BIN 381

Project Milestone 4

Members

Jo-Anne van der Wath (577394)
Henry Roux (577440)
Armandre Erasmus (577311)
Chaleigh Storm (577716)

Table of Contents

Tables of Figures

Table of Tables

Introduction

This milestone focuses on the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework's fourth phase, modelling, which is crucial for creating classification models that precisely identify people who meet the requirements for service offers based on demographic information. This milestone highlights how crucial it is to use the right supervised learning strategies to solve the specified business problem. The intention is to create preliminary models that can be improved upon in later milestones, guaranteeing a complete comprehension of the modelling procedure and its consequences for practical uses. This milestone focusses on improving on the models created in milestone 3 and selecting the most appropriate model for this project. Each models' characteristics and evaluation metrics will be analysed to create a conclusion.

Choosing appropriate modelling techniques, recording the selected algorithms, and outlining their assumptions are the activities listed for this milestone. This methodical approach will direct the creation of a strong Proof of Concept (PoC) that acts as a basis for finalizing the models and will assist in choosing the final model.

Modelling Technique

Logistic Regression

1. Definition
   Logistic regression is a statistical model used to predict a binary outcome from one or more predictor variables. Unlike linear regression, which predicts a continuous output, logistic regression is specifically designed for cases where the dependent variable is categorical, typically binary, such as variables which have yes/no values (Hosmer, et al., 2013). It uses a logistic function to model the relationship between the dependent variable and the independent variables, providing probabilities that fall between 0 and 1. This makes it particularly useful for classification problems (Kleinbaum & Klein, 1994).

2. Reasons for use
   One of the main reasons for using logistic regression is its interpretability. It is a relatively simple model, meaning its coefficients can be understood as representing the change in the odds of the dependent variable occurring with a one-unit change in the predictor variable.
   In the context of predicting eligibility, logistic regression works well because it can handle both continuous and categorical variables effectively. For example, a model might use continuous variables like age and income, along with categorical variables such as marital status, to predict whether someone is eligible for a service. The logistic regression model uses a logistic function to map the combination of these independent variables to a probability that the dependent variable will occur (Hosmer, et al., 2013). It should provide an accurate prediction model for determining the classes of eligibility for the customers.

3. Assumptions
   - **Binary Dependent Variable:** Logistic regression assumes that the outcome variable is binary (0/1). This makes it a natural choice for classification problems where the goal is to predict one of two outcomes

- **Linear Relationship Between Log Odds and Predictors**: Although logistic regression models a binary outcome, it assumes that the log odds (logarithm of the odds) of the outcome is linearly related to the predictor variables.
- **Independence of Errors:** It assumes that the observations are independent of each other and that there is no multicollinearity among the predictors (Hosmer, et al., 2013).
- **No Perfect Separation**: Logistic regression requires that the predictors are not perfectly correlated with the outcome. In other words, there should be no combination of predictor variables that perfectly predicts the outcome (Kleinbaum & Klein, 1994).
- **Sufficient Sample Size**: Logistic regression generally requires a large sample size to provide reliable estimates of the relationship between predictor variables and the outcome.

Decision Tree

1. Definition
   The C4.5 algorithm is a popular decision tree algorithm often used for classification tasks. It is known for its capability to generate visual models that help interpret relationships between input data and outcomes. In the context of predicting service eligibility, the C4.5 algorithm would allow for easy visualization of how demographic factors influence eligibility.

2. Reasons for Use

A key reason for selecting C4.5 is its versatility. As a non-parametric method, it does not require the data to follow any particular distribution, making it suitable for handling both categorical and numerical data (Kotsiantis, 2011). This makes it an excellent fit for datasets that have a mix of continuous and discrete features, such as demographic and financial information that are present in the customer dataset.

3. Assumptions

One of the strengths of the C4.5 algorithm is that it operates without strict assumptions about the underlying data. Unlike many statistical models, it does not assume linearity or normal distribution, making it flexible across a wide range of data types (Fouché & Langit, 2011).

Random Forest

1. Definition

Random Forest is an ensemble learning technique that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) from individual trees. It is designed to combine the simplicity of decision trees with the power of ensemble learning to create a more robust and accurate model. This method is useful for increasing prediction accuracy and preventing overfitting, a common issue with individual decision trees (Breiman, 2001).

2. Reasons for use

The key advantage of Random Forest is its ability to enhance generalization on unseen data. This is achieved by combining predictions from multiple trees, thereby limiting the variance that leads to overfitting. Overfitting occurs when a model captures noise or peculiarities in the training data,

resulting in poor performance on new data. Random Forest reduces this risk by averaging the predictions of numerous trees, ensuring the model is not overly dependent on any single tree's predictions (Breiman, 2001).

Another benefit of using Random Forest is its ability to evaluate the importance of various features in the dataset. This makes it a valuable tool for identifying the most relevant demographic characteristics in predicting eligibility for services, as it highlights which features contribute the most to the model's predictive power (Hastie, et al., 2009). Applying this to the customer dataset will not only allow for the most important attributes to be identified but will also limit overfitting which will improve the accuracy of the model.

3. Assumptions

Random Forest makes the assumption that the decision trees built from different samples of the data will collectively capture more information than any single tree alone. It also assumes that the more diverse the trees, the better the ensemble will perform. This is why Random Forest uses techniques like bagging (bootstrap aggregating) to create different subsets of the data for each tree. By using random samples and splitting points, Random Forest assumes that the aggregated result of these trees will provide a more accurate and generalized prediction (Breiman, 2001).

Test Design

This section describes how to evaluate three classification models: Logistic Regression, Decision Tree (C4.5), and Random Forest to find which model is most suitable for predicting client eligibility for service contracts. These models were chosen because they are well-suited for classification tasks, which are aimed at finding variables that affect customer service eligibility other than annual salary. The dataset will be separated into training and test sets to ensure unbiased evaluation to prevent 'overfitting'. Models will be compared using performance indicators such as accuracy, precision, recall, F1 score, and confusion matrices. The purpose is to find the model that best satisfies the project's business objectives.

Overfitting in machine learning is when an algorithm fits its training data too closely, or perhaps too precisely, producing a model that is unable to draw reliable conclusions or predictions from any other data (IBM, 2024).

Data Splitting Strategy

A conventional data splitting strategy to guarantee reliable model evaluation will be used:

- Training set: The model is trained using 80% of the dataset.
- Test set: 20% of the dataset is put aside for the last assessment of the model. By ensuring that model performance is evaluated on unknown data, this splitting technique reduces the possibility of overfitting and offers an objective assessment of the model's generalizability.

The code used to split the dataset is as follows:

```
set.seed(123)
```

Reproducible random numbers are obtained with this function. When a random function is called, it aids in producing the same random numbers each time. This aids in producing data sets for analysis that may be repeated (Biet, 2023).

```
train_index <- createDataPartition(custData$Eligible, p = 0.8, list = FALSE)
```

The object "train_index" is used to store the split dataset obtained from the "createDataPartition()" function. The parameters for this function are as follows:

- 'Cust$Eligible': This parameter must provide the target attribute of the dataset, which is in this case the "Eligible" attribute.
- 'p = 0.8': This parameter provides the percentage of the dataset that should be used for training the dataset, in this case it's 80%. The rest of the dataset (in this case 20%) will be used for testing.

```
train_data <- custData[train_index, ]
```

The object "training_set" is used to contain the training portion of the dataset. This is done by indexing the "custData" using the "train_index" object.

```
test_data <- custData[-train_index, ]
```

The object "test_set" is used to contain the training portion of the dataset. This is done by indexing the "custData" using the opposite of the "train_index" object.

Model Evaluation Metrics

The following measures will be used to assess each model's performance:

- **Accuracy**: The frequency with which a machine learning model accurately predicts the result is measured by its accuracy. Accuracy is computed by dividing the total number of predictions by the number of right predictions, accuracy may be computed (Evidently AI, 2024).
- **Precision**: The frequency with which a machine learning model accurately predicts the positive class (eligible customers) is measured by this statistic. By dividing the total number of occurrences the model predicted as positive (including true and false positives) by the number of accurate positive predictions (true positives), you can compute precision (Evidently AI, 2024).
- **Recall**: A machine learning model's recall is a metric that expresses how frequently it properly selects positive examples, or true positives, out of all the real positive samples in the dataset. Recall can be computed by dividing the total number of positive cases by the number of true positives. The latter consists of false negative results (missed cases) and true positives (successfully detected cases) (Evidently AI, 2024).
- **F1-Score**: An important machine learning metric that offers a fair assessment of a model's recall and precision is the F1 Score. The precision recall F1 score framework requires the F1 Score formula, which is obtained from the harmonic mean of precision and recall. When there is an imbalance in the distribution of classes, this statistic becomes especially helpful (Geeksforgeeks, 2024).
- **Confusion Matrix**: Two target classes are conceivable in binary classification; these are usually designated as "positive" and "negative," or "1" and "0." The target (positive class) in an example of spam is "spam," whereas the negative class is "not spam". One must consider both accurate and inaccurate predictions, ignoring the class designation, when assessing accuracy. It can be "correct" or "wrong" in binary classification, but, in two distinct ways. This includes the following four classifications (Jacob Murel Ph.D., 2024):
    - True positive (TP): An email that the model accurately classifies as spam, and it is actually spam.
    - True negative (TN): is an email that the model correctly classifies as non-spam, and it isn't spam.
    - False Positive (FP): An email that the model mistakenly classifies as spam but is actually not (a "false alarm").
    - False Negative (FN): An email that the model mistakenly classifies as non-spam but is in fact spam ("missed spam").

The layout of the confusion matrix is as follows:
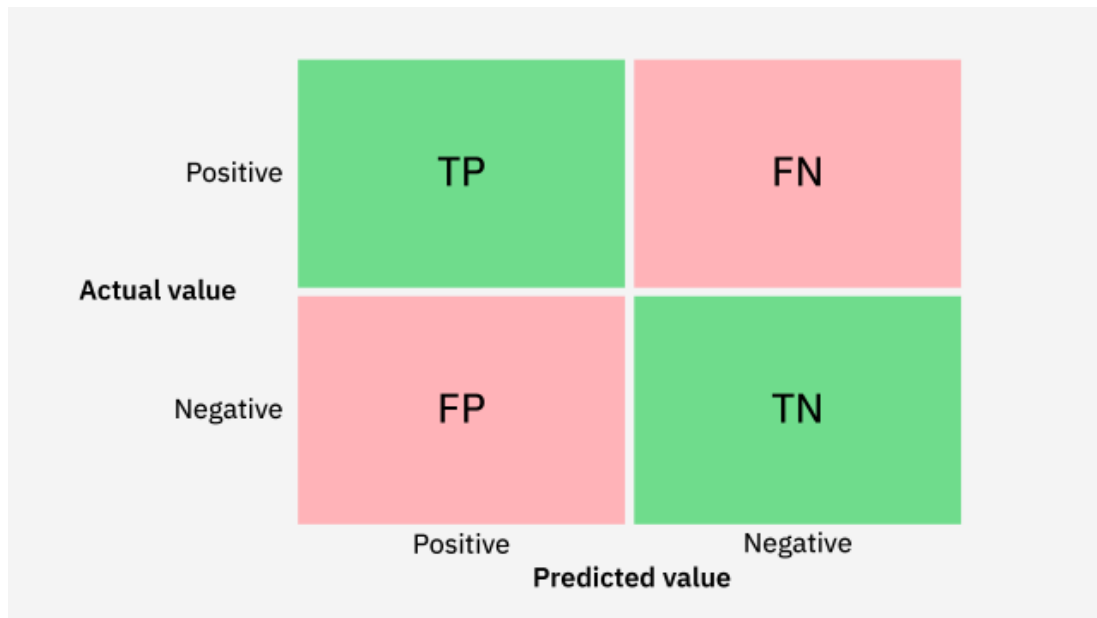
Figure 1: Confusion Matrix Layout (Jacob Murel Ph.D., 2024)

The confusion matrix is used to calculate come of the evaluation metrics. The formulas are given below.

$$Accuracy = \frac{TP + TF}{TP+FP+TF+FN}$$

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F = \frac{2PR}{P+R}$$

TP = True Positive
FP = False Positive
TN = True Negative
FN = False Negative

F = F1 Score
P = Precision
R = Recall

Figure 2: Evaluation Metrics Equations (Jacob Murel Ph.D., 2024)

Model Description

Splitting of Data

The following libraries will be used during the building of the models:

```
library(dplyr)
library(rpart)
library(rpart.plot)
library(caret)
library(caTools)
library(randomForest)
```

Figure 3 Libraries

- **dyplr:** This package is used for data manipulation. It will be used for factorisation, summarisation and other data manipulation needed to ensure the target attribute (Eligible) is ready for modelling (Data Carpentry, 2024).
- **rpart:** rpart is used to build the decision tree.
- **rpart.plot:** rpart.plot will be used to plot the decision tree.
- **caret:** The caret package will be used to build and train the logistic regression model (Kuhn, 2019).
- **caTools:** The caTools package can be used for splitting the datasets (geeksforgeeks, 2024).
- **randomForest:** The random forest package will be used to build the random forest model.

To start, the prepared dataset will be read and the structure of the data will be examined.

```
> #Load the dataset
> custData <- read.csv("CustData2_Prepared.csv")
>
> #Explore the structure of the dataset
> str(custData)
'data.frame':   191317 obs. of  26 variables:
 $ Annual_Salary                   : num  0.872 -0.156 -0.455 0.266 -0.621 ...
 $ Gross_Pay_Last_Paycheck         : num  0.591 -0.267 -0.511 0.432 -0.637 ...
 $ Gross_Year_To_Date              : num  0.827 -0.267 -0.884 0.602 -0.417 ...
 $ Gross_Year_To_Date_FRS_Contribution: num  0.823 -0.268 -0.884 0.599 -0.417 ...
 $ Age                             : num  2.088 0.021 0.688 2.022 -0.779 ...
 $ Household_Size                  : int  2 3 2 2 2 2 2 3 2 ...
 $ Years_Residence                 : int  2 5 2 3 4 4 4 5 3 ...
 $ Marital_Statusdivorced          : int  0 0 0 0 0 0 0 0 0 ...
 $ Marital_Statusmarried           : int  1 0 1 0 0 0 0 0 1 0 ...
 $ Marital_Statussingle            : int  0 1 0 1 1 1 1 1 0 1 ...
 $ Marital_Statuswidowed           : int  0 0 0 0 0 0 0 0 0 ...
 $ EducationBach.                  : int  0 1 0 1 0 0 0 0 1 1 ...
 $ EducationHS.grad                : int  1 0 1 0 0 0 0 0 0 0 ...
 $ EducationMasters                : int  0 0 0 0 1 1 1 1 0 0 ...
 $ OccupationCleric.               : int  1 0 1 0 0 0 0 0 0 0 ...
 $ OccupationExec.                 : int  0 1 0 0 0 0 0 0 1 0 ...
 $ OccupationProf.                 : int  0 0 0 0 1 1 1 1 0 0 ...
 $ OccupationSales                 : int  0 0 0 1 0 0 0 0 0 1 ...
 $ Salary_GroupLow                 : int  0 0 1 0 1 0 0 0 0 ...
 $ Salary_GroupMedium              : int  0 1 0 0 0 0 0 1 0 0 ...
 $ Salary_GroupHigh                : int  0 0 0 1 0 1 0 0 0 1 ...
 $ Salary_GroupVery.High           : int  1 0 0 0 0 0 1 0 1 0 ...
 $ Frequency_Title                 : int  8206 1030 28 67 1368 10809 7 723 1030 101 ...
 $ Frequency_Department            : int  16988 16925 6138 16925 8895 18158 14 29331 8895 16925 ...
 $ Frequency_Country_ID            : int  2079 2079 2079 2079 2079 2079 2079 2079 2079 2079 ...
 $ Eligible                        : int  1 1 0 1 0 1 1 1 1 1 ...
```

Figure 4 Load and Examine Dataset

As can be seen, the dataset contains only numerical attributes after preprocessing has been done.

```
> set.seed(123)
> train_index <- createDataPartition(custData$Eligible, p = 0.8, list = FALSE)
> train_data <- custData[train_index, ]
> test_data <- custData[-train_index, ]
```

Figure 5 Dataset Splitting

The splitting of the data can occur. The seed will be set to 123 to ensure that reproducibility is possible. This is done by setting a random seed.

The data is split using the createDataPartition function from the caret library. The dataset is split into 80% training data and 20% testing data. This will create a balanced split of data based on the Eligible attribute which is the target attribute.

Logistic Regression

Logistic regression is the first model that will be implemented and evaluated.

```
> logisticRegressionModel <- glm(formula = Eligible~ . -Annual_Salary,
+                                 data = train_data,family = 'binomial')
```

Figure 6 Logistic Regression Model

Parameters:

- **Function**: The logistic regression model will be built using the glm function.
- **Target**: This will create a generalised linear model that will find and model the relationship between the target attribute ("Eligible") and the independent attributes (datacamp, 2024).
- **Exclusion:** The Annual_Salary attribute has been excluded from the model to ensure that the model focuses on other attributes and not just Annual_Salary, as it was used to create the Eligibility attribute.
- **Data**: The model will be built on the training data.
- **Family**: The family = 'binomial' parameter assigns the logistic regression algorithm which is used when the target attribute is binary.

```
> logisticRegressionPrediction <- predict(logisticRegressionModel, newdata = test_data, type
='response')
> head(logisticRegressionPrediction)
           2            4            5            6           14           25
4.681861e-01 1.000000e+00 4.870882e-10 1.000000e+00 1.000000e+00 1.000000e+00
> logisticRegressionY_pred = ifelse(logisticRegressionPrediction >0.5, 1, 0)
```

Figure 7 Predict Eligibility using Logistic Regression

The logistic regression model can then be used to make predictions using the test data, as can be seen in the figure above. This will predict whether users will be eligible or not. If the prediction value is above 0.5 then they will be eligible, and if it is below 0.5 then they will not be eligible.

```
> #Confusion matrix of Logistic Regression
> logisticRegression_matrix <- table(actual = test_data$Eligible, predicted = logisticRegres
sionY_pred)
> logisticRegression_matrix
      predicted
actual     0     1
     0 12483  1004
     1   985 23791
>
> logisticRegression_truePositive <- logisticRegression_matrix[1, 1]
> logisticRegression_trueNegative <- logisticRegression_matrix[2, 2]
> logisticRegression_falsePositive <- logisticRegression_matrix[1, 2]
> logisticRegression_falseNegative <- logisticRegression_matrix[2, 1]
```

Figure 8 Confusion Matrix for Logistic Regression

A confusion matrix is generated that compares the test data values with the predicted values. The values in the matrix will then be used to find the number of true positives, true negatives, false positives and false negatives produced by the model.

```
> # Calculate Evaluation Metrics
> logisticRegression_accuracy <- round((sum(diag(logisticRegression_matrix)) / sum(log
isticRegression_matrix)), 2)
> logisticRegression_precision <- round(logisticRegression_truePositive / (logisticReg
ression_truePositive + logisticRegression_falsePositive), 2)
> logisticRegression_recall <- round(logisticRegression_truePositive / (logisticRegres
sion_truePositive + logisticRegression_falseNegative), 2)
> logisticRegression_f1_score <- round(2 * (logisticRegression_precision * logisticReg
ression_recall) / (logisticRegression_precision + logisticRegression_recall), 2)
```

Figure 9 Calculation of Performance Metric - Logistic Regression

The accuracy, precision, recall and f1 score for the logistic regression model is then calculated. This can be seen in the figure above.

Decision Tree

The Decision Tree model is the next model to be implemented and evaluated.

```
> ##DECISION TREE
> #Build Decision Tree Model
> decisionTreeModel <- rpart(Eligible ~ . -Annual_Salary, data = train_data, method =
'class')
```

Figure 10 Building of Decision Tree Model

To building of the Decision Tree model will be done using rpart.

Parameters:

- **Function**: rpart will be used to build the decision tree model.
- **Target**: The model will be predicting the target attribute ("Eligible").
- **Exclusion**: Annual_Salary will be excluded from the model to ensure that other attributes are considered as Annual_Salary will bias the data as it was used to calculate the Eligible attribute.
- **Data**: The model will be built on the training data.
- **Method**: The method = 'class' parameter specifies that a classification model should be built.

After the decision tree model has been built, the decision tree can be visualized. This can be seen in the following figure:
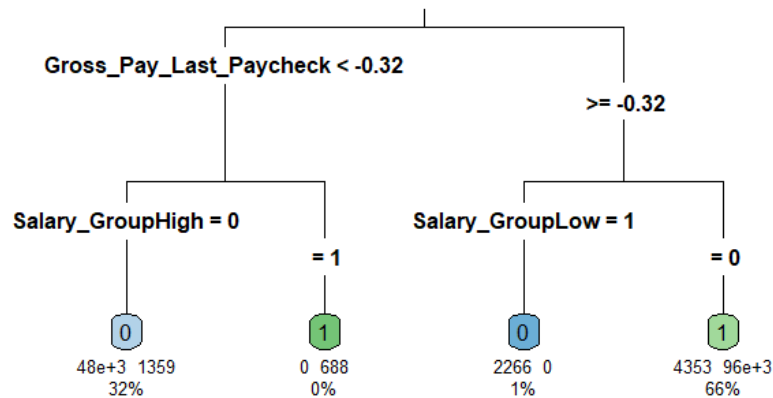
11

Figure 11 Decision Tree

The main attribute which determines user eligibility is Gross Pay Last Paycheck. The split is based on the scaled threshold of -0.32.

If the customer has a Gross_Pay_Last_Paycheck of less than -0.32, then the branch is further split into branches based on the Salary_GroupHigh.

If the Salary_GroupHigh = 0, this indicates that the customer does not have a high annual salary. If the Gross_Pay_Last_Paycheck is less than -0.32 and the Salary_Group is not high, then the customer will not be eligible for the services provided by LangaSat.

If the Salary_GroupHigh = 1, this indicates that the customer does have a high annual salary. If the Gross_Pay_Last_Paycheck is less than -0.32 and the Salary_Group is high, then the customer will be eligible for the services provided by LangaSat.

The second branching of the main branch is based on of the Gross_Pay_Last_Paycheck is more than the scaled threshold of -0.32. This branch further splits based on the whether or not the annual salary of the customer is low.

If the Salary_GroupLow = 0, this indicates that the customer does not have a low annual salary. If the Gross_Pay_Last_Paycheck is more than -0.32, and the Salary group is not low, then the customer is not eligible for the services provided by LangaSat.

If the Salary_GroupLow = 1, this indicates that the customer does have a low annual salary. If the Gross_Pay_Last_Paycheck is more than -0.32, and the Salary group is low, then the customer is eligible for the services provided by LangaSat.

The model can then be used to make predictions using the test data. This can be seen in the following figure:

```
> # Make predictions on the test data
> decisionTreePredictions <- predict(decisionTreeModel, newdata = test_
data. type = 'class')
```

Figure 12 Decision Tree Prediction

The confusion matrix for the decision tree model can be generated. This will compare the predicted values with the test values. The elements of the confusion matrix can be derived from the matrix to allow for evaluation metrics to be calculated.

```
> decisionTreeMatrix <- table(test_data$Eligible, decisionTreePredictions)
> print(decisionTreeMatrix)
   decisionTreePredictions
        0     1
  0 12450  1037
  1   314 24462
>
> decisionTreeTruePositive <- decisionTreeMatrix[1, 1]
> decisionTreeTrueNegative <- decisionTreeMatrix[2, 2]
> decisionTreeFalsePositive <- decisionTreeMatrix[1, 2]
> decisionTreeFalseNegative <- decisionTreeMatrix[2, 1]
```

Figure 13 Decision Tree Matrix

Finally, the performance metrics for the decision tree can be calculated. The accuracy, precision, recall and f1 will be calculated. All metrics will be rounded to two decimal places.

```
> decisionTreeAccuracy <- round((sum(diag(decisionTreeMatrix)) / sum(decisi
onTreeMatrix)), 2)
> decisionTreePrecision <- round(decisionTreeTruePositive / (decisionTreeTr
uePositive + decisionTreeFalsePositive), 2)
> decisionTreeRecall <- round(decisionTreeTruePositive / (decisionTreeTrueP
ositive + decisionTreeFalseNegative), 2)
> decisionTreeF1Score <- round(2 * (decisionTreePrecision * decisionTreeRec
all) / (decisionTreePrecision + decisionTreeRecall), 2)
```

Figure 14 Decision Tree Performance Metric Calculation

Random Forest

The splitting for the Random Forest model will differ slightly from the other models. For the random forest model, the target attribute "Eligible" will have to be categorized first using factoring. Thus, we will reread the prepared data file and categorise "Eligible".

```
> #Reload the dataset
> custData <- read.csv("CustData2_Prepared.csv")
> custData$Eligible <- as.factor(custData$Eligible)
```

Figure 15 Rereading Prepared Data

Then the dataset will be split the exact same way as previously done.

```
> #Split the dataset to 80% training data and 20% testing data
> set.seed(123)
> train_index <- createDataPartition(custData$Eligible, p = 0.8, list = FAL
SE)
> train_data <- custData[train_index, ]
> test_data <- custData[-train_index, ]
```

Figure 16 Split Data for Random Forest

Next, the model can be built:

```
> #Build Random Forest Model
> randomForest_model <- randomForest(Eligible ~ . -Annual_Salary, data = tr
ain_data, ntree = 100, mtry = 3, importance = TRUE)
```

Figure 17 Random Forest Model

Parameters:

- **Function**: The model will be build using the randomForest function.
- **Target**: The model will be built to predict the "Eligible" attribute.
- **Exclusion**: Annual Salary will not be used to build the model, as to avoid bias seeing as the Eligible attribute was created using the Annual Salary.
- **Data**: The model will be built using the training data.
- **Ntree**: ntree refers to the number of decision trees that will be created in the random forest. In this parameter, this number has been set to 100, so 100 decision trees will be generated. Although this may result in better performance, it will also require a lot of computational power which may result in slower results.
- **Mtry**: mtry refers to the number of variables that may be chosen at each split in a tree. This attribute can impact the fitting of the model (be it over or under). This has been set to 3, so three variables may be chosen at each split.
- **Importance**: Importance = true; This parameter forces the model to calculate the importance of the attributes.

The importance of the attributes derived calculated by the model can be visualised using the function varImpPlot.

```
> varImpPlot(randomForest_model)
```

Figure 18 Function for Plotting Importance
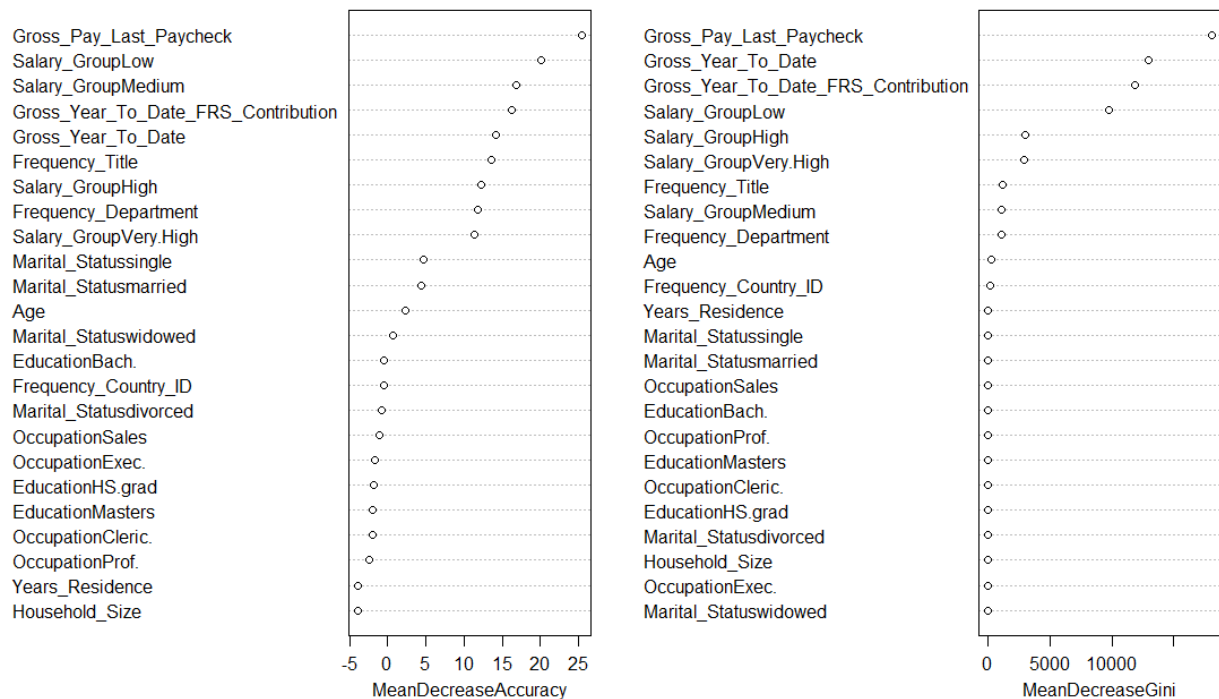
The plot will be as follows:

Figure 19 Plot for Attribute Importance

The first plot shows the mean decrease in model accuracy when an attribute is randomly shuffled. The higher the decrease in accuracy, the more important the attribute.

According to the first plot the three most important attributes are:

1. Gross_Pay_Last_Paycheck
2. Salary_GroupLow
3. Salary_GroupMedium

The second plot shows how much the impurity of the branch node decreases when a specific attribute is used to do the branching. If the decrease is higher, this indicates that the attribute is more important.

According to the second plot, the three most important attributes are:

1. Gross_Pay_Last_Paycheck
2. Gross_Year_To_Date
3. Gross_Year_To_Date_FRS_Contribution

From this, it can be surmised that Gross_Pay_Last_Paycheck is probably the most important attribute.

The random forest model can be used to make predictions using the test data.

```
> #Make Predictions Using Random Forest
> randomForest_predictions <- predict(randomForest_model, newdata = test_data)
```

Thereafter, a confusion matrix will be created using the predictions from the random forest model. The matrix will compare the predictions to the test data target values.

```
> #Matrix for Random Forest
> randomForest_cm <- confusionMatrix(as.factor(randomForest_predictions), as.factor(test_da
ta$Eligible))
> randomForest_matrix <- randomForest_cm$table
>
> randomForest_truePositive <- randomForest_matrix[1, 1]
> randomForest_trueNegative <- randomForest_matrix[2, 2]
> randomForest_falsePositive <- randomForest_matrix[1, 2]
> randomForest_falseNegative <- randomForest_matrix[2, 1]
```

Finally, the performance metrics for the random forest model can be calculated. The metrics that are calculated include the accuracy, precision, recall and f1 score.

```
> #Calculate Evaluation Metrics
> randomForest_accuracy <- round((sum(diag(randomForest_matrix)) / sum(randomForest_matri
x)), 2)
> randomForest_precision <- round(randomForest_truePositive / (randomForest_truePositive +
randomForest_falsePositive), 2)
> randomForest_recall <- round(randomForest_truePositive / (randomForest_truePositive + ran
domForest_falseNegative), 2)
> randomForest_f1_score <- round(2 * (randomForest_precision * randomForest_recall) / (rand
omForest_precision + randomForest_recall), 2)
```

Finally, we can print and examine the performance metrics for all three models.

The performance metrics for the logistic regression model will first be examined. As can be seen in the following figure, the logistic regression model has an accuracy of 95%.

```
> cat("Logistic Regression Accuracy:", logisticRegression_accuracy, "\n")
Logistic Regression Accuracy: 0.95
> cat("Logistic Regression Precision:", logisticRegression_precision, "\n")
Logistic Regression Precision: 0.93
> cat("Logistic Regression Recall:", logisticRegression_recall, "\n")
Logistic Regression Recall: 0.93
> cat("Logistic Regression F1-score:", logisticRegression_f1_score, "\n")
Logistic Regression F1-score: 0.93
```

Next, the performance metrics for the decision tree model will be examined. As can be seen in the figure below, the accuracy of the decision tree model is 96%.

```
> cat("Decision Tree Accuracy:", decisionTreeAccuracy, "\n")
Decision Tree Accuracy: 0.96
> cat("Decision Tree Precision:", decisionTreePrecision, "\n")
Decision Tree Precision: 0.92
> cat("Decision Tree Recall:", decisionTreeRecall, "\n")
Decision Tree Recall: 0.98
> cat("Decision Tree F1-score:", decisionTreeF1Score, "\n")
Decision Tree F1-score: 0.95
```

Figure 24 Decision Tree Performance Metrics

Finally, the performance metrics for the random forest can be examined. As seen in the following figure, the accuracy of this model is 97%.

```
> cat("Random Forest Accuracy:", randomForest_accuracy, "\n")
Random Forest Accuracy: 0.97
> cat("Random Forest Precision:", randomForest_precision, "\n")
Random Forest Precision: 0.98
> cat("Random Forest Recall:", randomForest_recall, "\n")
Random Forest Recall: 0.93
> cat("Random Forest F1-score:", randomForest_f1_score, "\n")
Random Forest F1-score: 0.95
```

Figure 25 Random Forest Performance Metrics

Finally, we can save the models to use later if we choose. This can be done in the following manner:

```
> #Save the logistic regression model
> saveRDS(logisticRegressionModel, file = "logistic_regression_model.rds")
> #Save the decision tree model
> saveRDS(decisionTreeModel, file = "decision_tree_model.rds")
> #Save the random forest model
> saveRDS(randomForest_model, file = "random_forest_model.rds")
```

Figure 26 Save the Models

Model Assessment:

In our project, the models were evaluated using **Accuracy**, **Precision**, **Recall**, and **F1-Score**, which are essential metrics for understanding the performance of machine learning models, particularly in the context of customer eligibility classification. These metrics provide insights into different aspects of the model's performance:

- **Accuracy** reflects the proportion of correct predictions, indicating the overall effectiveness of the model across all classes (Biet, 2023).
- **Precision** focuses on the correctness of positive predictions, helping assess how many of the customers predicted as eligible are truly eligible (Hosmer, Lemeshow & Sturdivant, 2013).
- **Recall** measures the model's ability to identify all relevant positive cases, ensuring that eligible customers are not missed (Hastie, Tibshirani & Friedman, 2009).
- **F1-Score** balances Precision and Recall, which is particularly useful when class distribution is uneven or when both metrics need to be considered equally (GeeksforGeeks, 2024).

The table below highlights the performance of each model in terms of these metrics:

Table 1: Comparing Metric Results

Metric	Logistic Regression	Decision Tree	Random Forest
Accuracy	94.8%	96.5%	97.00%
Precision	92.5%	92.3%	98.42%
Recall	92.6%	97.5%	93.07%
F1-Score	92.6%	94.8%	95.67%

Model Comparisons: Key Insights

Logistic Regression

- **Strengths**: Logistic Regression demonstrated a balanced performance between precision and recall, making it a reliable choice for identifying eligible customers without favoring false positives or false negatives. Due to its simplicity, it is particularly suitable for cases where relationships between predictors and the target variable are linear (Kleinbaum & Klein, 1994). Its interpretability is one of its greatest advantages, as it allows for straightforward understanding of how variables impact the outcome.
- **Weaknesses**: Despite its balanced performance, Logistic Regression's accuracy was lower compared to Decision Tree and Random Forest models, suggesting that it struggles to capture more complex interactions between variables. In datasets where non-linear relationships are present, Logistic Regression may not perform as well as other models that can handle such complexities (Hosmer, Lemeshow & Sturdivant, 2013).

Decision Tree

- **Strengths**: The Decision Tree model, constructed using the C4.5 algorithm, excelled in recall, capturing nearly all eligible customers (Kotsiantis, 2011). This high recall is essential in business scenarios where it is crucial not to miss any eligible customers, as

doing so could result in lost revenue or missed opportunities (Fouché & Langit, 2011). Additionally, its simplicity and ability to visually represent decision-making processes make it an excellent tool for understanding which variables influence customer eligibility.

- **Weaknesses**: However, the Decision Tree's precision, while similar to Logistic Regression, was slightly lower, meaning that it could misclassify some ineligible customers as eligible. This could lead to unnecessary business costs associated with offering services to non-eligible individuals (Hastie, Tibshirani & Friedman, 2009). Furthermore, Decision Trees can overfit the data if not properly pruned, making them less generalizable to new, unseen data (Kuhn, 2019).

Random Forest

- **Strengths**: Random Forest, an ensemble learning technique, significantly outperformed both Logistic Regression and Decision Tree in terms of precision and accuracy (Breiman, 2001). Its ability to aggregate multiple decision trees reduces overfitting and increases the model's robustness, making it ideal for handling complex datasets with high dimensionality (Hastie, Tibshirani & Friedman, 2009). Random Forest's high precision ensures that most customers predicted as eligible are indeed eligible, minimizing false positives, which is critical in business scenarios where such mistakes can lead to costly errors (Breiman, 2001).
- **Weaknesses**: Although Random Forest achieved the highest precision and accuracy, its recall was slightly lower than that of the Decision Tree. This means that it might miss some eligible customers, which could be problematic if the business's primary goal is to ensure that all eligible customers are identified (Kuhn, 2019). Additionally, Random Forest requires more computational resources and time for training, which may affect its scalability, especially in large datasets (Hastie, Tibshirani & Friedman, 2009).

Comparative Analysis and Business Impact

When comparing these models, each brings distinct strengths and trade-offs, making the choice dependent on the specific business goals.

- **Random Forest** stands out for its superior precision and accuracy. It minimizes false positives, which is crucial in scenarios where offering services to non-eligible customers could incur significant costs. In this case, its robustness and ability to handle complex, high-dimensional data make it the best choice when minimizing misclassification errors is critical (Breiman, 2001).
- **Decision Tree** excels in **recall**, making it the preferred model in cases where missing an eligible customer could lead to considerable revenue loss. Its ability to capture almost all eligible customers makes it invaluable for scenarios where maximizing customer engagement is the priority (Kotsiantis, 2011).
- **Logistic Regression**, although not as robust in terms of accuracy, remains a reliable and interpretable model that provides a balanced performance. Its simplicity and ability to handle both continuous and categorical variables make it a strong baseline model, particularly in cases where interpretability is crucial (Hosmer, Lemeshow & Sturdivant, 2013).

Final Recommendation

Based on the evaluation of all metrics, **Random Forest** is recommended for deployment due to its superior performance in both precision and accuracy. This makes it an ideal model for customer eligibility tasks, where minimizing false positives is essential to reduce unnecessary costs and improve business efficiency. However, if the business's priority is to ensure that no eligible customer is overlooked, the **Decision Tree** model would be a better option due to its exceptional recall performance.

In scenarios where interpretability and ease of understanding are more important, **Logistic Regression** should still be considered, especially as a benchmark or baseline model for comparison in future developments (Hosmer, Lemeshow & Sturdivant, 2013).

Conclusion

In this project, we implemented and evaluated three machine learning models—Logistic Regression, Decision Tree, and Random Forest—to predict customer eligibility. Each model provided unique insights and had distinct strengths and weaknesses. Logistic Regression offered a balanced performance between precision and recall but lagged behind in accuracy compared to more complex models. The Decision Tree model excelled in recall, identifying nearly all eligible customers, making it a strong contender for business applications where customer identification is critical. However, the Random Forest model emerged as the best performer in terms of accuracy and precision, reducing the risk of misclassification while maintaining high-quality predictions.

Given the business goal of accurately predicting customer eligibility with minimal errors, the Random Forest model is recommended for deployment due to its superior performance in accuracy and precision. This model's ability to reduce false positives ensures that fewer customers are wrongly classified, providing more reliable predictions. However, in cases where the business goal shifts toward maximizing recall, especially in scenarios where missing any eligible customer could have significant consequences, the Decision Tree model could serve as an alternative.

Future work could explore hyperparameter tuning to further optimize model performance and the inclusion of additional features to improve predictions. By leveraging the insights from this project, the business can deploy the most suitable model to enhance decision-making processes related to customer eligibility.

References

Biet, N., 2023. *Generate Data sets of same Random Values in R Programming – set.seed() Function.* [Online]
Available at: https://www.geeksforgeeks.org/generate-data-sets-of-same-random-values-in-r-programming-set-seed-function/
[Accessed 14 October 2024].

Breiman, L., 2001. Random Forests. In: *Machine Learning.* s.l.:SpringerLink, pp. 5-32.

Data Carpentry, 2024. *Aggregating and analyzing data with dplyr.* [Online]
Available at: https://datacarpentry.org/R-genomics/04-dplyr.html
[Accessed 14 October 2024].

datacamp, 2024. *glm: Fitting Generalized Linear Models.* [Online]
Available at: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/glm
[Accessed 15 October 2024].

Evidently AI, 2024. *Accuracy vs. precision vs. recall in machine learning: what's the difference?.* [Online]
Available at: https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall
[Accessed 14 October 2024].

Fouché, G. & Langit, L., 2011. Introduction to Data Mining. In: *Foundations of SQL Server 2008 R2 Business Intelligence.* s.l.:Apress, Berkeley, CA.

geeksforgeeks, 2024. *caTools Package in R.* [Online]
Available at: https://www.geeksforgeeks.org/catools-package-in-r/
[Accessed 14 October 2024].

Geeksforgeeks, 2024. *F1 Score in Machine Learning.* [Online]
Available at: https://www.geeksforgeeks.org/f1-score-in-machine-learning/
[Accessed 14 October 2024].

Hastie, T., Tibshirani, R. & Friedman, J., 2009. *The Elements of Statistical Learning.* 2nd ed. s.l.:SpringerLink.

Hosmer, D., Lemeshow, S. & Sturdivant, R., 2013. *Applied Logistic Regression.* s.l.:John Wiley & Sons..

IBM, 2024. *What is overfitting?.* [Online]
Available at: https://www.ibm.com/topics/overfitting
[Accessed 14 October 2024].

Jacob Murel Ph.D., E. K., 2024. *What is a confusion matrix?.* [Online]
Available at: https://www.ibm.com/topics/confusion-matrix
[Accessed 14 October 2024].

Kleinbaum, D. G. & Klein, M., 1994. *Logistic Regression A Self-Learning Text.* 3rd ed. s.l.:SpringerLink.

Kotsiantis, S. B., 2011. Decision trees: a recent overview. In: s.l.:SpringerLink, pp. 261-283.

Kuhn, M., 2019. *The caret Package*. [Online]
Available at: https://topepo.github.io/caret/
[Accessed 15 October 2024].