REPORT

Dataset Description: Fake Bills Classification

The dataset "Fake Bills Classification" comprises 1500 instances with 7 columns, each representing specific attributes of banknote features. The primary objective of this dataset is to discern the authenticity of banknotes based on certain characteristics. Below is a detailed description of the columns:

1. is_genuine (boolean)

   - This binary attribute indicates whether a banknote is genuine or fake.

   - A value of 1 signifies a genuine banknote, while a value of 0 represents a fake banknote.

2. diagonal (float):

   - The diagonal length of the banknote, measured in millimeters.

   - A continuous numerical feature that provides information about the physical dimensions of the banknote.

3. height_left (float):

   - The height of the banknote on the left side, measured in millimeters.

   - Another continuous numerical attribute that offers insights into the asymmetrical nature of the banknote.

4. height_right (float):

   - The height of the banknote on the right side, measured in millimeters.

   - Similar to "height_left," this attribute contributes to the banknote's asymmetry analysis.

5. margin_low (float):

   - The lower margin of the banknote, indicating the distance between the lower edge and features on the banknote, measured in millimeters.

   - A numerical attribute that helps assess the banknote's layout and design.

6. margin_upper (float):

- The upper margin of the banknote, indicating the distance between the upper edge and features on the banknote, measured in millimeters.

- Similar to "margin_low," this attribute contributes to the analysis of the banknote's layout and design.

7. length (float):

- The length of the banknote, measured in millimeters.

- A continuous numerical feature that provides additional information about the banknote's physical dimensions.

This dataset serves as a valuable resource for exploring and implementing classification algorithms, particularly those designed to distinguish between genuine and fake banknotes based on their unique characteristics. The dataset's features offer insights into the physical properties and layout of banknotes, which can contribute to the accurate classification of banknote authenticity.

Descriptive analysis of the code:

1. Load necessary libraries

library(class)

library(caret)

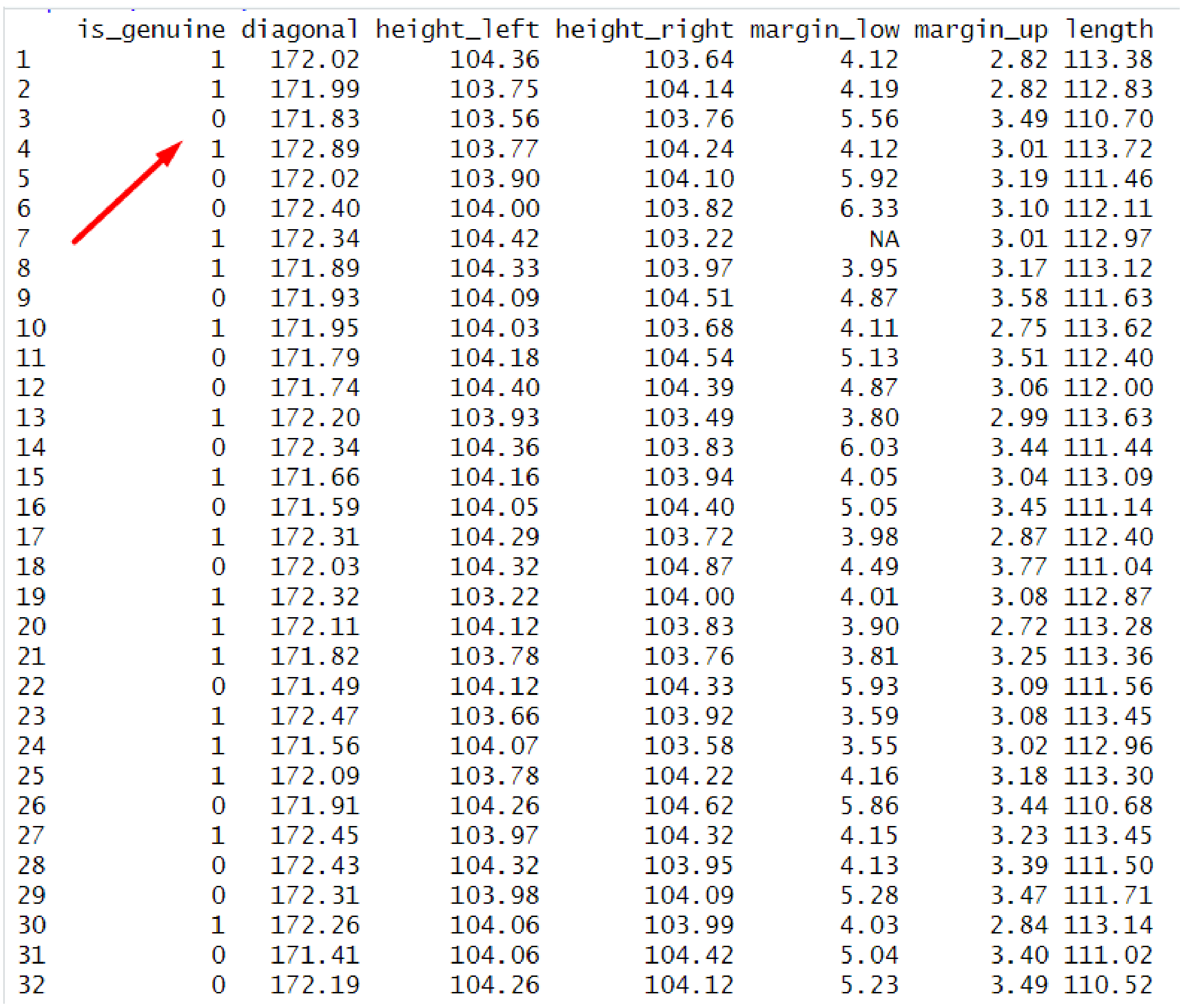
library(ggpubr)

These lines load the required R libraries: class for k-nearest neighbors (KNN) classification, caret for data preprocessing and model evaluation, and ggpubr for data visualization.

2. Load the dataset
dataset <- read.csv("fake_bills.csv")
This line reads a CSV file named "fake_bills.csv" and stores its content in the variable dataset.

3. Convert boolean to numeric (TRUE = 1, FALSE = 0)

dataset$is_genuine <- as.numeric(dataset$is_genuine)

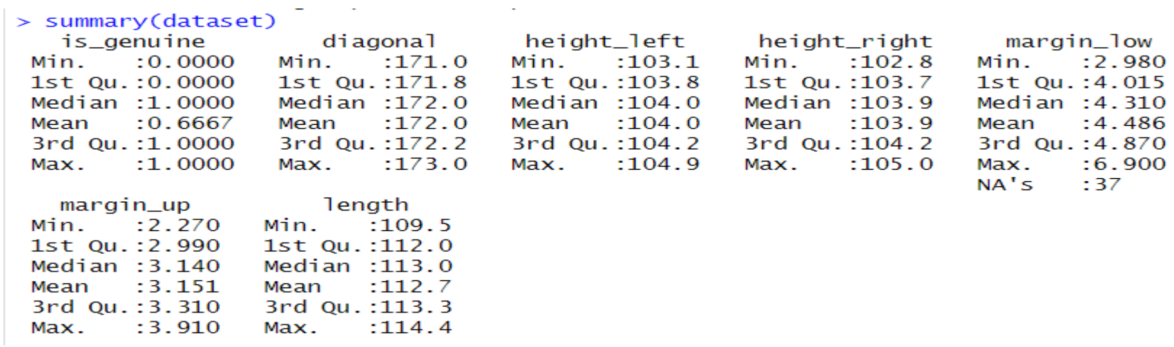


```
   is_genuine diagonal height_left height_right margin_low margin_up length
1           1   172.02      104.36       103.64       4.12      2.82 113.38
2           1   171.99      103.75       104.14       4.19      2.82 112.83
3           0   171.83      103.56       103.76       5.56      3.49 110.70
4           1   172.89      103.77       104.24       4.12      3.01 113.72
5           0   172.02      103.90       104.10       5.92      3.19 111.46
6           0   172.40      104.00       103.82       6.33      3.10 112.11
7           1   172.34      104.42       103.22         NA      3.01 112.97
8           1   171.89      104.33       103.97       3.95      3.17 113.12
9           0   171.93      104.09       104.51       4.87      3.58 111.63
10          1   171.95      104.03       103.68       4.11      2.75 113.62
11          0   171.79      104.18       104.54       5.13      3.51 112.40
12          0   171.74      104.40       104.39       4.87      3.06 112.00
13          1   172.20      103.93       103.49       3.80      2.99 113.63
14          0   172.34      104.36       103.83       6.03      3.44 111.44
15          1   171.66      104.16       103.94       4.05      3.04 113.09
16          0   171.59      104.05       104.40       5.05      3.45 111.14
17          1   172.31      104.29       103.72       3.98      2.87 112.40
18          0   172.03      104.32       104.87       4.49      3.77 111.04
19          1   172.32      103.22       104.00       4.01      3.08 112.87
20          1   172.11      104.12       103.83       3.90      2.72 113.28
21          1   171.82      103.78       103.76       3.81      3.25 113.36
22          0   171.49      104.12       104.33       5.93      3.09 111.56
23          1   172.47      103.66       103.92       3.59      3.08 113.45
24          1   171.56      104.07       103.58       3.55      3.02 112.96
25          1   172.09      103.78       104.22       4.16      3.18 113.30
26          0   171.91      104.26       104.62       5.86      3.44 110.68
27          1   172.45      103.97       104.32       4.15      3.23 113.45
28          0   172.43      104.32       103.95       4.13      3.39 111.50
29          0   172.31      103.98       104.09       5.28      3.47 111.71
30          1   172.26      104.06       103.99       4.03      2.84 113.14
31          0   171.41      104.06       104.42       5.04      3.40 111.02
32          0   172.19      104.26       104.12       5.23      3.49 110.52
```

This line converts the boolean values in the is_genuine column to numeric values: TRUE becomes 1 and FALSE becomes 0.

4. Summary of the dataset

summary(dataset)



```
> summary(dataset)
   is_genuine         diagonal        height_left      height_right      margin_low   
 Min.   :0.0000   Min.   :171.0   Min.   :103.1   Min.   :102.8   Min.   :2.980  
 1st Qu.:0.0000   1st Qu.:171.8   1st Qu.:103.8   1st Qu.:103.7   1st Qu.:4.015  
 Median :1.0000   Median :172.0   Median :104.0   Median :103.9   Median :4.310  
 Mean   :0.6667   Mean   :172.0   Mean   :104.0   Mean   :103.9   Mean   :4.486  
 3rd Qu.:1.0000   3rd Qu.:172.2   3rd Qu.:104.2   3rd Qu.:104.2   3rd Qu.:4.870  
 Max.   :1.0000   Max.   :173.0   Max.   :104.9   Max.   :105.0   Max.   :6.900  
                                                                  NA's   :37     
   margin_up         length     
 Min.   :2.270   Min.   :109.5  
 1st Qu.:2.990   1st Qu.:112.0  
 Median :3.140   Median :113.0  
 Mean   :3.151   Mean   :112.7  
 3rd Qu.:3.310   3rd Qu.:113.3  
 Max.   :3.910   Max.   :114.4  
```

This line generates a summary of the dataset, showing summary statistics for each column.

5. Check for missing values

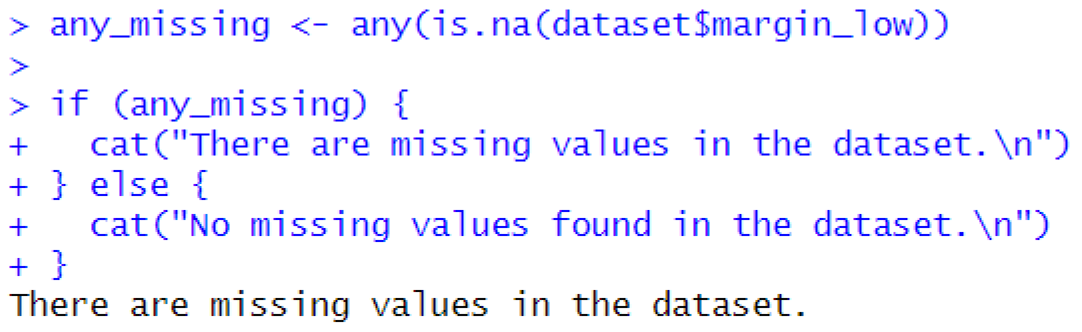
any_missing <- any(is.na(dataset$margin_low))

if (any_missing) {

  cat("There are missing values in the dataset.\n")

} else {

  cat("No missing values found in the dataset.\n")}

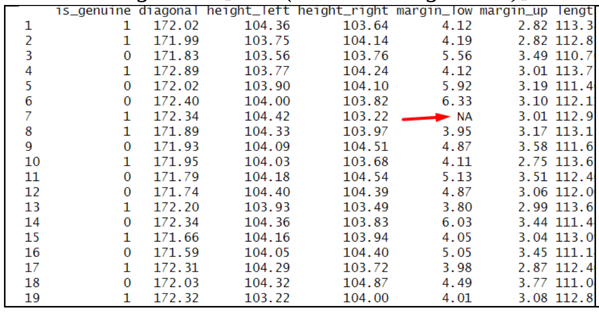


```
> any_missing <- any(is.na(dataset$margin_low))
>
> if (any_missing) {
+   cat("There are missing values in the dataset.\n")
+ } else {
+   cat("No missing values found in the dataset.\n")
+ }
There are missing values in the dataset.
```

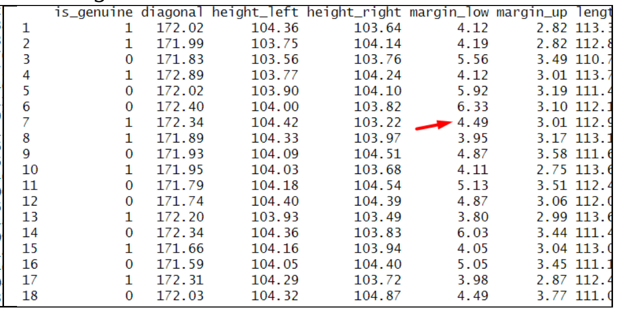
These lines check if there are any missing values in the margin_low column of the dataset and provide a message accordingly.

6. Replace missing values with the mean value

mean_margin_low <- round(mean(dataset$margin_low, na.rm = TRUE), 2)
dataset$margin_low[is.na(dataset$margin_low)] <- mean_margin_low



	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	lengt
1	1	172.02	104.36	103.64	4.12	2.82	113.3
2	1	171.99	103.75	104.14	4.19	2.82	112.8
3	0	171.83	103.56	103.76	5.56	3.49	110.7
4	1	172.89	103.77	104.24	4.12	3.01	113.7
5	0	172.02	103.90	104.10	5.92	3.19	111.4
6	0	172.40	104.00	103.82	6.33	3.10	112.1
7	1	172.34	104.42	103.22	NA	3.01	112.9
8	1	171.89	104.33	103.97	3.95	3.17	113.1
9	0	171.93	104.09	104.51	4.87	3.58	111.6
10	1	171.95	104.03	103.68	4.11	2.75	113.6
11	0	171.79	104.18	104.54	5.13	3.51	112.4
12	0	171.74	104.40	104.39	4.87	3.06	112.0
13	1	172.20	103.93	103.49	3.80	2.99	113.6
14	0	172.34	104.36	103.83	6.03	3.44	111.4
15	1	171.66	104.16	103.94	4.05	3.04	113.0
16	0	171.59	104.05	104.40	5.05	3.45	111.1
17	1	172.31	104.29	103.72	3.98	2.87	112.4
18	0	172.03	104.32	104.87	4.49	3.77	111.0
19	1	172.32	103.22	104.00	4.01	3.08	112.8



	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	lengt
1	1	172.02	104.36	103.64	4.12	2.82	113.
2	1	171.99	103.75	104.14	4.19	2.82	112.
3	0	171.83	103.56	103.76	5.56	3.49	110.
4	1	172.89	103.77	104.24	4.12	3.01	113.
5	0	172.02	103.90	104.10	5.92	3.19	111.
6	0	172.40	104.00	103.82	6.33	3.10	112.
7	1	172.34	104.42	103.22	4.49	3.01	112.
8	1	171.89	104.33	103.97	3.95	3.17	113.
9	0	171.93	104.09	104.51	4.87	3.58	111.
10	1	171.95	104.03	103.68	4.11	2.75	113.
11	0	171.79	104.18	104.54	5.13	3.51	112.
12	0	171.74	104.40	104.39	4.87	3.06	112.
13	1	172.20	103.93	103.49	3.80	2.99	113.
14	0	172.34	104.36	103.83	6.03	3.44	111.
15	1	171.66	104.16	103.94	4.05	3.04	113.
16	0	171.59	104.05	104.40	5.05	3.45	111.
17	1	172.31	104.29	103.72	3.98	2.87	112.
18	0	172.03	104.32	104.87	4.49	3.77	111.

This code calculates the mean of the margin_low column (ignoring missing values), rounds it to two decimal places, and replaces any missing values in that column with the calculated mean.

7. Normalize the dataset using min-max scaling

numeric_features <- c("diagonal", "height_left", "height_right", "margin_low", "margin_up", "length")
for (feature in numeric_features) {
  min_val <- min(dataset[[feature]], na.rm = TRUE)
  max_val <- max(dataset[[feature]], na.rm = TRUE)
  dataset[[feature]] <- (dataset[[feature]] - min_val) / (max_val - min_val)
}

```
   is_genuine   diagonal height_left height_right margin_low margin_up    length
1           1 0.49746193  0.70114943    0.3849765  0.2908163 0.3353659 0.7858586
2           1 0.48223350  0.35057471    0.6197183  0.3086735 0.3353659 0.6747475
3           0 0.40101523  0.24137931    0.4413146  0.6581633 0.7439024 0.2444444
4           1 0.93908629  0.36206897    0.6666667  0.2908163 0.4512195 0.8545455
5           0 0.49746193  0.43678161    0.6009390  0.7500000 0.5609756 0.3979798
6           0 0.69035533  0.49425287    0.4694836  0.8545918 0.5060976 0.5292929
7           1 0.65989848  0.73563218    0.1877934  0.3852041 0.4512195 0.7030303
8           1 0.43147208  0.68390805    0.5399061  0.2474490 0.5487805 0.7333333
9           0 0.45177665  0.54597701    0.7934272  0.4821429 0.7987805 0.4323232
10          1 0.46192893  0.51149425    0.4037559  0.2882653 0.2926829 0.8343434
11          0 0.38071066  0.59770115    0.8075117  0.5484694 0.7560976 0.5878788
12          0 0.35532995  0.72413793    0.7370892  0.4821429 0.4817073 0.5070707
13          1 0.58883249  0.45402299    0.3145540  0.2091837 0.4390244 0.8363636
14          0 0.65989848  0.70114943    0.4741784  0.7780612 0.7134146 0.3939394
15          1 0.31472081  0.58620690    0.5258216  0.2729592 0.4695122 0.7272727
```

This loop iterates over the specified numeric features, calculates the minimum and maximum values of each feature (ignoring missing values), and then scales each feature to the [0, 1] range using min-max scaling.

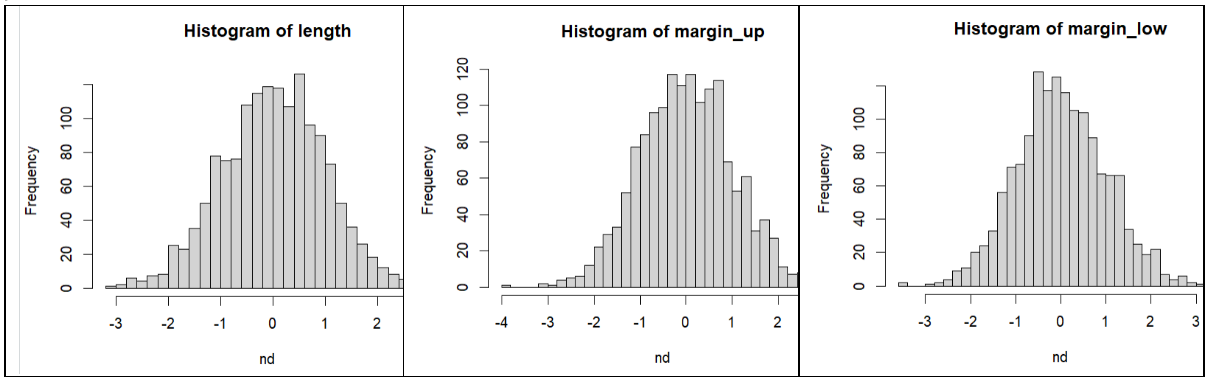
8. Calculate correlations with target variable

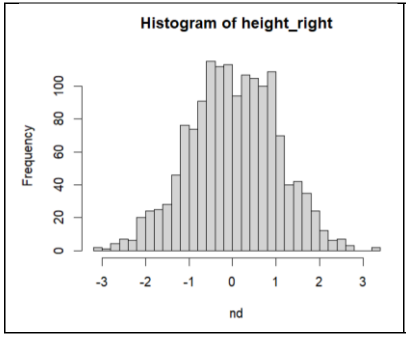
```
cor_with_target <- cor(dataset[, numeric_features], dataset$is_genuine)
```

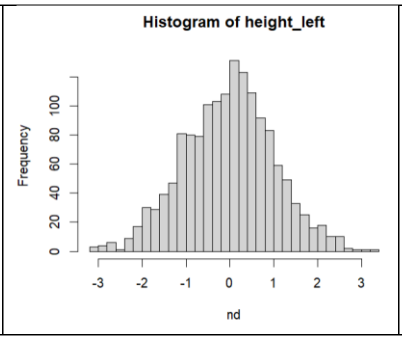
This line calculates the Pearson's correlation coefficients between the numeric features specified in numeric_features and the target variable is_genuine. The result is stored in the cor_with_target matrix.

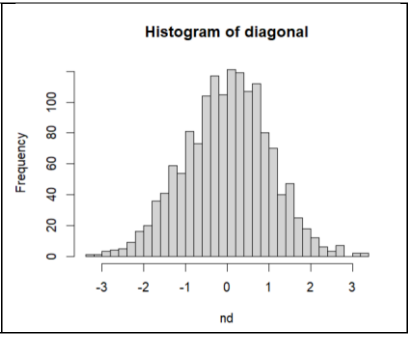
9. Skewness analysis

```
options(repr.plot.width = 4, repr.plot.height = 4)
for (data in numeric_features) {
  nd <- rnorm(dataset[[data]])
  hist(nd, breaks = 30, main = paste("Histogram of", data))
  var <- as.character(readline(prompt = "Continue(y/n): "))
  n <- "n"
  if (tolower(var) == n) {
    break
  }
}
```



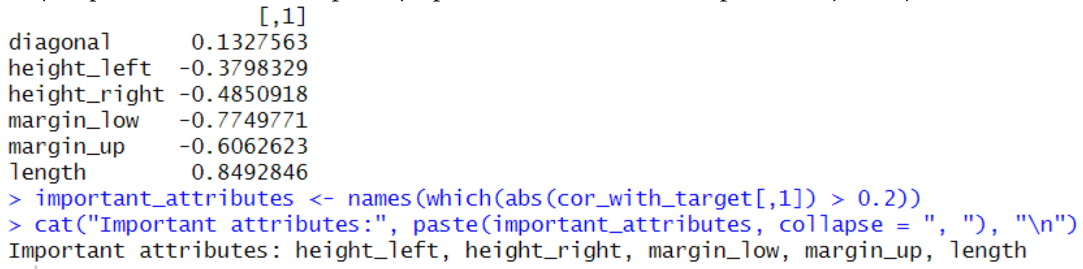






This loop iterates over the numeric features and performs a skewness analysis by generating histograms for each feature. It uses random data (rnorm) to generate histograms and asks for user input to continue to the next histogram or stop.

10. Correlation Analysis
important_attributes <- names(which(abs(cor_with_target[, 1]) > 0.2))
cat("Important attributes:", paste(important_attributes, collapse = ", "), "\n")



```
                   [,1]
diagonal      0.1327563
height_left  -0.3798329
height_right -0.4850918
margin_low   -0.7749771
margin_up    -0.6062623
length        0.8492846
> important_attributes <- names(which(abs(cor_with_target[,1]) > 0.2))
> cat("Important attributes:", paste(important_attributes, collapse = ", "), "\n")
Important attributes: height_left, height_right, margin_low, margin_up, length
```

This line identifies important attributes by selecting the column names where the absolute value of the correlation coefficient with the target variable (is_genuine) is greater than 0.2. The selected attributes' names are concatenated and printed.

11. Prepare data for KNN
knn_data <- dataset[, c("is_genuine", "height_left", "height_right", "margin_low", "margin_up", "length")]

This line creates a new dataset knn_data containing only the selected columns: is_genuine, height_left, height_right, margin_low, margin_up, and length. This subset of the dataset will be used for the KNN classification task.

12. Split data for training and testing
set.seed(123)
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[train_index,]
y_train <- y[train_index]
X_test <- X[-train_index,]
y_test <- y[-train_index]

These lines split the dataset into training and testing sets using a function from the caret package. The createDataPartition function partitions the data into an 80-20 split, creating indices for the training and testing data. The variables X_train, y_train, X_test, and y_test store the respective subsets of the data.

13. Train KNN model

```
k <- 5
knn_model <- knn(train = X_train, test = X_test, cl = y_train, k = k)
```

This line trains a KNN model using the training data (X_train and y_train). The k parameter is set to 5, which represents the number of neighbors considered during classification.

14. Calculate accuracy

```
accuracy <- sum(knn_model == y_test) / length(y_test)
```

This line calculates the accuracy of the KNN model by comparing the predicted class labels (knn_model) with the actual class labels (y_test). The accuracy is the ratio of correctly classified instances to the total number of instances in the test set.

15. Train KNN model using 10-fold cross-validation

```
knn_model_cv <- train(
  X, y, method = "knn",
  tuneGrid = expand.grid(k = 1:20),
  trControl = trainControl(method = "cv", number = 10)
)
```

These lines use the train function from the caret package to train the KNN model using 10-fold cross-validation. The parameter tuneGrid specifies a range of k values to be evaluated, and trControl defines the cross-validation method with 10 folds.

16. Calculate accuracy from cross-validation

```
accuracy_cv <- knn_model_cv$results$Accuracy[which.max(knn_model_cv$results$Accuracy)]
```

This line calculates the accuracy achieved by the KNN model through cross-validation. It extracts the accuracy values from the knn_model_cv object and selects the maximum accuracy achieved among the different k values.

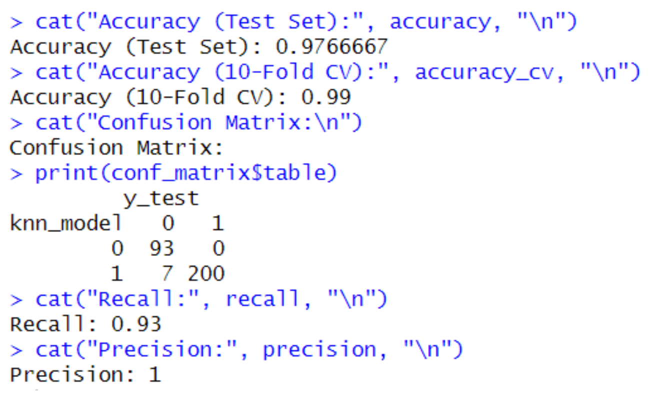
17. Confusion Matrix, Recall, and Precision

```
conf_matrix <- confusionMatrix(table(knn_model, y_test))
recall <- conf_matrix$byClass["Sensitivity"]
precision <- conf_matrix$byClass["Pos Pred Value"]
```

These lines involve computing the confusion matrix, recall, and precision using the confusionMatrix function from the caret package. The results are printed, providing insights into the KNN model's performance.

18. Print the results

```
cat("Accuracy (Test Set):", accuracy, "\n")
cat("Accuracy (10-Fold CV):", accuracy_cv, "\n")
cat("Confusion Matrix:\n")
print(conf_matrix$table)
cat("Recall:", recall, "\n")
cat("Precision:", precision, "\n")
```



```
> cat("Accuracy (Test Set):", accuracy, "\n")
Accuracy (Test Set): 0.9766667
> cat("Accuracy (10-Fold CV):", accuracy_cv, "\n")
Accuracy (10-Fold CV): 0.99
> cat("Confusion Matrix:\n")
Confusion Matrix:
> print(conf_matrix$table)
         y_test
knn_model   0   1
        0  93   0
        1   7 200
> cat("Recall:", recall, "\n")
Recall: 0.93
> cat("Precision:", precision, "\n")
Precision: 1
```

Analysis:

- Accuracy (Test Set): 0.9766667; The accuracy on the test set is approximately 0.9767, which translates to around 97.67%. This indicates that the KNN model correctly classified about 97.67% of instances in the test set, demonstrating its effectiveness in distinguishing between genuine and fake banknotes.

- Accuracy (10-Fold CV): 0.99; The accuracy achieved through 10-fold cross-validation is approximately 0.99 or 99%. This suggests that the KNN model consistently performs well across different folds of the dataset, reinforcing its reliability in making accurate predictions.

- Confusion Matrix:

```
              y_test
  knn_model   0   1
       0      93  0
       1      7   200
```

  In the confusion matrix:
  The cell (0,0) represents the number of instances that were truly class 0 and were correctly classified as class 0 by the model. There were 93 such instances.
  The cell (0,1) represents the number of instances that were actually class 1 but were incorrectly classified as class 0 by the model. There were 7 such instances.
  The cell (1,0) represents the number of instances that were actually class 0 but were incorrectly classified as class 1 by the model. There were 0 instances in this case.
  The cell (1,1) represents the number of instances that were truly class 1 and were correctly classified as class 1 by the model. There were 200 such instances.
  This matrix provides a detailed breakdown of how the model's predictions match with the actual classes in the test set.

- Recall: 0.93; The recall value is approximately 0.93 or 93%. This indicates that the model successfully identified around 93% of the actual positive instances (genuine banknotes) correctly. A higher recall value implies that the model is good at minimizing false negatives.

- Precision: 1; The precision value is 1, which means that all instances classified as positive by the model (predicted as genuine banknotes) were indeed genuine. A precision value of 1 indicates that there were no false positives.

Overall, the output showcases a high accuracy, consistent cross-validation performance, and favorable values for recall and precision. These results suggest that the KNN model is effective in accurately classifying genuine and fake banknotes based on the provided features.