

**Project Title**  
Face Expression Recognition Using Machine Learning Techniques

**Student Name**  
Dimitrios Bakatsis

**Course**  
DAMA51 – Data Science and Machine Learning

**Institution**  
Hellenic Open University (HOU)

**Programming Language/Tool Used**  
R

**Table of Contents**

**1 Introduction**

1. **Methodology**

2.1 **Dataset Description**  
2.2 **Data Preprocessing**  
2.3 **Feature Selection Techniques**

2.4 **Evaluation Metrics**

2.5 **Modeling Techniques**  
2.5 **Clustering Methods**

1. **General Conclusion**
2. **Introduction**

The topic of this project is called “face expression recognition”. It includes a dataset in excel format, that has 210 instances, each consisting of 25 measurements. These measurements capture information about the left and right eyebrow, eyes, mouth, and relationships between these facial features. The target variable, **Expression**, has seven unique classes representing different emotions: **Neutral, Disgust, Sadness, Fear, Surprise, Anger, Joy**.

The first objective of the analysis is to identify the most important features that contribute to face expression recognition.

1. **Methodology**
   1. **Dataset Description**

The dataset contains the following 25 features:

H1, H2, H3, H4, L1 (concern the left eyebrow)

H5, H6, H7, H8, L2 (concern the right eyebrow)

H9, H10, W1 (concern the left eye)

H11, H12, W2 (concern the right eye)

H13, H14, H15, W3, L3 (concern the mouth)

R1, R2, R3, R4 (concern relationships between the above parts)

* 1. **Data Preprocessing**

**Data Import:** The dataset was imported from an excel file in Rstudio using the “readxl” package.

**Noise Removal:** The “noise” was removed, which are some columns that were created once the excel file was viewed in R, that were useless.

**Data Standardization:** The data features were standardized using the z-score normalization.

* 1. **Feature Selection Techniques**

 **Principal Component Analysis (PCA)**: PCA was used to reduce the dimensionality of the dataset and identify the principal components that explain the maximum variance.

 **Chi-Squared Test**: This statistical test was used to evaluate the dependency between each feature and the target variable **Expression**. The higher the Chi-squared value, the more dependent the feature is on the target.

 **Correlation Matrix**: The correlation matrix was calculated to measure the linear relationship between each feature and the target variable. Features with higher absolute correlations were considered more important.

 **Random Forest**: Random Forest, an ensemble machine learning method, was used to evaluate feature importance based on how well each feature contributes to the predictive power of the model.

**Results**

**PCA Results**

The PCA technique was used to extract the principal components, as a first selection method, in order to check which measurements maximize variance in data.

Then with the help of a scree plot depicting the proportion of the variance for each principal component, it’s visible that from PCA number 11 it starts to make a straight line so that means that from this point and on the PCAs are not so important. However, there was a criss-cross check in that conclusion, by finding which principal component explains a very high percentage of variance. For this, the threshold that was chosen was at least 90% variance. If it equals or is greater than 0.9 it means that it is much less significant than another PCA value, because it doesn’t have anything else to “add” or it doesn’t give us any new information.

By finding that this happens on PCA number 11, it means that only the first 11 PCAs are significant. Now, in order to answer to the first question about specifying which features seem to be necessary or fundamental, the original values of each PCA for every measurement were calculated and then their absolute values were selected, in order to check the “contribution” that each measurement has in total.

After that, each row is summed horizontally in order to find the total value that each measurement has. By doing that, the importance of each measurement can be found. Finally, the measurements are sorted to a decreasing level in order to appear the most fundamental ones at first and the least ones after.

So, the most fundamental measurements are: W2, R1, W3, H8, W1, R3, H5, H11, L2, H12, H14, H13, R2, H4, H9, H1, H7, H15, R4, L3, L1, H2, H3, H10, H6 in that order.

> file\_path<-file.choose()

> library("readxl")

> face\_expression<-read\_xls(file\_path)

New names:

• `` -> `...26`

• `` -> `...27`

• `` -> `...28`

• `` -> `...30`

• `` -> `...31`

• `` -> `...32`

• `` -> `...33`

• `` -> `...34`

• `` -> `...35`

• `` -> `...36`

> face\_expression <- face\_expression[ ,-c(26,27,28,30,31,32,33,34,35,36)]

> face\_expression\_scaled <- scale(face\_expression[, 1:25])

> pca\_result <- prcomp(face\_expression\_scaled, center = TRUE, scale. = TRUE)

> pca\_variance<-summary(pca\_result)$importance[2,]

> plot(pca\_variance, type = "b", pch = 19, col = "blue",

+ xlab = "Principal Components", ylab = "Proportion of Variance",

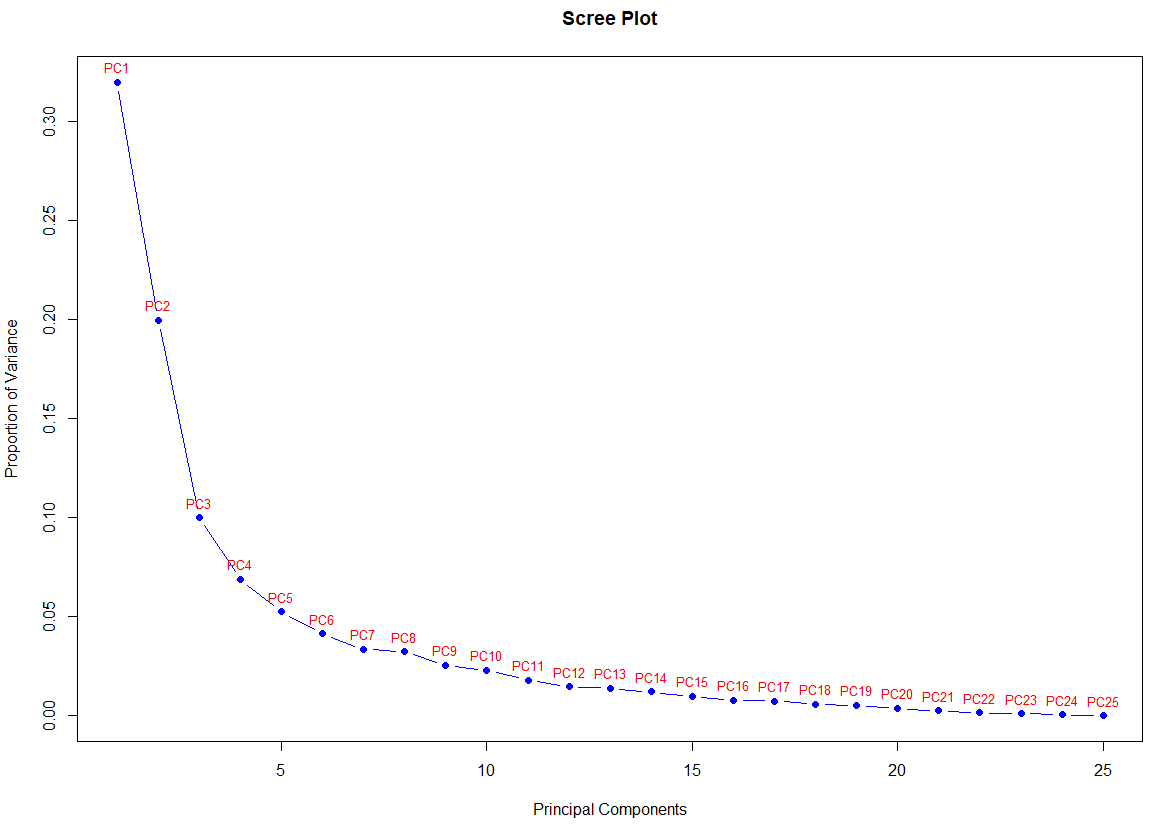
+ main = "Scree Plot")

> text(x = 1:length(pca\_variance),

+ y = pca\_variance,

+ labels = paste("PC", 1:length(pca\_variance), sep = ""),

+ pos = 3, cex = 0.8, col = "red")



> num\_components <- which(cumsum(pca\_variance) >= 0.90)[1]

> selected\_pcs <- pca\_result$x[, 1:num\_components]

> loadings <- pca\_result$rotation[, 1:num\_components]

> abs\_loadings <- abs(loadings)

> feature\_importance <- rowSums(abs\_loadings[, 1:num\_components])

> names(feature\_importance) <- rownames(abs\_loadings)

> sorted\_features\_pca <- sort(feature\_importance, decreasing = TRUE)

> sorted\_features\_pca <- names(sorted\_features\_pca)

> print(sorted\_features\_pca)

"W2" "R1" "W3" "H8" "W1" "R3" "H5" "H11" "L2" "H12" "H14" "H13" "R2" "H4" "H9" "H1" "H7" "H15" "R4" "L3" "L1" "H2" "H3" "H10" "H6"

**Chi-Squared Test Results**

The Chi-Squared measures dependency with the target for categorical data. At first, a chi-squared test is run for each feature with respect to the “Expression” column/target and then the results are sorted to a decreasing level. Finally, the most fundamental measurements according to the Chi-squared technique are: H12, W2, H14, R3, H9, H8, H11, L2, W3, H5, H2, H10, R2, H15, L3, H6, R4, H7, R1 in that order.

> chi\_sq\_p\_values <- chi\_squared\_results(face\_expression[,1:25],face\_expression$Expression)

> sorted\_chi\_sq\_p\_values <- sort(chi\_sq\_p\_values,decreasing=TRUE)

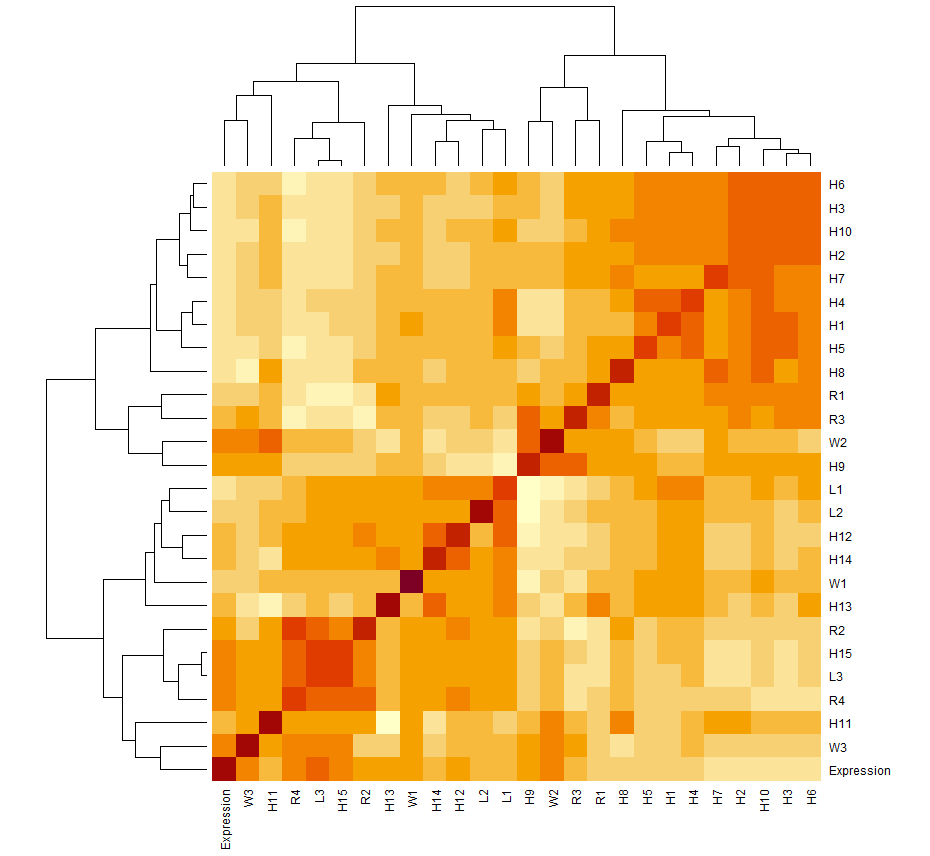
> sorted\_features\_chi\_sq <- names(sorted\_chi\_sq\_p\_values)

> print(sorted\_features\_chi\_sq)

"H12" "W2" "H14" "R3" "H9" "H8" "H11" "L2" "W3" "H5" "H2" "H10" "R2" "H15" "L3" "H6" "R4" "H7" "R1"

**Correlation Matrix Results**

The correlation matrix is selected in order to identify the linear relationships between the 25 measurements and the target “Expression”. Then, a heatmap plot is created that shows the correlation between all of the measurements, as below:



Secondly, the scaled dataset is combined with the numeric Expression column into a new dataframe. Then, the correlation of each feature with the variable “expression” is extracted and after that their absolute values are sorted in a decreasing level in order to appear the most fundamental ones at first and the least ones after, regarding their correlation levels. Obviously, from the sorted features the feature “Expression” is removed since it will always be the most correlated one and its value will be one. Finally, the most fundamental measurements according to the correlation technique are: H3, H7, H10, H6, H2, L3, H15, H5, R4, H1, H4, R1, H8, L1, W3, W2, R3, R2, L2, H9, H14, H11, H12, H13, W1 in that order.

> face\_expression$Expression\_numeric <- as.numeric(factor(face\_expression$Expression))

> data\_with\_target <- cbind(face\_expression\_scaled, Expression = face\_expression$Expression\_numeric)

> cor\_matrix <- cor(data\_with\_target)

> feature\_cor\_with\_target <- cor\_matrix[, "Expression"]

> heatmap(cor\_matrix)

> sorted\_features\_corr <- sort(abs(feature\_cor\_with\_target), decreasing = TRUE)

> sorted\_features\_corr<-names(sorted\_features\_corr)

> sorted\_features\_corr <- setdiff(sorted\_features\_corr, "Expression")

> print(sorted\_features\_corr)

"H3" "H7" "H10" "H6" "H2" "L3" "H15" "H5" "R4" "H1" "H4" "R1" "H8" "L1" "W3" "W2" "R3" "R2" "L2" "H9" "H14" "H11" "H12" "H13" "W1"

**Random Forest Model Results**

Finally, the last selection method that was used is the randomForest model, which is a collection of decision trees that work together to make predictions. Using the combined dataset, called “data with target” from the previous method, the randomForest model is executed and then the feature importance is extracted. Once that is done, the feature importance measurements are sorted in a decreasing level. So, the most fundamental measurements according to the randomForest model are: H7, H3, H6, R4, L3, L1, H15, R3, H9, H10, R1, H1, R2, H2, L2, H11, H8, W3, H14, H5, H12, H13, W2, H4, W1 in that order.

> rf\_model <- randomForest(Expression ~ ., data = data\_with\_target)

> feature\_importance\_rf <- rf\_model$importance

> sorted\_features\_rf <- names(sort(feature\_importance\_rf[, 1], decreasing = TRUE))

> print(sorted\_features\_rf)

"H7" "H3" "H6" "L3" "R4" "H10" "H15" "L1" "H9" "R3" "R1" "H1" "R2" "L2" "H2" "H11" "H8" "H14" "W3" "H5" "H12" "H13" "W2" "H4" "W1"

**Conclusion**

In conclusion, four different methods have been applied, with obviously different results from each one, since each method measures different aspects of the data. Now, the answer to the question regarding the most fundamental measurements of the dataset, will be based on the combination of these four different methods and the first five common features of them will be selected as most fundamental. So, the most fundamental measurements are: W2, R1, W3, H8, R3 in that order.

> common\_features <- intersect(intersect(intersect(sorted\_features\_pca, sorted\_features\_chi\_sq),sorted\_features\_corr),sorted\_features\_rf)

> head(common\_features,5)

[1] "W2" "R1" "W3" "H8" "R3"

* 1. **Evaluation Metrics**

For the second part of the project, the goal was to evaluate the performance of the dataset using the **Random Forest** classifier for predicting facial expression. To ensure that the model performs effectively, the data is split into three sets: training, validation, and test. The primary evaluation metrics used were **accuracy**, **precision**, **recall**, and **F1-score**.

Data Splitting

To evaluate the model's performance robustly, the dataset was divided into three distinct sets:

1. **Training set (60%)**: Used to train the Random Forest model.
2. **Validation set (20%)**: Used for hyperparameter tuning and model validation during the training process.
3. **Test set (20%)**: Used to evaluate the final performance of the trained model.

> train\_index <- createDataPartition(face\_expression$Expression, p = 0.6, list = FALSE)

> train\_data <- face\_expression[train\_index, ]

> temp\_data <- face\_expression[-train\_index, ]

> valid\_index <- createDataPartition(temp\_data$Expression, p = 0.5, list = FALSE)

> validation\_data <- temp\_data[valid\_index, ]

> test\_data <- temp\_data[-valid\_index, ]

> train\_data$Expression <- factor(train\_data$Expression)

> validation\_data$Expression <- factor(validation\_data$Expression)

> test\_data$Expression <- factor(test\_data$Expression)

Model Training and Prediction

A **Random Forest classifier** was chosen for this task, as it was also used in the first part of the project, due to the many features of the dataset. Once trained, the model was used to make predictions on the test set.

> rf\_model <- randomForest(Expression ~ ., data = train\_data)

> test\_pred <- predict(rf\_model, newdata = test\_data)

Model Evaluation

The evaluation of the model's performance on the test set was done using the following metrics:

**Confusion Matrix**: This was generated using the confusionMatrix function from the caret package. It provides detailed insights into the model’s performance, including the number of true positives, false positives, true negatives, and false negatives for each class. By finding and printing the confusion matrix, the metrics of accuracy, precision, recall and F1-score can also be calculated as below:

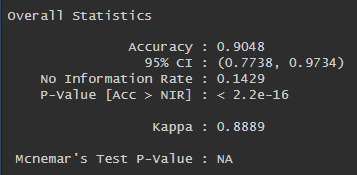
* **Accuracy**: The accuracy of the model was calculated as the proportion of correctly classified instances over the total number of instances. The formula for accuracy is:

> accuracy <- round(sum(diag(conf\_matrix$table)) / total,3)

> cat("Accuracy:", accuracy, "\n")

Accuracy: 0.905

However, this can be easily also seen from the output of the confusion matrix:



* **Precision**: Precision indicates the proportion of positive predictions that are actually correct. For each class, it is calculated as below:

> precision <- round(conf\_matrix$byClass[, "Precision"],3)

> cat("Precision:", precision, "\n")

Precision: 1 1 0.857 1 1 0.714 0.857

* **Recall**: Recall (also known as Sensitivity or True Positive Rate) measures the proportion of actual positive instances that are correctly identified. It is calculated as:

> recall <- round(conf\_matrix$byClass[, "Recall"],3)

> cat("Recall:", recall, "\n")

Recall: 1 0.833 1 1 0.667 0.833 1

* **F1-Score**: The F1-score is the harmonic mean of precision and recall. It balances the trade-off between precision and recall. It is calculated as:

> f1\_score <- round(2 \* (precision \* recall) / (precision + recall),3)

> cat("F1 Score:", f1\_score, "\n")

F1 Score: 1 0.909 0.923 1 0.8 0.769 0.923

Also, the average of the metrics precision, recall and F1-Score are calculated, as below in order to have a sum up review, as below:

> avg\_precision <- round(mean(precision, na.rm = TRUE),3)

> avg\_recall <- round(mean(recall, na.rm = TRUE),3)

> avg\_f1\_score <- round(mean(f1\_score, na.rm = TRUE),3)

> # Print results

> cat("Average Precision:", avg\_precision, "\n")

Average Precision: 0.918

> cat("Average Recall:", avg\_recall, "\n")

Average Recall: 0.905

> cat("Average F1 Score:", avg\_f1\_score, "\n")

Average F1 Score: 0.903

Results

The following metrics were calculated based on the confusion matrix and the caret package’s functionality:

1. **Accuracy**: The model’s accuracy was computed as the ratio of correctly classified instances to the total instances.
2. **Precision**: The average precision for all classes was computed.
3. **Recall**: The average recall for all classes was computed.
4. **F1-Score**: The F1-score was calculated for each class and the macro-average F1-score was derived.

**Conclusion**

In conclusion, the model shows a very high precision (0.918), meaning it rarely classifies negative instances as positive. Also, the model also performs well in terms of recall (0.905), meaning it is capable of identifying most of the actual positive instances. Finally, the F1 score is a balanced measure between precision and recall (0.903), and the high score suggests that the model achieves a good trade-off between correctly identifying positive cases and minimizing false positives. So, by splitting the data into training, validation, and test sets, and using key evaluation metrics, it is ensured that the model is both trained and tested effectively to avoid overfitting or underfitting.

* 1. **Modeling Techniques**

For the third part of the project, there will be applied a few classifiers on the dataset and then compared. Three different classifiers have been selected, which are the **Random Forest**, **Naive Bayes**, and **K-Nearest Neighbors (KNN)**, in order to determine how well they classify the target variable "Expression" in the given dataset.

1. **Random Forest**:

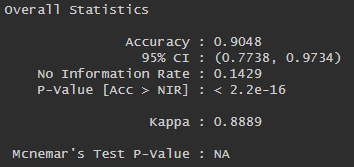
At first, as in the second part of the project, the random forest classifier has been selected and a confusion matrix was created between the actual and the predicted values of the data, with accuracy 0.905 as it is seen below:

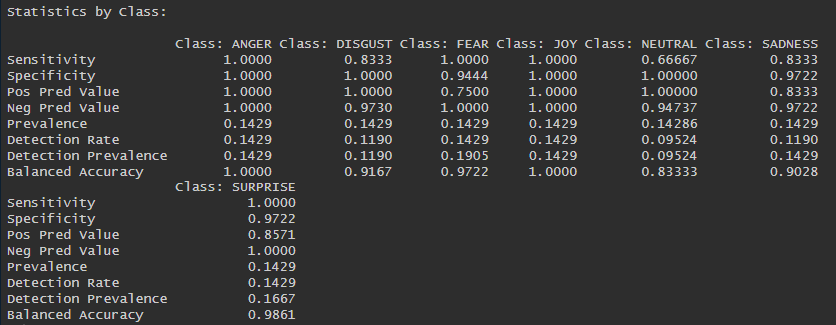
> rf\_model <- randomForest(Expression ~ ., data = train\_data)

> rf\_pred <- predict(rf\_model, newdata = test\_data)

> conf\_matrix\_rf <- confusionMatrix(factor(rf\_pred), factor(test\_data$Expression))

> print(conf\_matrix\_rf)





2. **Naive Bayes**:

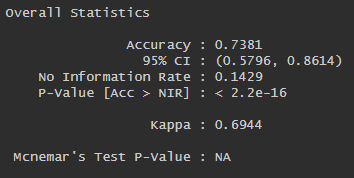
As a second classifier, the Naïve Bayes method was selected anda confusion matrix was created between the actual and the predicted values of the data, with accuracy 0.738 as it is seen below:

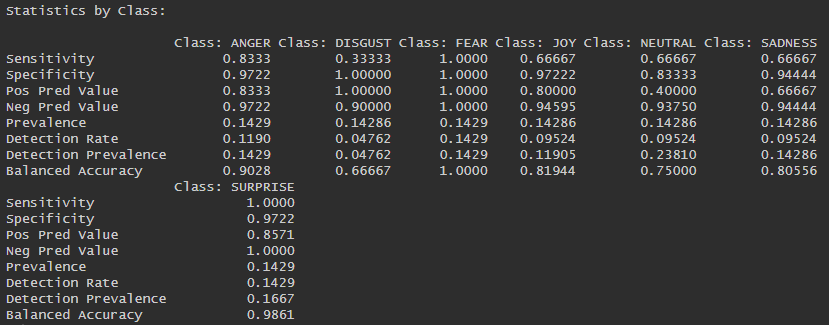
> nb\_model <- naiveBayes(Expression ~ ., data = train\_data)

> nb\_pred <- predict(nb\_model, newdata = test\_data)

> conf\_matrix\_nb <- confusionMatrix(factor(nb\_pred), factor(test\_data$Expression))

> print(conf\_matrix\_nb)





3. **K-Nearest Neighbors (KNN)**:

As a third classifier, the KNN method was selected, with k=5 the number of neighbors, and a confusion matrix was created between the actual and the predicted values of the data, with accuracy 0.333 as it is seen below:

> k\_value <- 5 # Choose the number of neighbors (k)

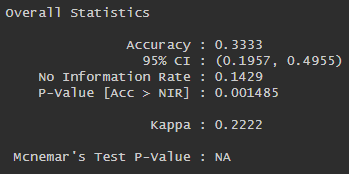
> knn\_pred <- knn(train = train\_data[, -which(names(train\_data) == "Expression")],

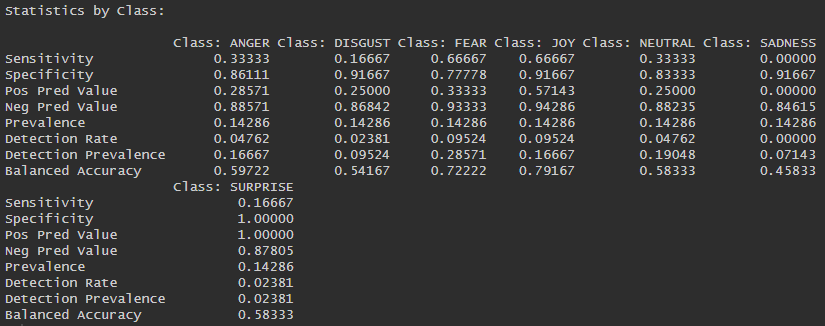
+ test = test\_data[, -which(names(test\_data) == "Expression")],

+ cl = train\_data$Expression, k = k\_value)

> knn\_conf\_matrix <- confusionMatrix(factor(knn\_pred), factor(test\_data$Expression))

> print(knn\_conf\_matrix)





After that, the accuracy for all three of the above selected classifiers was printed in order for the comparison to be clearer. It is obvious that the highest accuracy belongs to the Random Forest Classifier.

> rf\_accuracy <- round(sum(diag(conf\_matrix\_rf$table)) / sum(conf\_matrix\_rf$table),3)

> nb\_accuracy <- round(sum(diag(conf\_matrix\_nb$table)) / sum(conf\_matrix\_nb$table),3)

> knn\_accuracy <- round(sum(diag(knn\_conf\_matrix$table)) / sum(knn\_conf\_matrix$table),3)

> # Calculate Precision and Recall for each class

> precision\_knn <- round(knn\_conf\_matrix$byClass[, "Precision"],3)

> recall\_knn <- round(knn\_conf\_matrix$byClass[, "Recall"],3)

> f1\_score\_knn <- round(2 \* (precision\_knn \* recall\_knn) / (precision\_knn + recall\_knn),3)

> # Calculate macro-average Precision, Recall, and F1-Score

> avg\_precision\_knn <- round(mean(precision\_knn, na.rm = TRUE),3)

> avg\_recall\_knn <- round(mean(recall\_knn, na.rm = TRUE),3)

> avg\_f1\_score\_knn <- round(mean(f1\_score\_knn, na.rm = TRUE),3)

> # Print the results

> cat("Random Forest Accuracy:", rf\_accuracy, "\n")

Random Forest Accuracy: 0.905

> cat("Naive Bayes Accuracy:", nb\_accuracy, "\n")

Naive Bayes Accuracy: 0.738

> cat("KNN Accuracy:", knn\_accuracy, "\n")

KNN Accuracy: 0.333

> cat("Random Forest Average Precision:", avg\_precision\_rf, "\n")

Random Forest Average Precision: 0.92

> cat("Naive Bayes Average Precision:", avg\_precision\_nb, "\n")

Naive Bayes Average Precision: 0.794

> cat("KNN Average Precision:", avg\_precision\_knn, "\n")

KNN Average Precision: 0.384

> cat("Random Forest Average Recall:", avg\_recall\_rf, "\n")

Random Forest Average Recall: 0.905

> cat("Naive Bayes Average Recall:", avg\_recall\_nb, "\n")

Naive Bayes Average Recall: 0.738

> cat("KNN Average Recall:", avg\_recall\_knn, "\n")

KNN Average Recall: 0.333

> cat("Random Forest Average F1 Score:", avg\_f1\_score\_rf, "\n")

Random Forest Average F1 Score: 0.903

> cat("Naive Bayes Average F1 Score:", avg\_f1\_score\_nb, "\n")

Naive Bayes Average F1 Score: 0.736

> cat("KNN Average F1 Score:", avg\_f1\_score\_knn, "\n")

KNN Average F1 Score: 0.356

**Conclusion:**

In conclusion, based on the **accuracy** and performance metrics (recall, precision, F1 score), **Random Forest** is the most effective classifier for this particular problem. However, both **Naive Bayes** and **K-Nearest Neighbors (KNN)** performed reasonably well and may be useful in different contexts.

* 1. **Clustering Methods**

For the fourth and final part of the project, there will be applied a few clustering methods and then compared with the original clustering. Three different clustering methods have been selected, which are the **K-means**, **Hierarchical**, and **DBSCAN** in order to investigate to what degree the produced clusters coincide with the original ones (those that correspond to the original classes).

1. **K-means clustering method**:

At first, the K-means clustering method has been selected and the value of centers which has been selected is 7 which is the real value that the “Expression” class has **(“NEUTRAL”, “DISGUST”, “SADNESS”, “FEAR”, “SURPRISE”, “ANGER”, “JOY”)** as you can see below:

Applying K-Means Clustering method

> set.seed(123)

> kmeans\_result <- kmeans(face\_expression\_scaled, centers = length(unique(true\_labels)), nstart = 25)

> kmeans\_clusters <- kmeans\_result$cluster

> summary(kmeans\_clusters)

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.000 3.000 4.000 4.414 6.000 7.000

2. **Hierarchical clustering method**:

Secondly, the Hierarchical clustering method has been selected and the value of centers which has been selected is 7 which is the real value that the “Expression” as before:

Applying Hierarchical Clustering method

> dist\_matrix <- dist(face\_expression\_scaled)

> hc\_result <- hclust(dist\_matrix)

> hc\_clusters <- cutree(hclust(dist\_matrix), k = length(unique(true\_labels)))

> summary(hc\_clusters)

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.000 1.000 2.000 2.419 3.000 7.000

3. **DBSCAN clustering method**:

Finally, the Hierarchical clustering method has been selected and the value of centers which has been selected is 7 which is the real value that the “Expression” as before:

Applying DBSCAN Clustering method

> dbscan\_result <- dbscan(face\_expression\_scaled, eps = 2, minPts = 5)

> dbscan\_clusters <- dbscan\_result$cluster

> summary(dbscan\_clusters)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 0.000 0.000 1.133 1.000 8.000

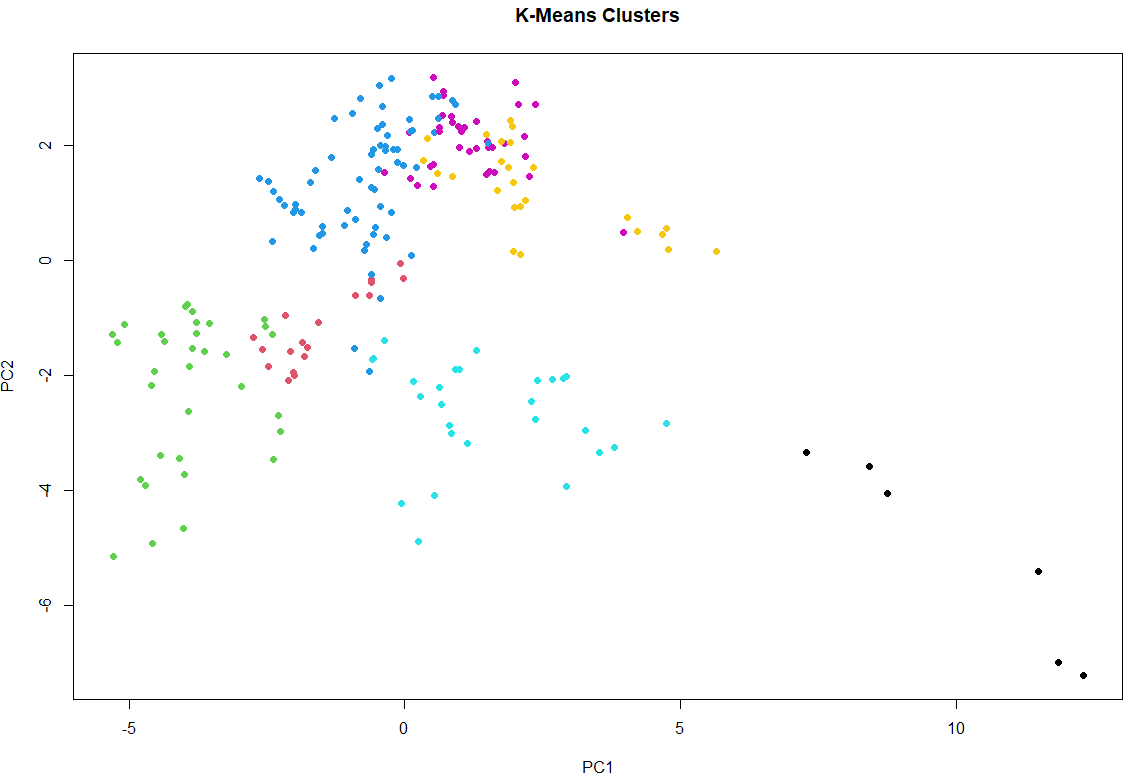
As a next step, a plot for each clustering method is created along with the plot with the true labels of “Expression” class in the 2-dimensional space by comparing the first two principal component analysis (PC1 and PC2), as below:

> # First two principal components

> pca\_2d <- prcomp(face\_expression\_scaled)$x[, 1:2]

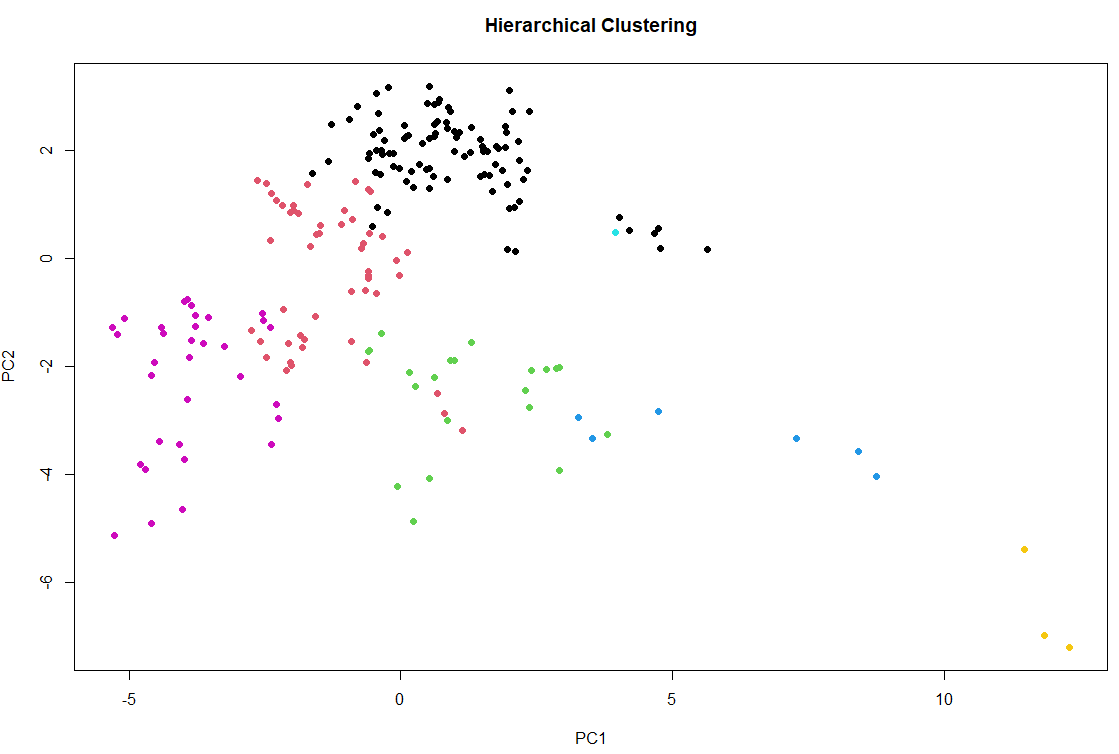
> # Plot K-means clusters

> plot(pca\_2d, col = kmeans\_clusters, main = "K-Means Clusters", pch = 19)



> # Plot Hierarchical clustering

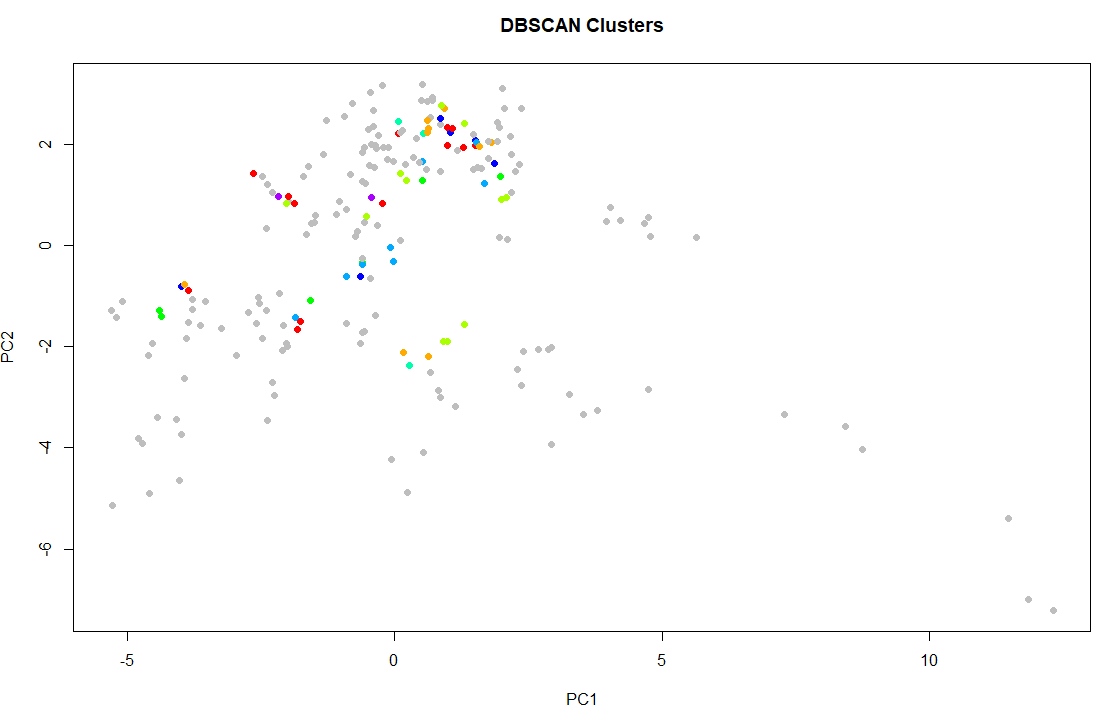
> plot(pca\_2d, col = hc\_clusters, main = "Hierarchical Clustering", pch = 19)



> # Plot DBSCAN clustering

> dbscan\_clusters\_color <- ifelse(dbscan\_clusters == 0, "gray", rainbow(length(unique(dbscan\_clusters)))[dbscan\_clusters])

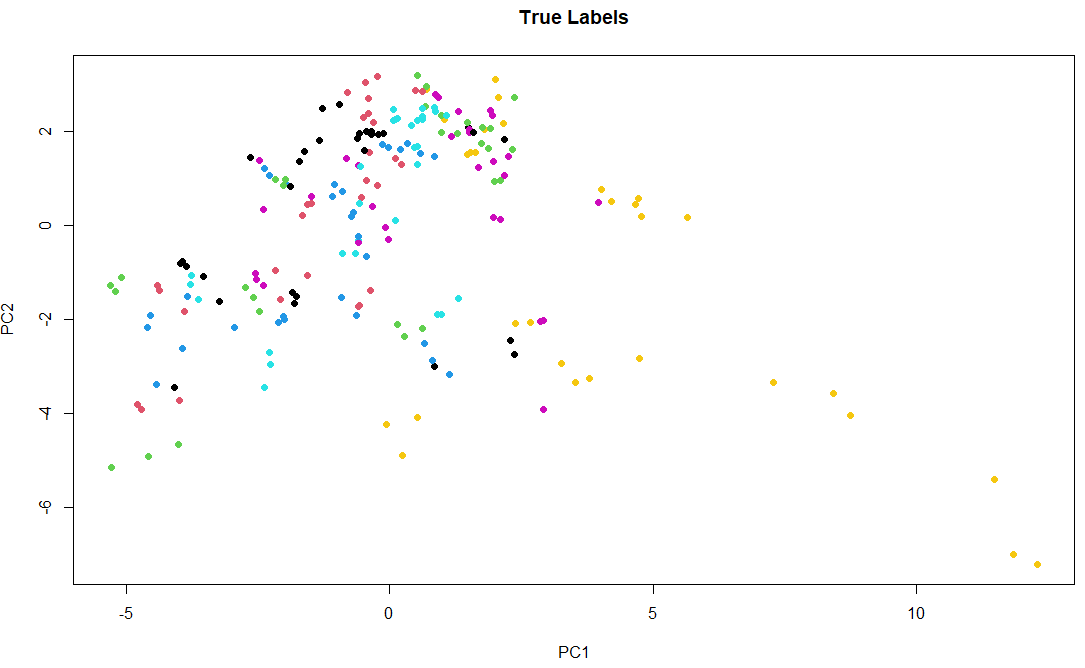
> plot(pca\_2d, col = dbscan\_clusters\_color, main = "DBSCAN Clusters", pch = 19)



> # Plot true labels

> label\_numeric <- as.numeric(factor(true\_labels))

> plot(pca\_2d, col = label\_numeric, main = "True Labels", pch = 19)



**Conclusion:**

In conclusion, after careful observation, it is clear that the resulting clusters did not closely align with the original class labels. The only clustering method that is a bit similar with the original clustering is K-means clustering. This suggests that the class structure in the dataset is not easily recoverable through unsupervised clustering. Possible reasons include overlapping features between classes, the data might not be naturally clustered, or the high dimensional nature of the data.

1. **General Conclusion**

This study conducted a thorough exploration of facial expression recognition using multiple analytical techniques. Feature selection through PCA, Chi-Square Test, Correlation Analysis, and Random Forest consistently highlighted W2, R1, W3, H8, and R3 as the most fundamental measurements. Classification models were evaluated using key metrics, with Random Forest standing out due to its superior precision, recall, and F1-score, although Naive Bayes and KNN also showed reasonable performance. In contrast, clustering analysis revealed that the dataset does not exhibit clear, natural groupings, with K-means only partially matching the true labels. These findings underline the importance of applying a variety of data analysis methods to gain a more complete and accurate understanding of complex datasets.