A blue and white logo

Description automatically generated

**EMBAA 2024-26**

**TERM-1**

**End term project**

# CRISP-DM Report: Forecasting Package Rejection

**Date of Submission: 8/18/2024**

**Assignment for the course- Programming in R**

**Topic – Parcel Rejection model**

**Submitted to- Prof. Venkata Raghavan Krishnaswamy**

**Submitted by-Group 9**

|  |  |
| --- | --- |
| **Students Name** | **Roll No** |
| **Rashmi Prasanna** | **EMBAA24033** |
| **Isha Jain** | **EMBAA24019** |
| **Chaitali Gore** | **EMBAA24013** |
| **Ruchika Vashistha** | **EMBAA24037** |
| **Vanshika Goel** | **EMBAA24056** |

**Certification of Authorship**

We certify that we are the authors of this paper and that any assistance we received in its preparation is fully acknowledged and disclosed in the paper.

**Faculty comment and Marks/Grade: -**

**Faculty Signature:**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Particulars** | **Pg. No.** |
| **1.** | **Business Understanding** | **2** |
| **2.** | **Data Understanding** | **2** |
| **3.** | **Data Preparation** | **4** |
| **4.** | **Modelling** | **10** |
| **5.** | **Evaluation** | **14** |
| **6.** | **Deployment** | **15** |
| **7.** | **Conclusion** | **15** |

**1. Business Understanding**

The logistics industry is constantly seeking methods to reduce inefficiencies and minimize costs. One such challenge is the rejection of parcels during processing, which can result in significant delays, operational inefficiencies, and financial losses. The objective of this project is to build a predictive model to determine whether a parcel will be rejected based on its attributes, such as weight, dimensions, time of registration, and various other factors.

The key objectives are:

- To reduce the number of rejected parcels.

- To better understand the factors that influence parcel rejection.

- To optimize logistics operations by predicting and preventing rejection cases.

By building a model capable of predicting parcel rejection, the logistics company can take proactive measures to handle at-risk parcels more effectively, saving time and resources while improving overall customer satisfaction.

**2. Data Understanding**

Data understanding is a crucial step in the CRISP-DM process because it allows you to familiarize yourself with the data, identify its quality, and discover patterns that may influence your modeling decisions[1]. Here's an expanded version of your initial explanation:

**2.1 Data Overview**

The dataset consists of 12,183 records related to parcels, with several features capturing various attributes:

- Parcel Attributes:

- `PackageWt`: Weight of the parcel in kilograms.

- `PackageCirc`, `PackageCircLength`, `PackageHeight`, `PackageVol`, `PackageWidth`: Dimensions of the parcel, which are important in determining the size and shape of the parcel.

- Time Attributes:

- `RegAddedDate`: The date when the parcel was registered.

- `RegTimestamp`: The timestamp showing the exact time of registration.

- Categorical Attributes:

- `ShiftType`: The type of shift during which the parcel was processed, such as day or night shifts.

- `PackageOverlap`: A categorical variable indicating whether the parcel overlaps with any constraints.

- Target Attribute:

- `Rejected`: The target variable, where 1 indicates the parcel was rejected and 0 indicates it was accepted.

This dataset is composed of both numerical and categorical features. The features help to form a holistic view of the parcel's characteristics, which can influence its rejection likelihood.

**2.2 Initial Data Exploration**

Before diving into any data processing, the initial exploration helps identify the following:[1]

- Class Distribution:

The distribution of the target variable (`Rejected`) shows class imbalance, where rejected parcels are significantly fewer compared to accepted ones. For instance, if only 10% of the parcels were rejected, this imbalance would need to be addressed during modeling.

- Missing Data:

A data quality report revealed missing values, particularly in the `PackageWt` column. The presence of missing values is often due to incorrect data capture or entry errors. Handling these missing values is essential for model accuracy and reliability.

**2.3 Data Visualization**

To enhance understanding, various plots were used:

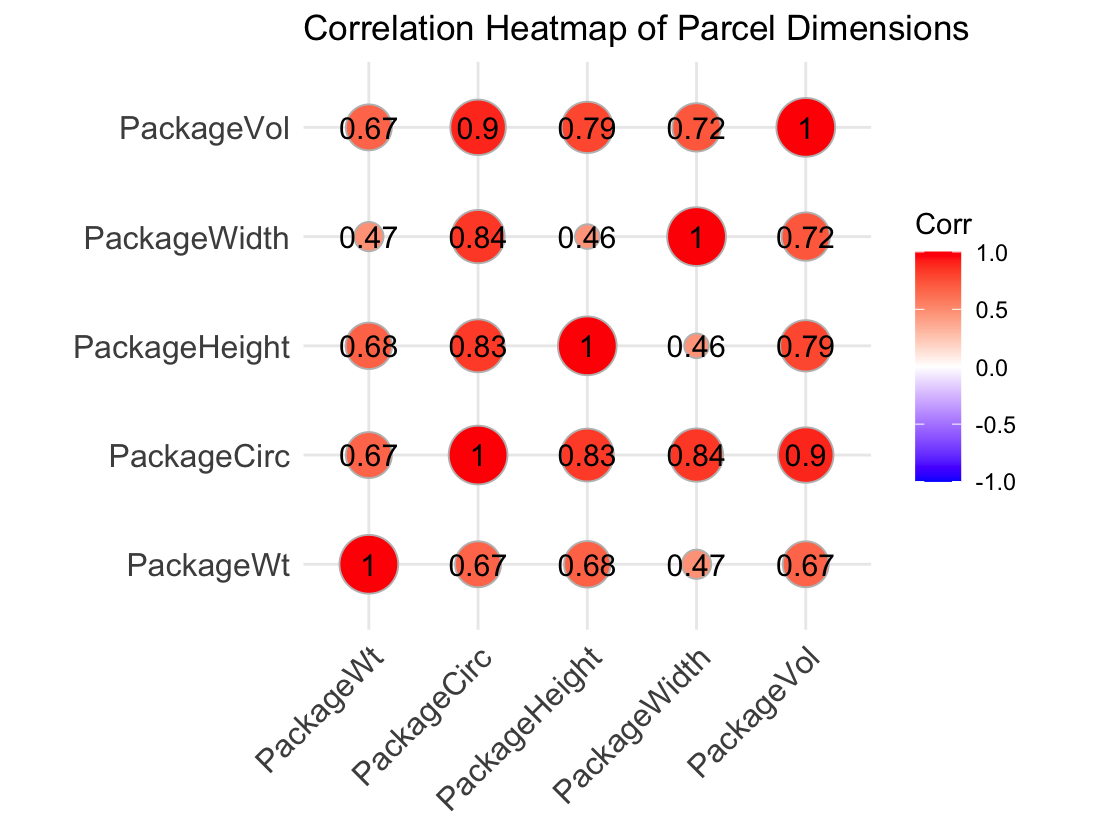
- Box Plots: To visualize the distribution of continuous variables like `PackageWt` and ‘PackageHeight’. These plots help in identifying skewness, kurtosis, and potential outliers.[1]

A graph with a bar graph

Description automatically generatedA graph with a box and a line

Description automatically generated with medium confidence

- Correlation Heatmaps: Used to identify relationships between variables, such as correlations between parcel dimensions (width, height, and weight). This could reveal multicollinearity or help with feature selection.[1]



**3. Data Preparation**

This stage transforms raw data suitable for modeling. It includes handling missing data, outliers, feature engineering, and scaling,

as well as addressing class imbalance. Here's a more detailed explanation of the steps involved:

**3.1 Handling Missing Values**

Missing Data Strategy:

1. Mean/Mode Imputation:

- Numeric Columns: For numerical columns other than `PackageWt`, missing values were replaced by the mean of the particular column.

- Categorical Columns: For categorical columns like `ShiftType` and `PackageOverlap`, missing values were imputed using the mode (most frequent value).

# Perform data cleaning (handling missing values)

fill\_missing\_values <- function(data) {

data$RegAddedDate <- parse\_date\_time(data$RegAddedDate, orders = "ymd HMS", tz = "UTC")

data$RegTimestamp <- parse\_date\_time(data$RegTimestamp, orders = "ymd HMS", tz = "UTC")

for (col in colnames(data)) {

if (is.character(data[[col]]) || is.logical(data[[col]])) {

mode\_value <- as.character(names(sort(table(data[[col]]), decreasing = TRUE)[1]))

data[[col]][is.na(data[[col]])] <- mode\_value

} else if (is.numeric(data[[col]]) && col != "PackageWt") {

mean\_value <- mean(data[[col]], na.rm = TRUE)

data[[col]][is.na(data[[col]])] <- mean\_value

}

}

return(data)

}

2. Ordinary Least Squares (OLS) Regression:

- PackageWt Imputation: Since weight is an important predictor for parcel rejection, missing values in this column were corrected using OLS regression. The idea was to predict the missing weights based on other correlated parcel dimensions (`PackageCirc`, `PackageCircLength`, `PackageHeight`, `PackageVol`, `PackageWidth`).

# Performing OLS for filling package weight column based on other dimensions of parcels

data\_cleaned <- data %>% drop\_na(all\_of(columns\_to\_check))

sample\_data <- data\_cleaned %>% dplyr::select(PackageCirc, PackageCircLength, PackageHeight, PackageVol, PackageWidth, PackageWt)

train\_data <- sample\_data %>% drop\_na(PackageWt)

beta <- train\_OLS(train\_data)

data <- predict\_OLS(beta, data)

\*Code Reference

- Rationale: The OLS model assumes a linear relationship between the weight and dimensions of the parcels, allowing the estimation of missing weights based on this relationship.

**3.2 Handling Outliers**

Outlier Detection:

Outliers in numerical columns were detected using Z-scores. Any value with a Z-score greater than 3 was considered an outlier, as it falls outside the typical range of the data.[1]

- Handling Strategy: Detected outliers were treated by replacing them with missing values. These missing values were then re-imputed using mean or mode imputation, ensuring that extreme values did not negatively impact the model.[1]

**3.3 Feature Engineering**

Feature engineering consists of creating new variables that better represent the underlying relationships in the data:

1. Parcel Size Categorization: New categorical features were created to represent the size of the parcel.[1] For example:

- Small: Parcels with smaller dimensions and weight below a certain threshold.

- Medium: Parcels falling within a middle range of size and weight.

- Large/Heavy: Parcels that are particularly large or heavy.

\*Code reference:

# Transform data as needed (adding new features based on existing features)

feature\_engineering <- function(df\_cleaned) {

df\_cleaned <- df\_cleaned %>%

dplyr::select(-Barcode, -PackageAdviceSystem, -LocIDSecondary, -PackageType) %>%

mutate(PackageHeight = PackageHeight \* 1000,

Package\_size\_width = case\_when(

PackageWidth < 0.3 ~ "Small",

PackageWidth < 0.6 ~ "Medium",

TRUE ~ "Large"),

Package\_size\_height = case\_when(

PackageHeight < 150 ~ "Small",

PackageHeight < 270 ~ "Medium",

TRUE ~ "Large"),

Package\_size\_weight = case\_when(

PackageWt < 5 ~ "Small",

PackageWt < 25 ~ "Medium",

TRUE ~ "Heavy"),

day\_of\_week = wday(RegAddedDate, label = TRUE))

return(df\_cleaned)

}

- Rationale: Parcel size can be a strong indicator of whether a parcel is rejected, as larger and heavier parcels may be more prone to processing issues.

2. Time Features:

- Day of the Week: Extracted from the `RegAddedDate` column to capture patterns in parcel rejection based on the day of the week (e.g., more rejections on weekends or certain busy days).[1]

3. One-Hot Encoding: Applied to categorical variables like `ShiftType` and `PackageOverlap` to convert them into numerical representations suitable for machine learning algorithms.

4. Label Encoding: For other categorical features that have a natural ordinal relationship, label encoding was used to map each category to an integer.

A pie chart with numbers and a few different colored circles

Description automatically generated with medium confidence

**3.4 Handling Class Imbalance**

\*Code Reference

# Performing upsampling using SMOTE

library(smotefamily)

handle\_imbalance <- function(data, numerical\_columns) {

# Ensure all columns are numeric

numerical\_columns <- names(data)[sapply(data, is.numeric)]

# Separate features and target

features <- data %>% dplyr::select(all\_of(numerical\_columns))

label <- data$Rejected

# Apply SMOTE

smote\_output <- SMOTE(features, label, K = 5)

# Extract synthetic data

smote\_data <- smote\_output$data

X\_smote <- smote\_data[, 1:(length(numerical\_columns) - 1)]

y\_smote <- smote\_data[, length(numerical\_columns)]

# Combine features and labels

combined\_df <- cbind(X\_smote, y\_smote)

return(combined\_df)

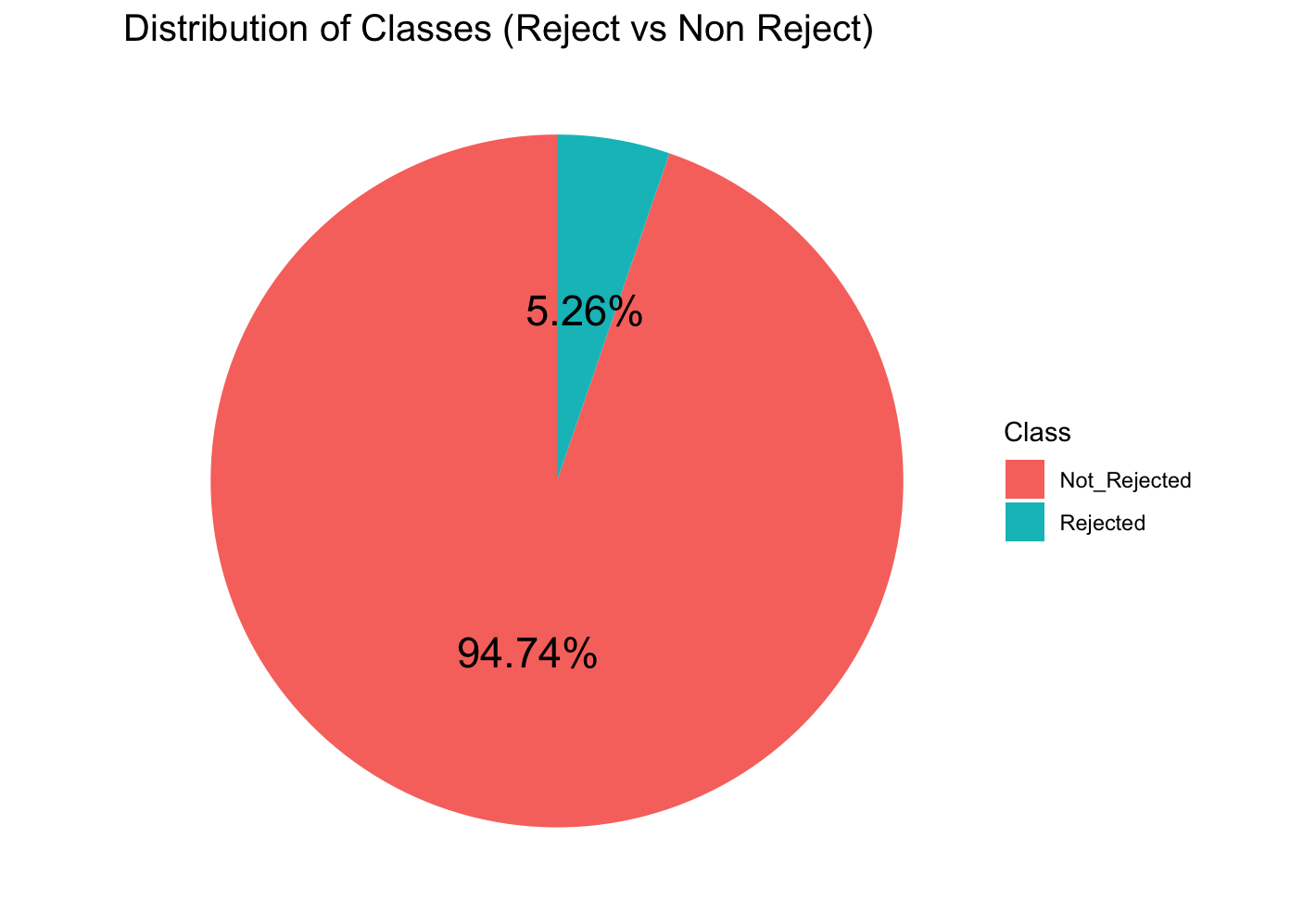
}

Class imbalance was a significant challenge since the number of rejected parcels was much lower than the number of accepted ones. SMOTE (Synthetic Minority Over-sampling Technique) was applied to balance the classes:

- Before SMOTE: The dataset was heavily skewed toward accepted parcels, with very few rejected parcels. This imbalance could lead to a model that is biased toward predicting that a parcel will be accepted.

A graph with a blue rectangle

Description automatically generated



- After SMOTE: The dataset was balanced by generating synthetic examples of rejected parcels. This ensured that both classes were represented equally during model training, reducing bias.

A graph showing a blue rectangular object

Description automatically generated with medium confidence

A diagram of a number of classes

Description automatically generated

4. Modeling

**4.1. Model Selection:**

The following models were selected for training and evaluation:

1. Decision Tree Classifier: A simple yet effective model capable of capturing non-linear relationships in the data.

2. Logistic Regression: A linear model suited for binary classification tasks.

Both models were trained on the SMOTE-balanced dataset.

**4.2. Decision Tree:**

- Training and Evaluation: The model is trained using 70% of the dataset, and the remaining 30% was used for testing.

- Metrics: Evaluation is performed using the confusion matrix and the area under the curve (AUC).

train\_decision\_tree <- function(data) {

# Split the data into training and testing sets

trainIndex <- createDataPartition(data$Rejected, p = .7, list = FALSE)

train\_data <- data[trainIndex, ]

test\_data <- data[-trainIndex, ]

# Train the decision tree model

model <- rpart(Rejected ~ ., data = train\_data, method = "class")

# Make predictions on the test set: class labels

prediction <- predict(model, newdata = test\_data, type = "class")

# Make predictions on the test set: probabilities for the positive class

prediction\_probs <- predict(model, newdata = test\_data, type = "prob")[, 2]

# Generate the confusion matrix

confusion <- confusionMatrix(as.factor(prediction), as.factor(test\_data$Rejected))

# Compute AUC using ROCR package

pred <- prediction(prediction\_probs, test\_data$Rejected)

auc <- performance(pred, measure = "auc")@y.values[[1]]

return(list(model = model, confusion = confusion, auc = auc))

}

Confusion Matrix:

- True Positives: 513

- False Positives: 1023

- True Negatives: 2310

- False Negatives: 258

- Accuracy: 97.28%

- Sensitivity (Recall): 98.35%

- Specificity: 96.14%

- Positive Predictive Value (Precision): 95.31%

- Balanced Accuracy: 97.24%

- Kappa: 0.9455

These results indicate that the decision tree model performs very well across all metrics, with a particularly high sensitivity, specificity, and overall accuracy.

**4.3. Logistic Regression:**

The Logistic Regression model was also trained using 70% of the data, with the remaining 30% used for testing. Below are the code and results for the Logistic Regression model:

Confusion Matrix:

- True Positives (1, 1): 5969

- False Positives (1, 2): 98

- False Negatives (2, 1): 143

- True Negatives (2, 2): 209

Performance Metrics:

- Accuracy: 96.25%

- Sensitivity (Recall): 97.66%

- Specificity: 68.08%

- Positive Predictive Value (Precision): 98.38%

- Negative Predictive Value: 59.38%

- Balanced Accuracy: 82.87%

- Kappa: 0.6146

These metrics show that while Logistic Regression performs well in terms of accuracy and sensitivity, it struggles with specificity and balanced accuracy. This means it is less effective at identifying negative cases.

\*Code Reference

run\_logistic\_regression <- function(data) {

# Print the dimensions of the dataset

print(dim(data))

# Separate features and labels

X <- data %>% dplyr::select(-Rejected)

Y <- data$Rejected

# Split the data into training and testing sets

set.seed(123)

trainIndex <- createDataPartition(Y, p = .8, list = FALSE)

X\_train <- X[trainIndex, ]

X\_test <- X[-trainIndex, ]

y\_train <- Y[trainIndex]

y\_test <- Y[-trainIndex]

# Train Logistic Regression model

lr <- cv.glmnet(as.matrix(X\_train), y\_train, family = "binomial", alpha = 1, standardize = TRUE)

# Make predictions on the test set

y\_pred <- predict(lr, newx = as.matrix(X\_test), s = "lambda.min", type = "class")

# Convert predictions to numeric if they are not already

y\_pred <- as.numeric(y\_pred)

y\_test <- as.numeric(y\_test)

# Evaluate the model

accuracy <- mean(y\_pred == y\_test)

print(paste("Accuracy:", accuracy))

print("---------------------------------------")

f1 <- F1\_Score(y\_test, y\_pred)

print(paste("F1 Score:", f1))

print("---------------------------------------")

# Calculate Mean Squared Error (MSE) and Root Mean Squared Error (RMSE)

mse <- mean((y\_test - y\_pred)^2)

rmse <- sqrt(mse)

print(paste("RMSE:", rmse))

print("---------------------------------------")

# Generate and print the confusion matrix

conf\_matrix <- confusionMatrix(as.factor(y\_pred), as.factor(y\_test))

print("Confusion Matrix:")

print(conf\_matrix)

}

5. Evaluation

Comparison of Models

1. Decision Tree:

- Accuracy: 97.28%

- Sensitivity (Recall): 98.35%

- Specificity: 96.14%

- Balanced Accuracy: 97.24%

- Kappa: 0.9455

2. Logistic Regression:

- Accuracy: 96.25%

- Sensitivity (Recall): 97.66%

- Specificity: 68.08%

- Balanced Accuracy: 82.87%

- Kappa: 0.6146

From these metrics, the decision tree outperforms the logistic regression model in almost every category, particularly in terms of specificity, balanced accuracy, and Kappa score, suggesting that it provides a more balanced and accurate prediction across both classes.

However, if higher sensitivity and precision are more crucial for the business context (i.e., identifying rejected parcels), Logistic Regression may still be considered viable despite its lower specificity.

A graph of different colored bars

Description automatically generated

6. Deployment

Recommendations:  
- Decision Tree Model: Given its higher accuracy and balanced metrics, the Decision Tree model is recommended for deployment. This model can be integrated into the company's logistics system to flag parcels likely to be rejected.  
- Logistic Regression Model: Still a valid choice, especially if model simplicity and interpretability are preferred.

Next Steps:  
- Deploy the Decision Tree model in a production environment.  
- Continuously monitor model performance and retrain as needed with new data.

Data Export:  
- The final processed dataset was exported to a Parquet file for further analysis or model training.

This could trigger interventions such as re-measuring parcels, adjusting packaging, or notifying the customer of potential issues.

Conclusion

The Decision Tree model exhibited better performance in terms of accuracy, specificity, and overall balanced accuracy, making it the better choice for this use case. However, the Logistic Regression model also provides a strong alternative, particularly in cases where sensitivity and precision are of higher importance.

References

[1]<https://www.researchgate.net/publication/341347141_A_Predictive_Analytics_Model_for_E-commerce_Sales_Transactions_to_Support_Decision_Making_A_Case_Study>