "XGBoostPlot.png"
  )
}

plot_rf_roc <- function() {
  plot_model_roc(
    model_folder  = "Random_Forest",
    file_prefix   = "RandomForest",
    plot_title    = "Random Forest model AUC ROC Plot",
    output_filename = "RandomForestPlot.png"
  )
}
```

```
plot_svm_roc()       # reads SVM CSVs, plots, and saves SVMPlot.png
plot_xgboost_roc()   # same for XGBoost
plot_rf_roc()        # same for Random Forest
```

```
# Load the datasets based on the models we chose
# from our empirical model analysis
trainNoFeatureEngineeringPCA<-
  read_csv("Prepped_Data/testNoFeatureEngineeringAblationPCA.csv",
```

```
                                          show_col_types = FALSE)
testNoFeatureEngineeringPCA<-
  read_csv("Prepped_Data/testNoFeatureEngineeringStackingPCA.csv",
                                          show_col_types = FALSE)
trainNoFeatureEngineering<-
  read_csv("Prepped_Data/testNoFeatureEngineeringAblation.csv",
                                          show_col_types = FALSE)
testNoFeatureEngineering <-
  read_csv("Prepped_Data/testNoFeatureEngineeringStacking.csv",
                                          show_col_types = FALSE)
trainWithFeatureEngineering<-
  read_csv("Prepped_Data/testWithFeatureEngineeringAblation.csv",
                                          show_col_types = FALSE)
testWithFeatureEngineering <-
  read_csv("Prepped_Data/testWithFeatureEngineeringStacking.csv",
                                          show_col_types = FALSE)

# load respective models
XGBoostNoFeatureEngineeringPCA <-
  readRDS('Base_Models/XGBoostNoFeatureEngineeringPCA.rds')
SVMNoFeatureEngineering <-
  readRDS('Base_Models/SVMNoFeatureEngineering.rds')
RandomForestWithFeatureEngineering<-
  readRDS('Base_Models/RandomForestWithFeatureEngineering.rds')
```

```
# generate chosen base model predictions
# For XGBoost and SVM, predictions are assumed to come out
# directly without CLAHE and HFE
preds_train_XGB <- predict(
  XGBoostNoFeatureEngineeringPCA,
  newdata = trainNoFeatureEngineeringPCA,
  type    = "prob"
)[, 2]

preds_test_XGB  <- predict(
  XGBoostNoFeatureEngineeringPCA,
  newdata = testNoFeatureEngineeringPCA,
  type    = "prob"
)[, 2]

preds_train_SVM <- predict(SVMNoFeatureEngineering,
                           newdata = trainNoFeatureEngineering,
```

```
                            type = "prob")[,2]
preds_test_SVM  <- predict(SVMNoFeatureEngineering,
                            newdata = testNoFeatureEngineering,
                            type = "prob")[,2]

# For the Random Forest model (using caret, with probability output)
preds_train_RF <- predict(RandomForestWithFeatureEngineering,
                           newdata = trainWithFeatureEngineering,
                           type = "prob")[,2]
preds_test_RF  <- predict(RandomForestWithFeatureEngineering,
                           newdata = testWithFeatureEngineering,
                           type = "prob")[,2]
```

```
# Build Meta Training and Test Sets
meta_train <- data.frame(
  XGB   = preds_train_XGB,
  SVM   = preds_train_SVM,
  RF    = preds_train_RF,
  label = trainNoFeatureEngineeringPCA$label
)

meta_test <- data.frame(
  XGB   = preds_test_XGB,
  SVM   = preds_test_SVM,
  RF    = preds_test_RF,
  label = testNoFeatureEngineeringPCA$label

)

# save for reloading
write.csv(meta_train, "Prepped_Data/StackedTrain.csv", row.names = FALSE)
write.csv(meta_test, "Prepped_Data/StackedTest.csv", row.names = FALSE)
```

```
# ---- 0. Set up parallel backend and trainControl once ----
num_cores    <- detectCores() - 1
cl           <- makeCluster(num_cores)
registerDoParallel(cl)

train_control <- trainControl(
  method          = "cv",
  number          = 5,
  summaryFunction = defaultSummary,
```

```
  classProbs      = TRUE,
  savePredictions = "all"
)

# ---- 1. The single wrapper function ----
train_and_save_meta_model <- function(meta_train,
                                      meta_test,
                                      method,
                                      model_name,
                                      family=NULL # for glm
) {
  # Ensure label is factor with consistent levels
  meta_train$label <- factor(meta_train$label)
  meta_test$label  <- factor(meta_test$label,
                             levels = levels(meta_train$label))
  positive_class   <- levels(meta_test$label)[1]

  # Train
  meta_model <- train(
    label ~ .,
    data      = meta_train,
    method    = method,
    family    = family,
    metric    = "Accuracy",
    trControl = train_control
  )

  rds_path<- file.path("Stacked_Models", paste0(model_name, ".rds"))
  params_path<- file.path("Stacked_Models_Data",
                          model_name,
                          paste0("BestHyperparams_", model_name, ".csv"))
  cm_path <- file.path("Stacked_Models_Data",
                       model_name,
                       paste0("ConfusionMatrix_", model_name, ".csv"))
  metrics_path<- file.path("Stacked_Models_Data",
                           model_name,
                           paste0("Metrics_",model_name, ".csv"))
  rocdata_path<- file.path("Stacked_Models_Data",
                           model_name,
                           paste0("ROCData_", model_name, ".csv"))
  rocplot_path <- file.path("Stacked_Models_Data",
                            model_name,
```

```
                          paste0("ROC_",model_name, ".png"))

# Make sure output dirs exist
dir.create(dirname(rds_path),     recursive = TRUE, showWarnings = FALSE)
dir.create(dirname(params_path),  recursive = TRUE, showWarnings = FALSE)

# 2. Save model
saveRDS(meta_model, rds_path)
cat("Model saved as '", rds_path, "'. To load it later, use:\n", sep = "")
cat("  loaded_model <- readRDS('", rds_path, "')\n\n", sep = "")

# 3. Save bestTune
best_tune_df <- cbind(model = model_name, meta_model$bestTune)
write.csv(best_tune_df, params_path, row.names = FALSE)
cat("BestTune hyperparameters saved as '", params_path, "'.\n\n", sep = "")

# 4. Predict & Confusion matrix
preds <- factor(predict(meta_model, newdata = meta_test),
                      levels = levels(meta_test$label))
cm <- confusionMatrix(data = preds, reference = meta_test$label)
print(cm)

tbl  <- cm$table
TP <- tbl[positive_class, positive_class]
FP <- tbl[positive_class, levels(meta_test$label)[2]]
FN <- tbl[levels(meta_test$label)[2], positive_class]
TN <- tbl[levels(meta_test$label)[2], levels(meta_test$label)[2]]

cm_df      <- data.frame(model = model_name,
                         TP = TP, FP = FP, FN = FN, TN = TN)
write.csv(cm_df, cm_path, row.names = FALSE)
cat("Confusion matrix saved as '", cm_path, "'.\n\n", sep = "")

# 5. Compute & save metrics + AUC
accuracy  <- unname(cm$overall["Accuracy"])
recall    <- unname(cm$byClass["Sensitivity"])
precision <- unname(cm$byClass["Pos Pred Value"])
f1_score  <- 2 * (precision * recall) / (precision + recall)

probs     <- predict(meta_model,
                     newdata = meta_test,
                     type = "prob")[[positive_class]]
```

```
  roc_obj   <- roc(response = meta_test$label, predictor = probs)
  auc_val   <- as.numeric(auc(roc_obj))

  metrics_df <- data.frame(
    model     = model_name,
    Accuracy  = accuracy,
    Precision = precision,
    Recall    = recall,
    FOne      = f1_score,
    AUC       = auc_val
  )
  write.csv(metrics_df, metrics_path, row.names = FALSE)
  cat("Performance metrics saved as '", metrics_path, "'.\n\n", sep = "")

  # 6. Save ROC data
  roc_data <- data.frame(
    fpr = 1 - roc_obj$specificities,
    tpr = roc_obj$sensitivities
  )
  write.csv(roc_data, rocdata_path, row.names = FALSE)
  cat("ROC Data for plot building saved as '",
      rocdata_path,
      "'.\n\n", sep = "")

  # 7. Plot & save ROC curve
  roc_plot <- ggplot(roc_data, aes(x = fpr, y = tpr)) +
    geom_line() +
    geom_abline(slope = 1, intercept = 0, linetype = "dotted") +
    labs(
      x     = "False Positive Rate (1 - Specificity)",
      y     = "True Positive Rate (Sensitivity)",
      title = paste("ROC Curve for", model_name)
    ) +
    theme_minimal() +
    annotate("text", x = 0.75, y = 0.95,
             label = paste("AUC =", format(round(auc_val, 3), nsmall = 3)),
             color = "red", size = 5)

  ggsave(filename = rocplot_path, plot = roc_plot,
         width = 7, height = 7, dpi = 300)
  cat("ROC plot saved as '", rocplot_path, "'.\n\n", sep = "")
}
```

```r
# ---- 2. Call it for each meta-learner ----
models_to_run <- list(
  list(method = "glm",       model_name = "LogReg",      family = binomial),
  list(method = "svmLinear", model_name = "LinSVM",      family = NULL),
  list(method = "xgbTree",   model_name = "XGBoost",     family = NULL),
  list(method = "rf",        model_name = "RandomForest", family = NULL)
)

for (m in models_to_run) {
  train_and_save_meta_model(meta_train, meta_test,
                            method     = m$method,
                            model_name = m$model_name,
                            family     = m$family)
}

# ---- 3. Tear down parallel backend ----
stopCluster(cl)
registerDoSEQ()
```

```r
# Final plot constructions
# Generic combiner for any stacked-model folder
combine_stacked_reports <- function(model_folder,
                                    base_dir = "Stacked_Models_Data") {
  base_path  <- file.path(base_dir,
                          model_folder)
  metrics_files  <- list.files(base_path,
                               "^Metrics_.*\\.csv$",
                               full.names = TRUE)
  confusion_files <- list.files(base_path,
                                "^ConfusionMatrix_.*\\.csv$",
                                full.names = TRUE)
  hyperparam_files <- list.files(base_path,
                                 "^BestHyperparams_.*\\.csv$",
                                 full.names = TRUE)

  # helper to strip prefix & .csv
  extract_label <- function(path, prefix) {
    basename(path) %>%
      str_remove(paste0("^", prefix)) %>%
      str_remove("\\.csv$")
  }
```

```
  # map labels to confusion matrix files
  confusion_map <- set_names(
    confusion_files,
    map_chr(confusion_files, extract_label, prefix = "ConfusionMatrix_")
  )

  # join metrics and confusion
  classification_list <- map(metrics_files, function(mf) {
    label <- extract_label(mf, "Metrics_")
    if (!label %in% names(confusion_map)) {
      warning("No ConfusionMatrix found for '", label, "'.")
      return(NULL)
    }
    cf <- confusion_map[[label]]
    metrics_df   <- read_csv(mf, show_col_types = FALSE)
    confusion_df <- read_csv(cf, show_col_types = FALSE)
    inner_join(metrics_df, confusion_df, by = "model")
  }) %>% compact()

  combined_classification  <- bind_rows(classification_list)
  combined_hyperparameters <- map_dfr(hyperparam_files,
                                      ~ read_csv(.x, show_col_types = FALSE))

    write_csv(combined_classification,
            file.path(base_path,
                      paste0(model_folder, "CombinedClassificationReport.csv")))
  write_csv(combined_hyperparameters,
            file.path(base_path,
                      paste0(model_folder, "Hyperparameters.csv")))

  message("   Reports written to ", base_path)
  invisible(list(
    classification  = combined_classification,
    hyperparameters = combined_hyperparameters
  ))
}

combine_logreg_reports <- function() {
  combine_stacked_reports("LogReg")
}

combine_linsvm_reports <- function() {
```

```
  combine_stacked_reports("LinSVM")
}

combine_xgboost_reports <- function() {
  combine_stacked_reports("XGBoost")
}

combine_randomforest_reports <- function() {
  combine_stacked_reports("RandomForest")
}
```

```
combine_logreg_reports()
combine_linsvm_reports()
combine_xgboost_reports()
combine_randomforest_reports()
```

```
#combine the reports for the stacked models
rf_report <- "Stacked_Models_Data/RandomForest/RandomForestCombinedClassificationReport.csv"
xgb_report <- "Stacked_Models_Data/XGBoost/XGBoostCombinedClassificationReport.csv"
svm_report <- "Stacked_Models_Data/LinSVM/LinSVMCombinedClassificationReport.csv"
logreg_report <- "Stacked_Models_Data/LogReg/LogRegCombinedClassificationReport.csv"

rf_df <- read_csv(rf_report, show_col_types = FALSE)
xgb_df <- read_csv(xgb_report, show_col_types = FALSE)
svm_df <- read_csv(svm_report, show_col_types = FALSE)
lr_df <- read_csv(logreg_report, show_col_types = FALSE)


final_report <- bind_rows(rf_df, xgb_df, svm_df,lr_df) %>%
  arrange(desc(Accuracy), desc(AUC))

# Define the output path and write the final combined data frame as CSV
output_path <- "Stacked_Models_Data/FinalStackedModelReport.csv"
write_csv(final_report, output_path)
```

```
# construct final dataframe
rf_report <- "Stacked_Models_Data/RandomForest/RandomForestHyperparameters.csv"
xgb_report <- "Stacked_Models_Data/XGBoost/XGBoostHyperparameters.csv"
svm_report <- "Stacked_Models_Data/LinSVM/LinSVMHyperparameters.csv"
lr_report <- "Stacked_Models_Data/LogReg/LogRegHyperparameters.csv"

# Read each CSV into a data frame
```

```r
rf_df <- read_csv(rf_report, show_col_types = FALSE)
xgb_df <- read_csv(xgb_report, show_col_types = FALSE)
svm_df <- read_csv(svm_report, show_col_types = FALSE)


# Define the output path and write the final combined data frame as CSV
write_csv(rf_df, "Stacked_Models_Data/RandomForestHyperparameters.csv")
write_csv(xgb_df, "Stacked_Models_Data/XGBoostHyperparameters.csv")
write_csv(svm_df, "Stacked_Models_Data/SVMHyperparameters.csv")
```

```r
# make 3d-plot for report
calc_auc <- function(x, y) {
  sum(diff(x) * (head(y, -1) + tail(y, -1)) / 2)
}

roc_files <- c(
  "Stacked_Models_Data/XGBoost/ROCData_XGBoost.csv",
  "Stacked_Models_Data/LinSVM/ROCData_LinSVM.csv",
  "Stacked_Models_Data/LogReg/ROCData_LogReg.csv",
  "Stacked_Models_Data/RandomForest/ROCData_RandomForest.csv"
)

curve_labels <- c(
  "Stacked XGBoost",
  "Stacked Linear SVM",
  "Stacked Logistic Regression",
  "Stacked Random Forest"
)

# 3) Read, compute AUC, store results
roc_list <- list()
auc_values <- numeric(length(roc_files))

for(i in seq_along(roc_files)) {
  df <- read_csv(roc_files[i], show_col_types = FALSE) %>%
    arrange(fpr)  # ensure ascending fpr

  this_auc <- calc_auc(df$fpr, df$tpr)
  auc_values[i] <- this_auc

  df <- df %>%
    mutate(AUC = this_auc,
```

```
            Curve = curve_labels[i])

  roc_list[[i]] <- df
}

roc_data <- bind_rows(roc_list)

auc_summary <- roc_data %>%
  group_by(Curve) %>%
  summarize(AUC = unique(AUC), .groups = "drop") %>%
  arrange(desc(AUC))

roc_data$Curve <- factor(roc_data$Curve,
                         levels = auc_summary$Curve,
                         ordered = TRUE)
color_vec <- c("green", "blue", "purple", "red")

ann_df <- auc_summary %>%
  mutate(
    x = 0.98,
    y = seq(0.15, by = -0.05, length.out = nrow(auc_summary)),
    label = paste("AUC =", round(AUC, 3))
  )

# 8) Plot all curves
p <- ggplot(
  roc_data,
  aes(
    x = fpr,
    y = tpr,
    color = Curve,
    group = Curve  # ensures ggplot doesn't use shapes for grouping
  )
) +
  geom_line(size = 1, key_glyph = "path") +
  geom_abline(intercept = 0,
              slope = 1,
              linetype = "dotted",
              color = "black") +
  labs(
    title = "Stacked model AUC ROC Plot",
    x = "False Positive Rate",
```

```
    y = "True Positive Rate",
    color = "Model"
  ) +
  theme_minimal() +
  scale_color_manual(values = color_vec) +
  scale_shape_discrete(guide = "none") +
  guides(color = guide_legend(override.aes = list(
    shape = NA,
    linetype = 1,
    key_glyph = "path"
  ))) +
  geom_text(
    data = ann_df,
    aes(x = x, y = y, label = label, color = Curve),
    hjust = 1,
    size = 4
  )

ggsave(
  filename = "PlotsAndPictures/Combined_AUCROCPlots/StackedPlot.png",
  plot = p,
  width = 8,
  height = 6
)
```

```
df <- read_csv("Prepped_Data/StackedTrain.csv")

# make sure your labels are in the right order for coloring
df$state <- factor(df$label,
                   levels = c("Natural","Drowsy"),
                   labels = c("Natural","Drowsy"))

# interactive 3D scatter
fig <- plot_ly(
  df,
  x     = ~XGB,
  y     = ~SVM,
  z     = ~RF,
  color = ~state,
  colors= c("blue","red"),
  type  = "scatter3d",
  mode  = "markers",
```

```
  marker = list(size = 4)
) %>%
  layout(
    title = "3D Scatter: Natural (blue) vs Drowsy (red)",
    scene = list(
      camera = list(
        eye = list(x = 2, y =-2, z = 2.5)
      ),
      xaxis = list(title = "XGBoost Prediction"),
      yaxis = list(title = "SVM Prediction"),
      zaxis = list(title = "Random Forest Prediction")
    )
  )
```