Code Appendix

```r
#|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",
```

```r
            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,
                          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
```

```r
  # 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)
```

```
# 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)
```

```r
# 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)
```

```
# 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)
```