Supplementary Materials R Code

Jenny Xu

2025-05-26

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(ggplot2)
library(pheatmap)
library(caret)
```

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

```r
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```r
library(e1071)
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
```

```
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(class)
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'
##
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
##
## The following object is masked from 'package:base':
##
##     Recall
```

```r
library(yardstick)
```

```
##
## Attaching package: 'yardstick'
##
## The following objects are masked from 'package:caret':
##
##     precision, recall, sensitivity, specificity
##
## The following object is masked from 'package:readr':
##
##     spec
```

```r
library(dplyr)
library(pROC)
```

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

```r
library(scales)
```

```
##
## Attaching package: 'scales'
##
## The following object is masked from 'package:purrr':
##
##     discard
##
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

Exploratory Data Analysis

```
# Exploratory Data Analysis
# Read data
data = read.csv("Breast_GSE45827.csv")

# View shape and basic structure of the dataset
dim(data)
```

```
## [1]   151 54677
```

```
head(data[,1:10])
```

```
##   samples  type X1007_s_at X1053_at  X117_at  X121_at X1255_g_at X1294_at
## 1      84 basal   9.850040 8.097927 6.424728 7.353027   3.029122 6.880079
## 2      85 basal   9.861357 8.212222 7.062593 7.685578   3.149468 7.542283
## 3      87 basal  10.103478 8.936137 5.735970 7.687822   3.125931 6.562369
## 4      90 basal   9.756875 7.357148 6.479183 6.986624   3.181638 7.802344
## 5      91 basal   9.408330 7.746404 6.693980 7.333426   3.169923 7.610457
## 6      92 basal   7.505488 8.802820 6.235074 7.202227   2.987976 7.985281
##   X1316_at X1320_at
## 1 4.963740 4.408328
## 2 5.129607 4.584418
## 3 4.813449 4.425195
## 4 5.490982 4.567956
## 5 5.372469 4.424426
## 6 5.413368 4.465616
```

```
# Check the unique values and levels of outcome variable
table(data$type)
```

```
##
##     basal cell_line       HER luminal_A luminal_B    normal
##        41        14        30        29        30         7
```

```
unique(data$type)
```

```
## [1] "basal"     "HER"       "cell_line" "normal"    "luminal_A" "luminal_B"
```

```
# Check missing values -- there's no missing values in our dataset
sum(is.na(data))
```

```
## [1] 0
```

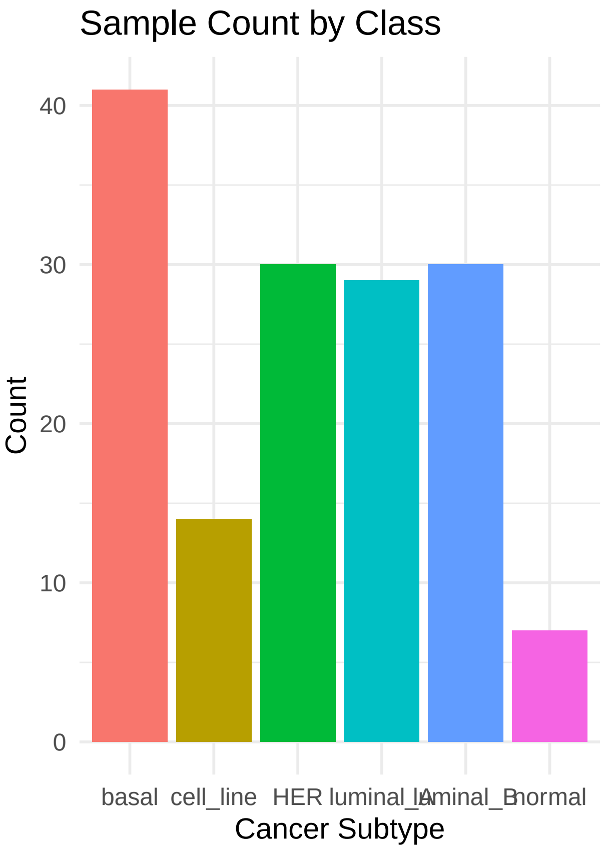
Dataset Composition

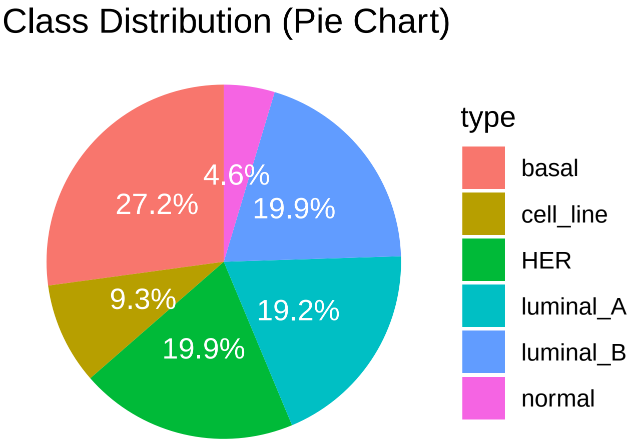
```
# Dataset Composition
# Count per class
type_counts <- data %>%
  count(type) %>%
  mutate(Percentage = n / sum(n) * 100)
```

```r
# Bar plot of class counts
bar_plot <- ggplot(type_counts, aes(x = type, y = n, fill = type)) +
  geom_bar(stat = "identity") +
  labs(title = "Sample Count by Class", x = "Cancer Subtype", y = "Count") +
  theme_minimal() +
  theme(legend.position = "none")

# Pie chart of class proportions
pie_plot <- ggplot(type_counts, aes(x = "", y = Percentage, fill = type)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar("y") +
  labs(title = "Class Distribution (Pie Chart)") +
  theme_void() +
  geom_text(aes(label = paste0(round(Percentage, 1), "%")),
            position = position_stack(vjust = 0.5), color = "white", size = 4)

# Combine the two plots using gridExtra
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:randomForest':
##
##     combine

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
grid.arrange(bar_plot, pie_plot, ncol = 2)
```





Descriptive Statistics by Gene

```
# Descriptive Statistics by Gene
# remove first 2 columns (samples and types)
gene_data = data[, -(1:2)]

gene_summary = tibble(
  Gene = colnames(gene_data),
  Mean = colMeans(gene_data, na.rm = TRUE),
  Median = apply(gene_data, 2, median, na.rm = TRUE),
  SD = apply(gene_data, 2, sd, na.rm = TRUE),
  Min = apply(gene_data, 2, min, na.rm = TRUE),
  Max = apply(gene_data, 2, max, na.rm = TRUE),
)

head(gene_summary)
```

```
## # A tibble: 6 x 6
##   Gene         Mean Median    SD   Min   Max
##   <chr>       <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 X1007_s_at 10.3    10.4  0.613  7.51 11.7
## 2 X1053_at    7.63    7.53 0.706  5.86  9.63
## 3 X117_at     6.22    6.24 0.645  4.76  8.36
## 4 X121_at     7.34    7.33 0.331  6.63  8.37
## 5 X1255_g_at  3.19    3.19 0.159  2.76  3.61
## 6 X1294_at    7.31    7.42 0.642  5.46  8.57
```

Descriptive Statistics by Sample

```r
# Compute summary statistics per row (i.e., per sample)
sample_summary <- tibble(
  SampleID = data$samples,
  Type     = data$type,
  Mean     = apply(gene_data, 1, mean, na.rm = TRUE),
  Median   = apply(gene_data, 1, median, na.rm = TRUE),
  SD       = apply(gene_data, 1, sd, na.rm = TRUE),
  Min      = apply(gene_data, 1, min, na.rm = TRUE),
  Max      = apply(gene_data, 1, max, na.rm = TRUE)
)

# Preview the result
head(sample_summary)
```

```
## # A tibble: 6 x 7
##   SampleID Type   Mean Median    SD   Min   Max
##      <int> <chr> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1       84 basal  5.66   5.17  2.16  2.44  14.8
## 2       85 basal  5.67   5.19  2.18  2.51  14.7
## 3       87 basal  5.65   5.15  2.17  2.34  14.7
## 4       90 basal  5.66   5.19  2.16  2.41  14.7
## 5       91 basal  5.66   5.19  2.17  2.46  14.7
## 6       92 basal  5.67   5.20  2.16  2.38  14.7
```

Descriptive Statistics by Breast Cancer Subtype

```r
# Pivot to long format
data_long = data %>%
  pivot_longer(cols = -(samples:type), names_to = "Gene", values_to = "Expression")

# Compute stats by class and gene
grouped_summary = data_long %>%
  group_by(type, Gene) %>%
  summarise(
    Mean = mean(Expression, na.rm = TRUE),
    Median = median(Expression, na.rm = TRUE),
    SD = sd(Expression, na.rm = TRUE),
    .groups = "drop"
  )

# Preview
head(grouped_summary)
```

```
## # A tibble: 6 x 5
##   type  Gene              Mean Median    SD
##   <chr> <chr>            <dbl>  <dbl> <dbl>
## 1 HER   AFFX.BioB.3_at    7.70   7.62 0.257
## 2 HER   AFFX.BioB.5_at    7.85   7.72 0.334
## 3 HER   AFFX.BioB.M_at    8.36   8.32 0.265
## 4 HER   AFFX.BioC.3_at    9.49   9.43 0.279
## 5 HER   AFFX.BioC.5_at    9.18   9.12 0.233
## 6 HER   AFFX.BioDn.3_at 11.8    11.8  0.217
```

Overall Descriptive Statistics

```r
# Overall Descriptive Statistics
# Flatten to a vector
all_values = unlist(gene_data, use.names = FALSE)

overall_summary = tibble(
  # Gene = colnames(gene_data),
  Mean = mean(all_values, na.rm = TRUE),
  Median = median(all_values, na.rm = TRUE),
  SD = sd(all_values, na.rm = TRUE),
  Min = min(all_values, na.rm = TRUE),
  Max = max(all_values, na.rm = TRUE),
)

head(overall_summary)
```

```
## # A tibble: 1 x 5
##    Mean Median    SD   Min   Max
##   <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1  5.65   5.17  2.13  2.17  15.0
```

```r
print(paste("Mean:", overall_summary$Mean))
```

```
## [1] "Mean: 5.65347002674015"
```

```r
print(paste("Median:", overall_summary$Median))
```

```
## [1] "Median: 5.17401561951104"
```

```r
print(paste("Standard Deviation:", overall_summary$SD))
```

```
## [1] "Standard Deviation: 2.12764795656006"
```

```r
print(paste("Min:", overall_summary$Min))
```

```
## [1] "Min: 2.17109972663613"
```

```r
print(paste("Max:", overall_summary$Max))
```

```
## [1] "Max: 14.9701002149474"
```

Max/Min of Mean Gene Expression

```r
# Sample with highest mean expression
max_mean_sample <- sample_summary %>%
  filter(Mean == max(Mean, na.rm = TRUE))

# Sample with lowest mean expression
min_mean_sample <- sample_summary %>%
  filter(Mean == min(Mean, na.rm = TRUE))

# Print results
cat("Sample with MAX mean expression:\n")
```

```
## Sample with MAX mean expression:
```

```r
print(max_mean_sample)
```

```
## # A tibble: 1 x 7
##   SampleID Type        Mean Median    SD   Min   Max
##      <int> <chr>      <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1      183 luminal_B  5.68   5.18  2.19  2.45  14.7
```

```r
cat("\nSample with MIN mean expression:\n")
```

```
##
## Sample with MIN mean expression:
```

```r
print(min_mean_sample)
```

```
## # A tibble: 1 x 7
##   SampleID Type        Mean Median    SD   Min   Max
##      <int> <chr>      <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1      220 luminal_A  5.59   5.15  2.05  2.44  14.7
```

Max/Min of Absolute Gene Expression

```r
# Sample with highest expression
max_sample <- sample_summary %>%
  filter(Max == max(Max, na.rm = TRUE))

# Sample with lowest expression
min_sample <- sample_summary %>%
  filter(Min == min(Min, na.rm = TRUE))

# Print results
cat("Sample with MAX mean expression:\n")
```

```
## Sample with MAX mean expression:
```

```r
print(min_sample)
```

```
## # A tibble: 1 x 7
##   SampleID Type   Mean Median    SD   Min   Max
##      <int> <chr> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1      147 basal  5.64   5.19  2.08  2.17  14.8
```

```r
cat("\nSample with MIN mean expression:\n")
```

```
##
## Sample with MIN mean expression:
```

```r
print(max_sample)
```

```
## # A tibble: 1 x 7
##   SampleID Type   Mean Median    SD   Min   Max
##      <int> <chr> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1      151 HER    5.66   5.17  2.16  2.37  15.0
```

Frequency distribution of Gene Expression - Why we need normalization/scaling

```r
# Distribution of Gene Expression - Why we need normalization/scaling
all_values = unlist(gene_data, use.names = FALSE)
```

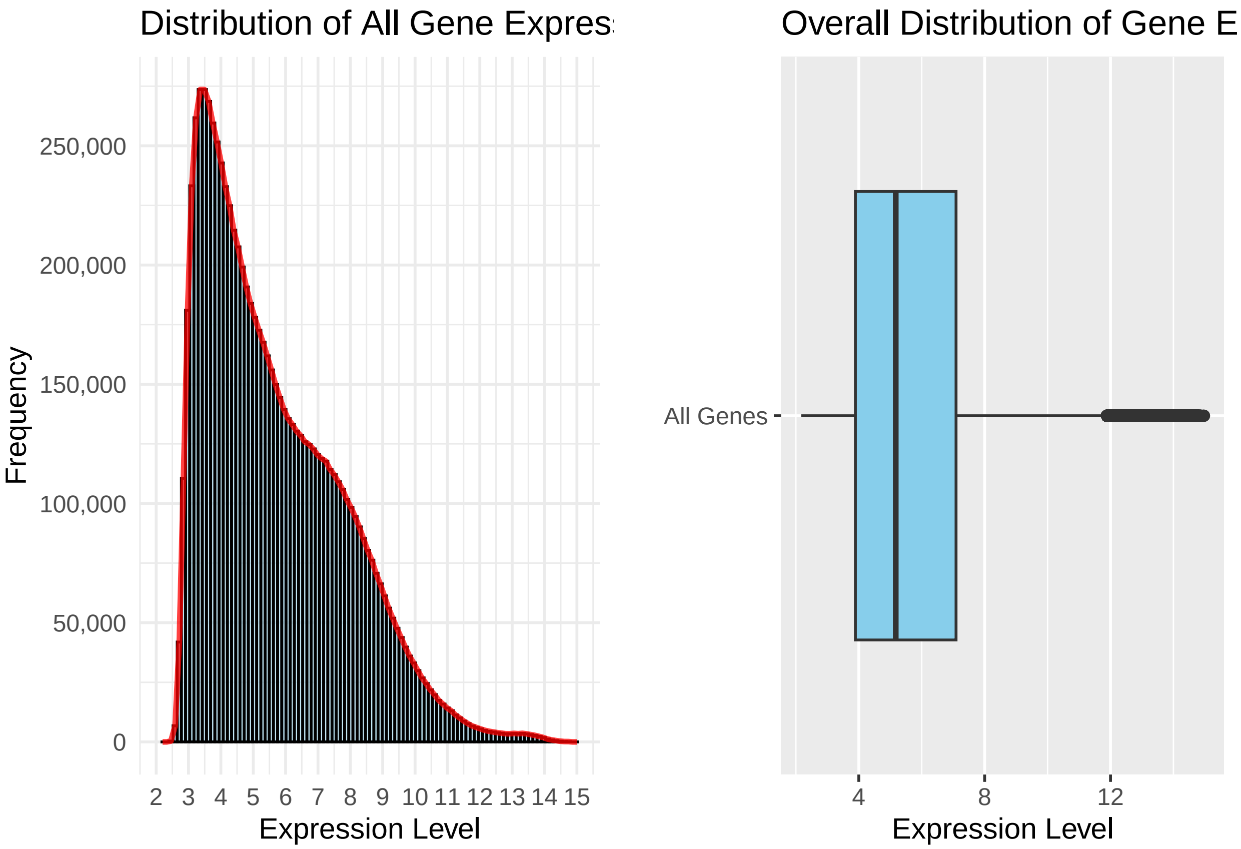
```r
expression_df = tibble(Expression = all_values)

frequency = ggplot(expression_df, aes(x = Expression)) +
  geom_histogram(bins = 100, fill = "skyblue", color = "black", alpha = 0.5) +
  geom_line(
    stat = "bin",
    bins = 100,
    aes(y = ..count..),
    color = "red",
    size = 1,
    alpha = 0.75
  ) +
  labs(title = "Distribution of All Gene Expression Values",
       x = "Expression Level", y = "Frequency") +
  scale_x_continuous(breaks = seq(0, 20, by = 1)) +
  scale_y_continuous(
    labels = scales::label_comma(),
    breaks = seq(0, 500000, by = 50000)
  ) +
  theme_minimal()
```

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

Boxplot of Gene Expression - Why we need normalization/scaling

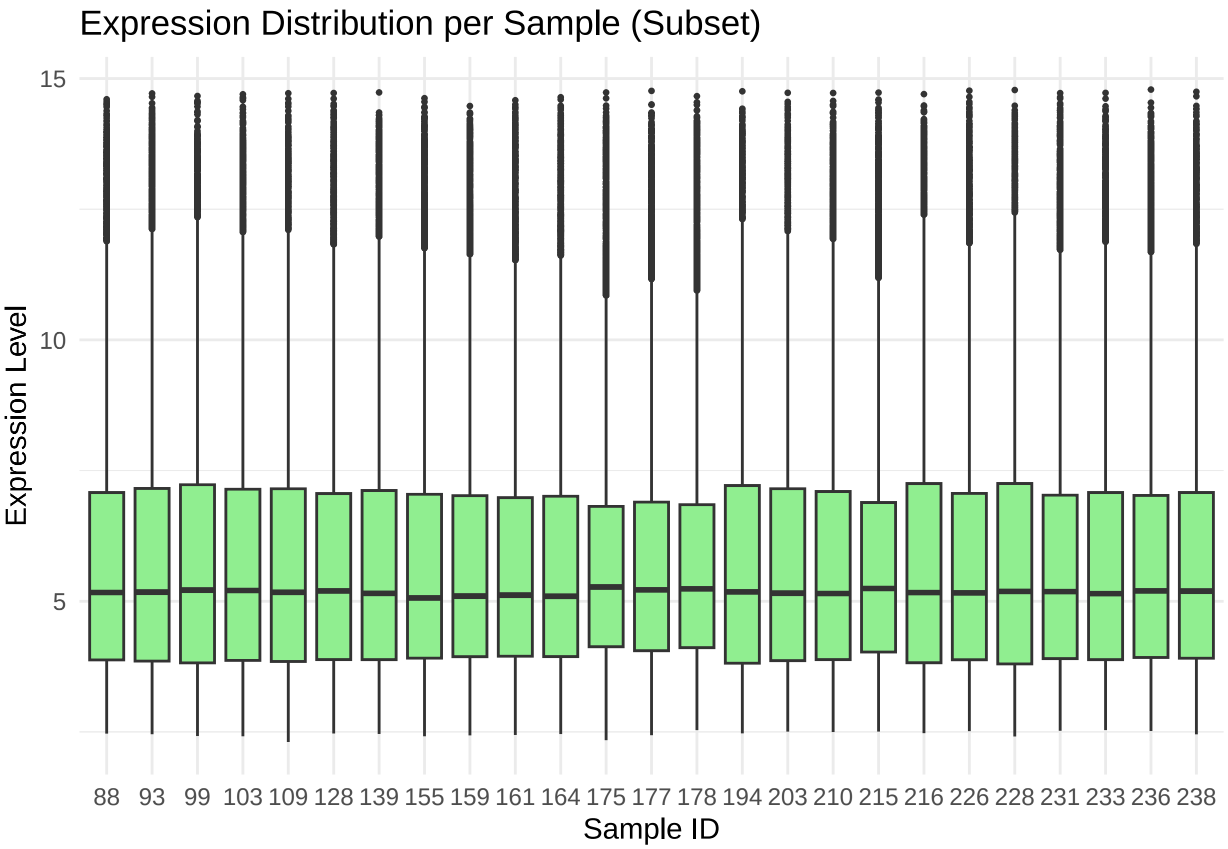
```r
# Boxplot of Gene Expression - Why we need normalization/scaling
# Prepare data: expression as y-axis
expression_df = tibble(Expression = unlist(data[ , -(1:2)], use.names = FALSE))

# Add dummy variable for plotting
expression_df$Group = "All Genes"

boxplot = ggplot(expression_df, aes(y = Group, x = Expression)) +
  geom_boxplot(fill = "skyblue") +
  labs(title = "Overall Distribution of Gene Expression Values",
       y = "", x = "Expression Level")

grid.arrange(frequency, boxplot, ncol = 2)
```

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



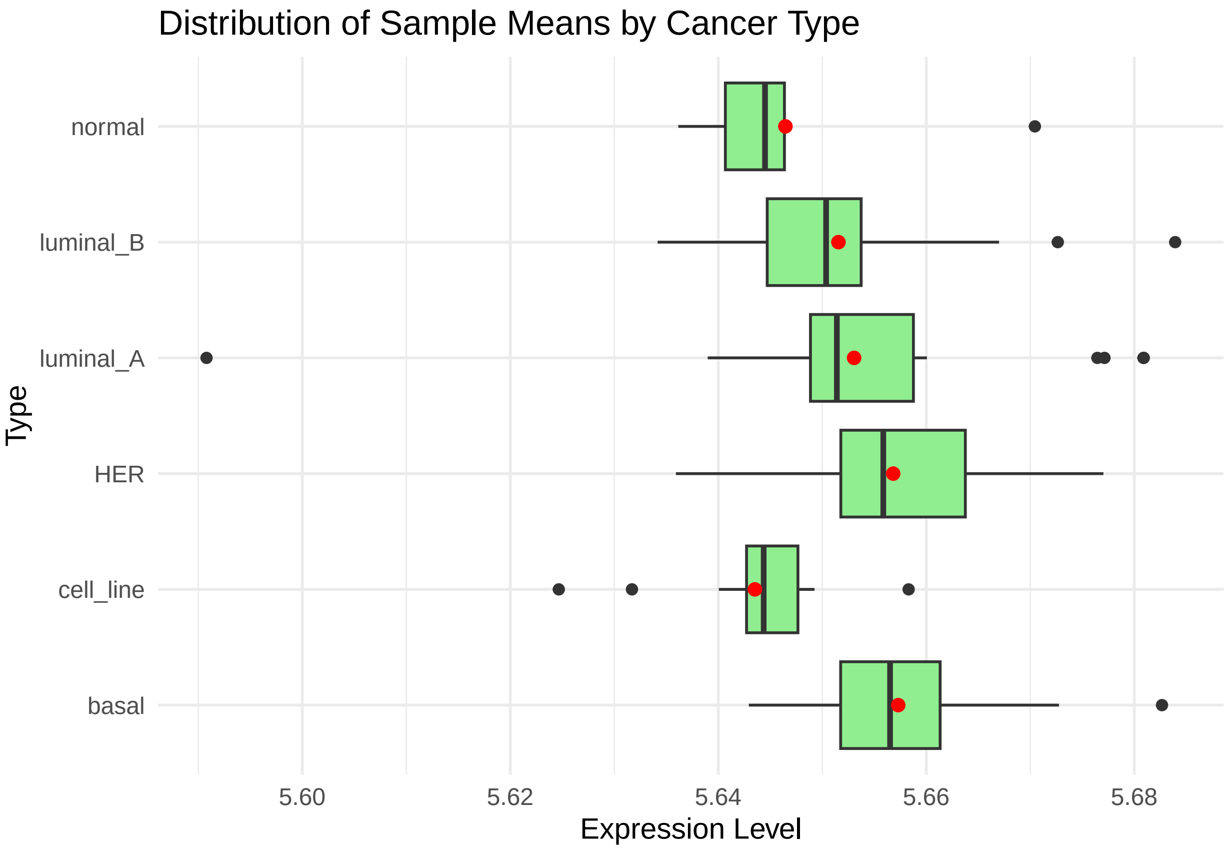
Boxplot of Gene Expression by Sample

```r
set.seed(123)
# Boxplot for a few random samples to keep it readable
sample_subset = sample(unique(data_long$samples), 25)
plot_data = data_long %>% filter(samples %in% sample_subset)

# plot_data = data_long %>% filter(samples %in% data_long$samples)

ggplot(plot_data, aes(x = factor(samples), y = Expression)) +
  geom_boxplot(outlier.size = 0.5, fill = "lightgreen") +
  labs(title = "Expression Distribution per Sample (Subset)",
       x = "Sample ID", y = "Expression Level") +
  theme_minimal()
```



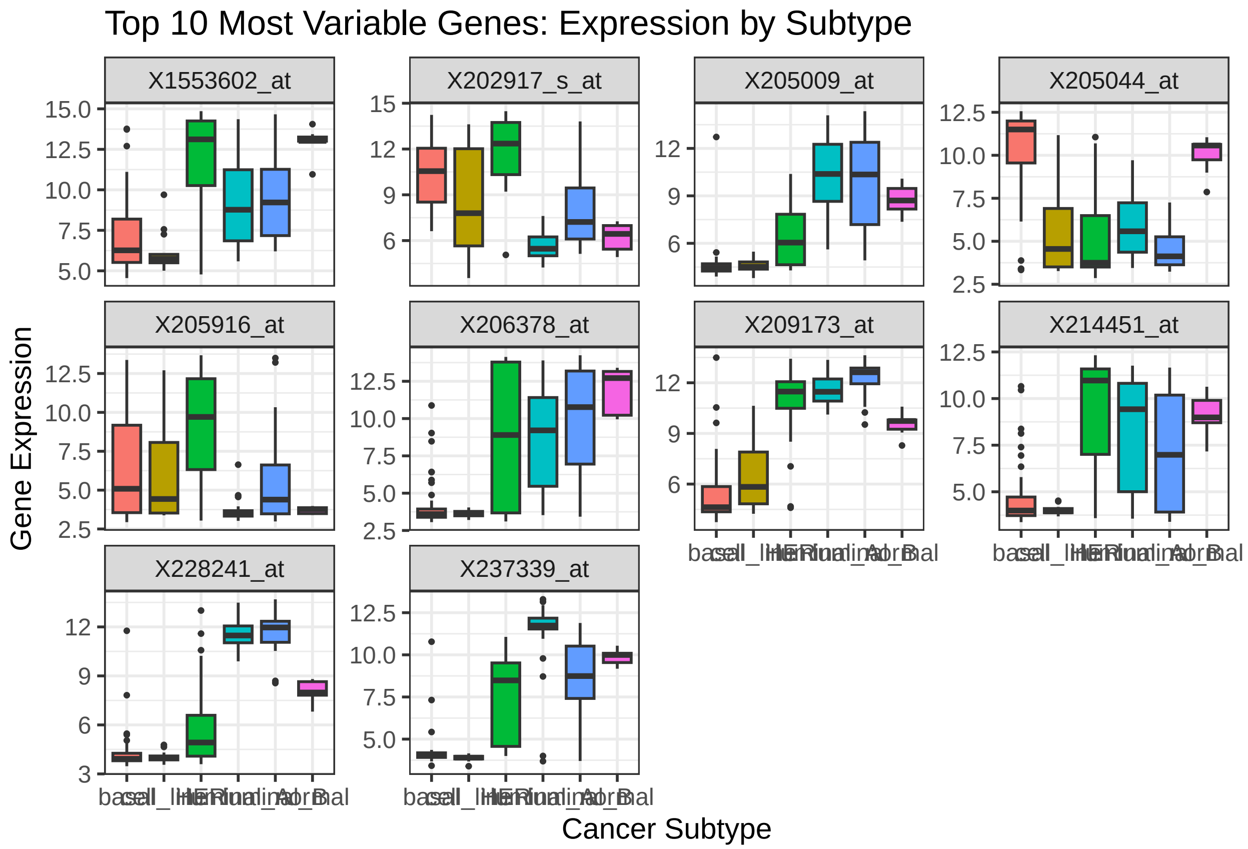
Boxplot of Gene Expression by Cancer Type

```r
ggplot(sample_summary, aes(y = Type, x = Mean)) +
  geom_boxplot(fill = "lightgreen") +
  stat_summary(fun = mean, geom = "point", shape = 20, size = 3, color = "red") +
  labs(title = "Distribution of Sample Means by Cancer Type",
       x = "Expression Level") +
  theme_minimal()
```



Top 10 Variable Genes - Do their gene expression differ by cancer subtype?

```
gene.variances = apply(gene_data, 2, var)
top.genes = sort(gene.variances, decreasing = TRUE)
top.10.genes.name = names(top.genes)[1:10]

# Create a long-format dataframe for plotting
plot_data = lapply(top.10.genes.name, function(gene) {
  data.frame(
    Gene = gene,
    Expression = gene_data[[gene]],
    Subtype = data$type
  )
}) %>% bind_rows()

# Create combined boxplot
ggplot(plot_data, aes(x = Subtype, y = Expression, fill = Subtype)) +
  geom_boxplot(outlier.size = 0.5) +
  facet_wrap(~ Gene, scales = "free_y") +
  labs(title = "Top 10 Most Variable Genes: Expression by Subtype",
       x = "Cancer Subtype",
       y = "Gene Expression") +
  theme_bw() +
  theme(legend.position = "none")
```



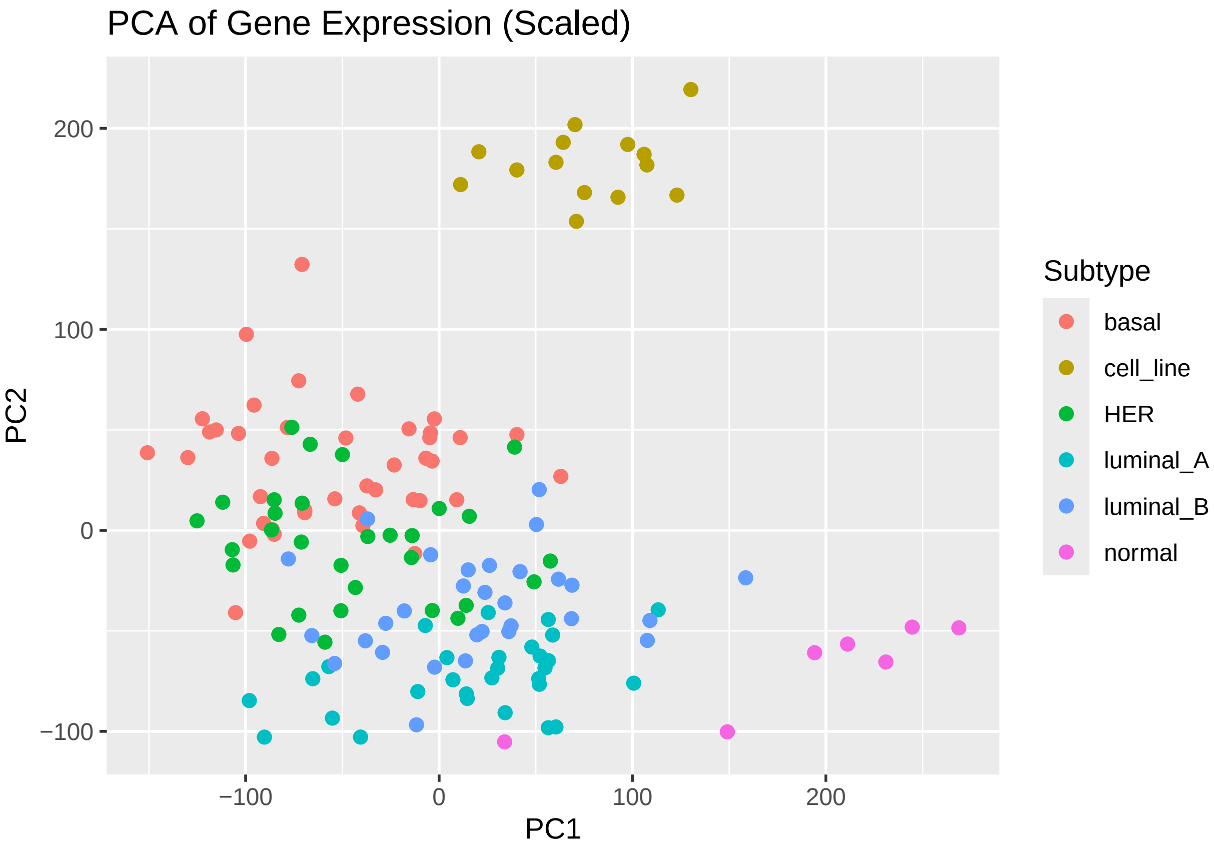
```
# Anova table
anova_results <- lapply(top.10.genes.name, function(gene) {
  model <- aov(gene_data[[gene]] ~ data$type)
  summary_df <- summary(model)[[1]]
  data.frame(
    Gene = gene,
    F_value = summary_df[["F value"]][1],
    p_value = summary_df[["Pr(>F)"]][1]
  )
}) %>% bind_rows()

# Print to console
print(anova_results)
```

```
##             Gene   F_value      p_value
## 1    X206378_at  21.58804 4.125586e-16
## 2    X228241_at 134.88529 1.053774e-52
## 3    X205916_at  12.04787 9.138208e-10
## 4    X237339_at  56.74219 1.909145e-32
## 5   X1553602_at  18.87886 1.989393e-14
## 6    X209173_at  86.16018 1.178749e-41
## 7    X214451_at  18.90565 1.912701e-14
## 8    X205009_at  44.16762 1.643301e-27
## 9    X205044_at  35.16204 1.948368e-23
## 10 X202917_s_at  29.47704 1.438317e-20
```

```
# for (gene in top.10.genes.name){
#   # New data frame
#   data.plot = data.frame(
#     expression = gene_data[[gene]],
#     subtype = data$type
#   )
#
#   # Boxplot of how gene expression differ by subtype for highly variable gene
#   print(
#     ggplot(data.plot, aes(x = subtype, y = expression, fill = subtype)) +
#       geom_boxplot() +
#       labs(title = paste("Expression of", gene, "by Subtype"),
#            x = "Subtype",
#            y = "Expression Level")
#   )
#   # One-way ANOVA for one top gene
#   print(summary(aov(expression~subtype, data = data.plot)))
# }
```

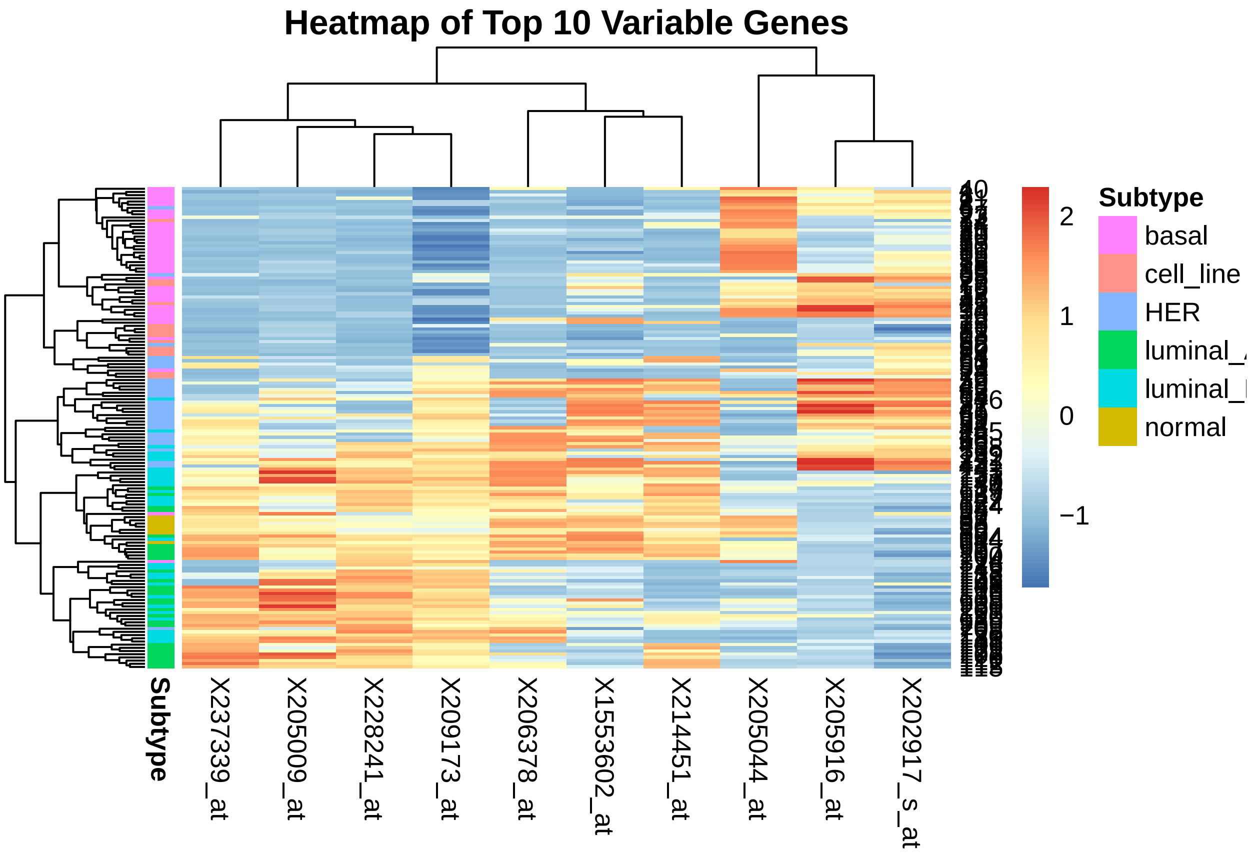
PCA Plot - Sample Cluster by Cancer Subtype

```
# PCA plot -- samples cluster by cancer subtype
# Standardize all genes (mean 0, sd 1)
scaled.data = scale(gene_data)
# PCA
pca.result = prcomp(scaled.data)
pca.plot = data.frame(
  PC1 = pca.result$x[,1],
  PC2 = pca.result$x[,2],
  Subtype = data$type
)
ggplot(pca.plot, aes(x = PC1, y = PC2, color = Subtype)) +
  geom_point(size = 2) +
  labs(title = "PCA of Gene Expression (Scaled)", x = "PC1", y = "PC2")
```



Heatmap - Sample Cluster by Subtype

```
# Heatmap of 10 top variable genes
top.10.expression = gene_data[,top.10.genes.name]
top.10.scaled = scale(top.10.expression)
rownames(top.10.scaled) = rownames(data)
row.annotation = data.frame(Subtype = data$type)
rownames(row.annotation) = rownames(top.10.scaled)

pheatmap(top.10.scaled,
         annotation_row = row.annotation,
         cluster_rows = TRUE,
         cluster_cols = TRUE,
         main = "Heatmap of Top 10 Variable Genes")
```



Methodology

Data Pre-processing

```
# Methodology (Pre-processing steps)
# Step 1: Split into Training and Test sets
set.seed(141)
idx <- createDataPartition(data$type, p = 0.80, list = FALSE)
train.data <- data[idx, ]
test.data  <- data[-idx, ]

table(test.data$type)
```

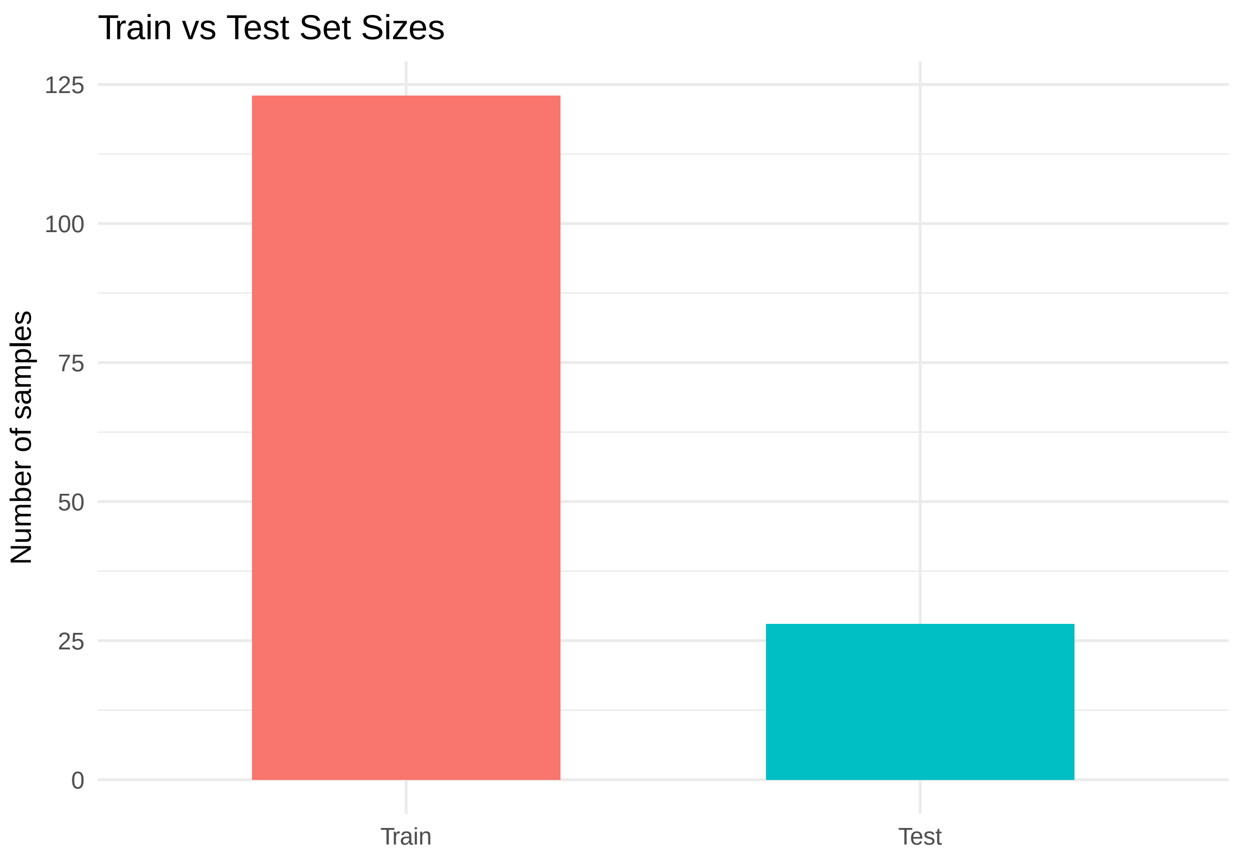
```
##
##     basal cell_line       HER luminal_A luminal_B    normal
##         8         2         6         5         6         1
```

```
# Step 2: Extract gene expression matrix and subtype labels
x.train.raw = train.data[,-(1:2)]
y.train = train.data$type
x.test.raw = test.data[,-(1:2)]
y.test = test.data$type

# Step 3: Scale training data only and test data using training parameters
scaled.train = scale(x.train.raw)
train.center = attr(scaled.train, "scaled:center")
```

```r
train.scale = attr(scaled.train, "scaled:scale")
scaled.test = scale(x.test.raw,
                    center = train.center,
                    scale = train.scale)

# Step 4: Handle class imbalance by upsampling scaled training data
y.train = factor(train.data$type)

train.upsampled = upSample(
  x = as.data.frame(scaled.train),
  y = y.train,
  yname = "target"
)

x.train.resampled = train.upsampled[, -ncol(train.upsampled)]
y.train.resampled = train.upsampled$target
```

80% Training vs. 20% Test Split (Before Upsampling)

```r
# Before upsamping, 80% Train set vs. 20% Test set
# build a tiny data frame
n_train = length(y.train)
n_test = length(y.test)
df <- data.frame(
  Set   = factor(c("Train","Test"), levels = c("Train","Test")),
  Count = c(n_train, n_test)
)

# plot
ggplot(df, aes(x = Set, y = Count, fill = Set)) +
  geom_col(width = 0.6) +               # geom_col is the same as geom_bar(stat="identity")
  labs(title = "Train vs Test Set Sizes",
       x     = NULL,
       y     = "Number of samples") +
  theme_minimal() +
  theme(legend.position = "none")
```



Unbalanced vs. Balanced Class (After Upsampling)

```
# After upsampling, balanced class in training set
# train.upsampled$target is the factor of up-sampled labels
table(train.upsampled$target)
```

```
##
##     basal cell_line       HER luminal_A luminal_B    normal
##        33        33        33        33        33        33
```

```
# class counts in your up-sampled test set
table(y.train.resampled)
```

```
## y.train.resampled
##     basal cell_line       HER luminal_A luminal_B    normal
##        33        33        33        33        33        33
```

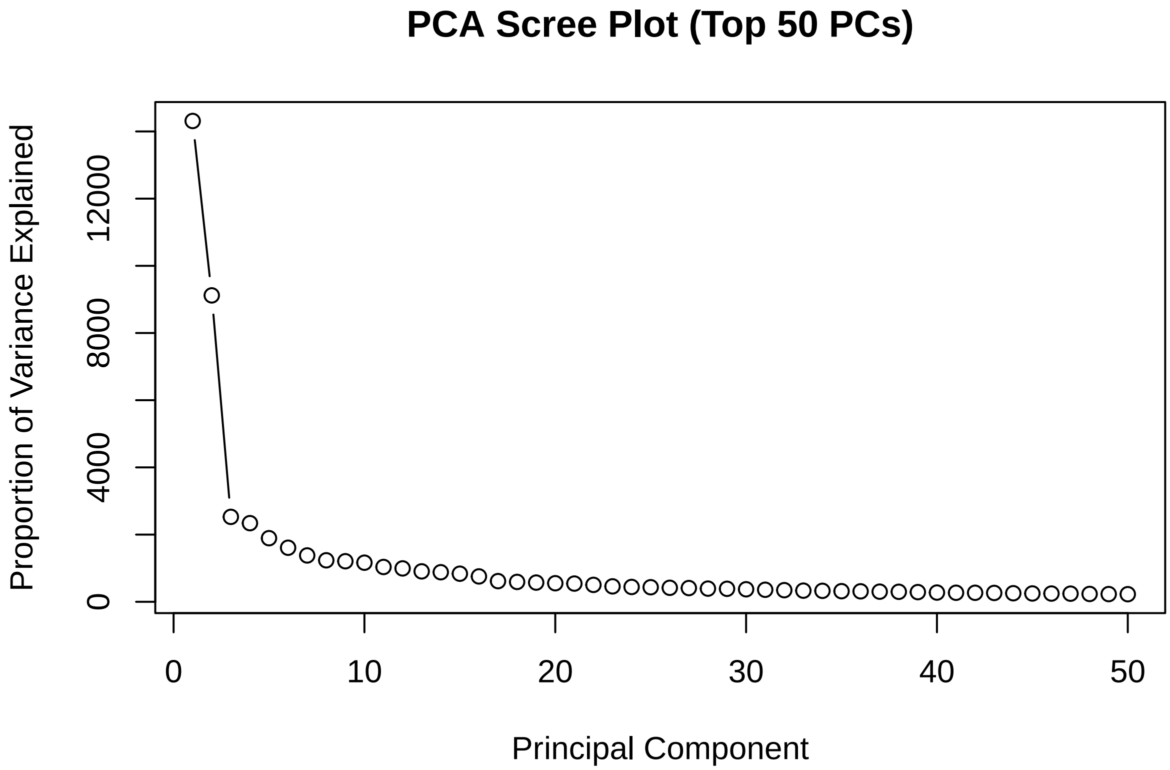
```
table(test.data$type)
```

```
##
##     basal cell_line       HER luminal_A luminal_B    normal
##         8         2         6         5         6         1
```

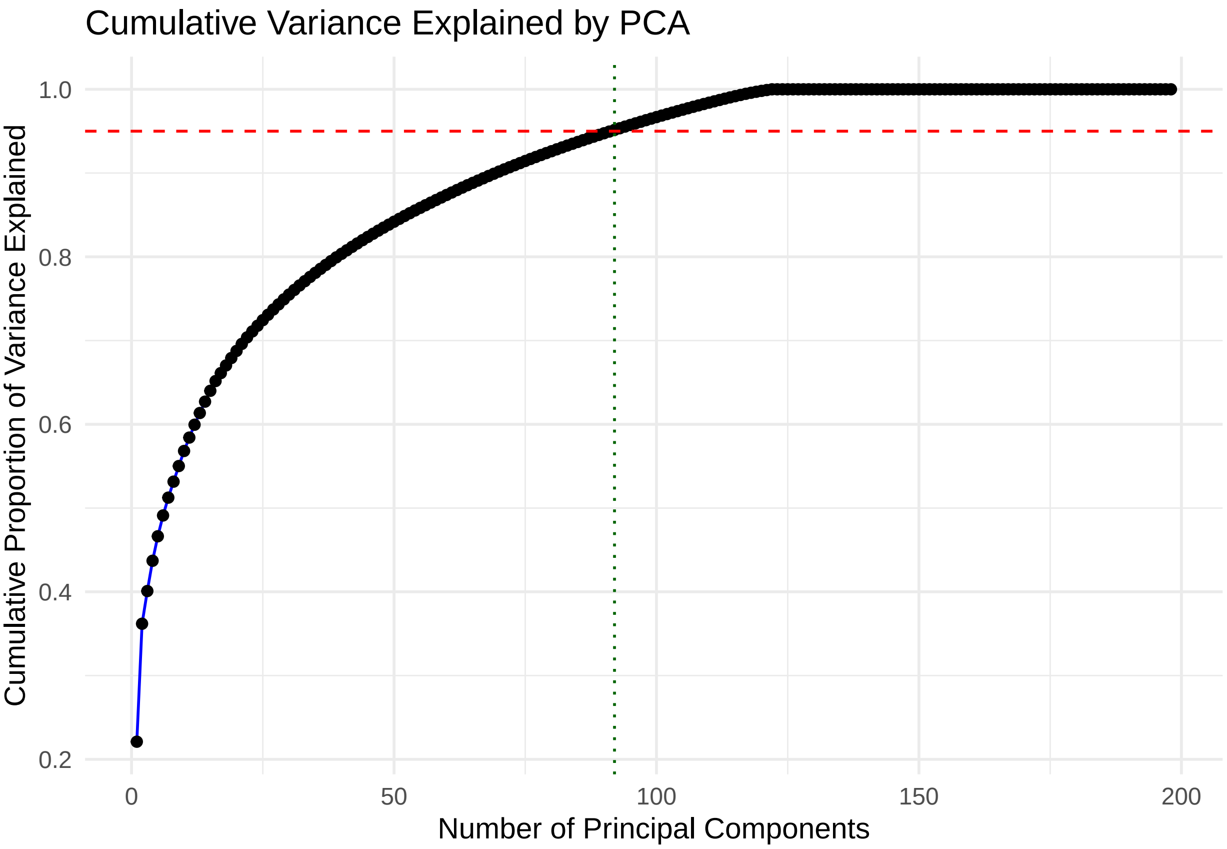
PCA

```
# Methodology (PCA)
# Step 1: Apply PCA to scaled training data
pca.model = prcomp(x.train.resampled, center = FALSE, scale. = FALSE)
```

```r
# Step 2: Examine variance explained by each component
explained.variance = pca.model$sdev^2
proportion.variance = explained.variance/sum(explained.variance)
cumulative.variance = cumsum(proportion.variance)

# Step 3: Find number of PCs that explain more than 95% of variance
num.pc.95 = which(cumulative.variance >= 0.95)[1]

# Step 4: Create PCA-transformed training data
pca.train = as.data.frame(pca.model$x[,1:num.pc.95])
pca.train$Subtype = y.train.resampled

# Step 5: Apply same PCA to test data
pca.test.raw = predict(pca.model, newdata = scaled.test)
pca.test = as.data.frame(pca.test.raw[,1:num.pc.95])

# Step 6: Scree plot: proportion of variance
plot(explained.variance[1:50], type = "b",
     xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     main = "PCA Scree Plot (Top 50 PCs)")
```



```r
# Step 7: Cumulative variance plot
scree.data = data.frame(
  PC = 1:length(cumulative.variance),
  CumulativeVariance = cumulative.variance
)

ggplot(scree.data, aes(x = PC, y = CumulativeVariance)) +
  geom_line(color = "blue") +
  geom_point() +
  geom_hline(yintercept = 0.95, linetype = "dashed", color = "red") +
  geom_vline(xintercept = num.pc.95, linetype = "dotted", color = "darkgreen") +
  theme_minimal() +
  labs(
    title = "Cumulative Variance Explained by PCA",
    x = "Number of Principal Components",
    y = "Cumulative Proportion of Variance Explained"
  )
```



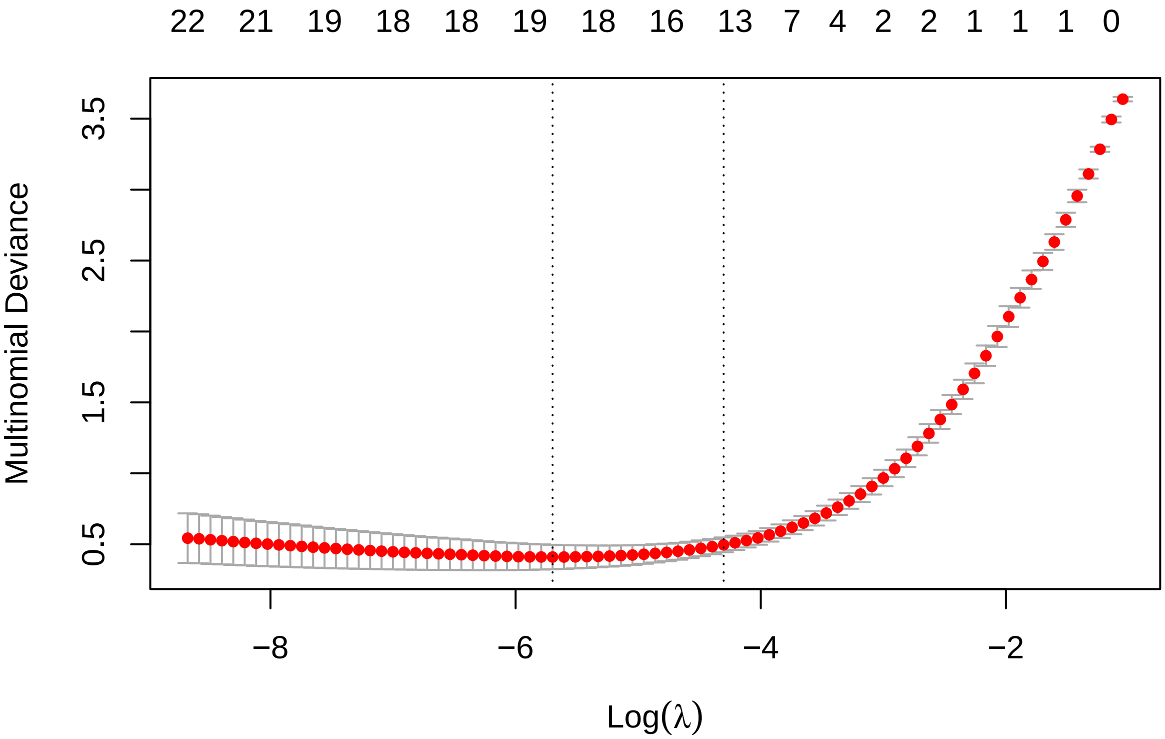
```
# What cumulative variance do those PCs explain?
cat("Cumulative variance at PC", num.pc.95, ":",
    round(cumulative.variance[num.pc.95], 4), "\n")
```

```
## Cumulative variance at PC 92 : 0.9515
```

Lasso

```
# Methodology (Lasso)
# Step 1: Prepare training and test data in matrix form
x.lasso.train = as.matrix(pca.train[,-ncol(pca.train)])
y.lasso.train = pca.train$Subtype

x.lasso.test = as.matrix(pca.test)

# Step 2: Fit Lasso Model
```

```r
lasso.model = glmnet(x.lasso.train, y.lasso.train,
                     alpha = 1, family = "multinomial")

# Step 3: Use Cross-validation to find optimal lambda
set.seed(141)
cv.lasso = cv.glmnet(x.lasso.train, y.lasso.train,
                     alpha = 1,family = "multinomial")
plot(cv.lasso)
```



```r
best.lambda = cv.lasso$lambda.min

# Step 4: Predict on test data
lasso.pred = predict(cv.lasso, newx = x.lasso.test,
                     s = best.lambda, type = "class")

# Step 5: Evaluate Model Performance
table(Predicted = lasso.pred, Actual = test.data$type)
```

```
##             Actual
## Predicted   basal cell_line HER luminal_A luminal_B normal
##   basal         8         0   0         0         0      0
##   cell_line     0         2   0         0         0      0
##   HER           0         0   6         0         1      0
##   luminal_A     0         0   0         5         1      0
##   luminal_B     0         0   0         0         4      0
##   normal        0         0   0         0         0      1
```

```r
lasso.accruacy = mean(lasso.pred == test.data$type)
lasso.accruacy
```

```
## [1] 0.9285714
```

Which gene is most predictive for each breast cancer subtype using Lasso?

```
# Which gene is most predictive for each breast cancer subtype using Lasso?
# No scaling of raw data, just upsampling to avoid bias
# Step 1: Upsample raw data
set.seed(141)
train.raw.upsampled = upSample(
  x = x.train.raw,
  y = factor(y.train),
  yname = "target"
)

# Step 2: Prepare training and test data for Lasso
x.raw.lasso.train = as.matrix(train.raw.upsampled[, -ncol(train.raw.upsampled)])
y.raw.lasso.train = train.raw.upsampled$target

# Step 3: Fit lasso on raw gene data
lasso.raw.model = glmnet(x.raw.lasso.train, y.raw.lasso.train,
                         alpha = 1, family = "multinomial")

# Step 4: Cross-validation for lambda
cv.raw.lasso = cv.glmnet(x.raw.lasso.train, y.raw.lasso.train,
                         alpha = 1, family = "multinomial")
best.raw.lambda = cv.raw.lasso$lambda.min

# Step 5: Extract coefficients at best lambda
coef.raw = coef(cv.raw.lasso, s = best.raw.lambda)

# Step 6: Find top predictive genes per subtype
top.genes.by.subtype = list()

for (subtype in names(coef.raw)){
  coef.matrix = coef.raw[[subtype]]
  nonzero.idx = which(coef.matrix !=0)[-1]

  if (length(nonzero.idx) >0){
    gene.name = rownames(coef.matrix)[nonzero.idx]
    abs.coef = abs(as.vector(coef.matrix[nonzero.idx]))
    names(abs.coef) = gene.name

    top.genes = sort(abs.coef, decreasing = TRUE)
    top.genes.by.subtype[[subtype]] = top.genes
  }else{
    top.genes.by.subtype[[subtype]] = NULL
    }
}

top.genes.by.subtype
```

```
## $basal
##   X1553315_at  X241044_x_at    X233730_at  X230226_s_at    X231374_at
##   0.644895206   0.486732596   0.460282225   0.329445630   0.279968846
##    X204580_at    X232440_at  X208358_s_at    X237246_at X1557809_a_at
##   0.279843949   0.228475730   0.202975141   0.174167453   0.172691455
##    X205376_at X1564676_a_at    X220612_at    X214772_at    X238865_at
##   0.150992785   0.149102300   0.141474729   0.133711273   0.126005714
##    X223315_at  X222457_s_at   X1554840_at    X233405_at    X227349_at
##   0.116169825   0.115616115   0.091852214   0.089881590   0.079107452
## X1568574_x_at    X205143_at X1553989_a_at    X226206_at    X208154_at
##   0.071489789   0.055545286   0.040303851   0.039905312   0.038286879
##    X205549_at  X205029_s_at
##   0.007430235   0.001241784
##
## $cell_line
## X202878_s_at X201721_s_at   X211990_at X207365_x_at  X1569041_at   X214722_at
##   0.32624585   0.23698968   0.21831347   0.20950763   0.08842685   0.07470837
##   X235847_at   X217757_at   X230332_at X210809_s_at   X219926_at
##   0.05808671   0.05013165   0.02278132   0.02077457   0.01994783
##
## $HER
##    X241884_at   X1565819_at  X210930_s_at    X236522_at   X1556923_at
##    1.43625764    1.07760441    0.99043397    0.40609026    0.21526034
##    X206793_at    X244162_at   X1557758_at X1552590_a_at X1560556_a_at
##    0.16814060    0.14644032    0.14067245    0.13317818    0.11574505
##    X229306_at    X229194_at    X242275_at    X215802_at  X207284_s_at
##    0.06348842    0.05663565    0.05411003    0.05172053    0.04515138
##  X204915_s_at X1554712_a_at  X216917_s_at
##    0.04467974    0.03588161    0.03075034
##
## $luminal_A
##   X215014_at   X238625_at X201235_s_at   X229160_at   X243605_at   X206638_at
##   0.52484685   0.36448613   0.34387789   0.32474664   0.23392198   0.22426145
## X223721_s_at   X227742_at   X215856_at   X233977_at X231002_s_at   X227182_at
##   0.19448390   0.17139572   0.14332353   0.11030388   0.10366811   0.09468187
##   X209123_at   X218613_at   X229110_at X229461_x_at X202174_s_at X205908_s_at
##   0.06338289   0.04710698   0.03878650   0.03428616   0.03411821   0.02759677
##   X243334_at   X242301_at
##   0.01911009   0.01581751
##
## $luminal_B
## X205477_s_at   X228405_at  X1556654_at  X78047_s_at   X217724_at X221836_s_at
## 0.9434511886 0.6873265248 0.5970686385 0.5945719647 0.5288954543 0.5191250060
##   X225090_at   X217351_at   X221811_at   X239278_at   X239612_at   X226727_at
## 0.4560141897 0.4453464140 0.4446205967 0.3335340055 0.3141100145 0.2382750585
##   X219401_at   X234046_at X234927_s_at X211712_s_at X219051_x_at   X203623_at
## 0.2155730559 0.1441536298 0.1132527216 0.1065677388 0.0692334567 0.0672823694
##   X214858_at X225350_s_at   X206107_at   X220148_at   X204378_at
## 0.0642107162 0.0486018096 0.0159176783 0.0056914940 0.0000139867
##
## $normal
##  X231598_x_at    X242641_at X1552509_a_at  X243689_s_at    X218872_at
##    0.67650609    0.57641822    0.49052294    0.28673783    0.22638400
##  X206093_x_at   X1561754_at    X211565_at
##    0.13479423    0.12250978    0.08008514
```

SVM

```r
# Methodology (SVM)
# Step 1: Prepare training and test data from PCA transformed data
x.svm.train = as.matrix(pca.train[, -ncol(pca.train)])
y.svm.train = pca.train$Subtype

x.svm.test = as.matrix(pca.test)

# Step 2: Train SVM Model
set.seed(141)
svm.model = svm(
  x = x.svm.train,
  y = y.svm.train,
  kernel = "linear",
  probability = TRUE
)

# Step 3: Predict on Test data
svm.pred = predict(svm.model, newdata = x.svm.test)

# Step 4: Evaluate Accruacy
table(Predicted = svm.pred, Actual = test.data$type)
```

```
##            Actual
## Predicted   basal cell_line HER luminal_A luminal_B normal
##   basal         8         0   0         0         0      0
##   cell_line     0         2   0         0         0      0
##   HER           0         0   6         0         0      0
##   luminal_A     0         0   0         5         2      0
##   luminal_B     0         0   0         0         4      0
##   normal        0         0   0         0         0      1
```

```r
svm.accruacy = mean(svm.pred == test.data$type)
svm.accruacy
```

```
## [1] 0.9285714
```

Decision Tree

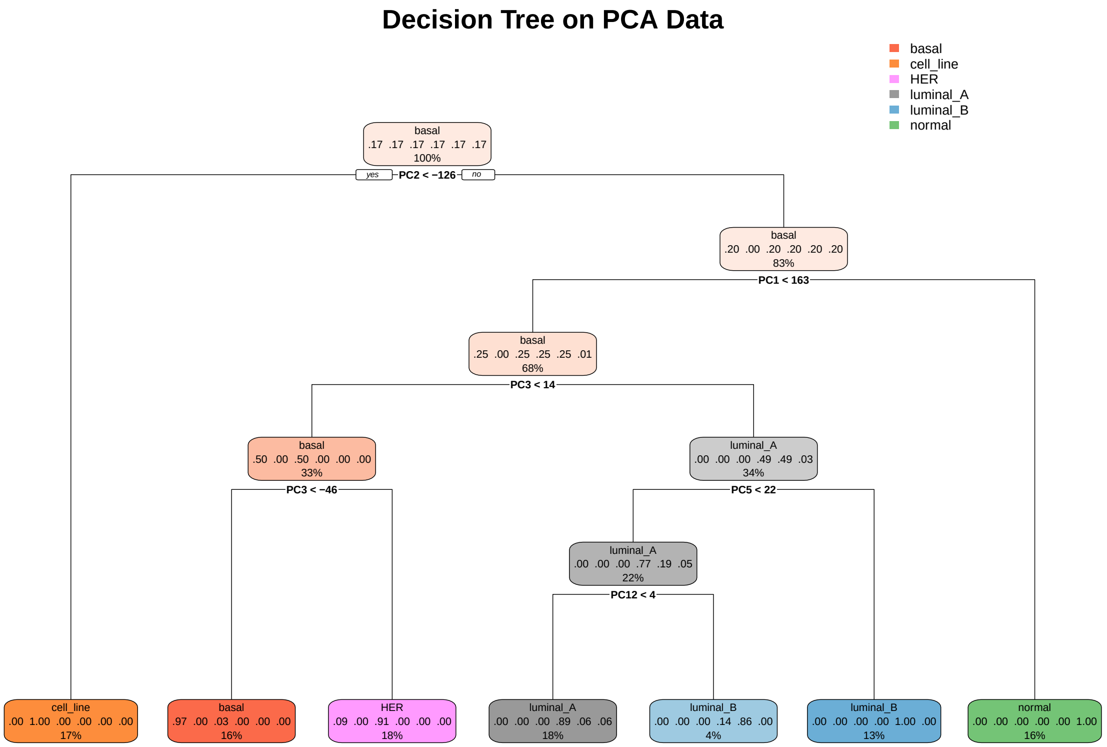
```r
# Methodology (Decision Tree)
# Step 1: Prepare training and test data from PCA transformed data
tree.train.data = pca.train
tree.test.data = pca.test

# Step 3: Fit tree model
tree.model = rpart(Subtype~., data = tree.train.data, method = "class")

# Step 4: Predict on test data
tree.pred = predict(tree.model, newdata = tree.test.data, type = "class")

# Step 5: Confusion matrix + accuracy
table(Predicted = tree.pred, Actual = test.data$type)
```

```
##            Actual
## Predicted   basal cell_line HER luminal_A luminal_B normal
##   basal         8         0   0         0         0      0
##   cell_line     0         2   0         0         0      0
##   HER           0         0   6         0         0      0
##   luminal_A     0         0   0         4         0      0
##   luminal_B     0         0   0         1         6      0
##   normal        0         0   0         0         0      1
```

```
tree.accriacy = mean(tree.pred == test.data$type)
tree.accriacy
```

```
## [1] 0.9642857
```

```
# Step 6: Visualize the tree
rpart.plot(tree.model, main = "Decision Tree on PCA Data")
```



Random Forest

```
# Methodology (Random Forest)
# Step 1: Prepare training and test data from PCA transformed data
rf.train.data = pca.train
rf.test.data = pca.test

# Step 3: Fit random forest model
set.seed(123)
rf.model = randomForest(Subtype~., data = rf.train.data, ntree = 500)

# Step 4: Predict on Test data
rf.pred = predict(rf.model, newdata = rf.test.data)

# Step 5: Confusion matrix + accuracy
```

```
table(Predicted = rf.pred, Actual = test.data$type)
```

```
##            Actual
## Predicted   basal cell_line HER luminal_A luminal_B normal
##   basal         8         0   0         0         0      0
##   cell_line     0         2   0         0         0      0
##   HER           0         0   5         0         1      0
##   luminal_A     0         0   0         4         1      0
##   luminal_B     0         0   1         1         4      0
##   normal        0         0   0         0         0      1
```

```
rf.accuracy = mean(rf.pred == test.data$type)
rf.accuracy
```

```
## [1] 0.8571429
```

KNN

```
# Methodology (KNN)
# Step 1: Prepare training and test data from PCA transformed data
x.knn.train = as.matrix(pca.train[, -ncol(pca.train)])
y.knn.train = pca.train$Subtype

x.knn.test = as.matrix(pca.test)

# Step 2: fit KNN model (manually choose k)
set.seed(141)
knn.pred = knn(train = x.knn.train, test = x.knn.test,
               cl = y.knn.train, k = 5)

# Step 3: Confusion matrix + accuracy
table(Predicted = knn.pred, Actual = test.data$type)
```

```
##            Actual
## Predicted   basal cell_line HER luminal_A luminal_B normal
##   basal         7         0   1         0         0      0
##   cell_line     0         2   0         0         0      0
##   HER           1         0   2         0         0      0
##   luminal_A     0         0   3         5         5      0
##   luminal_B     0         0   0         0         1      0
##   normal        0         0   0         0         0      1
```

```
knn.accuracy = mean(knn.pred == test.data$type)
knn.accuracy
```

```
## [1] 0.6428571
```

Main Results

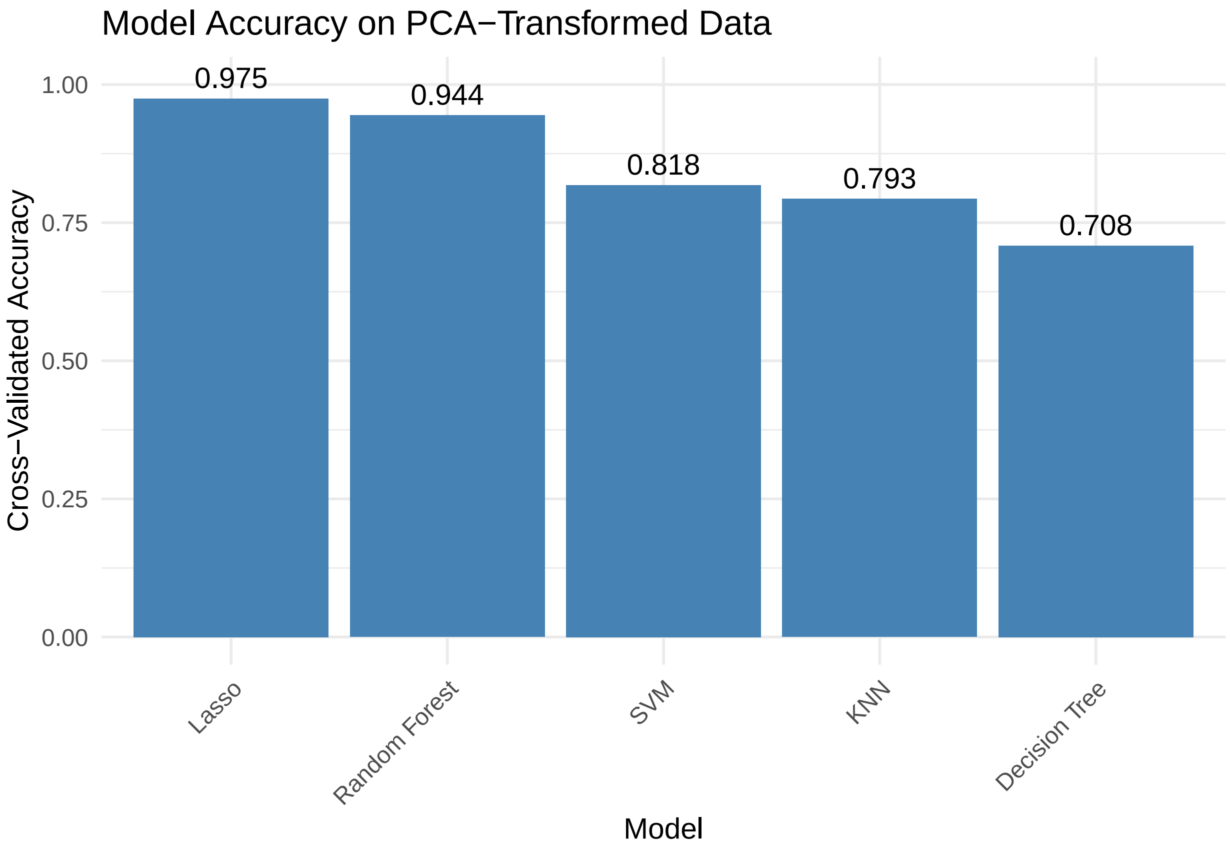
Accuracy

```
# Main Results (Model Comparison using Caret pacakage)
# Step 1: Extract features and target from PCA data
x.pca = pca.train[, -ncol(pca.train)]
```

```
y.pca = pca.train$Subtype

# Step 2: Define 5-fold CV and upsamling
ctrl <- trainControl(
  method   = "cv",
  number   = 5,
  sampling = "up"       # <-- this makes caret up-sample the minority classes in each fold
)

model.results.pca = list()

set.seed(141)
model.results.pca[["Random Forest"]] <- train(
  x         = x.pca,
  y         = y.pca,
  method    = "rf",
  trControl = ctrl
)


# Step 3: Train models using PCA data
set.seed(141)
model.results.pca = list()

model.results.pca[["Lasso"]] = train(
  x = x.pca, y = y.pca,
  method = "glmnet",
  trControl = ctrl,
  tuneLength = 10
)

model.results.pca[["SVM"]] = train(
  x = x.pca, y =y.pca,
  method = "svmLinear",
  trControl = ctrl
)

model.results.pca[["Decision Tree"]] = train(
  x = x.pca,y =y.pca,
  method = "rpart",
  trControl = ctrl
)

model.results.pca[["Random Forest"]] = train(
  x = x.pca,y =y.pca,
  method = "rf",
  trControl = ctrl
)

model.results.pca[["KNN"]] = train(
  x = x.pca,y =y.pca,
  method = "knn",
  trControl = ctrl,
  tuneLength = 5
)

# Step 4: Compare PCA-based model accuracy
model.accuracies.pca = sapply(model.results.pca,
                              function(m) max(m$results$Accuracy))
model.accuracies.pca
```

```
##         Lasso           SVM Decision Tree Random Forest           KNN
##     0.9747373     0.8178947     0.7082610     0.9444737     0.7930206
```

```
# Step 5: Plot
# Extract cross-validated accuracy from each caret model
model.accuracies.pca <- sapply(model.results.pca, function(m) {
  if ("Accuracy" %in% colnames(m$results)) {
    max(m$results$Accuracy)
  } else {
    NA  # Handle models like SVM (if probs or tuning failed)
  }
})

# Create data frame
accuracy_df <- data.frame(
  Model = names(model.accuracies.pca),
  Accuracy = as.numeric(model.accuracies.pca)
)

ggplot(accuracy_df, aes(x = reorder(Model, -Accuracy), y = Accuracy)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = round(Accuracy, 3)), vjust = -0.5, size = 4) +
  ylim(0, 1) +
  theme_minimal() +
  labs(
    title = "Model Accuracy on PCA-Transformed Data",
    x = "Model",
    y = "Cross-Validated Accuracy"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



F1 & F2 & ROC Curve/AUC Score

Random Forest

```
# Main Results (Model Evaluation: Random Forest)
# Step 1: Store predicted labels and true test labels
true.label = factor(test.data$type)
rf.prediction = predict(model.results.pca[["Random Forest"]], pca.test)

# Step 2: Create dataframe for evaluation
evaluate.data = data.frame(
  truth = true.label,
  prediction = factor(rf.prediction, levels = levels(true.label))
)

# Step 3: F1
f1.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 1)

# Step 4: F2
f2.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 2)

# Step 5: Probabilities for AUC
rf.probs = predict(model.results.pca[["Random Forest"]], pca.test, type = "prob")
roc.multiclass = multiclass.roc(response = true.label,
                                predictor = as.matrix(rf.probs))
```

```r
auc.value = auc(roc.multiclass)

# Step 5b: Compute per-class AUCs
class_levels <- colnames(rf.probs)

auc_values <- sapply(class_levels, function(cls) {
  # Build a binary response: 1 for this class, 0 otherwise
  bin_truth <- as.numeric(true.label == cls)
  # Compute ROC and then its AUC
  roc_obj <- roc(bin_truth, rf.probs[[cls]], quiet = TRUE)
  auc(roc_obj)
})

# Make a little table
auc_table <- data.frame(
  Class = class_levels,
  AUC   = round(auc_values, 3)
)
print(auc_table)
```

```
##                Class   AUC
## basal          basal 1.000
## cell_line cell_line 1.000
## HER              HER 1.000
## luminal_A luminal_A 0.991
## luminal_B luminal_B 0.985
## normal       normal 1.000
```

```r
# Step 6: Output
f1.macro
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.908
```

```r
f2.macro
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.910
```

```r
auc.value
```

```
## Multi-class area under the curve: 0.9967
```

```r
# Step 7: ROC Curve Plot
class_levels = colnames(rf.probs)

roc_df <- do.call(rbind, lapply(class_levels, function(class) {
  binary_response <- as.numeric(true.label == class)
    # Only compute ROC if both classes are present
  if (length(unique(binary_response)) < 2) {
    return(NULL)  # Skip this class
  }
  roc_obj <- roc(binary_response, rf.probs[[class]], quiet = TRUE)

  data.frame(
    fpr = 1 - roc_obj$specificities,
    tpr = roc_obj$sensitivities,
    class = class,
    auc = rep(auc(roc_obj), length(roc_obj$sensitivities))
  )
}))

random.forest = ggplot(roc_df, aes(x = fpr, y = tpr)) +
  geom_line(linewidth = 1.2, color = "steelblue") +
  geom_abline(linetype = "dashed", color = "gray") +
  facet_wrap(~ class, ncol = 3) +
  theme_minimal() +
  labs(
    title = "Test Set: One-vs-Rest ROC Curves for Each Class (Random Forest)",
    x = "False Positive Rate",
    y = "True Positive Rate"
  )
```

Lasso

```r
# Main Results (Model Evaluation: Lasso)
# Step 1: Store predicted labels and true test labels
true.label = factor(test.data$type)
lasso.prediction = predict(model.results.pca[["Lasso"]], pca.test)

# Step 2: Create dataframe for evaluation
evaluate.data = data.frame(
  truth = true.label,
  prediction = factor(lasso.prediction, levels = levels(true.label))
)

# Step 3: F1
f1.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 1)

# Step 4: F2
f2.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 2)

# Step 5: Probabilities for AUC
lasso.probs = predict(model.results.pca[["Lasso"]], pca.test, type = "prob")
roc.multiclass = multiclass.roc(response = true.label,
                                predictor = as.matrix(lasso.probs))
auc.value = auc(roc.multiclass)

# After Step 5, you have:
lasso.probs <- predict(model.results.pca[["Lasso"]], pca.test, type = "prob")
true.label  <- factor(test.data$type)

# Compute per-class AUCs:
```

```
class_levels <- colnames(lasso.probs)

auc_values <- sapply(class_levels, function(cls) {
  bin_truth <- as.numeric(true.label == cls)
  roc_obj   <- roc(bin_truth, lasso.probs[[cls]], quiet = TRUE)
  auc(roc_obj)
})

# Build and print a table of AUCs
auc_table <- data.frame(
  Class = class_levels,
  AUC   = round(auc_values, 3)
)
print(auc_table)
```

```
##               Class AUC
## basal         basal   1
## cell_line cell_line   1
## HER             HER   1
## luminal_A luminal_A   1
## luminal_B luminal_B   1
## normal       normal   1
```

```
# Step 6: Output
f1.macro
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.970
```

```
f2.macro
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.971
```

```
auc.value
```

```
## Multi-class area under the curve: 1
```

```
# Step 7: ROC Curve Plot
class_levels = colnames(lasso.probs)

roc_df <- do.call(rbind, lapply(class_levels, function(class) {
  binary_response <- as.numeric(true.label == class)
    # Only compute ROC if both classes are present
  if (length(unique(binary_response)) < 2) {
    return(NULL)  # Skip this class
  }

  roc_obj <- roc(binary_response, lasso.probs[[class]], quiet = TRUE)

  data.frame(
    fpr = 1 - roc_obj$specificities,
    tpr = roc_obj$sensitivities,
    class = class,
    auc = rep(auc(roc_obj), length(roc_obj$sensitivities))
  )
}))

lasso = ggplot(roc_df, aes(x = fpr, y = tpr)) +
  geom_line(linewidth = 1.2, color = "steelblue") +
  geom_abline(linetype = "dashed", color = "gray") +
  facet_wrap(~ class, ncol = 3) +
  theme_minimal() +
  labs(
    title = "Test Set: One-vs-Rest ROC Curves for Each Class (Lasso)",
    x = "False Positive Rate",
    y = "True Positive Rate"
  )

# Step 8: Confusion matrix
conf_matrix <- conf_mat(evaluate.data, truth = truth, estimate = prediction)

# Print the matrix
print(conf_matrix)
```

```
##             Truth
## Prediction  basal cell_line HER luminal_A luminal_B normal
##   basal         8         0   0         0         0      0
##   cell_line     0         2   0         0         0      0
##   HER           0         0   6         0         0      0
##   luminal_A     0         0   0         5         1      0
##   luminal_B     0         0   0         0         5      0
##   normal        0         0   0         0         0      1
```

Decision Tree

```
# Main Results (Model Evaluation: Decision Tree)
# Step 1: Store predicted labels and true test labels
true.label = factor(test.data$type)
tree.prediction = predict(model.results.pca[["Decision Tree"]], pca.test)

# Step 2: Create dataframe for evaluation
evaluate.data = data.frame(
  truth = true.label,
  prediction = factor(tree.prediction, levels = levels(true.label))
)

# Step 3: F1
f1.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 1)
```

```
## Warning: While computing multiclass `precision()`, some levels had no predicted events
## (i.e. `true_positive + false_positive = 0`).
## Precision is undefined in this case, and those levels will be removed from the
## averaged result.
## Note that the following number of true events actually occurred for each
## problematic event level:
```

```
## 'HER': 6, 'luminal_B': 6
```

```
# Step 4: F2
f2.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 2)
```

```
## Warning: While computing multiclass `precision()`, some levels had no predicted events
## (i.e. `true_positive + false_positive = 0`).
## Precision is undefined in this case, and those levels will be removed from the
## averaged result.
## Note that the following number of true events actually occurred for each
## problematic event level:
## 'HER': 6, 'luminal_B': 6
```

```
# Step 5: Probabilities for AUC
tree.probs = predict(model.results.pca[["Decision Tree"]], pca.test, type = "prob")
roc.multiclass = multiclass.roc(response = true.label,
                                predictor = as.matrix(tree.probs))
auc.value = auc(roc.multiclass)

# Step 6: Output
f1.macro
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.838
```

```
f2.macro
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.919
```

```
auc.value
```

```
## Multi-class area under the curve: 0.9333
```

```
# Step 7: ROC Curve Plot
class_levels = colnames(tree.probs)

roc_df <- do.call(rbind, lapply(class_levels, function(class) {
  binary_response <- as.numeric(true.label == class)
    # Only compute ROC if both classes are present
  if (length(unique(binary_response)) < 2) {
    return(NULL)  # Skip this class
  }

  roc_obj <- roc(binary_response, tree.probs[[class]], quiet = TRUE)

  data.frame(
    fpr = 1 - roc_obj$specificities,
    tpr = roc_obj$sensitivities,
    class = class,
    auc = rep(auc(roc_obj), length(roc_obj$sensitivities))
  )
}))
```

```r
decision.tree = ggplot(roc_df, aes(x = fpr, y = tpr)) +
  geom_line(linewidth = 1.2, color = "steelblue") +
  geom_abline(linetype = "dashed", color = "gray") +
  facet_wrap(~ class, ncol = 3) +
  theme_minimal() +
  labs(
    title = "Test Set: One-vs-Rest ROC Curves for Each Class (Decision Tree)",
    x = "False Positive Rate",
    y = "True Positive Rate"
  )
```

KNN

```r
# Main Results (Model Evaluation: KNN)
# Step 1: Store predicted labels and true test labels
true.label = factor(test.data$type)
knn.prediction = predict(model.results.pca[["KNN"]], pca.test)

# Step 2: Create dataframe for evaluation
evaluate.data = data.frame(
  truth = true.label,
  prediction = factor(knn.prediction, levels = levels(true.label))
)

# Step 3: F1
f1.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 1)

# Step 4: F2
f2.macro = f_meas(evaluate.data, truth = truth,
                  estimate = prediction, beta = 2)

# Step 5: Probabilities for AUC
knn.probs = predict(model.results.pca[["KNN"]], pca.test, type = "prob")
roc.multiclass = multiclass.roc(response = true.label,
                               predictor = as.matrix(knn.probs))
auc.value = auc(roc.multiclass)

# Step 6: Output
f1.macro
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.725
```

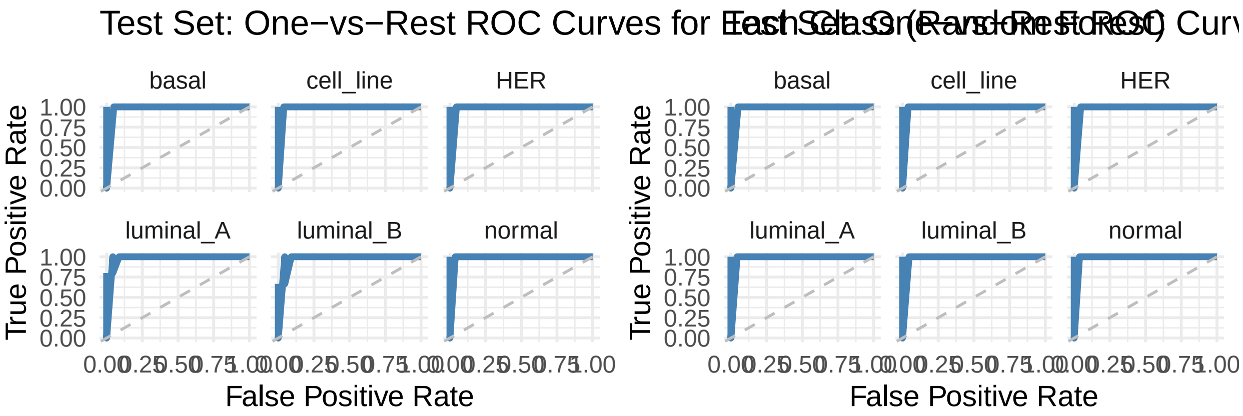
```r
f2.macro
```

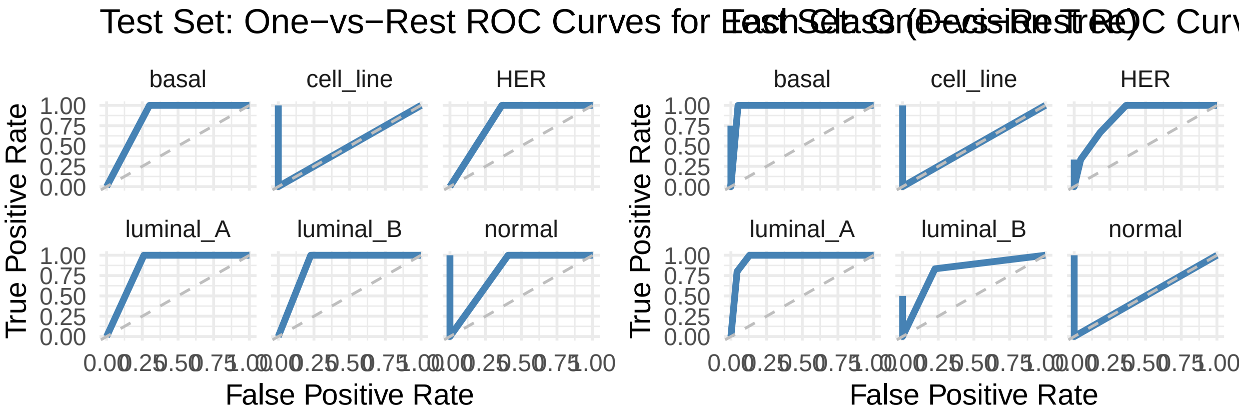
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 f_meas  macro          0.732
```

```r
auc.value
```

```
## Multi-class area under the curve: 0.9623
```

```r
# Step 7: ROC Curve Plot
class_levels = colnames(knn.probs)

roc_df <- do.call(rbind, lapply(class_levels, function(class) {
  binary_response <- as.numeric(true.label == class)
    # Only compute ROC if both classes are present
  if (length(unique(binary_response)) < 2) {
    return(NULL)  # Skip this class
  }

  roc_obj <- roc(binary_response, knn.probs[[class]], quiet = TRUE)

  data.frame(
    fpr = 1 - roc_obj$specificities,
    tpr = roc_obj$sensitivities,
    class = class,
    auc = rep(auc(roc_obj), length(roc_obj$sensitivities))
  )
}))

knn = ggplot(roc_df, aes(x = fpr, y = tpr)) +
  geom_line(linewidth = 1.2, color = "steelblue") +
  geom_abline(linetype = "dashed", color = "gray") +
  facet_wrap(~ class, ncol = 3) +
  theme_minimal() +
  labs(
    title = "Test Set: One-vs-Rest ROC Curves for Each Class (KNN)",
    x = "False Positive Rate",
    y = "True Positive Rate"
  )

grid.arrange(random.forest, lasso, decision.tree, knn, ncol = 2)
```


Test Set: One−vs−Rest ROC Curves for Each Class (Random Forest)
Test Set: One−vs−Rest ROC Curv
basal
cell_line
HER
luminal_A
luminal_B
normal
True Positive Rate
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
False Positive Rate


Test Set: One−vs−Rest ROC Curves for Each Class (Decision Tree)
Test Set: One−vs−Rest ROC Curv
basal
cell_line
HER
luminal_A
luminal_B
normal
True Positive Rate
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
False Positive Rate