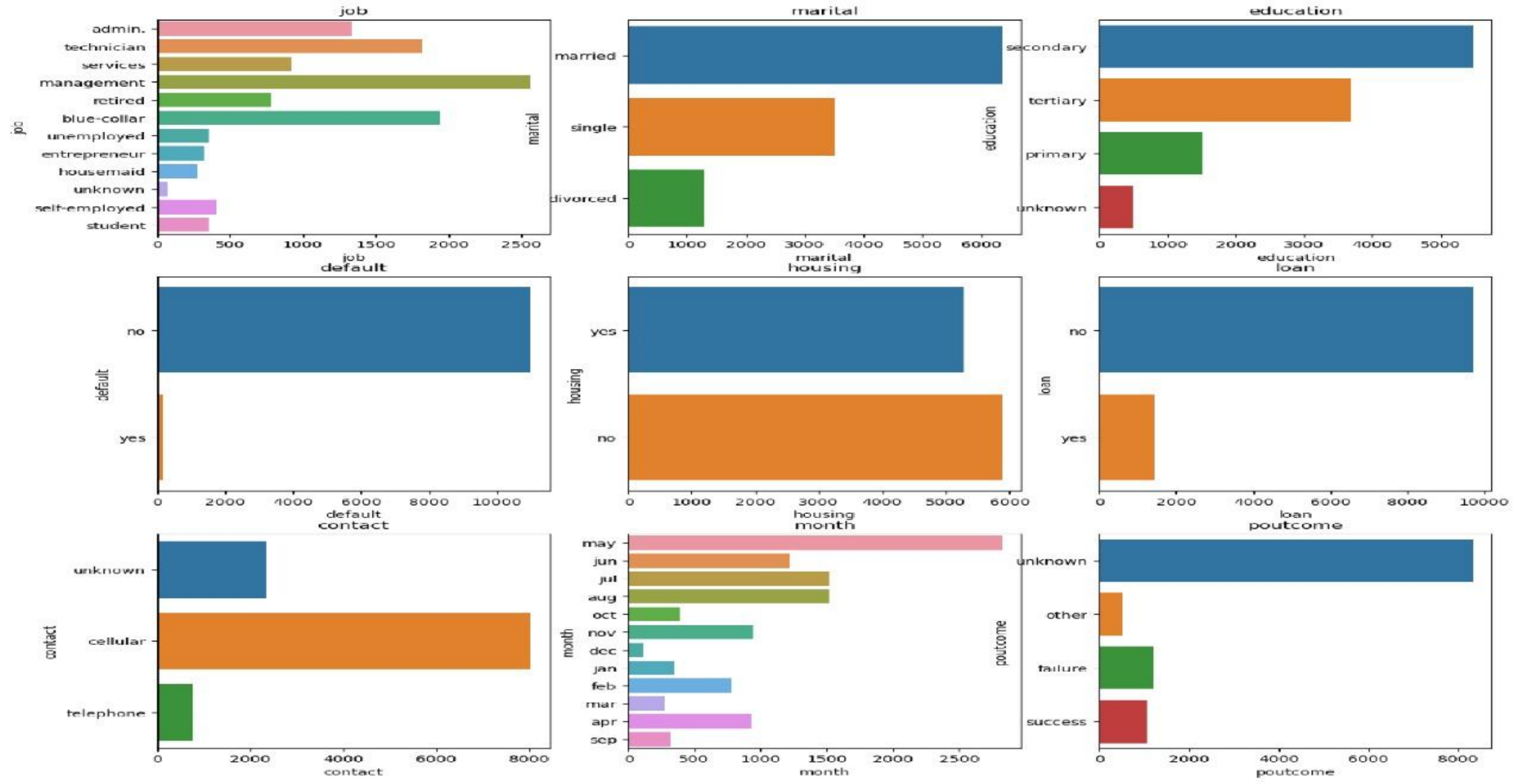
BANK MARKETING ANALYSIS

PRESENTED BY:
AQUILA,KRISHNA

Overview ..

- **The Dataset comprises 41,188 rows and 21 columns.**
- **The dataset contains various features representing client attributes, communication details, and economic indicators.**
- **Numerical features include age, duration of contact, number of contacts performed during the campaign, and economic factors such as employment variation rate, consumer price index, and consumer confidence index.**
- **Categorical features encompass client attributes like job type, marital status, education level, and communication channels such as contact type, month, and day of the week of the last contact.**
- **We will be showcasing implementations in both R and Python for various classification algorithms and feature importance calculation.**

CATEGORICAL VARIABLE VIEW



DESCRIPTIVE ANALYSIS

FEATURE DESCRIPTION:

Encompasses client attributes, contact details, economic indicators, and campaign information.

Provides insights into demographic characteristics, communication methods, economic environment, and campaign effectiveness

IMBALANCED ANALYSIS

Examines the class distribution where one class significantly outnumbers others.

Poses challenges for predictive modeling due to biased algorithms and overlooked minority classes.

TARGET DEFINITION :

Focuses on the "y" variable indicating term deposit subscription.

Main objective is to accurately predict subscription based on client attributes and campaign interactions.

CLASSIFICATION

Decision Tree, Naive Bayes, Random Forest, Bagging, and Gradient Boosting are classification algorithms used in data analysis.

Decision Tree splits data based on attributes, while Naive Bayes is probabilistic.

Random Forest combines multiple decision trees, Bagging averages them, and Gradient Boosting sequentially corrects errors.

We will be showing Code snippets in R and Python that demonstrate model training and confusion matrix for each algorithm

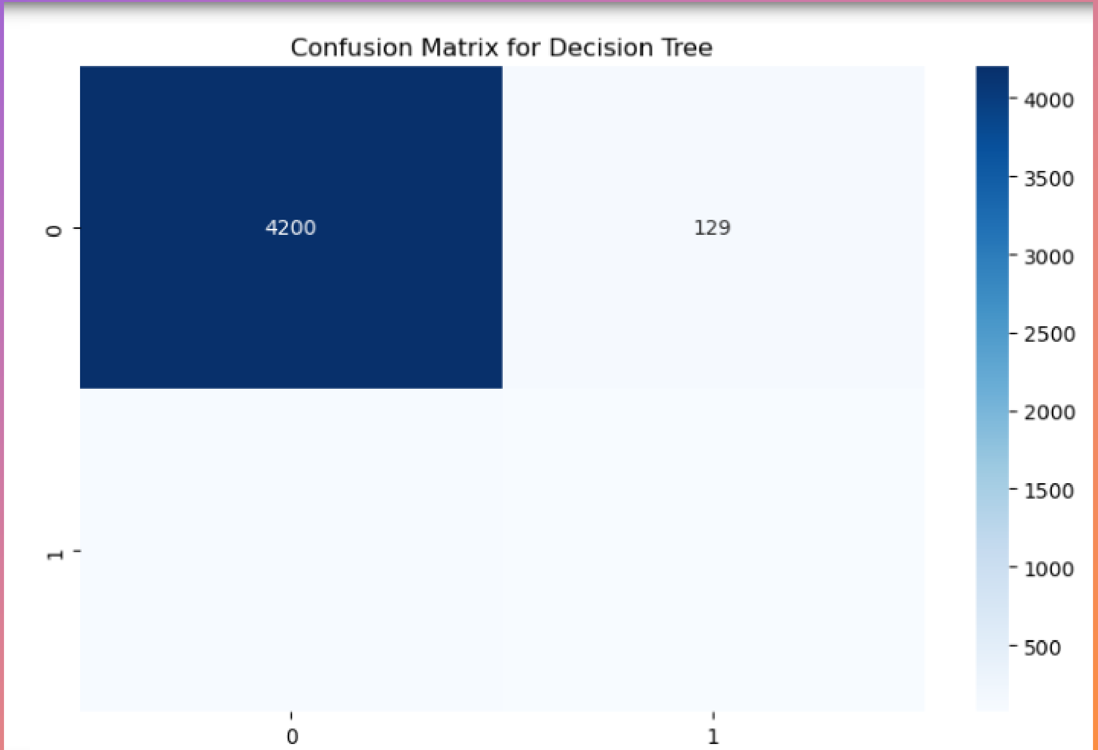
Along with feature selection.

DECISION TREE

IMPLEMENTATION IN PYTHON

```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

```
Statistics for Decision Tree classifier:
Confusion Matrix:
 [[4192  137]
 [ 103   68]]
Accuracy: 0.9466666666666667
Precision: 0.33170731707317075
Recall: 0.39766081871345027
F1 Score: 0.3617021276595745
```

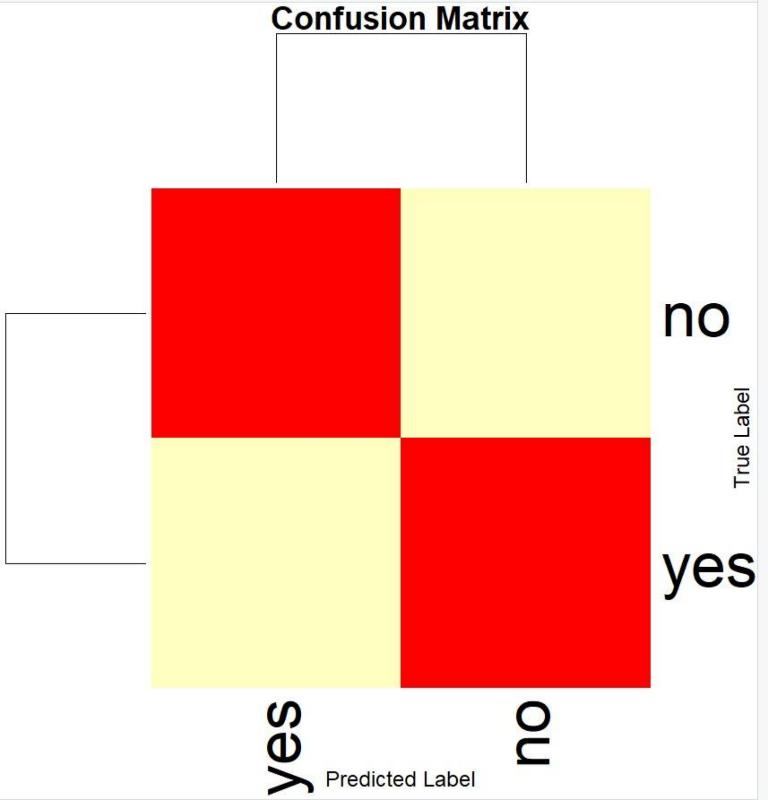


The decision tree is constructed iteratively by selecting the best attribute to split the data at each node.

IMPLEMENTATION IN R

```r
# Training Decision Tree model
model <- rpart(subscription ~ ., data = train_data, method = "class")
```

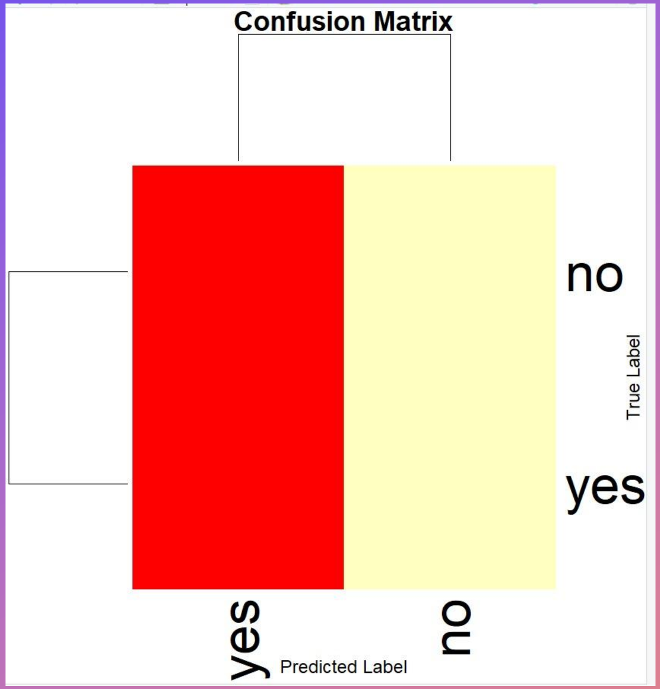
```
+ }
> # Iterate over models and train/evaluate
> for (model_name in names(models_cat)) {
+   train_and_evaluate_cat_model(model_name, models_cat[[model_name]], train_data_c
at, test_data_cat)
+ }
Confusion Matrix for Decision Tree :
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  4239   90
       yes   79   90

               Accuracy : 0.9624
                 95% CI : (0.9565, 0.9678)
    No Information Rate : 0.96
    P-Value [Acc > NIR] : 0.2134

                  Kappa : 0.4962

 Mcnemar's Test P-Value : 0.4418

            Sensitivity : 0.9817
            Specificity : 0.5000
         Pos Pred Value : 0.9792
         Neg Pred Value : 0.5325
             Prevalence : 0.9600
         Detection Rate : 0.9424
   Detection Prevalence : 0.9624
      Balanced Accuracy : 0.7409

       'Positive' Class : no
```



```r
model <- naive_bayes(subscription ~ ., data = train_data)
```

NAÏVE BAYES

IMPLEMENTATION IN R



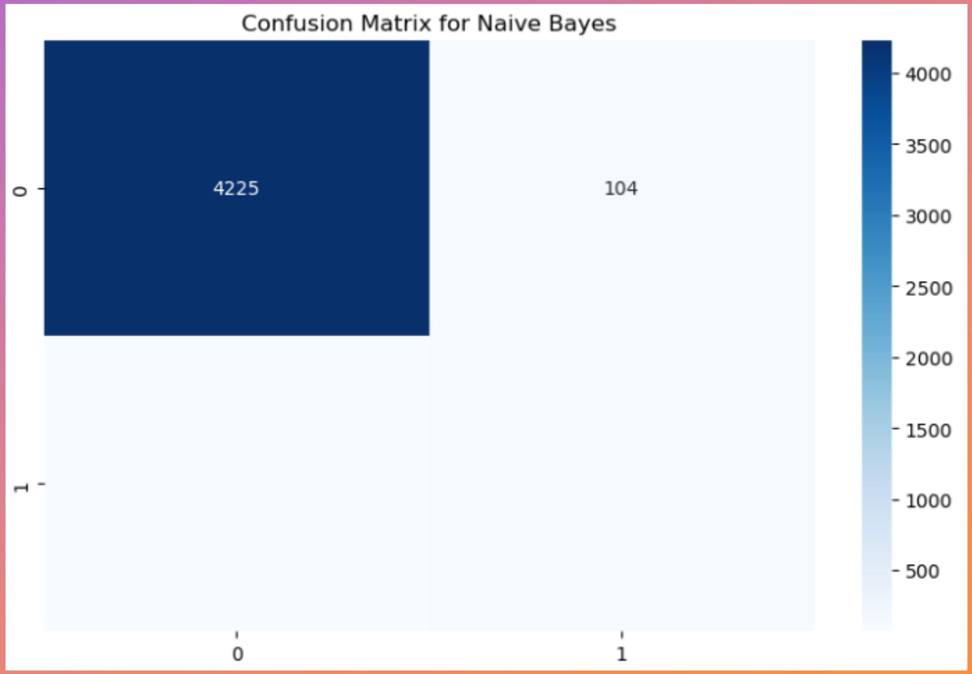

```
Confusion Matrix for Naive Bayes :
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  4318  180
       yes    0    0

               Accuracy : 0.96
                 95% CI : (0.9538, 0.9655)
    No Information Rate : 0.96
    P-Value [Acc > NIR] : 0.5198

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.00
            Specificity : 0.00
         Pos Pred Value : 0.96
         Neg Pred Value :  NaN
             Prevalence : 0.96
         Detection Rate : 0.96
   Detection Prevalence : 1.00
      Balanced Accuracy : 0.50

       'Positive' Class : no
```

IMPLEMENTATION IN PYTHON

IT ASSUMES THAT THE FEATURES ARE CONDITIONALLY INDEPENDENT GIVEN THE CLASS, HENCE THE "NAIVE" ASSUMPTION

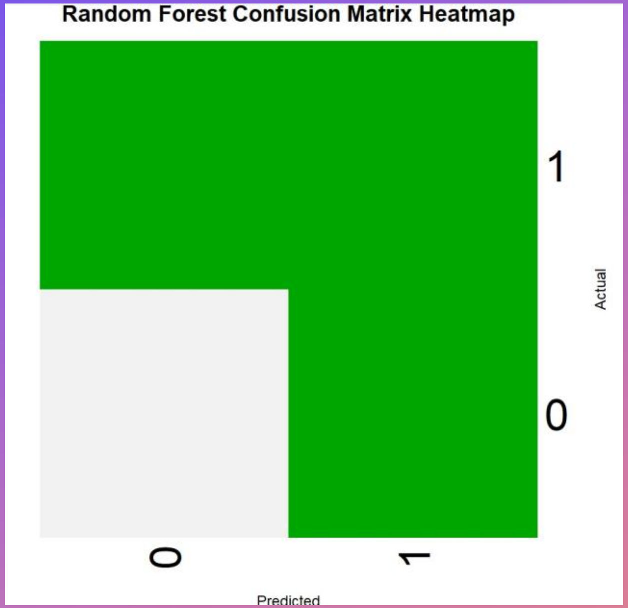
```
model = GaussianNB()
model.fit(X_train, y_train)
```

```
Statistics for Naive Bayes classifier:
Confusion Matrix:
 [[4225  104]
 [  68  103]]
Accuracy: 0.9617777777777777
Precision: 0.4975845410628019
Recall: 0.6023391812865497
F1 Score: 0.544973544973545
```



RANDOM FOREST

IMPLEMENTATION IN R.

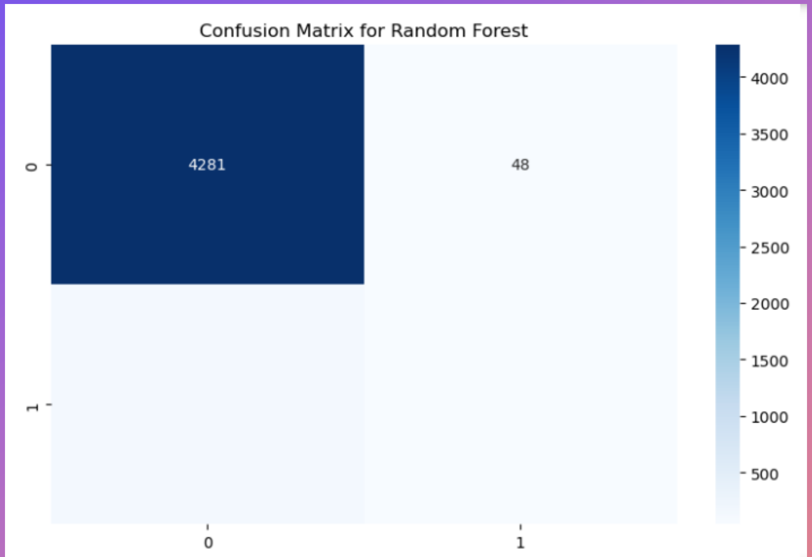



```r
rf_model <- randomForest(subscription~., data=train, ntree=50,
                         ntry=6, importance=TRUE, replace=FALSE)
```

```
> confusionMatrix (original, rf_pred)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4785   24
         1  136   16

               Accuracy : 0.9677
                 95% CI : (0.9624, 0.9725)
    No Information Rate : 0.9919
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1559

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9724
            Specificity : 0.4000
         Pos Pred Value : 0.9950
         Neg Pred Value : 0.1053
             Prevalence : 0.9919
         Detection Rate : 0.9645
   Detection Prevalence : 0.9694
      Balanced Accuracy : 0.6862

       'Positive' Class : 0
```

IMPLEMENTATION IN PYTHON

IT CONSTRUCTS MULTIPLE DECISION TREES DURING TRAINING AND OUTPUTS THE MODE OF THE CLASSES (CLASSIFICATION) OR MEAN PREDICTION (REGRESSION) OF THE INDIVIDUAL TREES



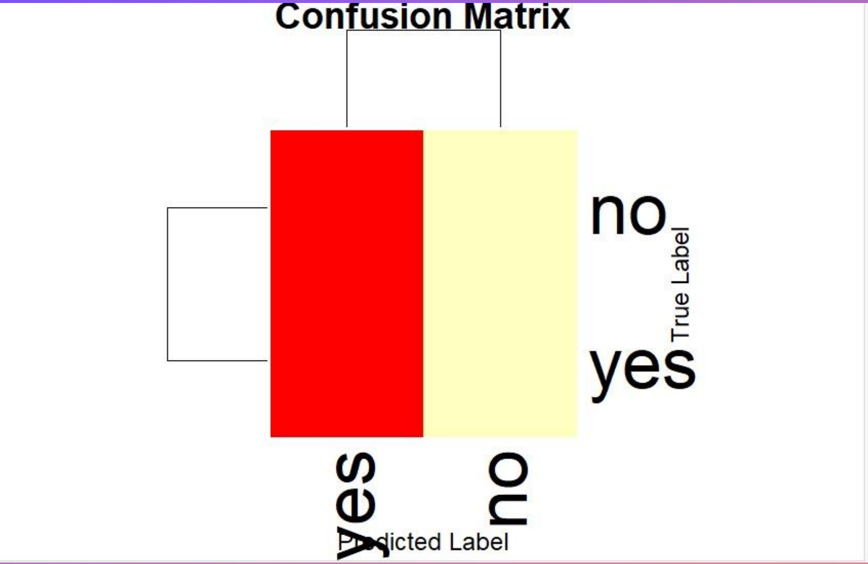

```python
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

```
Statistics for Random Forest classifier:
Confusion Matrix:
 [[4283   46]
 [ 133   38]]
Accuracy: 0.9602222222222222
Precision: 0.4523809523809524
Recall: 0.2222222222222222
F1 Score: 0.2980392156862745
```

GRADIENT BOOSTING

IMPLEMENTATION IN R

```
model <- gbm(subscription ~ ., data = train_data, n.trees = 100, interaction.depth = 4)
```



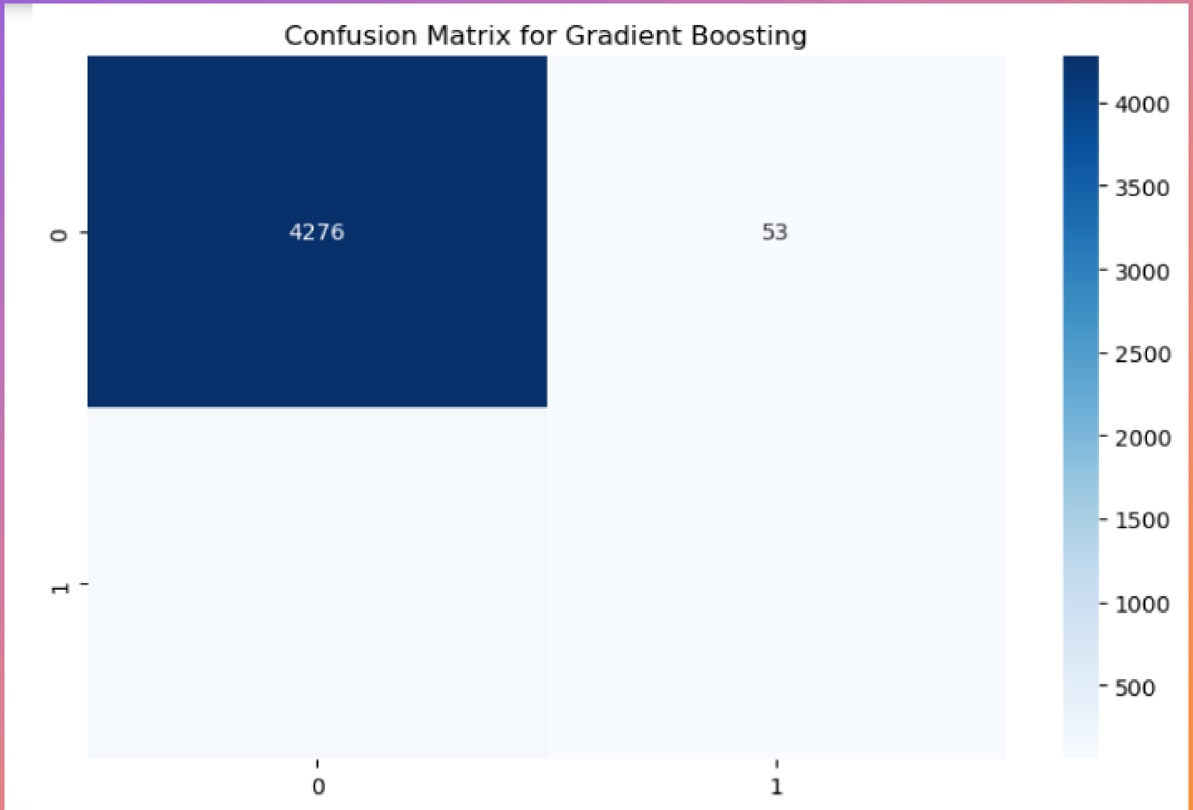

```
     9        0.1930             nan     0.1000    0.0014
    10        0.1882             nan     0.1000    0.0021
    20        0.1666             nan     0.1000    0.0007
    40        0.1576             nan     0.1000   -0.0000
    50        0.1551             nan     0.1000   -0.0001

Confusion Matrix for Gradient Boosting :
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  4272  122
       yes   46   58

               Accuracy : 0.9627
                 95% CI : (0.9567, 0.968)
    No Information Rate : 0.96
    P-Value [Acc > NIR] : 0.1915

                  Kappa : 0.3906

 Mcnemar's Test P-Value : 7.192e-09

            Sensitivity : 0.9893
            Specificity : 0.3222
         Pos Pred Value : 0.9722
         Neg Pred Value : 0.5577
             Prevalence : 0.9600
         Detection Rate : 0.9498
   Detection Prevalence : 0.9769
      Balanced Accuracy : 0.6558

       'Positive' Class : no
```

IMPLEMENTATION IN PYTHON

Gradient Boosting is effective for a wide range of predictive modeling tasks and is known for its high predictive accuracy

```
model = GradientBoostingClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

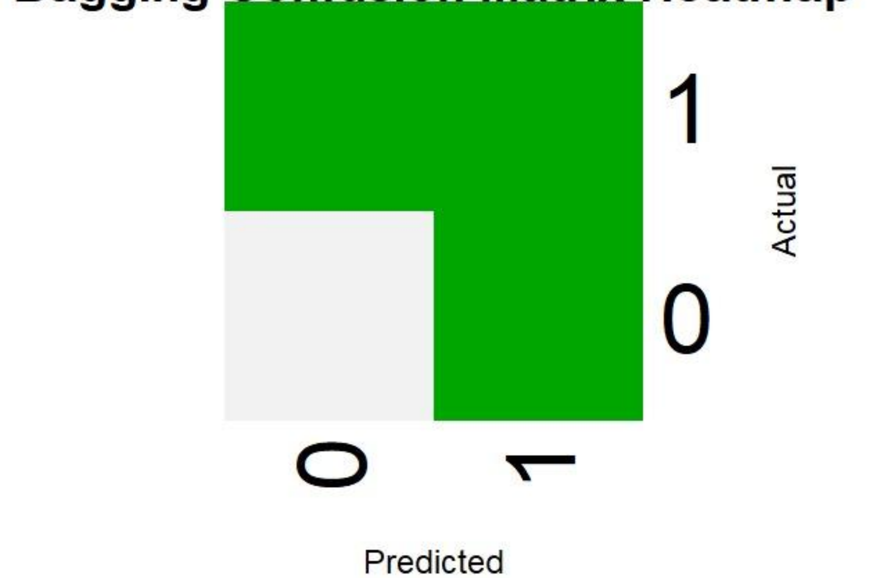
```
Statistics for Gradient Boosting classifier:
Confusion Matrix:
 [[4276   53]
 [ 113   58]]
Accuracy: 0.9631111111111111
Precision: 0.5225225225225225
Recall: 0.3391812865497076
F1 Score: 0.41134751773049644
```



BAGGING

```
bag_model <- bagging(formula = subscription~., data = train,
                     nbagg=50, coob=TRUE)
```

IMPLEMENTATION IN R



Bagging Confusion Matrix Heatmap

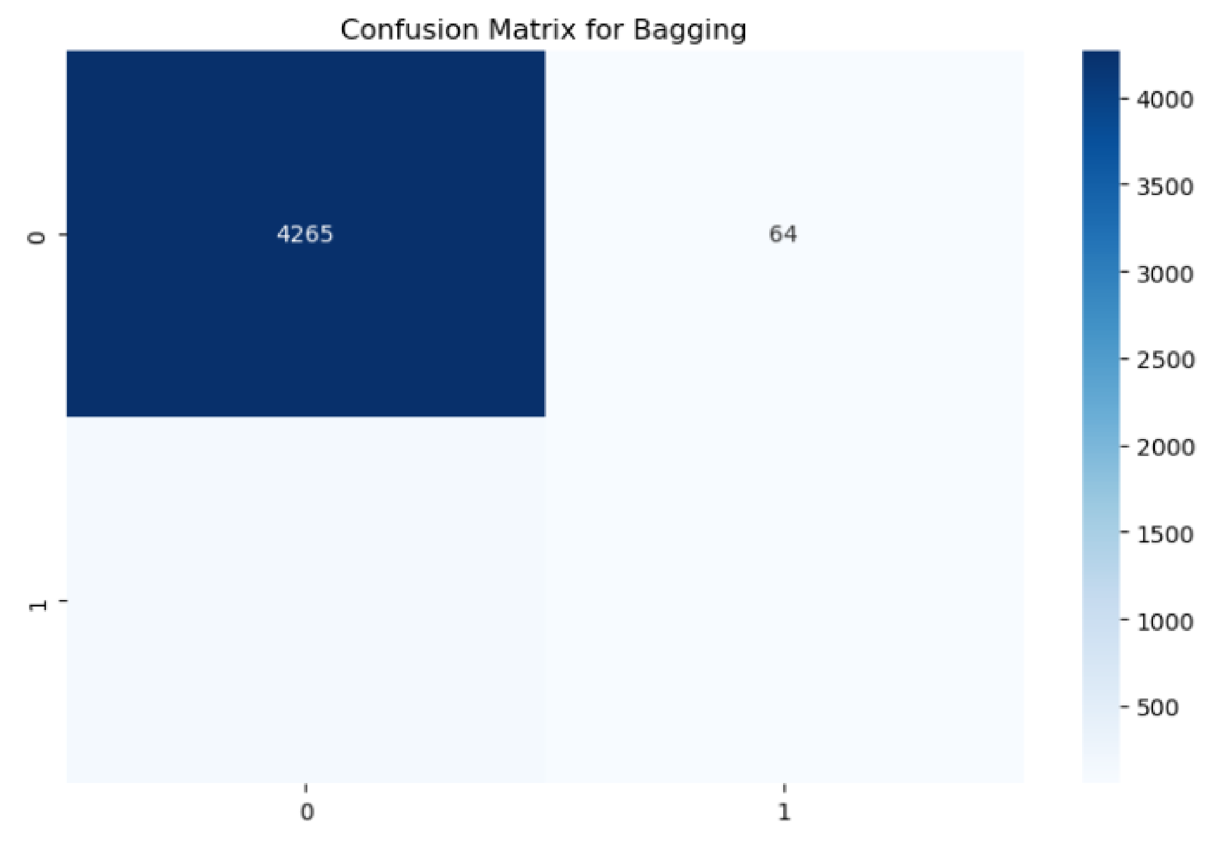
```
> confusionMatrix (original, bag_pred)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 4745   64
         1  103   49

               Accuracy : 0.9663
                 95% CI : (0.9609, 0.9712)
    No Information Rate : 0.9772
    P-Value [Acc > NIR] : 0.999999

                  Kappa : 0.3529

 Mcnemar's Test P-Value : 0.003277

            Sensitivity : 0.9788
            Specificity : 0.4336
         Pos Pred Value : 0.9867
         Neg Pred Value : 0.3224
             Prevalence : 0.9772
         Detection Rate : 0.9565
   Detection Prevalence : 0.9694
      Balanced Accuracy : 0.7062

       'Positive' Class : 0
```

IMPLEMENTATION IN PYTHON

It Builds multiple models independently and then combines their predictions through averaging (for regression) or voting.

```
model <- bagging(subscription ~ ., data = train_data, nbagg = 50)
```

```
Statistics for Bagging classifier:
Confusion Matrix:
 [[4263   66]
 [ 126   45]]
Accuracy: 0.9573333333333334
Precision: 0.40540540540540543
Recall: 0.2631578947368421
F1 Score: 0.3191489361702128
```

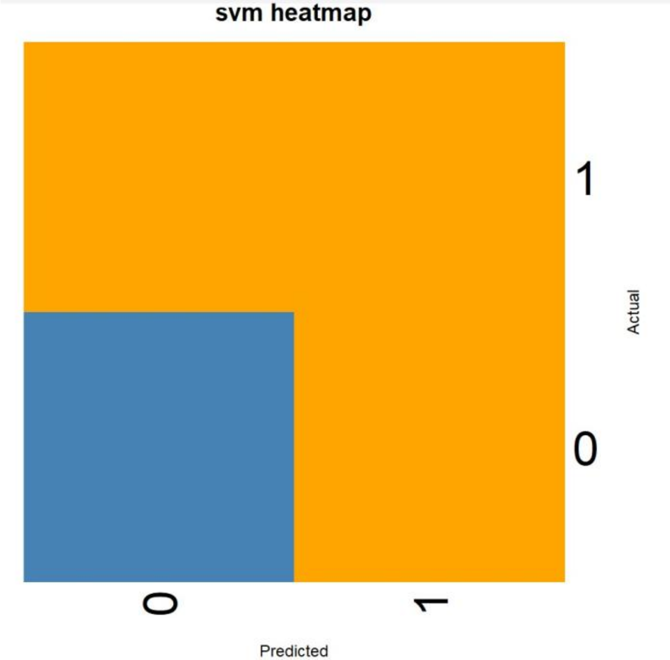


SUPPORT VECTOR MACHINE

IMPLEMENTATION IN R

```
svm1 <- svm(subscription~., data = train, kernel = "radial")
```

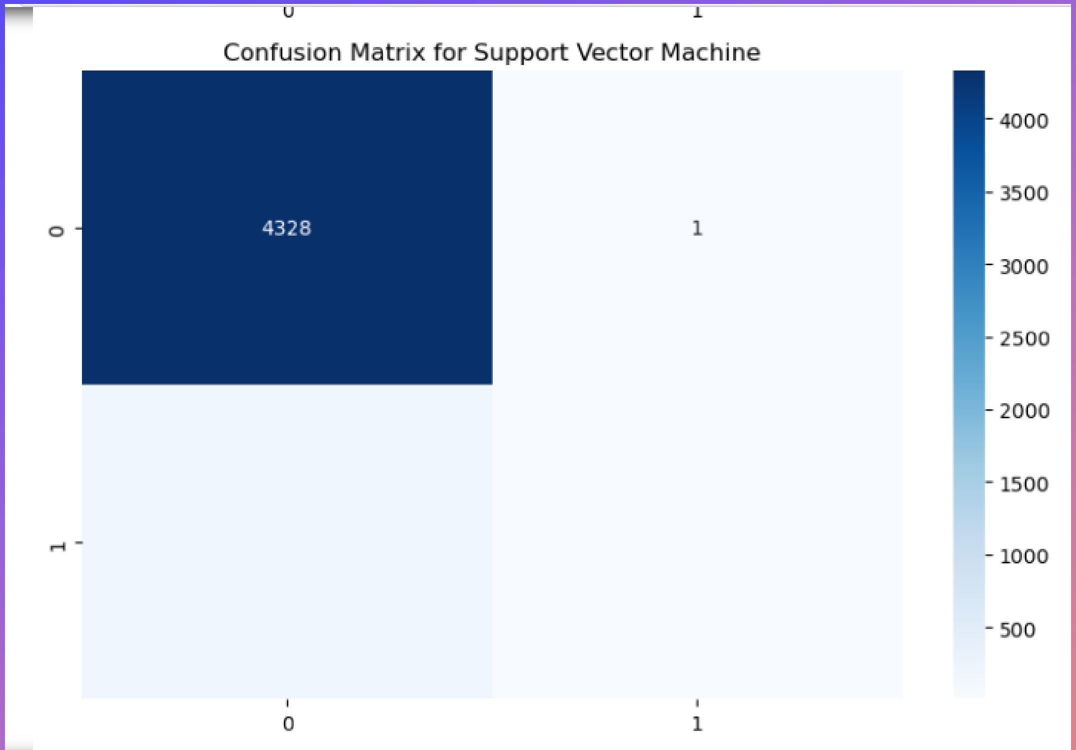
```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 11626   855
         1   524   724

               Accuracy : 0.8996
                 95% CI : (0.8944, 0.9045)
    No Information Rate : 0.885
    P-Value [Acc > NIR] : 2.711e-08

                  Kappa : 0.4571

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9569
            Specificity : 0.4585
         Pos Pred Value : 0.9315
         Neg Pred Value : 0.5801
             Prevalence : 0.8850
         Detection Rate : 0.8468
   Detection Prevalence : 0.9091
      Balanced Accuracy : 0.7077

       'Positive' Class : 0
```



IMPLEMENTATION IN PYTHON

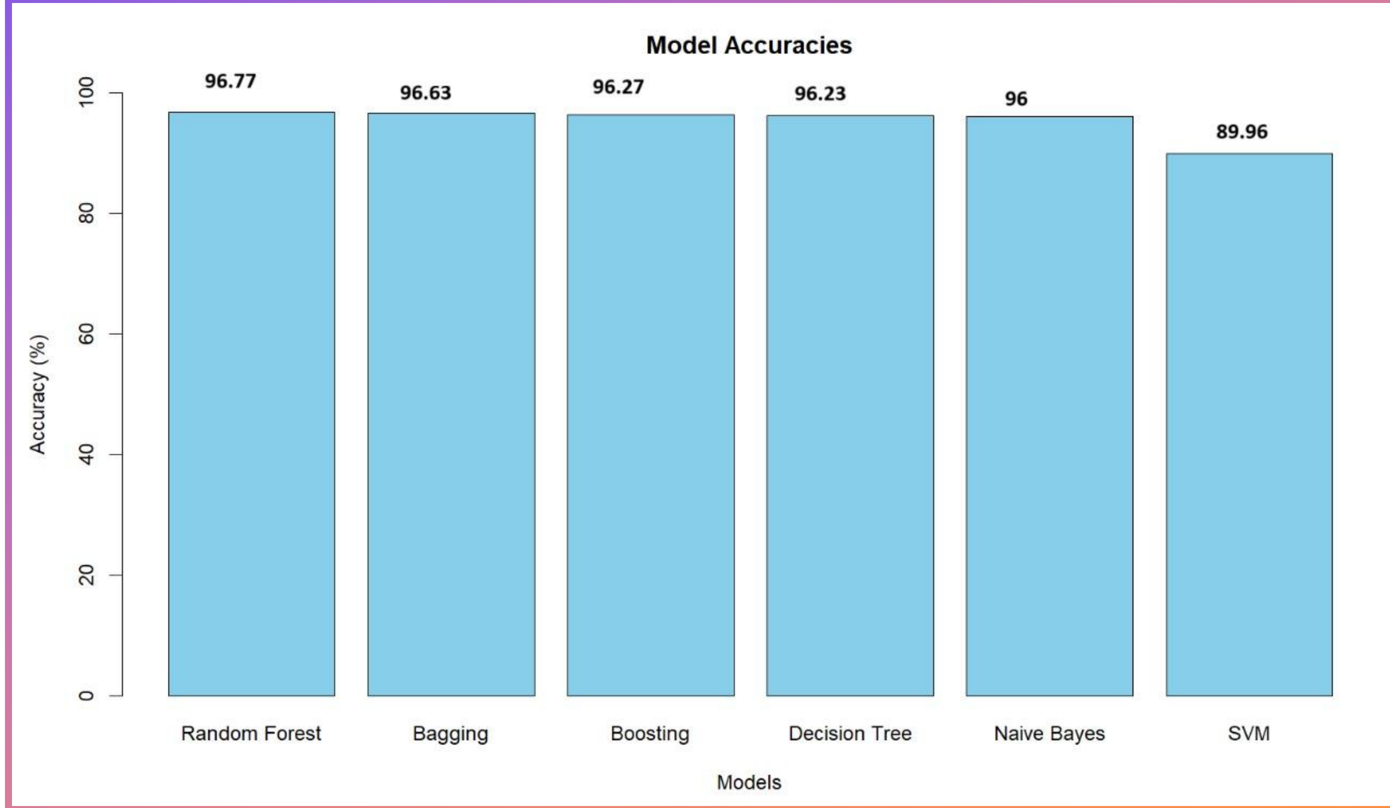
SVM aims to maximize the margin between classes, making it robust to outliers and effective in high-dimensional spaces.

```
"Support Vector Machine": SVC()
svm_classifier.fit(X_train, y_train)
```




```
Statistics for Support Vector Machine classifier:
Confusion Matrix:
 [[4328    1]
 [ 168    3]]
Accuracy: 0.9624444444444444
Precision: 0.75
Recall: 0.017543859649122806
F1 Score: 0.03428571428571429
```

COMPARISON OF ALL CATEGORICAL MODELS



NUMERICAL VALUES CLASSIFICATION

METHODS USED IN R

KNN - The 'k' nearest data points, determined by a distance metric such as Euclidean distance, are considered for classification, We Implemented this classification with various k values and found the best accuracy for k=22

NAÏVE BAYES – We have tried this specific method for comparison of the accuracy with other numerical categories.

METHODS USED IN PYTHON

KNN

NAÏVE BAYES

DECISION TREE

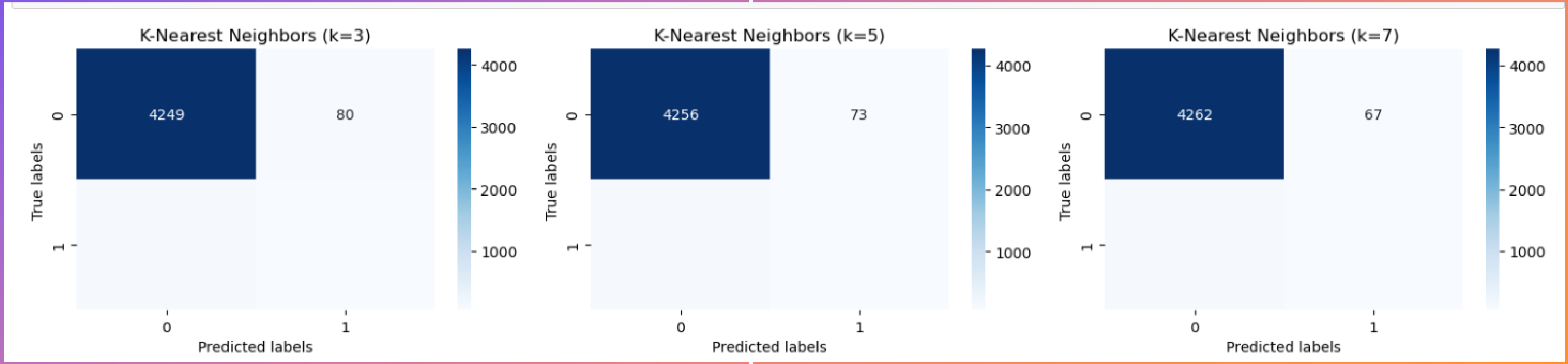
This iterates over various k values (ODD NUMBERS – 3,5,7) for KNN and creates multiple instances of Decision Tree and Naive Bayes classifiers. By training and evaluating these models on the same dataset, it enables direct comparison of their performance metrics, aiding in selecting the most suitable algorithm for the classification task.

KNN

IN PYTHON

```python
classifiers = {
    "K-Nearest Neighbors": [KNeighborsClassifier(n_neighbors=k) for k in k_values],
```

```
Statistics for K-Nearest Neighbors (Model 1):
Confusion Matrix:
 [[4249   80]
 [ 112   59]]
Accuracy: 0.9573333333333334
Precision: 0.4244604316546763
Recall: 0.34502923976608185
F1 Score: 0.38064516129032255

Statistics for K-Nearest Neighbors (Model 2):
Confusion Matrix:
 [[4256   73]
 [ 112   59]]
Accuracy: 0.9588888888888889
Precision: 0.44696969696969696
Recall: 0.34502923976608185
F1 Score: 0.38943894389438943

Statistics for K-Nearest Neighbors (Model 3):
Confusion Matrix:
 [[4262   67]
 [ 116   55]]
Accuracy: 0.9593333333333334
Precision: 0.45081967213114754
Recall: 0.3216374269005848
F1 Score: 0.37542662116040953
```

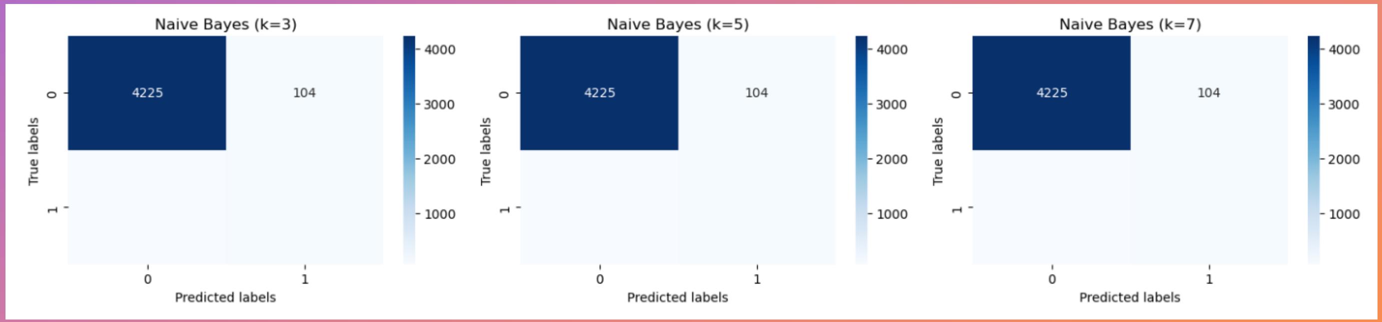


NAÏVE BAYES

IN PYTHON

```python
classifiers = {
    "Naive Bayes": [GaussianNB() for _ in range(len
}
```

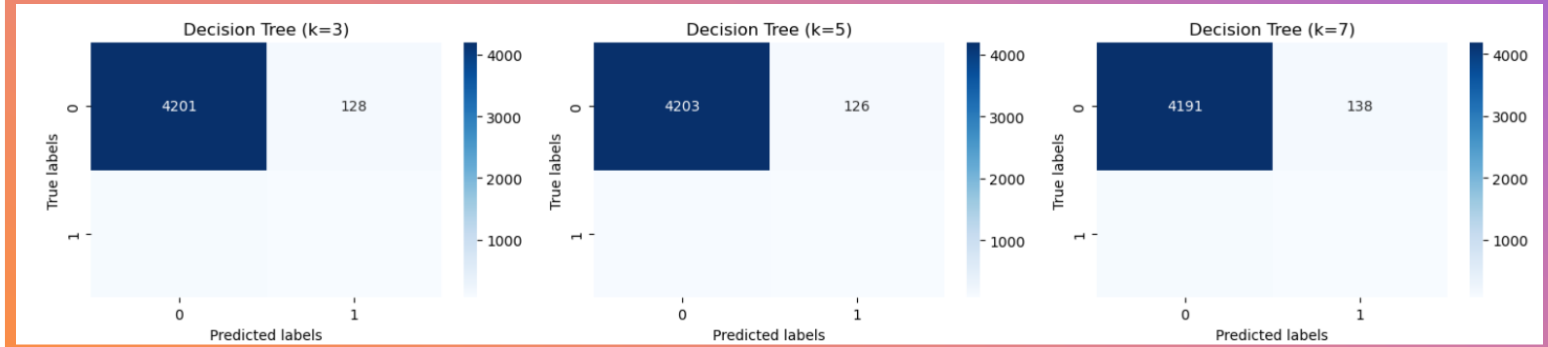
```
Statistics for Naive Bayes (Model 1):
Confusion Matrix:
 [[4225  104]
 [  68  103]]
Accuracy: 0.9617777777777777
Precision: 0.4975845410628019
Recall: 0.6023391812865497
F1 Score: 0.544973544973545

Statistics for Naive Bayes (Model 2):
Confusion Matrix:
 [[4225  104]
 [  68  103]]
Accuracy: 0.9617777777777777
Precision: 0.4975845410628019
Recall: 0.6023391812865497
F1 Score: 0.544973544973545

Statistics for Naive Bayes (Model 3):
Confusion Matrix:
 [[4225  104]
 [  68  103]]
Accuracy: 0.9617777777777777
Precision: 0.4975845410628019
Recall: 0.6023391812865497
F1 Score: 0.544973544973545
```



```python
classifiers = {

    "Decision Tree": [DecisionTreeClassifier() for _ in range(len
```

DECISION TREE

IN PYTHON

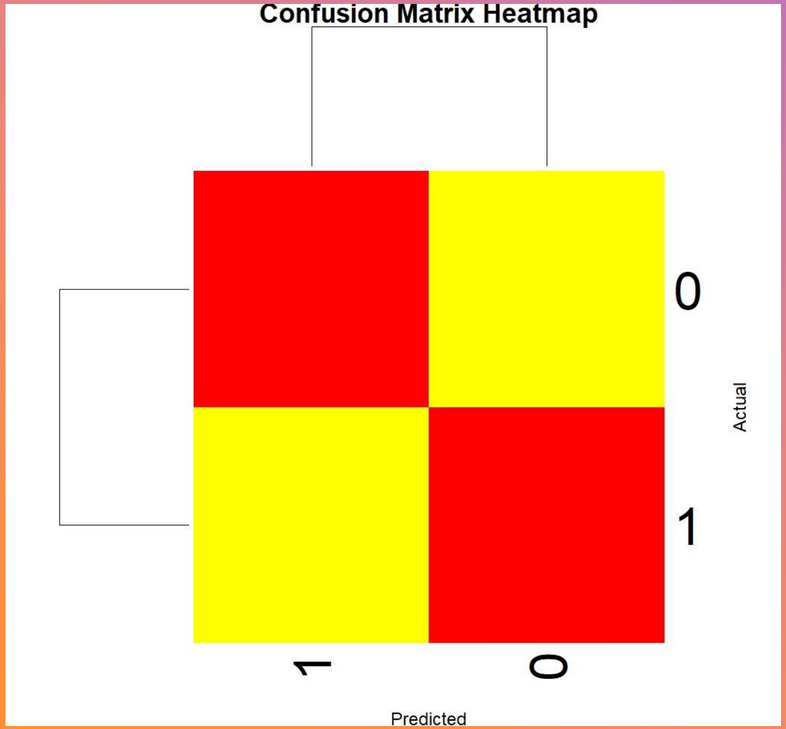



```
Statistics for Decision Tree (Model 1):
Confusion Matrix:
 [[4195  134]
 [ 109   62]]
Accuracy: 0.946
Precision: 0.3163265306122449
Recall: 0.36257309941520466
F1 Score: 0.33787465940054495

Statistics for Decision Tree (Model 2):
Confusion Matrix:
 [[4197  132]
 [ 107   64]]
Accuracy: 0.9468888888888889
Precision: 0.32653061224489793
Recall: 0.3742690058479532
F1 Score: 0.34877384196185285

Statistics for Decision Tree (Model 3):
Confusion Matrix:
 [[4207  122]
 [ 103   68]]
Accuracy: 0.95
Precision: 0.35789473684210527
Recall: 0.39766081871345027
F1 Score: 0.3767313019390582
```

KNN

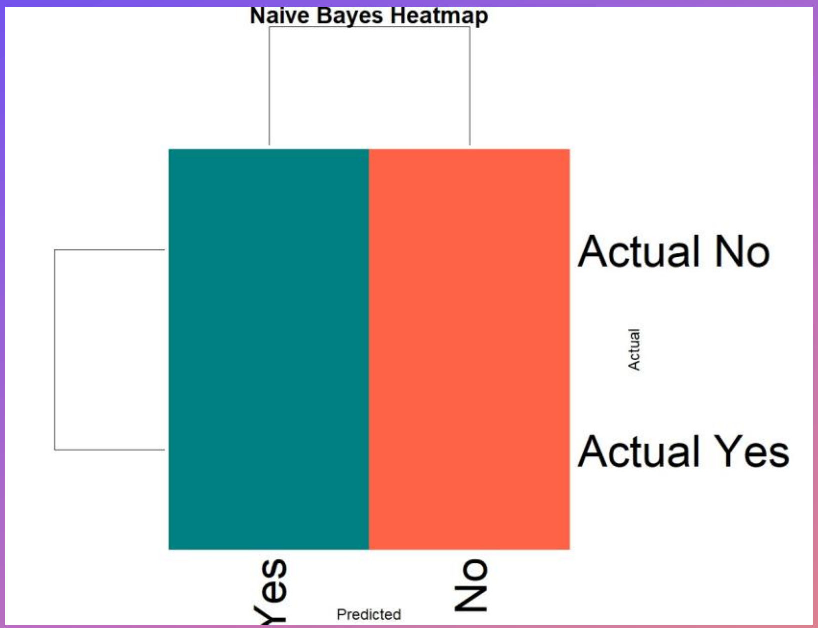
IMPLEMENTATION IN R

```r
# Fitting KNN Model to training dataset
classifier_knn <- knn(train = train_scale, test = test_scale,
                      cl = train$subscription, k = 5)
classifier_knn <- knn(train = train_scale, test = test_scale,
                      cl = train$subscription, k = 10)
classifier_knn <- knn(train = train_scale, test = test_scale,
                      cl = train$subscription, k = 22)
```




```
> confusionMatrix(classifier_knn, actual_labels)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 11798   844
         1   374   713

               Accuracy : 0.9113
                 95% CI : (0.9064, 0.916)
    No Information Rate : 0.8866
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.492

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9693
            Specificity : 0.4579
         Pos Pred Value : 0.9332
         Neg Pred Value : 0.6559
             Prevalence : 0.8866
         Detection Rate : 0.8593
   Detection Prevalence : 0.9208
      Balanced Accuracy : 0.7136

       'Positive' Class : 0
```

NAÏVE BAYES IMPLEMENTATION IN R

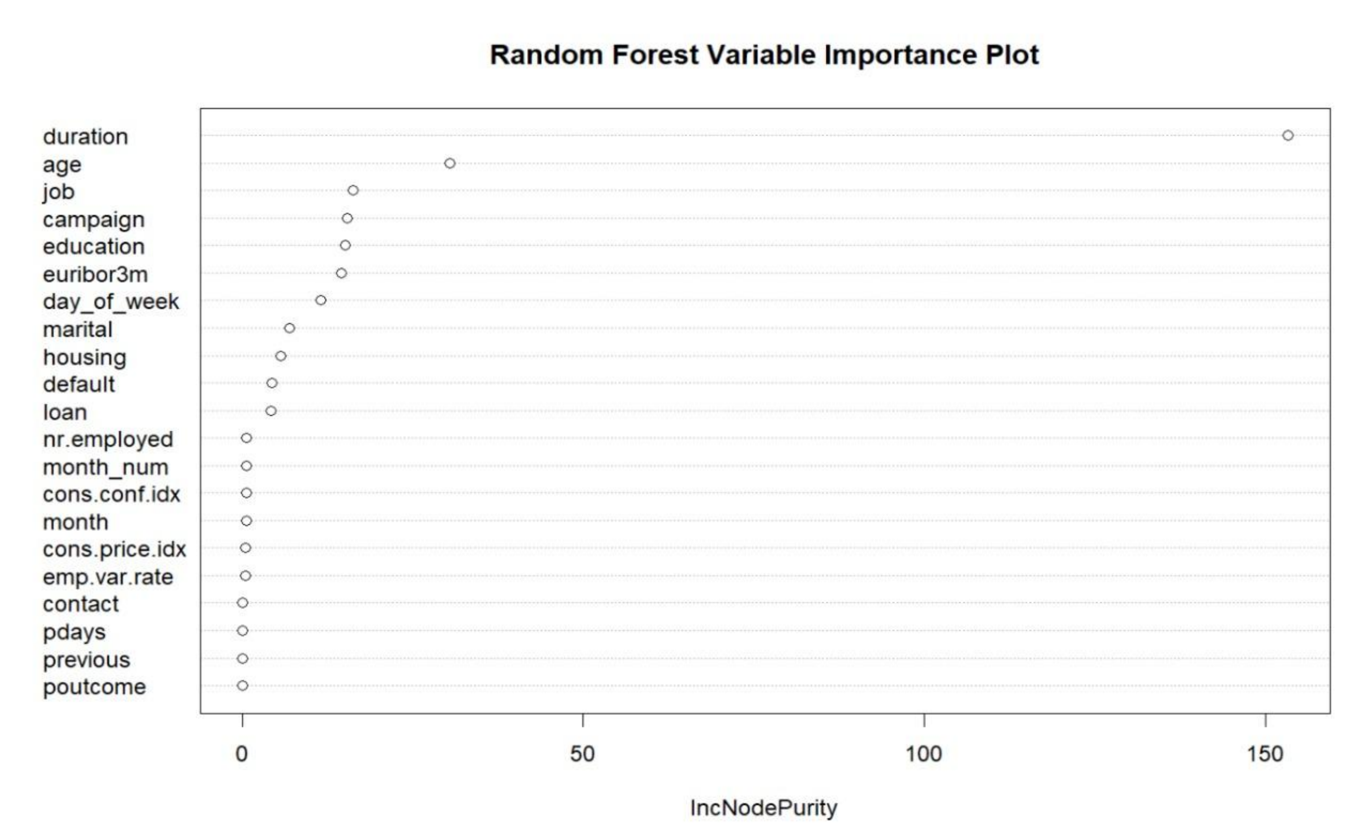



```
# Fitting Naive Bayes Model to training dataset
classifier_nb <- naiveBayes(x = train_scale, y = train$subscription)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 11067   634
         1  1105   923

               Accuracy : 0.8733
                 95% CI : (0.8677, 0.8789)
    No Information Rate : 0.8866
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4435

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9092
            Specificity : 0.5928
         Pos Pred Value : 0.9458
         Neg Pred Value : 0.4551
             Prevalence : 0.8866
         Detection Rate : 0.8061
   Detection Prevalence : 0.8523
      Balanced Accuracy : 0.7510

       'Positive' Class : 0
```

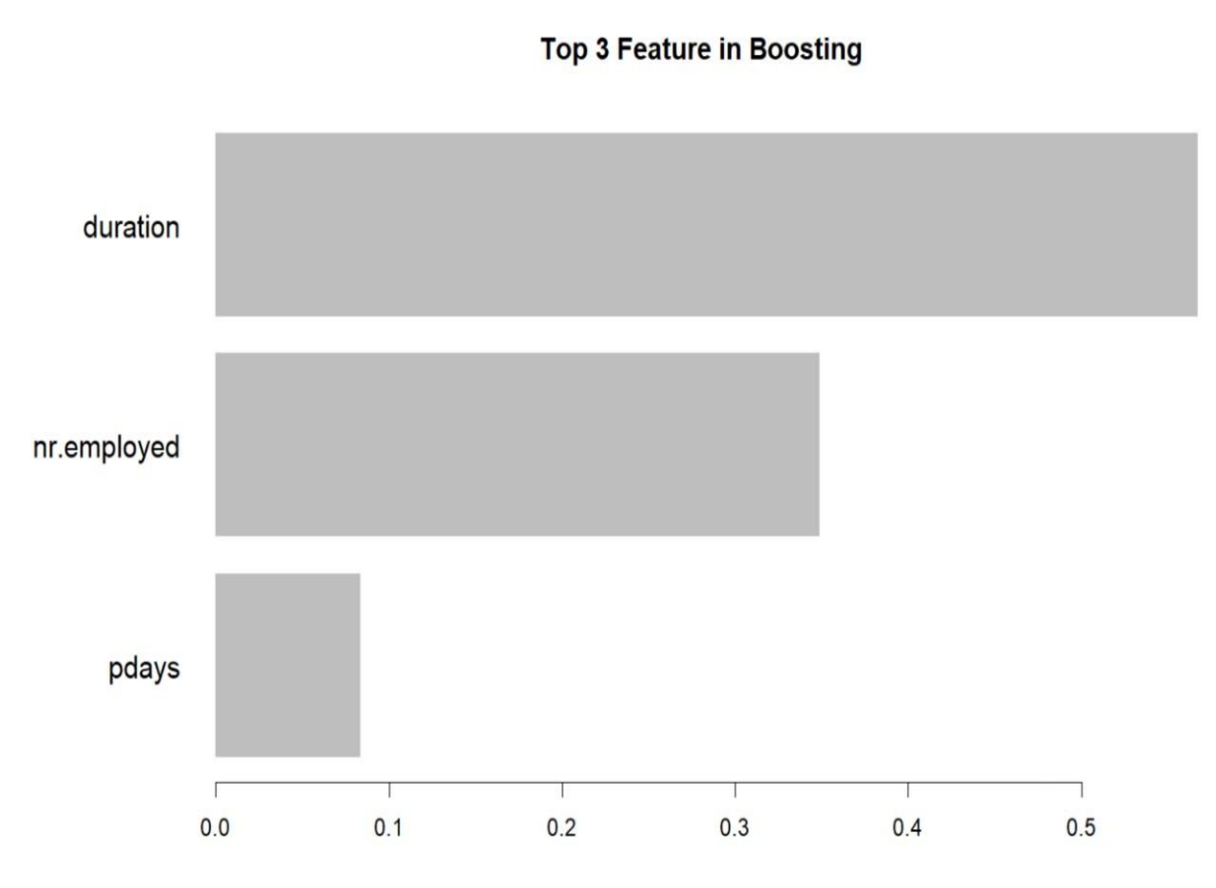
FEATURE IMPORTANCE CALCULATION USING RANDOM FOREST METHOD

```
# Fitting Random Forest Model
rf_model <- randomForest(subscription ~ ., data = bank_data)
# Plotting variable importance
varImpPlot(rf_model, main = "Random Forest Variable Importance Plot")
```



FEATURE IMPORTANCE CALCULATION USING BOOSTING

```
# Plotting feature importance
importance_matrix <- xgb.importance(feature_names = colnames(train_data[, -1]),
                                    model = boost_model)
xgb.plot.importance(importance_matrix[1:3], main = "Top 3 Feature in Boosting")
```

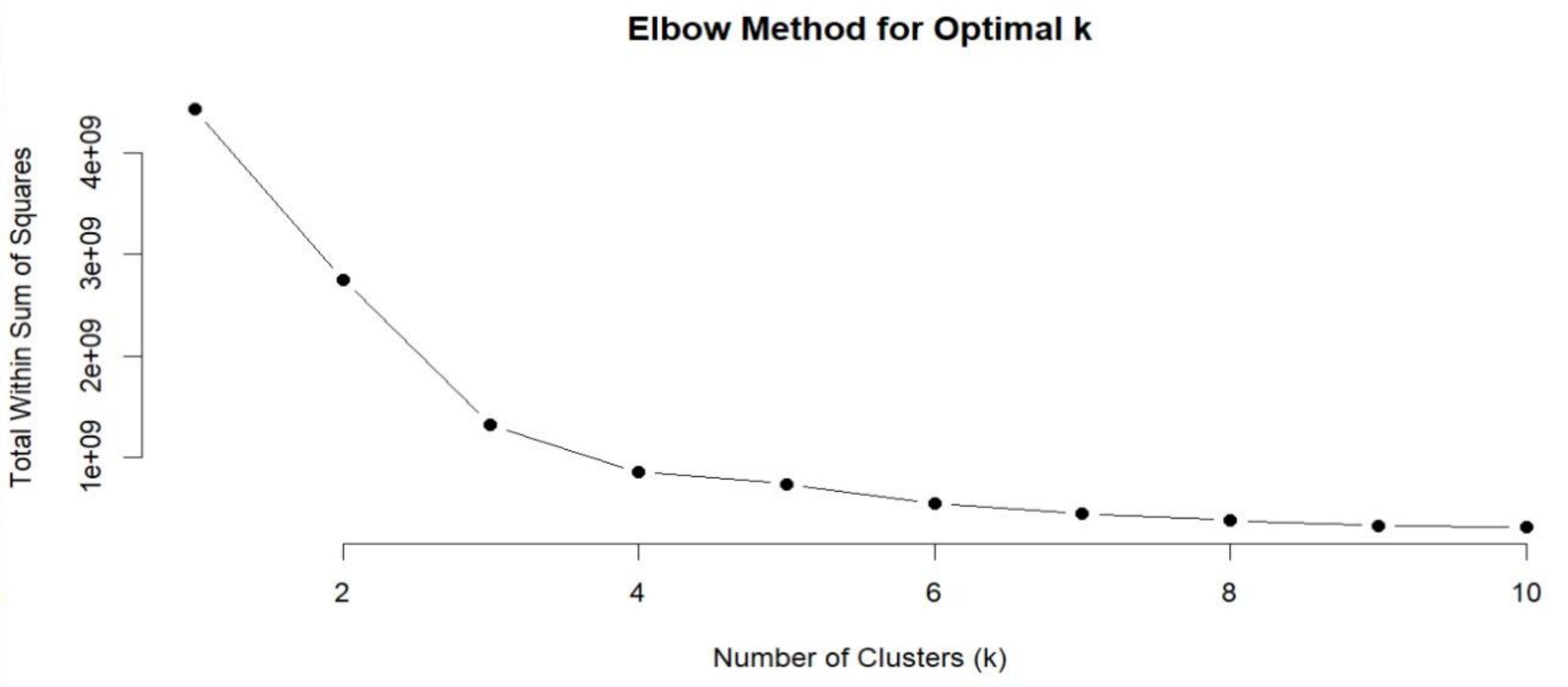


```r
# Identify categorical and numerical variables
categorical_vars <- data[, sapply(data, is.factor)]
numerical_vars <- data[, sapply(data, is.numeric)]

# Create datasets for categorical and numerical variables
categorical_data <- data[, sapply(data, is.factor)]
numerical_data <- data[, sapply(data, is.numeric)]
```

CLUSTERING

- K MEANS - KMeans clustering is a popular unsupervised machine learning algorithm used for clustering data points into a predefined number of clusters. The algorithm aims to partition the data into clusters such that the similarity within each cluster is maximized, while the similarity between clusters is minimized.

```r
# Function to calculate total within-cluster sum of squares
wss <- function(k) {
  kmeans_model <- kmeans(numerical_data, centers = k)
  return(sum(kmeans_model$withinss))
}

# Compute within-cluster sum of squares for different values of k
k_values <- 1:10
wss_values <- sapply(k_values, wss)

# Plot the elbow curve
plot(k_values, wss_values, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Clusters (k)", ylab = "Total Within Sum of Squares",
     main = "Elbow Method for Optimal k")
```



```r
#K means for clusters = 3 from elbow curve
kmeans_model <- kmeans(numerical_data,3)
kmeans_model
```

R 4.3.2 · ~/R Studio Works/ALY 6040/Final Project/

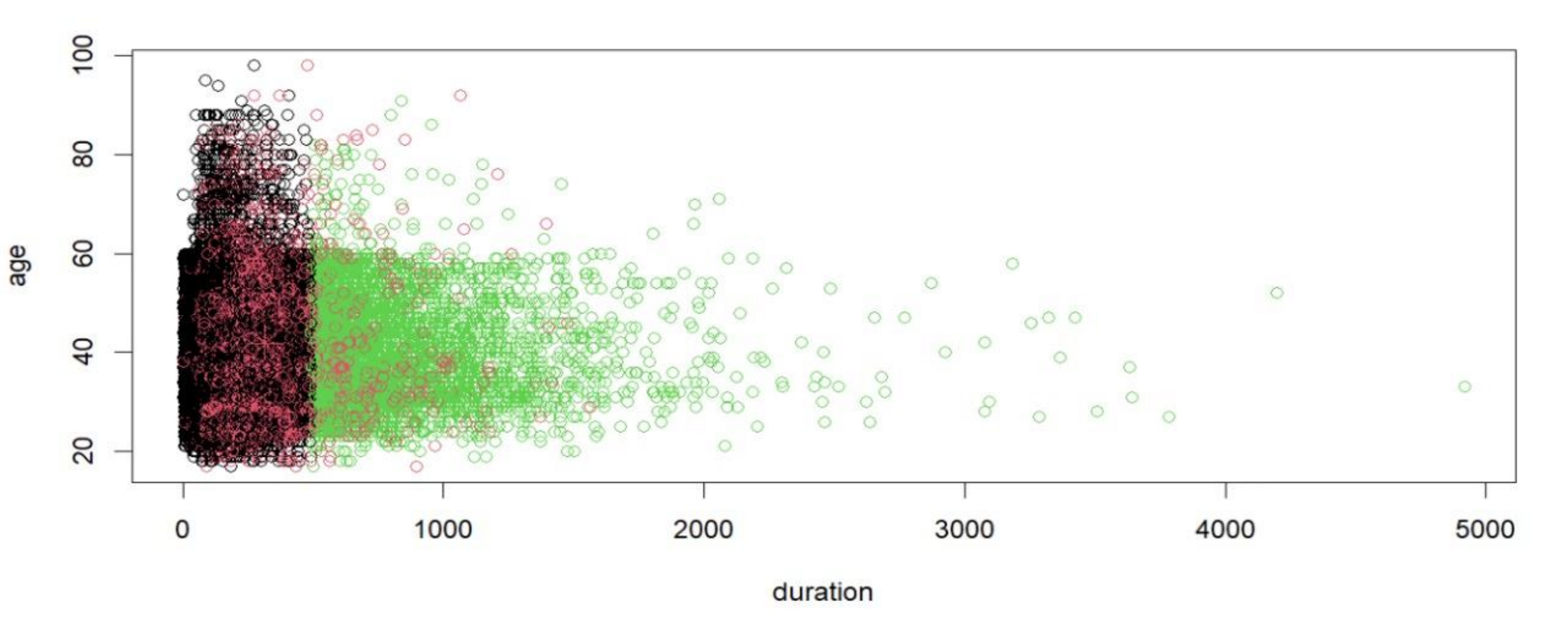
```
> kmeans_model <- kmeans(numerical_data,3)
> kmeans_model
K-means clustering with 3 clusters of sizes 34859, 1509, 4820

Cluster means:
       age duration campaign days_after_contacted_p  previous previous_interest_rate
1 39.94765 180.5454 2.617201               999.0000 0.1181331              0.1662842
2 41.86547 314.5427 1.821074                 6.0000 1.6626905             -2.0964877
3 40.00021 802.8973 2.442531               997.7685 0.1031120              0.1534855
  cons.price.idx cons.conf.idx euribor3m nr.employed          y
1       93.58295     -40.55035 3.7252459    5172.459 0.05020224
2       93.34247     -38.33227 0.9859768    5029.251 0.63817097
3       93.59599     -40.83676 3.6945102    5170.954 0.39979253

Clustering vector:
   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1
  [41] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 3 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1
  [81] 1 1 1 3 1 1 3 1 3 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 3 1 1
 [121] 1 1 1 1 1 1 1 1 1 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 3
 [161] 1 1 1 3 3 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 3 3
 [201] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
 [241] 1 1 1 3 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 3 1 1 3 1 1 1
 [281] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [321] 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1
 [361] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 1 1 1 1 3 1 1 1 1
 [401] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1 3 1 1 1 3 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1
```

Console | Terminal | Background Jobs

R 4.3.2 · ~/R Studio Works/ALY 6040/Final Project/

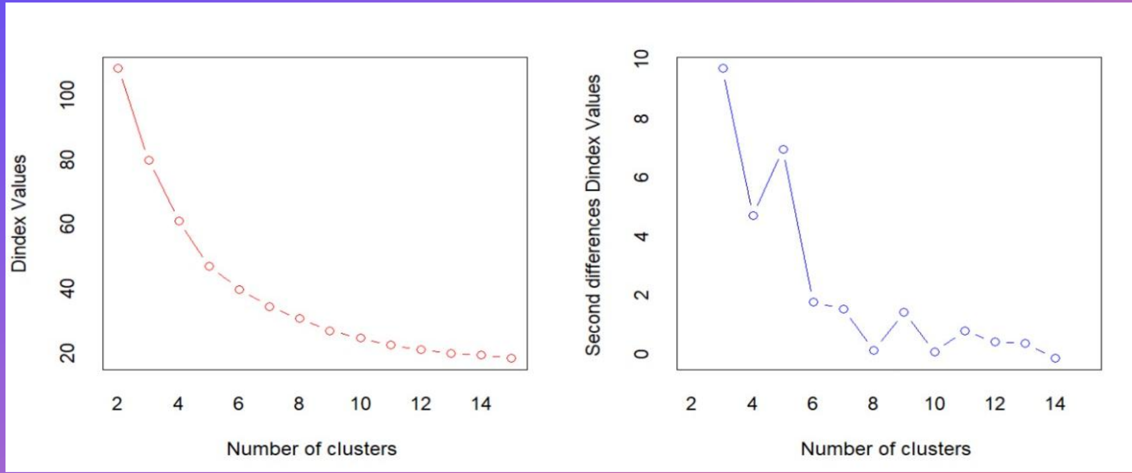
```
 [521] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1
 [561] 1 3 1 1 1 1 1 1 1 1 3 1 1 1 1 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
 [601] 1 1 1 1 1 1 1 1 1 1 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 3 1 1
 [641] 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [681] 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 3 1 1 1 3
 [721] 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [761] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [801] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1
 [841] 3 1 1 1 3 3 1 1 1 1 1 1 3 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 3 1 1 1
 [881] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1
 [921] 1 1 1 1 1 3 1 1 1 1 1 3 1 1 1 1 1 3 1 1 1 1 1 3 1 1 1 1 3 1 1 1 3 3 1 1 1 1 1 1
 [961] 3 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1
 [ reached getOption("max.print") -- omitted 40188 entries ]

Within cluster sum of squares by cluster:
[1] 614685211  81769087 624339349
 (between_SS / total_SS =  70.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
>
```

```
> table(numerical_data$y,kmeans_model$cluster)

         1     2     3
  0 33109   546  2893
  1  1750   963  1927
> plot(numerical_data[c("duration","age")], col=kmeans_model$cluster)
> points(kmeans_model$centers[,c("duration","age")], col=1:3, pch=8, cex=2)
> 
```

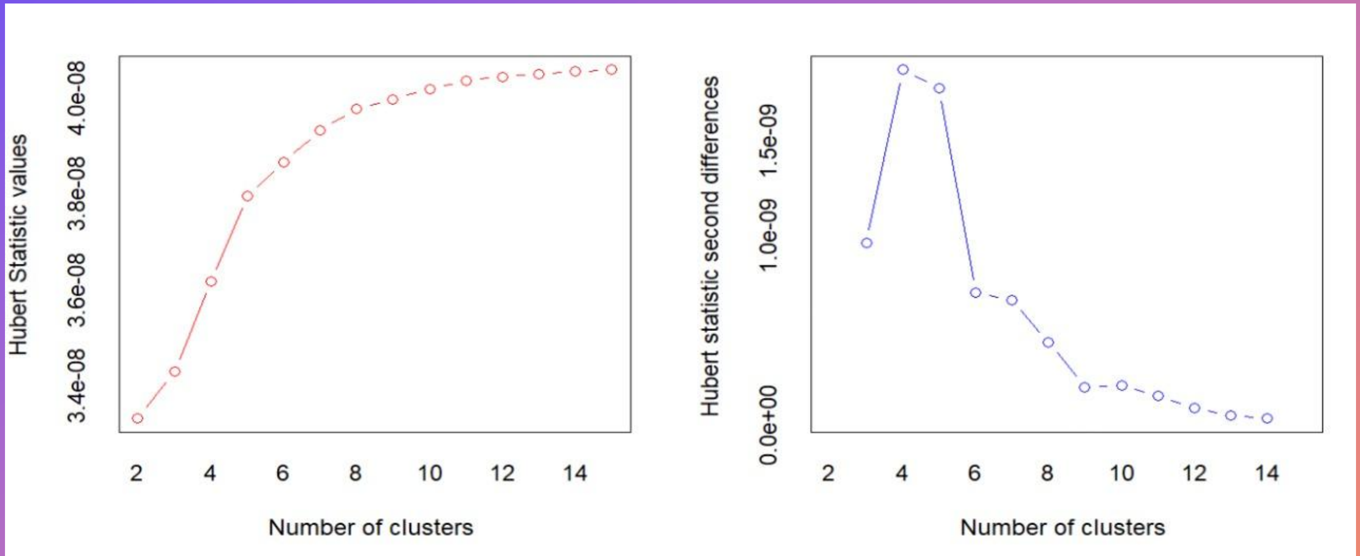


NB CLUST METHOD

```r
library(cluster)
library(NbClust)

# Perform NbClust analysis
numerical_data_subset <- numerical_data[1:1000,1:3 ]

library(caret)

  # Perform cluster analysis
nbclust_result <- NbClust(data = numerical_data_subset, min.r
Bar<-table(nbclust_result$Best.n[1,])
Bar
```




```
> Bar<-table(nbclust_result$Best.n[1,])
> Bar

 0  2  3  4  6  9 11 15 
 2  8  4  1  1  1  7  2 
> 
```

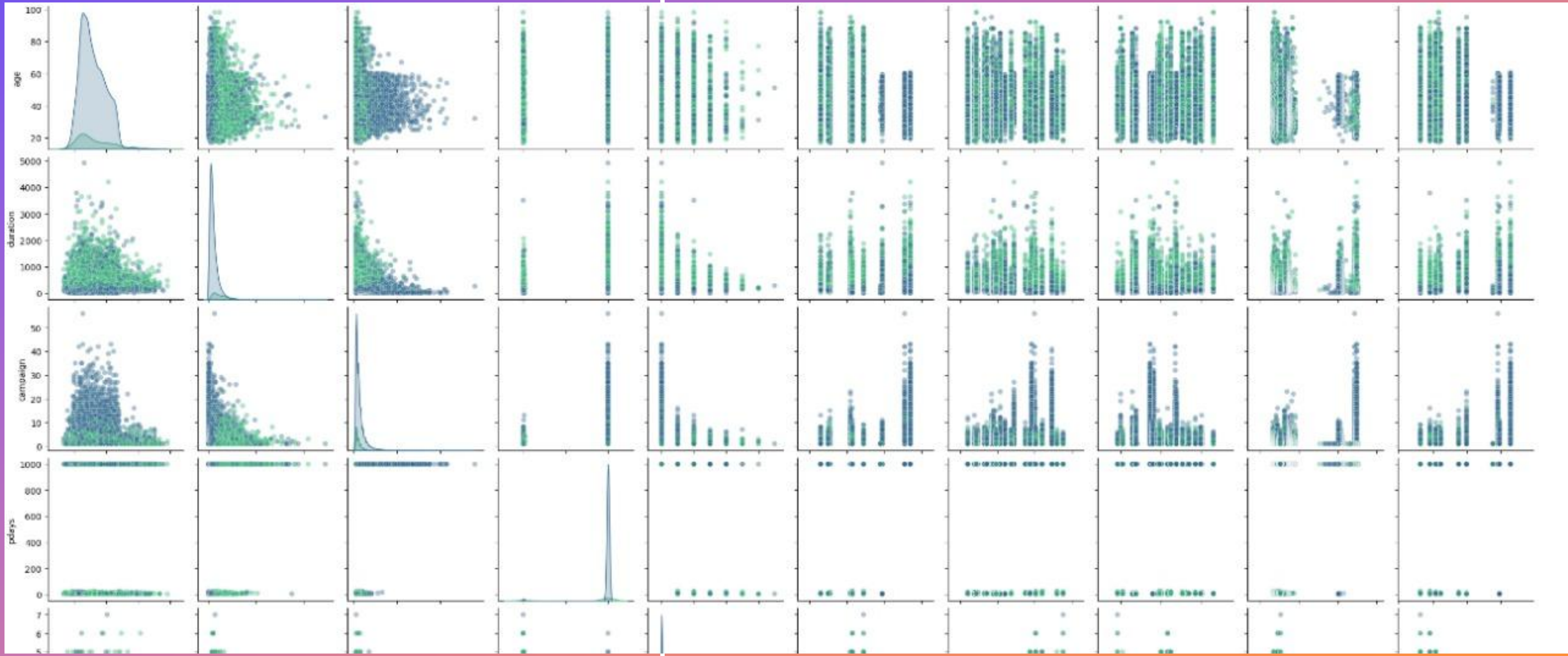



```
> nbclust_result <- NbClust(data = numerical_data_subset, min.nc = 2, max.nc = 15, method = "kmean
s")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a 
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot. 
 
*** : The D index is a graphical method of determining the number of clusters. 
                In the plot of D index, we seek a significant knee (the significant peak in Dindex
                second differences plot) that corresponds to a significant increase of the value of
                the measure. 
 
******************************************************************* 
* Among all indices:                                                
* 8 proposed 2 as the best number of clusters 
* 4 proposed 3 as the best number of clusters 
* 1 proposed 4 as the best number of clusters 
* 1 proposed 6 as the best number of clusters 
* 1 proposed 9 as the best number of clusters 
* 7 proposed 11 as the best number of clusters 
* 2 proposed 15 as the best number of clusters 

                   ***** Conclusion *****                            
 
* According to the majority rule, the best number of clusters is  2 
```

IMPLEMENTATION IN PYTHON

```
data['y'] = data['y'].apply(lambda x: 1 if x == 'yes' else 0)
```

```
sns.pairplot(data, hue="y", palette="viridis", plot_kws={"alpha": 0.4})
plt.show()
```



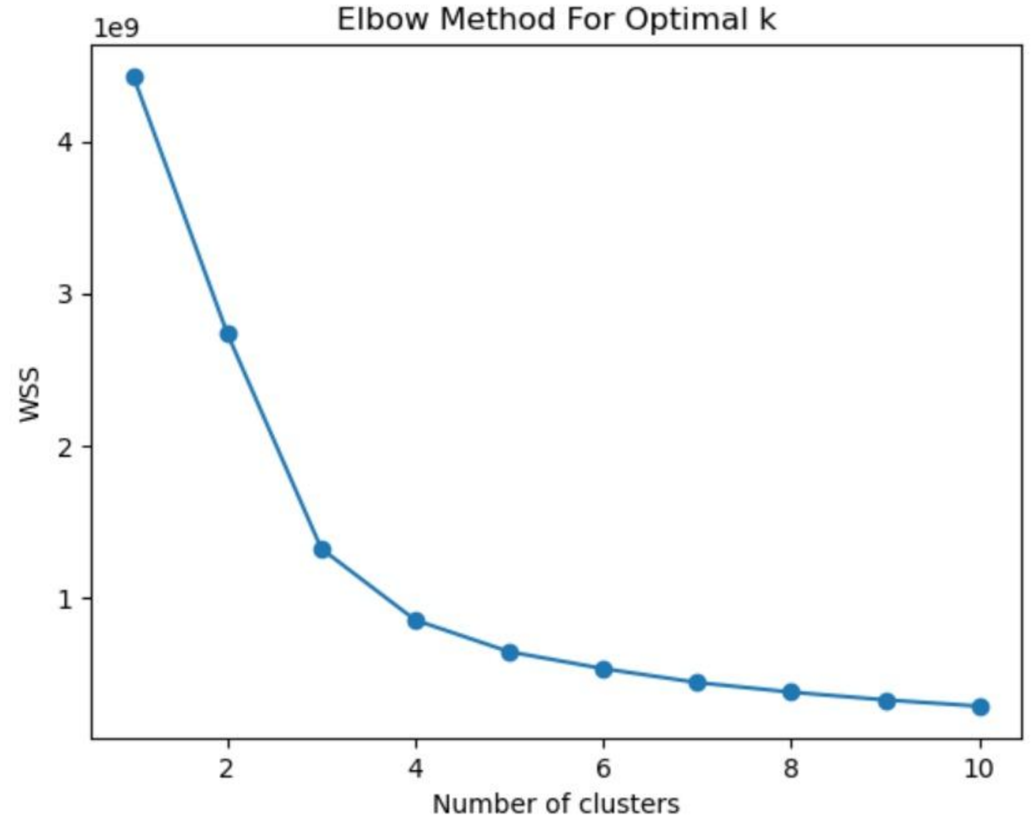
WITHIN CLUSTER SUM OF SQUARE

```python
wss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)  # Explicitly set n_init
    kmeans.fit(numerical_data)
    wss.append(kmeans.inertia_)
```

```python
plt.plot(range(1, 11), wss, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('WSS')
plt.show()
```

```python
k_opt = 3
kmeans_opt = KMeans(n_clusters=k_opt, random_state=42)
kmeans_opt.fit(numerical_data)
print("Cluster means:")
print(kmeans_opt.cluster_centers_)
```

```
C:\Users\trill\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init`
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Cluster means:
[[ 3.99981351e+01  8.02509532e+02  2.44135930e+00  9.97769996e+02
   1.03398259e-01  1.50497306e-01  9.35950253e+01 -4.08389142e+01
   3.69140489e+00  5.17084279e+03  3.99709905e-01]
 [ 3.99479241e+01  1.80491981e+02  2.61739305e+00  9.99000000e+02
   1.18096003e-01  1.66700141e-01  9.35830800e+01 -4.05499986e+01
   3.72568120e+00  5.17247435e+03  5.01535019e-02]
 [ 4.18654738e+01  3.14542744e+02  1.82107356e+00  6.00000000e+00
   1.66269052e+00 -2.09648774e+00  9.33424679e+01 -3.83322730e+01
   9.85976806e-01  5.02925070e+03  6.38170974e-01]]
```



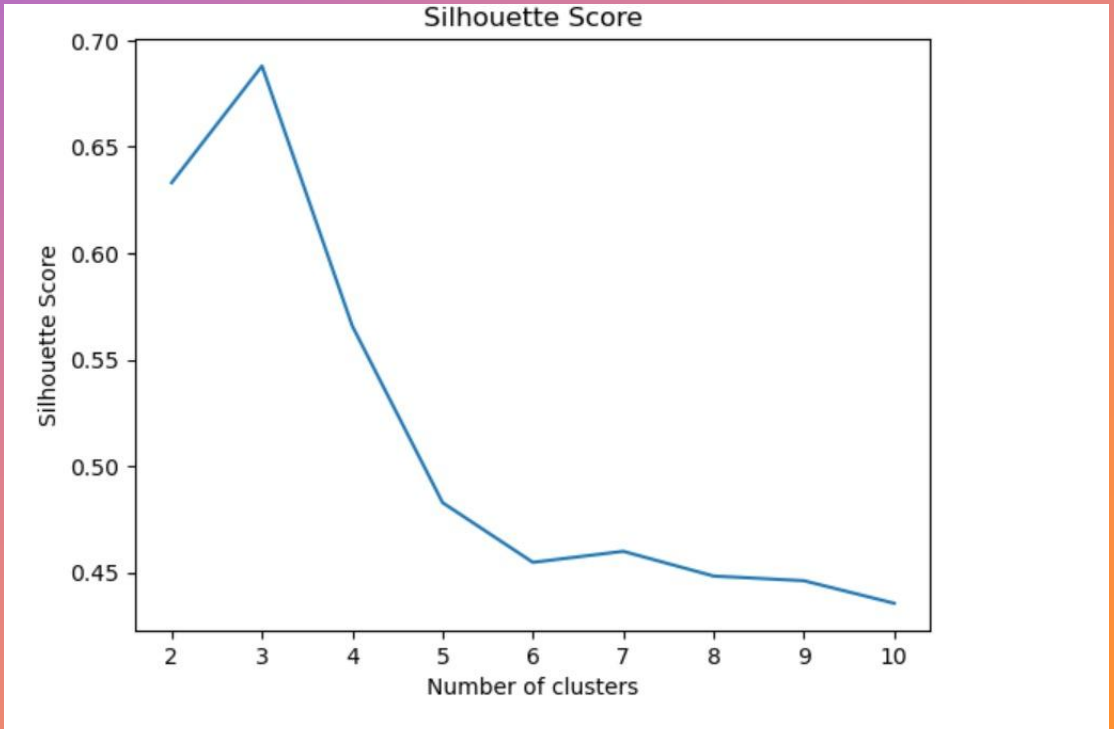
SILHOUETTE METHOD

```python
# Silhouette Score
silhouette_scores = []
for n_clusters in range(2, 11):
    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', max_iter=300, n_init=10, random_state=0)
    cluster_labels = kmeans.fit_predict(numerical_data)
    silhouette_avg = silhouette_score(numerical_data, cluster_labels)
    silhouette_scores.append(silhouette_avg)
silhouette = silhouette_scores
```

```
[0.6331208146929761, 0.6879613893051137, 0.5658051326698728, 0.48280792718349985, 0.45476621438284814, 0.45994780696935533, 0.4483395658146482, 0.4461410
425019763, 0.435545925189039]
```

```python
print(silhouette)
```

```python
plt.plot(range(2, 11), silhouette_scores)
plt.title('Silhouette Score')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



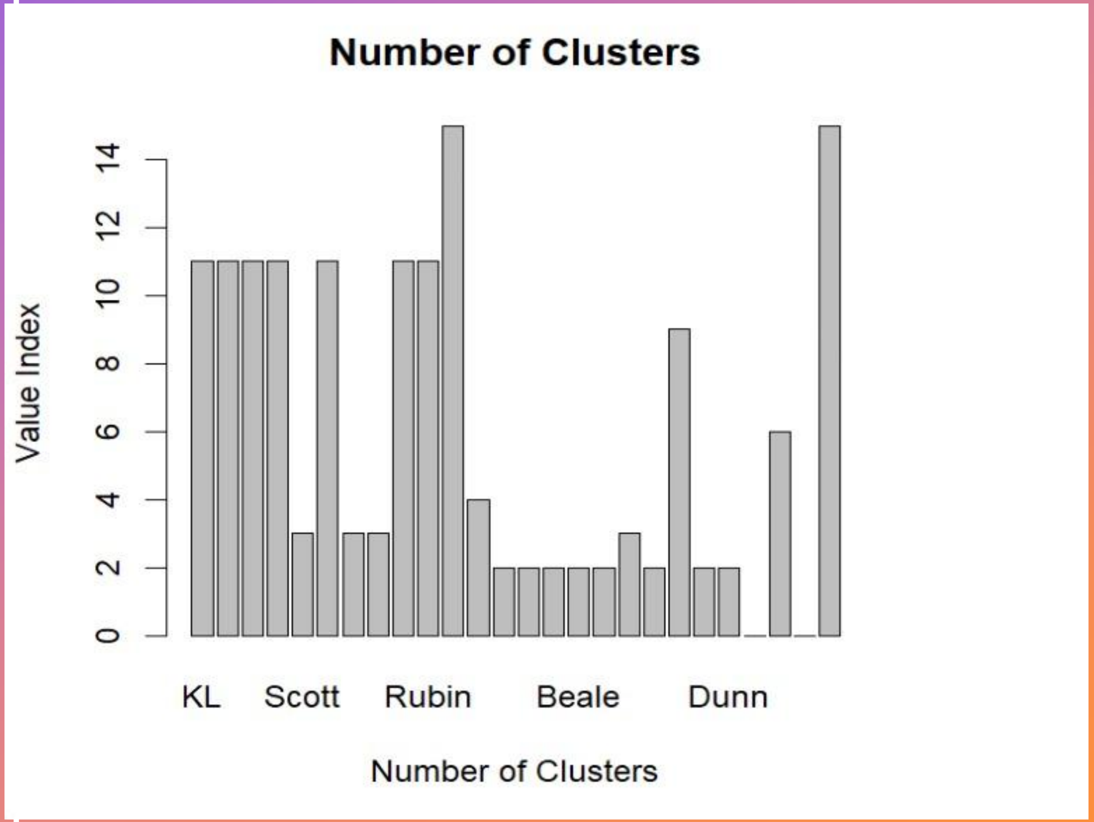

```python
k_opt = 3
kmeans_opt = KMeans(n_clusters=k_opt, random_state=42)
kmeans_opt.fit(numerical_data)
print("Cluster means:")
print(kmeans_opt.cluster_centers_)
```

```
C:\Users\trill\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Cluster means:
[[ 3.99981351e+01  8.02509532e+02  2.44135930e+00  9.97769996e+02
   1.03398259e-01  1.50497306e-01  9.35950253e+01 -4.08389142e+01
   3.69140489e+00  5.17084279e+03  3.99709905e-01]
 [ 3.99479241e+01  1.80491981e+02  2.61739305e+00  9.99000000e+02
   1.18096003e-01  1.66700141e-01  9.35830800e+01 -4.05499986e+01
   3.72568120e+00  5.17247435e+03  5.01535019e-02]
 [ 4.18654738e+01  3.14542744e+02  1.82107356e+00  6.00000000e+00
   1.66269052e+00 -2.09648774e+00  9.33424679e+01 -3.83322730e+01
   9.85976806e-01  5.02925070e+03  6.38170974e-01]]
```

IMPLEMENTATION IN R

PARTITIONING AROUND MEDOIDS WITH **K-MEDOIDS**

```
numerical_data_subset <- numerical_data[1:1000,1:3 ]
nbclust_result <- NbClust(data = numerical_data_subset, min.nc = 2, max.nc = 15, meth

str(nbclust_result$Best.nc)

# Extract the number of clusters from NbClust results
num_clusters <- nbclust_result$Best.nc[1, ]

# Create a bar plot
barplot(num_clusters, main = "Number of Clusters",
        xlab = "Number of Clusters", ylab = "Value Index")
```



IN PYTHON

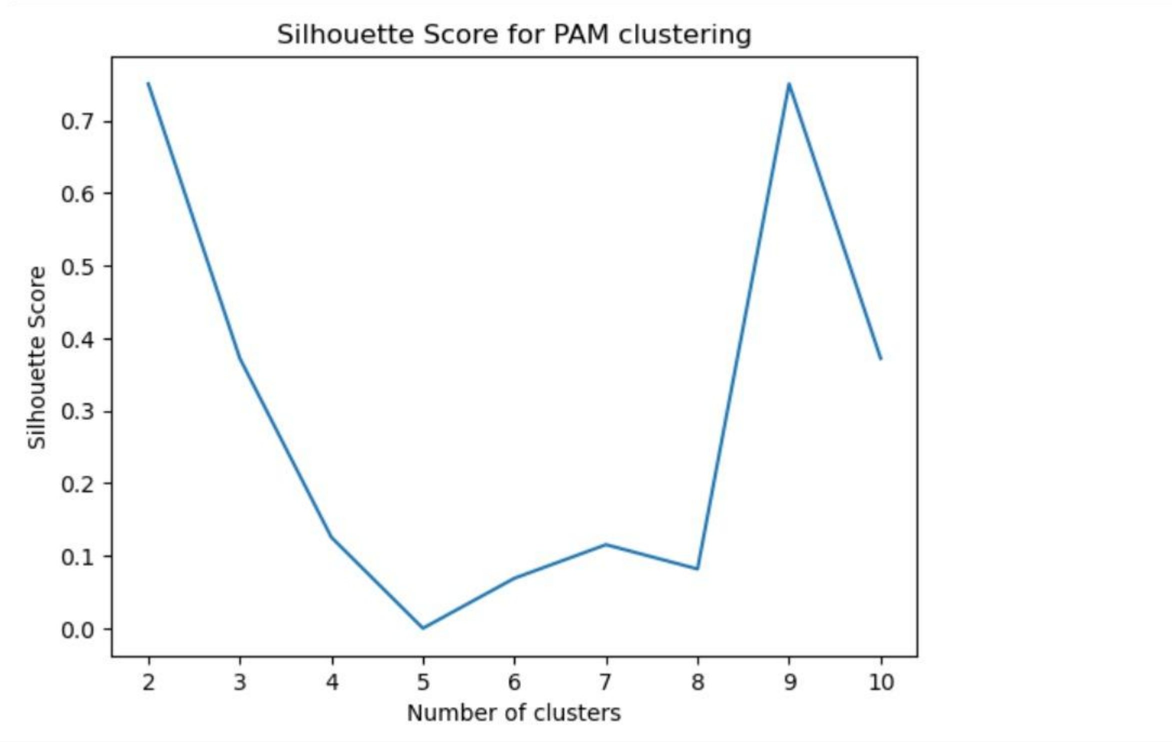
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn_extra.cluster import KMedoids
from sklearn.metrics import silhouette_score
```

```
silhouette_array = np.array(silhouette)
# Calculate silhouette scores for different numbers of clusters
n_clusters_range = range(2, 11)
silhouette_scores = []
```

```
for n_clusters in n_clusters_range:
    if n_clusters <= len(silhouette):  # Check if the number of clusters is valid
        kmedoids = KMedoids(n_clusters=n_clusters, random_state=42)
        cluster_labels = kmedoids.fit_predict(silhouette.reshape(-1, 1))

        if len(np.unique(cluster_labels)) < 2 or len(np.unique(cluster_labels)) >= len(silhouette):
            silhouette_avg = np.nan  # Set silhouette score to NaN
        else:
            silhouette_avg = silhouette_score(silhouette.reshape(-1, 1), cluster_labels)
    else:
        silhouette_avg = np.nan  # Set silhouette score to NaN

    silhouette_scores.append(silhouette_avg)

# Print silhouette scores
print(silhouette_scores)
```

```
[0.7504423966663645, 0.3720195503389193, 0.12546674711425723, 0.00014474006572864288, 0.06910587800231835, 0.1153747745128902, 0.08182048973199998, 0.750
4423966663645, 0.3720195503389193, 0.12546674711425723, 0.00014474006572864288, 0.06910587800231835, 0.1153747745128902, 0.08182048973199998, nan, nan]
```

```
n_clusters_range = range(2, 11)

# Plot silhouette scores corresponding to the range of clusters
plt.plot(n_clusters_range, silhouette_scores[:len(n_clusters_range)])
plt.title('Silhouette Score for PAM clustering')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



DENSITY BASED SPATIALCLUSTERING OF APPLICATIONS WITH NOISE

```
install.packages("dbscan")
library(dbscan)

# Perform DBSCAN clustering
dbscan_result <- dbscan(numerical_data_subset, eps = 0.5, minPts = 5)

# Extract cluster assignments and noise points
cluster_assignments <- dbscan_result$cluster
noise_points <- numerical_data_subset[dbscan_result$cluster == 0, ]

# Analyze the clusters and noises
num_clusters <- length(unique(cluster_assignments)) - 1  # Subtract 1 for noise clust
num_noises <- sum(cluster_assignments == 0)
```

```
> # Print the number of clusters and noises
> cat("Number of clusters:", num_clusters, "\n")
Number of clusters: 3
> cat("Number of noise points:", num_noises, "\n")
Number of noise points: 0
>
```

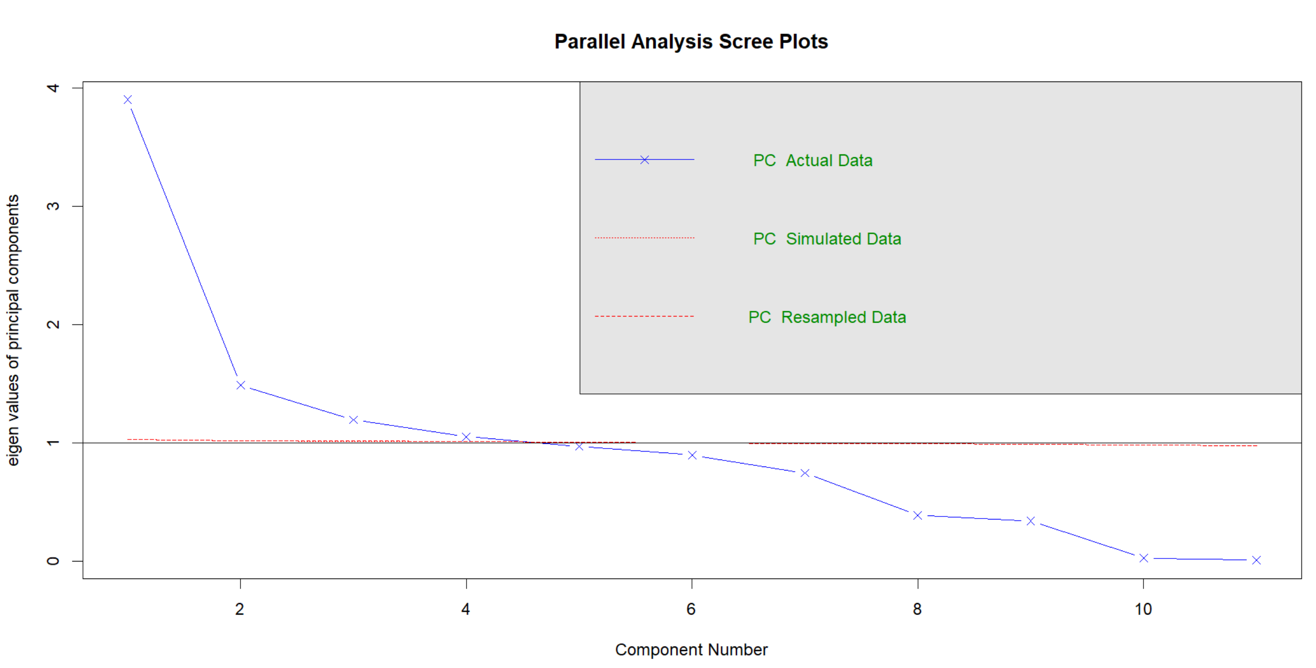
IN PYTHON

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import numpy as np

# Input data: silhouette values
silhouette = np.array([0.63312081, 0.68796139, 0.56580513, 0.48280793, 0.45476621, 0.45994781, 0.44833957, 0.4461

# Reshape silhouette data to fit DBSCAN input format
X = silhouette.reshape(-1, 1)

# DBSCAN clustering
dbscan = DBSCAN(eps=0.1, min_samples=2)
cluster_labels = dbscan.fit_predict(X)

# Analyzing clusters and noises
n_clusters_ = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)  # Number of clusters, ignoring noise
n_noises_ = list(cluster_labels).count(-1)  # Number of noise points

print('Estimated number of clusters:', n_clusters_)
print('Estimated number of noise points:', n_noises_)
print('Cluster labels:', cluster_labels)
```

```
Estimated number of clusters: 1
Estimated number of noise points: 0
Cluster labels: [0 0 0 0 0 0 0 0 0]
```

PCA IMPLEMENTATION IN R

```
# PCA Scree plot
fa.parallel(bank_data[,-1], fa="pc", n.iter=100,show.legend = TRUE)
```



We are choosing 4 as best component because that is where the elbow point of the curve is and the curve starts to flatten out.

CREATING DATA FRAME FOR 4 COMPONENTS AND APPLYING RF MODEL TO FIND THE ACCURACY

```
> confusionMatrix(table(rf_pred, test_labels))
Confusion Matrix and Statistics

       test_labels
rf_pred    0    1
      0 8815  574
      1  370  538

               Accuracy : 0.9083
                 95% CI : (0.9026, 0.9138)
    No Information Rate : 0.892
    P-Value [Acc > NIR] : 2.667e-08

                  Kappa : 0.4824

 Mcnemar's Test P-Value : 3.920e-11

            Sensitivity : 0.9597
            Specificity : 0.4838
         Pos Pred Value : 0.9389
         Neg Pred Value : 0.5925
             Prevalence : 0.8920
         Detection Rate : 0.8561
   Detection Prevalence : 0.9118
      Balanced Accuracy : 0.7218

       'Positive' Class : 0
```

```
# Perform PCA
pca_model <- prcomp(bank_data[, -1], scale = TRUE)
pca <- as.data.frame(pca_model$x[, 1:4])
```

```
# Random Forest
rf_model <- randomForest(subscription ~ ., data =
                           train_data_with_sub, ntree = 50)
```

CONCLUSION

This project analyzed a bank marketing dataset employing a comprehensive range of machine learning techniques including Decision Trees, SVM, Random Forest, Bagging, Boosting, Naive Bayes for categorical data, KNN, Random Forest, and Naive Bayes for numerical data, alongside clustering.

Insights gleaned facilitated personalized marketing campaigns, leveraging customer segmentation, predictive modeling, and ensemble learning. Recommendations emphasize continuous model refinement and integration into marketing strategies for enhanced customer engagement and conversion. By embracing data-driven methodologies, the project aims to empower the bank with actionable insights to optimize marketing efforts, bolster customer relationships, and drive sustainable growth in the financial services sector.

REFERENCES

Moro, S., Cortez, P., & Rita, P. (2014). A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems.

Zanella, A., Olsina, L., & Salto, C. (2017). Predictive Models for Bank Telemarketing Campaigns. Expert Systems with Applications.

Silva, A., & Ribeiro, R. (2018). Predictive Modeling in Direct Marketing: An Application Using Data Mining Techniques. Journal of Business Research.