```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")
    )
  )
```

```
library(ggplot2)
library(png)
library(gridExtra)
library(grid)
base_dir <- "PlotsAndPictures/Combined_AUCROCPlots/"
img_paths <- c("RandomForestPlot.png","XGBoostPlot.png","SVMPlot.png")
img_paths <- file.path(base_dir,img_paths)

img_grobs <- lapply(img_paths, function(img){
  rasterGrob(readPNG(img),interpolate = T)
})

combined_plot <- grid.arrange(grobs = img_grobs, ncol = 3)
ggsave("PlotsAndPictures/CombinedBasePlots.png",
       combined_plot)
```

```
base_dir <- "PlotsAndPictures/PCA_ScreePlots/"
img_paths <- c("NoPreprocessingScree.png","PreprocessingScree.png")
img_paths <- file.path(base_dir,img_paths)

img_grobs <- lapply(img_paths, function(img){
  rasterGrob(readPNG(img),interpolate = T)
})

combined_plot <- grid.arrange(grobs = img_grobs, ncol = 2)
ggsave("PlotsAndPictures/CombinedScreePlots.png",
       combined_plot,
       width = 8,
```

```
       height= 4,
       dpi = 300)
```

```
base_dir <- "PlotsAndPictures/Faces/"
img_paths <- c("trainNoFeatureEngineering.png","trainWithFeatureEngineering.png")
img_paths <- file.path(base_dir,img_paths)

img_grobs <- lapply(img_paths, function(img){
  rasterGrob(readPNG(img),interpolate = T)
})

combined_plot <- grid.arrange(grobs = img_grobs, ncol = 2)
ggsave("PlotsAndPictures/Faces.png",
       combined_plot,
       width = 8,
       height= 4,
       dpi = 300)
```

```
df <- read_csv("Base_Models_Data/finalBaseModelReport.csv")
df <- df %>%
  select(-TP,-FP,-FN,-TN)
kable(df,format="html")
```

```
df <- read_csv("Stacked_Models_Data/finalStackedModelReport.csv")
df <- df %>%
  select(-TP,-FP,-FN,-TN)
kable(df,format="html")
```