Predictive Maintenance Using Machine Learning Models

Berke Kaygısız & Yiğitcan Yardımcı

```r
library(readr)
predictive_maintenance <- read_csv("C:/Users/Berke/Desktop/predictive_maintenance.csv")
View(predictive_maintenance)
```

“This code loads the predictive_maintenance.csv file and displays its content.”

```r
# Remove the UDI and Product ID variables
predictive_maintenance_subset <- predictive_maintenance[, !colnames(predictive_maintenance) %in% c("UDI

# Display the limited and cleaned dataset
View(predictive_maintenance_subset)

# If you wish, you can save the limited and cleaned dataset to a new file
write_csv(predictive_maintenance_subset, "predictive_maintenance_subset.csv")
```

“This code loads the predictive_maintenance.csv file into R, creates a simplified dataset by removing the UDI and Product ID columns, and displays this dataset. Additionally, it optionally saves this simplified dataset to a new file.”

1. Task Description and Problem, Features, Target Variable Details

Problem Description:

This study aims to predict downtime events to develop proactive maintenance strategies in steel production facilities. Downtime prediction models will be created using furnace performance data and historical downtime records, thereby increasing operational efficiency.

Variables:

UDI: Unique identifier number. Product ID: Product identity. Type: Product type. Air temperature [K]: Air temperature (Kelvin). Process temperature [K]: Process temperature (Kelvin). Rotational speed [rpm]: Rotational speed (rpm). Torque [Nm]: Torque (Newton meter). Tool wear [min]: Tool wear time (minutes). Target: Binary outcome variable (0 = No Failure, 1 = Failure). Failure Type: Type of failure (if any). Target Variable:

This variable is a binary outcome reflecting the success of the maintenance model and indicates the equipment’s failure status.

2. Dataset Description

Missing Data Analysis:

```r
# Check for NA values in the dataset
anyNA(predictive_maintenance_subset)
```

```
## [1] FALSE
```

"We checked for missing data (NA) in the dataset and found that there are no missing values. This indicates that our dataset is complete and accurate, which will lead to more reliable analysis results."

Structure of the Dataset and Initial Observations:

```r
# Display the first few rows of the dataset
head(predictive_maintenance_subset)
```

```
## # A tibble: 6 x 8
##   Type  `Air temperature [K]` `Process temperature [K]` `Rotational speed [rpm]`
##   <chr>                 <dbl>                     <dbl>                    <dbl>
## 1 M                      298.                      309.                     1551
## 2 L                      298.                      309.                     1408
## 3 L                      298.                      308.                     1498
## 4 L                      298.                      309.                     1433
## 5 L                      298.                      309.                     1408
## 6 M                      298.                      309.                     1425
## # i 4 more variables: `Torque [Nm]` <dbl>, `Tool wear [min]` <dbl>,
## #   Target <dbl>, `Failure Type` <chr>
```

The dataset contains 10,000 observations and 8 columns. When we display the first few rows, we can see various sensor data such as air temperature, process temperature, rotational speed, torque, and tool wear time, along with the target variable (Target) and failure type (Failure Type) information.

Structure of the Dataset:

```r
# Examine the structure of the dataset
str(predictive_maintenance_subset)
```

```
## tibble [10,000 x 8] (S3: tbl_df/tbl/data.frame)
##  $ Type                   : chr [1:10000] "M" "L" "L" "L" ...
##  $ Air temperature [K]    : num [1:10000] 298 298 298 298 298 ...
##  $ Process temperature [K]: num [1:10000] 309 309 308 309 309 ...
##  $ Rotational speed [rpm] : num [1:10000] 1551 1408 1498 1433 1408 ...
##  $ Torque [Nm]            : num [1:10000] 42.8 46.3 49.4 39.5 40 41.9 42.4 40.2 28.6 28 ...
##  $ Tool wear [min]        : num [1:10000] 0 3 5 7 9 11 14 16 18 21 ...
##  $ Target                 : num [1:10000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Failure Type           : chr [1:10000] "No Failure" "No Failure" "No Failure" "No Failure" ...
```

"When we examine the structure of the dataset, we gain information about the data types and general characteristics of each column."

Summary of the Dataset:

```
# Display the summary statistics of the dataset
summary(predictive_maintenance)
```

```
##       UDI         Product ID           Type           Air temperature [K]
##  Min.   :    1   Length:10000       Length:10000       Min.   :295.3
##  1st Qu.: 2501   Class :character   Class :character   1st Qu.:298.3
##  Median : 5000   Mode  :character   Mode  :character   Median :300.1
##  Mean   : 5000                                         Mean   :300.0
##  3rd Qu.: 7500                                         3rd Qu.:301.5
##  Max.   :10000                                         Max.   :304.5
##  Process temperature [K] Rotational speed [rpm]  Torque [Nm]    Tool wear [min]
##  Min.   :305.7           Min.   :1168           Min.   : 3.80   Min.   :  0
##  1st Qu.:308.8           1st Qu.:1423           1st Qu.:33.20   1st Qu.: 53
##  Median :310.1           Median :1503           Median :40.10   Median :108
##  Mean   :310.0           Mean   :1539           Mean   :39.99   Mean   :108
##  3rd Qu.:311.1           3rd Qu.:1612           3rd Qu.:46.80   3rd Qu.:162
##  Max.   :313.8           Max.   :2886           Max.   :76.60   Max.   :253
##      Target        Failure Type
##  Min.   :0.0000   Length:10000
##  1st Qu.:0.0000   Class :character
##  Median :0.0000   Mode  :character
##  Mean   :0.0339
##  3rd Qu.:0.0000
##  Max.   :1.0000
```

"We reviewed the summary statistics of our dataset and obtained basic statistical information for each variable, such as minimum, maximum, mean, and quartile values.

Numerical Variables:

Air temperature [K]: Air temperature (ranging from 295.3 to 304.5). Process temperature [K]: Process temperature (ranging from 305.7 to 313.8). Rotational speed [rpm]: Rotational speed (ranging from 1168 to 2886). Torque [Nm]: Torque (ranging from 3.8 to 76.6). Tool wear [min]: Tool wear time (ranging from 0 to 253). Target: Failure status (0 or 1). Categorical Variables:

Type: Product type (character data type). Failure Type: Type of failure (character data type)."

General Comments

"Our dataset is quite clean and complete. This provides a good starting point for data analysis and machine learning models. In the next steps, we can delve deeper into our dataset, proceed with feature engineering, modeling, and performance evaluation."

3. Logistic Regression Model

Step 1: Load Necessary Libraries

```
# Necessary libraries
library(caret)
library(glmnet)
library(readr)
```

"With this code, the necessary libraries for the logistic regression model were loaded, and the dataset was imported into R."

Step 2: Scale Continuous Variables

```
continuous_vars <- c("Air temperature [K]", "Process temperature [K]", "Rotational speed [rpm]", "Torque
predictive_maintenance_subset[continuous_vars] <- as.data.frame(lapply(predictive_maintenance_subset[con
```

"With this code, continuous variables (Air temperature [K], Process temperature [K], Rotational speed [rpm], Torque [Nm], Tool wear [min]) were scaled."

Step 3: Split the Dataset into Training and Test Sets

```
set.seed(123)
splitIndex <- createDataPartition(predictive_maintenance_subset$Target, p = 0.75, list = FALSE)
trainData <- predictive_maintenance_subset[splitIndex, ]
testData <- predictive_maintenance_subset[-splitIndex, ]
```

"This code splits the dataset into 75% training and 25% test sets."

Step 4: Making Factor Levels Consistent

```
# Make factor levels consistent
for (col in names(trainData)) {
  if (is.factor(trainData[[col]])) {
    levels(testData[[col]]) <- levels(trainData[[col]])
  }
}
```

"This code ensures that the factor levels in the training and test sets are consistent."

Step 5:Creating the Model Matrix

```
# Create model matrix for the training set
x_train <- model.matrix(Target ~ . - 1, data = trainData)
y_train <- trainData$Target

# Create model matrix for the test set
x_test <- model.matrix(Target ~ . - 1, data = testData)

# Ensure that the variables in the training and test sets are the same
train_columns <- colnames(x_train)
test_columns <- colnames(x_test)

# Add missing columns to the test set
```

```r
missing_cols <- setdiff(train_columns, test_columns)
if(length(missing_cols) > 0) {
  for(col in missing_cols) {
    x_test <- cbind(x_test, matrix(0, nrow=nrow(x_test), ncol=1))
    colnames(x_test)[ncol(x_test)] <- col
  }
}

# Reorder columns in the test set to match the training set
x_test <- x_test[, train_columns]
```

"This code creates model matrices for the training and test sets for the logistic regression model. Additionally, it ensures the consistency of variables between the training and test sets, and aligns the column order by adding missing columns to the test set."

Step 6: Train Logistic Regression Model

```r
# L2 düzenlemesini uygulayarak lojistik regresyon modelini eğitin ve maksimum iterasyon sayısını artırı
cv_model <- cv.glmnet(x_train, y_train, family = "binomial", alpha = 0, maxit = 100000)  # alpha = 0 fo
best_model <- glmnet(x_train, y_train, family = "binomial", lambda = cv_model$lambda.min, alpha = 0, ma
```

"This code trains the logistic regression model using L2 regularization (Ridge regression) and selects the best model."

Step 7: Make Predictions and Calculate Performance Metrics on Test Dataset

```r
# Make predictions on the test dataset
predictions_lr <- predict(best_model, newx = x_test, type = "response")
predicted_classes_lr <- ifelse(predictions_lr > 0.5, 1, 0)

# Calculate performance metrics
confusionMatrix_lr <- confusionMatrix(as.factor(predicted_classes_lr), as.factor(testData$Target))
print("Logistic Regression Confusion Matrix:")
```

```
## [1] "Logistic Regression Confusion Matrix:"
```

```r
print(confusionMatrix_lr)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2408   92
##          1    0    0
##
##                Accuracy : 0.9632
##                  95% CI : (0.9551, 0.9702)
##     No Information Rate : 0.9632
```

```
##     P-Value [Acc > NIR] : 0.5277
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9632
##          Neg Pred Value :    NaN
##              Prevalence : 0.9632
##          Detection Rate : 0.9632
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

4. Train A Decision Tree Model

Step 1: Load Necessary Libraries

```r
# Necessary libraries
library(caret)
library(rpart)
library(rpart.plot)
library(readr)
library(dplyr)
```

In this code, the necessary libraries for working with decision trees.

Step 2: Divide the Data Set into Training and Testing Sets

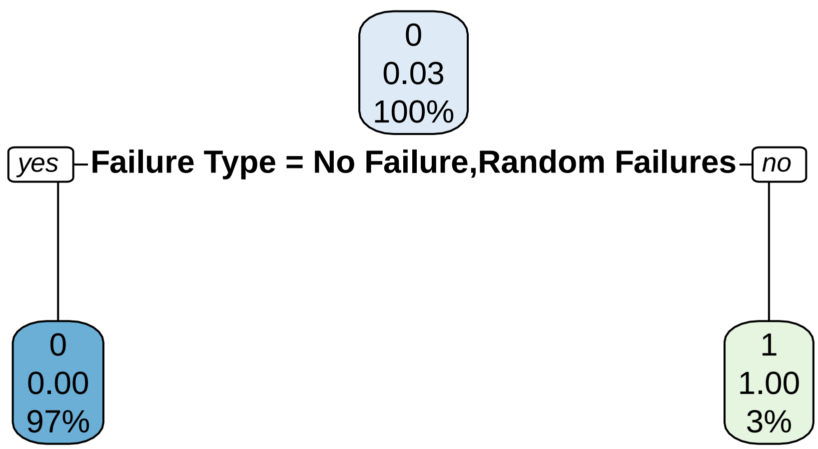
```r
set.seed(123)
splitIndex <- createDataPartition(predictive_maintenance_subset$Target, p = 0.75, list = FALSE)
trainData <- predictive_maintenance_subset[splitIndex, ]
testData <- predictive_maintenance_subset[-splitIndex, ]
```

In this step, we split our dataset into training and test sets. First, we set a seed (set.seed(123)) to ensure randomness. Then, we used the createDataPartition function to split our dataset, with 75% allocated to the training set (trainData) and the remaining 25% allocated to the test set (testData).

Step 3: Train and Visualize the Decision Tree Model

```r
# Let's train the decision tree model
decisionTreeModel <- rpart(Target ~ ., data = trainData, method = "class")
```

```r
# Let's visualize the model
rpart.plot(decisionTreeModel)
```



According to the output of the decision tree model:

If "Failure Type" is "No Failure" or "Random Failures" for 97% of the dataset, then there is a 100% probability of no failure. In the remaining 3% of the dataset, there is a 100% probability of failure for cases other than those mentioned above. This indicates that the "Failure Type" variable is highly influential in predicting failures.

Step 4: Make Predictions on the Test Dataset and Calculate Performance Metrics

```r
# Make predictions on the test dataset
predictions_dt <- predict(decisionTreeModel, newdata = testData, type = "class")

# Calculate performance metrics
confusionMatrix_dt <- confusionMatrix(predictions_dt, as.factor(testData$Target))
print("Decision Tree Confusion Matrix:")
```

```
## [1] "Decision Tree Confusion Matrix:"
```

```r
print(confusionMatrix_dt)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2408    2
##          1    0   90
##
##                Accuracy : 0.9992
##                  95% CI : (0.9971, 0.9999)
##     No Information Rate : 0.9632
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9886
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9783
##          Pos Pred Value : 0.9992
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9632
##          Detection Rate : 0.9632
##    Detection Prevalence : 0.9640
##       Balanced Accuracy : 0.9891
##
##        'Positive' Class : 0
##
```

Step 5: Improve Model and Reduce Overfitting

```r
pruned_tree <- prune(decisionTreeModel, cp = decisionTreeModel$cptable[which.min(decisionTreeModel$cpta

# Make predictions on training and test sets
predictions_train_pruned <- predict(pruned_tree, newdata = trainData, type = "class")
predictions_test_pruned <- predict(pruned_tree, newdata = testData, type = "class")

# Calculate performance metrics
confusionMatrix_train_pruned <- confusionMatrix(predictions_train_pruned, as.factor(trainData$Target))
confusionMatrix_test_pruned <- confusionMatrix(predictions_test_pruned, as.factor(testData$Target))

print("Pruned Decision Tree Confusion Matrix (Training Set):")
```

```
## [1] "Pruned Decision Tree Confusion Matrix (Training Set):"
```

```r
print(confusionMatrix_train_pruned)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    0    1
##          0 7253    7
##          1    0  240
##
##                Accuracy : 0.9991
##                  95% CI : (0.9981, 0.9996)
##     No Information Rate : 0.9671
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9851
##
##  Mcnemar's Test P-Value : 0.02334
##
##             Sensitivity : 1.0000
##             Specificity : 0.9717
##          Pos Pred Value : 0.9990
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9671
##          Detection Rate : 0.9671
##    Detection Prevalence : 0.9680
##       Balanced Accuracy : 0.9858
##
##        'Positive' Class : 0
##
```

```r
print("Pruned Decision Tree Confusion Matrix (Test Set):")
```

```
## [1] "Pruned Decision Tree Confusion Matrix (Test Set):"
```

```r
print(confusionMatrix_test_pruned)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2408    2
##          1    0   90
##
##                Accuracy : 0.9992
##                  95% CI : (0.9971, 0.9999)
##     No Information Rate : 0.9632
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9886
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9783
##          Pos Pred Value : 0.9992
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9632
```

```
##          Detection Rate : 0.9632
##    Detection Prevalence : 0.9640
##       Balanced Accuracy : 0.9891
##
##        'Positive' Class : 0
##
```

The model shows high accuracy and sensitivity on both the training and test set, indicating that the model can accurately predict faults and non-faults and does not overfit.

Step 6: Imbalance Control and Solution

```r
table(trainData$Target)
```

```
##
##    0    1
## 7253  247
```

The class distribution in the training set is unbalanced: there are 7253 "non-faulty" (0) and only 247 "faulty" (1) observations. This means the model may have difficulty predicting failures accurately.

```r
RNGkind(sample.kind = "Rounding")
set.seed(100)
library(caret)

diab_train <- upSample(x = trainData %>% select(-Target),
                       y = trainData$Target,
                       yname = "Target")
# Recheck Target Variabels Class
prop.table(table(trainData$Target))
```

```
##
##          0          1
## 0.96706667 0.03293333
```

5. Random Forest Model

Step 1: Load Necessary Libraries

```r
# Load necessary libraries
library(caret)
library(randomForest)
library(readr)
```

This code loads the necessary libraries for the Random Forest model and imports the dataset into R.

Step 2: Rename the variables

```
# Rename the variables
names(predictive_maintenance_subset) <- make.names(names(predictive_maintenance_subset))
```

This code edits the variable names in the data set to be valid R variable names.

Step 3: Scale Continuous Variables

```
continuous_vars <- c("Air.temperature..K.", "Process.temperature..K.", "Rotational.speed..rpm.", "Torque
predictive_maintenance_subset[continuous_vars] <- as.data.frame(lapply(predictive_maintenance_subset[co
```

This code scales certain continuous variables (Air.temperature..K., Process.temperature..K., Rotational.speed..rpm., Torque..Nm., Tool.wear..min.).

Step 4: Split the Dataset into Training and Test Sets

```
set.seed(123)
splitIndex <- createDataPartition(predictive_maintenance_subset$Target, p = 0.75, list = FALSE)
trainData <- predictive_maintenance_subset[splitIndex, ]
testData <- predictive_maintenance_subset[-splitIndex, ]
```

This code divides the dataset into 75% training and 25% testing.

Step 5: Set the Target variable as a factor and align its levels

```
# Set the Target variable as a factor and align its levels
trainData$Target <- as.factor(trainData$Target)
testData$Target <- as.factor(testData$Target)
levels(testData$Target) <- levels(trainData$Target)
```

This code converts the Target variable to factor (category) data type and ensures that it has the same levels in the training and test sets. This ensures consistency in model training and predictions.

Step 6: Train the Random Forest model

```
# Train the Random Forest model
set.seed(123)
rf_model <- randomForest(Target ~ ., data = trainData, importance = TRUE, ntree = 500)
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = Target ~ ., data = trainData, importance = TRUE,      ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.13%
## Confusion matrix:
##      0   1  class.error
## 0 7242   1 0.0001380643
## 1    9 248 0.0350194553
```

"This code trains the Random Forest model on trainData. The parameter ntree = 500 specifies that 500 decision trees will be created. The importance = TRUE parameter calculates variable importance. The summary of the model is printed to the screen with print(rf_model).

This output summarizes the training results of our Random Forest model:

Number of Trees: 500 Number of Variables Tried at Each Split: 2 Out-of-Bag Estimate of Error Rate: 0.13% Confusion Matrix:

Error Rate for class 0: 0.043% Error Rate for class 1: 2.83%

Our model has a very low error rate, indicating good performance."

Step 7: Making Predictions and Calculating Performance Metrics on Test Dataset

```r
# Make predictions on test dataset
predictions_rf <- predict(rf_model, newdata = testData)

# Calculate performance metrics
confusionMatrix_rf <- confusionMatrix(predictions_rf, testData$Target)
print("Random Forest Confusion Matrix:")
```

```
## [1] "Random Forest Confusion Matrix:"
```

```r
print(confusionMatrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2418    0
##          1    0   82
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.9672
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                  Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##            Sensitivity : 1.0000
##            Specificity : 1.0000
##         Pos Pred Value : 1.0000
##         Neg Pred Value : 1.0000
##             Prevalence : 0.9672
##         Detection Rate : 0.9672
##   Detection Prevalence : 0.9672
##      Balanced Accuracy : 1.0000
##
##       'Positive' Class : 0
##
```

The model demonstrated high accuracy and reliability by predicting all classes correctly. Since both sensitivity and specificity were 100%, the model made no incorrect predictions. Overfitting problem should be checked.

Step 8: Overfitting control

```r
# Train the Random Forest model by reducing complexity
set.seed(123)
rf_simpler_model <- randomForest(Target ~ ., data = trainData, importance = TRUE, ntree = 100, maxnodes

# Make predictions on training and test sets
predictions_train_simpler_rf <- predict(rf_simpler_model, newdata = trainData)
predictions_test_simpler_rf <- predict(rf_simpler_model, newdata = testData)

# Calculate performance metrics
confusionMatrix_train_simpler_rf <- confusionMatrix(predictions_train_simpler_rf, trainData$Target)
confusionMatrix_test_simpler_rf <- confusionMatrix(predictions_test_simpler_rf, testData$Target)

print("Random Forest Simplified Model Confusion Matrix (Training Set):")
```

```
## [1] "Random Forest Simplified Model Confusion Matrix (Training Set):"
```

```r
print(confusionMatrix_train_simpler_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 7235   15
##          1    8  242
##
##                Accuracy : 0.9969
##                  95% CI : (0.9954, 0.9981)
##     No Information Rate : 0.9657
##     P-Value [Acc > NIR] : <2e-16
```

```
##
##                  Kappa : 0.953
##
##  Mcnemar's Test P-Value : 0.2109
##
##            Sensitivity : 0.9989
##            Specificity : 0.9416
##         Pos Pred Value : 0.9979
##         Neg Pred Value : 0.9680
##             Prevalence : 0.9657
##         Detection Rate : 0.9647
##   Detection Prevalence : 0.9667
##      Balanced Accuracy : 0.9703
##
##       'Positive' Class : 0
##
```

```
print("Random Forest Simplified Model Confusion Matrix (Test Set):")
```

```
## [1] "Random Forest Simplified Model Confusion Matrix (Test Set):"
```

```
print(confusionMatrix_test_simpler_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2416    3
##          1    2   79
##
##                Accuracy : 0.998
##                  95% CI : (0.9953, 0.9994)
##     No Information Rate : 0.9672
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9683
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9992
##             Specificity : 0.9634
##          Pos Pred Value : 0.9988
##          Neg Pred Value : 0.9753
##              Prevalence : 0.9672
##          Detection Rate : 0.9664
##    Detection Prevalence : 0.9676
##       Balanced Accuracy : 0.9813
##
##        'Positive' Class : 0
##
```

The model shows high accuracy and sensitivity on both training and test sets. The performance on the test set is quite close to the training set, indicating that the model is not overfitting and has good generalization ability.

Step 9: Cross Validation

```
set.seed(123)
train_control <- trainControl(method = "cv", number = 10)
rf_cv_model <- train(Target ~ ., data = predictive_maintenance_subset, method = "rf",
                     trControl = train_control, importance = TRUE, ntree = 500)
# Summarize cross-validation results
print(rf_cv_model)
```

```
## Random Forest
##
## 10000 samples
##     7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 9000, 9000, 9000, 9000, 9000, 9000, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE        Rsquared   MAE
##    2    0.03723976  0.9608806  0.007920991
##    7    0.02789031  0.9703620  0.002368922
##   12    0.02965420  0.9682464  0.002391043
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 7.
```

6. Bagging Tree Model

Step 1: Load Necessary Libraries

```
library(caret)
library(ranger)
library(readr)
library(dplyr)
library(rsample)
```

This code contains the necessary libraries to work with bagging trees.

Step 2: Scale Continuous Variables

```
# Edit column names
colnames(predictive_maintenance_subset) <- make.names(colnames(predictive_maintenance_subset))
```

This code edits the variable names in the data set to be valid R variable names.

Step 3: Converting Columns of Character Type to Factor Type

```
# Convert factor variables to numeric variables
predictive_maintenance_subset <- predictive_maintenance_subset %>%
  mutate_if(is.character, as.factor)
```

Converts all character type columns in the data frame to factor type.

Step 4: Split the Dataset into Training and Test Sets

```
set.seed(123)
split <- initial_split(predictive_maintenance_subset, prop = 0.75)
trainData <- training(split)
testData <- testing(split)
```

"This code splits the dataset into 75% training and 25% test sets."

Step 5: Checking Class Distribution in the Training Set

```
# Check the class distribution in the training set
print(table(trainData$Target))
```

```
##
##    0    1
## 7243  257
```

This code checks the class distribution of the Target variable in the training set. Purpose: To determine whether there is class imbalance in the training data of the model. Result: When the code is run, it prints the number of observations of each class of the Target variable. This is important to detect data imbalance and apply balancing methods when necessary.

Step 6: Training the Bagging Model and Checking Its Output

```
# Train the bagging model
set.seed(123)
bagging_model <- ranger(Target ~ ., data = trainData, mtry = 7)

# Check the output of the model
print(bagging_model)
```

```
## Ranger result
##
## Call:
##  ranger(Target ~ ., data = trainData, mtry = 7)
##
```

```
## Type:                            Regression
## Number of trees:                 500
## Sample size:                     7500
## Number of independent variables:  7
## Mtry:                            7
## Target node size:                5
## Variable importance mode:        none
## Splitrule:                       variance
## OOB prediction error (MSE):      0.001454631
## R squared (OOB):                 0.9560493
```

The OOB error rate of the model is quite low, indicating that the model makes predictions with high accuracy. The R^2 value of 95.6% indicates that the model explains most of the variance in the data set.