# Telco Customer Churn Analysis Project

Feven Tefera & Maliha Wahedi

MA 5790
December 2024

## Abstract

Customer retention is a crucial challenge for businesses, as customer churn directly impacts revenue and growth. Churn occurs when customers stop using a company's services, making it vital for organizations to understand and address its causes. This analysis focuses on the Telco Customer Churn dataset, which includes detailed customer demographics, account details, and service usage information, to uncover patterns and key factors driving churn.

The main goal of this study is to predict customer churn using predictive modeling techniques to help businesses develop effective retention strategies. By analyzing variables such as service preferences, tenure, and payment methods, the study identifies the factors that influence churn behavior. Linear and non-linear models are applied to provide insights into identifying at-risk customers and improving retention efforts.

The results show that models like Logistic Regression and Mixture Discriminant Analysis (MDA) are effective for predicting churn, with tenure emerging as a particularly influential variable. These findings offer practical guidance for businesses to reduce churn, improve customer satisfaction, and foster long-term loyalty. The study highlights how data-driven approaches can support organizations in tackling churn and promoting sustainable growth.

# Contents

# 1 Background

Customer churn, which refers to the percentage of customers discontinuing a company's product or service within a specific period, poses a significant challenge for the telecommunications industry. High churn rates directly affect customer retention, revenue stability, and overall growth. Research shows that businesses lose 20–30% of their customers annually, making it difficult to maintain consistent operations and revenue. In the U.S. telecom sector, churn rates range from 25% to 40%, with one in four or five customers leaving each year. This leads to substantial revenue losses and impacts profitability, making churn management critical for maintaining a stable customer base.

Reducing churn can have a considerable financial impact. Studies suggest that even a 5% reduction in churn can increase profits by 25–95%. Retaining existing customers is also more cost-effective than acquiring new ones, which is why businesses prioritize managing churn. By analyzing customer behavior and identifying churn patterns, companies can improve retention rates and reduce losses.

The reasons behind churn are varied and interconnected. Factors like poor customer service, high costs, subpar service quality, and attractive offers from competitors are common drivers. Customers experiencing frequent disruptions or negative support interactions are more likely to leave. Additionally, long-term customers who do not receive service upgrades are at risk of churning. Understanding these patterns helps businesses identify dissatisfaction and address it early.

Proactively managing churn involves leveraging customer data to predict risks and develop targeted retention strategies. Service usage, feedback, and interaction history can be analyzed to build predictive models that identify at-risk customers. These insights allow businesses to provide personalized offers, improve service quality, and strengthen customer support. Effective churn management not only minimizes revenue loss but also boosts customer satisfaction and fosters stronger relationships.

This project aims to identify the best predictive model for customer churn and determine the key factors influencing it. These insights will help businesses implement effective retention strategies and make informed decisions. By addressing churn proactively, companies can optimize resources, enhance loyalty, and achieve sustainable growth.

# 2 Variable Introduction and Definitions

The dataset used for the Telco customer churn prediction project comprises 7,043 observations and 21 variables, including a mix of 18 categorical and 3 continuous predictors. These variables provide detailed information on customer demographics, service usage, and financial aspects, which are critical for understanding churn behavior. The target variable, *Churn*, is binary, indicating whether a customer has churned (*Yes*) or not (*No*). Below is a summary of the variables included in the analysis:

| Variable Name | Description |
|---|---|
| customerID | Unique identifier for each customer |
| gender | Gender of the customer (*Male*, *Female*) |
| SeniorCitizen | Indicates if the customer is a senior citizen (binary) |
| Partner | Whether the customer has a partner (*Yes*, *No*) |

| | |
|---|---|
| Dependents | Whether the customer has dependents (*Yes*, *No*) |
| tenure | Number of months the customer has stayed with the company |
| PhoneService | Indicates if the customer has phone service (*Yes*, *No*) |
| MultipleLines | Indicates if the customer has multiple lines (*Yes*, *No*, *No phone service*) |
| InternetService | Type of internet service (*DSL*, *Fiber optic*, *No*) |
| OnlineSecurity | Whether the customer has online security (*Yes*, *No*) |
| OnlineBackup | Whether the customer has online backup (*Yes*, *No*) |
| DeviceProtection | Whether the customer has device protection (*Yes*, *No*) |
| TechSupport | Whether the customer has tech support (*Yes*, *No*) |
| StreamingTV | Whether the customer streams TV (*Yes*, *No*) |
| StreamingMovies | Whether the customer streams movies (*Yes*, *No*) Contract Type of contract (*Month-to-month*, *One year*, *Two year*) |
| PaperlessBilling | Whether billing is paperless (*Yes*, *No*) |
| PaymentMethod | Payment method (*Electronic check*, *Mailed check*, *Bank transfer*, *Credit card*) |
| MonthlyCharges | Monthly charges for the customer (*Continuous*) |
| TotalCharges | Total charges for the customer (*Continuous*) |
| Churn | Target variable indicating if the customer churned (*Yes*, *No*) |

These variables form the foundation for identifying patterns and factors that influence customer churn. To develop predictive models, the relationships between these predictors and the target variable will be analyzed. However, before modeling, the dataset must be prepared and preprocessed to ensure accuracy and reliability.

# 3 Preprocessing of Predictors

The preprocessing of predictors included organized steps to prepare the dataset for ac- curate predictive modeling. Each step was aimed at improving data quality, making it suitable for machine learning algorithms, and boosting model performance.

## Dataset Adjustments

To enhance the dataset and focus only on meaningful predictors, the `customerID` variable was removed. This variable, serving only as an identifier, held no predictive value. This left the dataset to have only 20 predictors including the target variable.

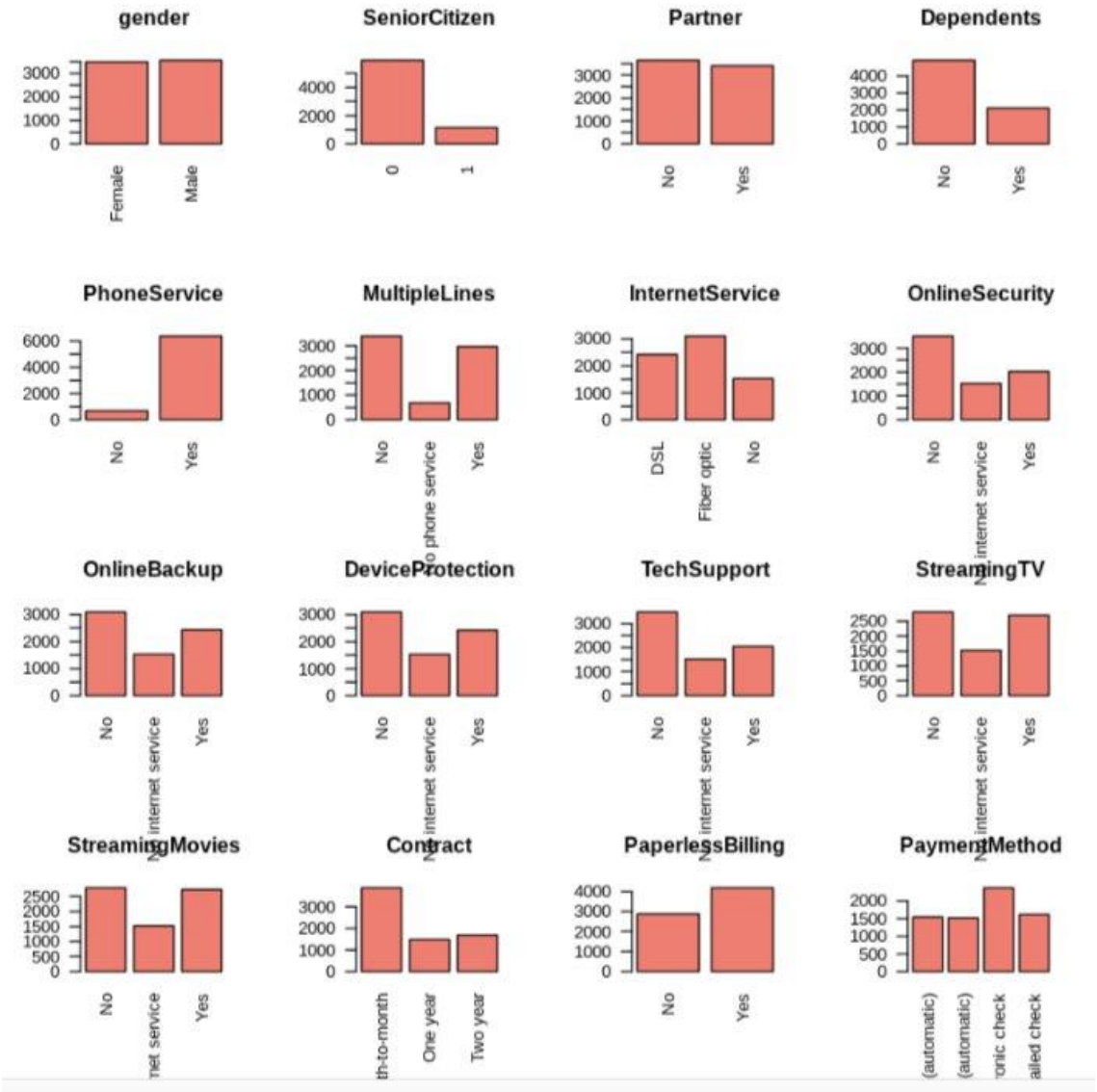
## Handling Missing Values

The dataset originally included 11 rows with missing values. These rows were removed to maintain data integrity and ensure compatibility with predictive models. This adjustment reduced the dataset to 7,032 observations, allowing for more reliable analysis and model development.

## Categorical Encoding

Categorical variables were encoded into numerical formats to facilitate their use in the models. Binary encoding was applied to variables with two levels. For example, the `gender` variable was encoded as `1` for *Female* and `0` for *Male*. Similarly, variables such as `Partner`, `Dependents`, `PhoneService`, and `PaperlessBilling` were encoded as `1` for *Yes* and `0` for *No*.

Multi-level categorical variables, including `MultipleLines`, `InternetService`, and `PaymentMethod`, were transformed into dummy variables. This process expanded the dataset to a total of 30 predictors. Ensuring that all categorical data could be effectively utilized by both linear and nonlinear models.




## Addressing Multicollinearity

To address multicollinearity, a correlation threshold of 75% was applied. Predictors with correlations above this threshold were systematically removed, reducing redundancy in the dataset and minimizing the risk of overfitting. This adjustment resulted in the elimination of 9 highly correlated predictors, leaving 21 predictors for models sensitive to multicollinearity, such as Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Regularized Discriminant Analysis (RDA), Mixture Discriminant Analysis (MDA), Neural Networks (NN), and Flexible Discriminant Analysis (FDA). How- ever, for models inherently robust to multicollinearity, such as Logistic Regression, Partial Least Squares Discriminant Analysis (PLSDA), Penalized Models, Support Vector Ma- chines (SVM), and K-Nearest

Neighbors (KNN), all 30 predictors were retained.

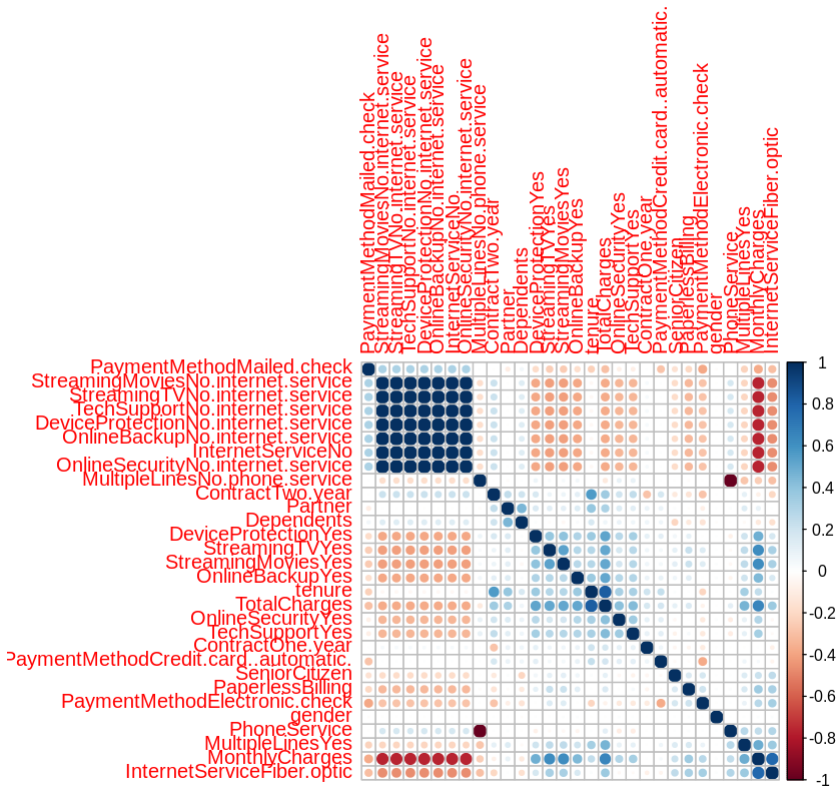



Figure 1: Correlation Matrix

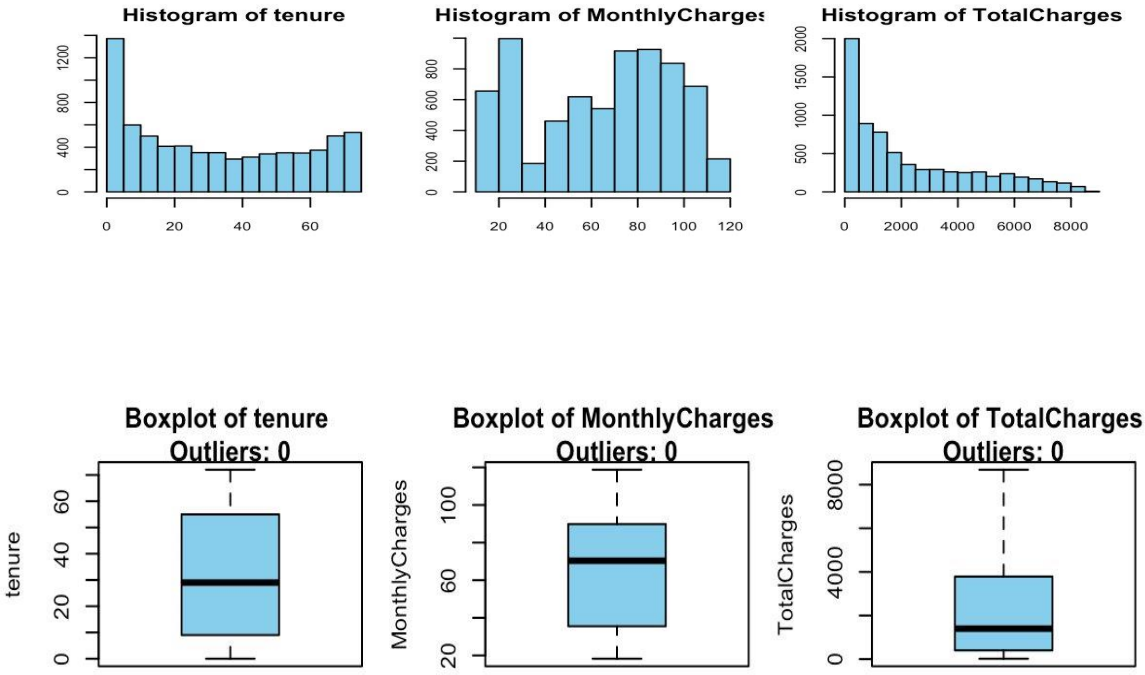
## Centering and Scaling

Centering and scaling were applied to standardize predictors, ensuring that all variables were on a comparable scale. This step was essential for models sensitive to the scale of input variables.

## Transformations of Continuous Predictors

Continuous predictors were examined using histograms and box plots, which revealed one predictor with significant skewness to the right. While no outliers were identified, a **Box-Cox transformation** was applied to normalize the distribution of the skewed predictor for the Naive Bayes (NB) model. This transformation improved the suitability of the data for modeling by achieving more normal-looking distributions.

These transformations, tailored to the requirements of specific models, ensured that continuous predictors contributed effectively to predictive accuracy. These preprocessing steps, ensured that the dataset was clean, standardized, and well-prepared for predictive modeling of customer churn.




# 4 Data Partitioning

## Target Variable Imbalance

The target variable, `Churn`, was identified as imbalanced, with 5,163 instances of 'No' (non-churn) and 1,869 instances of 'Yes' (churn). An imbalanced target variable can pose challenges during model training, as models may be biased towards the majority class. To address this issue, stratified sampling was used. A bar plot below visualizes this imbalance, providing insight into the distribution of the target variable.

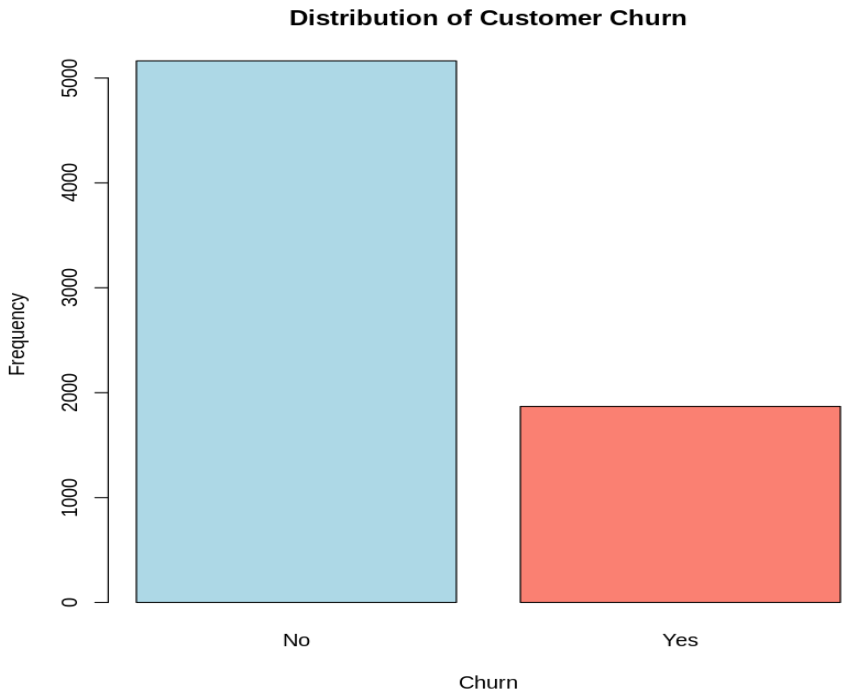



Figure 2: Target Variable Plot

## Training/Testing Split

The dataset was divided into training and testing sets using a stratified 80/20 split to maintain class balance within the `Churn` target variable. Stratification ensured that the proportions of churned and non-churned customers in both the training and testing sets were representative of the entire dataset. This approach minimized the risk of bias toward the majority class, which is essential for achieving fair and accurate model performance.

## Resampling

To ensure reliable model evaluation, 10-fold cross-validation was applied to the training set. During this process, the training data was randomly divided into 10 equally sized subsets, or folds. For each fold, the model was trained on 9 folds and tested on the remaining fold, iterating through all folds to calculate an average model performance metric. This technique provided a robust evaluation of the model's performance, reducing the risk of overfitting and offering a reliable estimate of accuracy, particularly for imbalanced datasets.

## Evaluation Metric

To address the imbalance in the target class, `Kappa` was chosen as the primary evaluation metric. Unlike accuracy, which can be misleading in imbalanced datasets, `Kappa` measures the agreement between predicted and actual classes while accounting for the chance agreement. This makes it a more reliable metric for evaluating model performance in scenarios with skewed class distributions. By using `Kappa`, we aimed to ensure that the models accurately predicted both churned and non-churned customers, providing a fair assessment of their effectiveness.

# 5 Model Fitting

## Linear Classification Models

The linear classification models employed for churn prediction include Logistic Regression, Linear Discriminant Analysis (LDA), Partial Least Squares Discriminant Analysis (PLSDA), and a Penalized Model. Each model was tuned with different parameters to optimize their performance. Among these models, Logistic Regression achieved the highest performance, with a training Kappa of 0.4627 and a testing Kappa of 0.5034. Linear Discriminant Analysis (LDA) did not show as strong performance, with a training Kappa of 0.4542 and a testing Kappa of 0.4803. Partial Least Squares Discriminant Analysis (PLSDA) with 25 components yielded a training Kappa of 0.4529 and a testing Kappa of 0.4783. The Penalized Model, tuned with an alpha of 0.1 and a lambda of 0.01, had a training Kappa of 0.4544 and a testing Kappa of 0.4903, indicating its improved predictive ability compared to the other models. However, Logistic Regression outperformed the other methods, making it the most effective model for predicting churn in this dataset.

| **Model** | **Best Tuning Parameter** | **Training Kappa** | **Testing Kappa** |
|---|---|---|---|
| Logistic Regression | N/A | 0.4627 | 0.5034 |
| Linear Discriminant Analysis (LDA) | N/A | 0.4542 | 0.4803 |
| Partial Least Squares Discriminant Analysis (PLSDA) | ncomp = 25 | 0.4529 | 0.4783 |
| Penalized Models (PM) | Alpha = 0.1, lambda = 0.01 | 0.4544 | 0.4903 |

Table 2: Performance Metrics for Linear Models

## Non-Linear Classification Models

For the non-linear models used in churn prediction, a variety of methods were evaluated with different tuning parameters. These included Quadratic Discriminant Analysis (QDA), Mixture Discriminant Analysis (MDA) with subclasses, Regularized Discriminant Analysis (RDA) with varying gamma and lambda values, Neural Networks (NN) with different sizes and decay rates, Flexible Discriminant Analysis (FDA), Support Vector Machines (SVM) with specific sigma and C parameters, K-Nearest Neighbors (KNN) with varying k values, and Naive Bayes (NB). Among these models, the Regularized Discriminant Analysis (RDA) with a gamma of 0.01 and a lambda of 0.6352632 achieved the highest performance on the training set, resulting in a training Kappa of 0.4836 and a testing Kappa of 0.4785. This highlights its effectiveness in managing the churn pre- diction task. Mixture Discriminant Analysis (MDA) with subclasses set to 2 also showed strong performance, achieving a testing Kappa of 0.4876. Neural Networks (NN) with a single hidden layer and a decay parameter of 0.1 provided a testing Kappa of 0.4856. Quadratic Discriminant Analysis (QDA), with no tuning parameters, had a testing Kappa of 0.465. However, the model with the highest testing Kappa was MDA, showcasing its ability to handle the imbalanced nature of the churn data effectively. Other models, such as Support Vector Machines (SVM) and K-Nearest Neighbors (KNN), had lower performance metrics, with testing Kappa values of 0.4575 and 0.4459 respectively, indicating their limitations in handling churn prediction in this dataset.

Table 3: Performance Metrics for Non-Linear Models

| **Model** | **Best Tuning Parameter** | **Training Kappa** | **Testing Kappa** |
|---|---|---|---|
| Quadratic Discriminant Analysis (QDA) | N/A | 0.4528 | 0.4650 |
| Mixture Discriminant Analysis (MDA) | subclasses = 2 | 0.4620 | 0.4876 |
| Regularized Discriminant Analysis (RDA) | gamma = 0.01, lambda = 0.635 | 0.4836 | 0.4785 |
| Neural Networks (NN) | size = 1, decay = 0.1 | 0.4612 | 0.4856 |
| Flexible Discriminant Analysis (FDA) | degree = 2, nprune = 21 | 0.4620 | 0.4710 |
| Support Vector Machines (SVM) | sigma = 0.017, C = 2 | 0.4147 | 0.4575 |
| K-Nearest Neighbors (KNN) | k = 49 | 0.4584 | 0.4459 |
| Naïve Bayes (NB) | N/A | 0.4401 | 0.4725 |

# 6 Summary

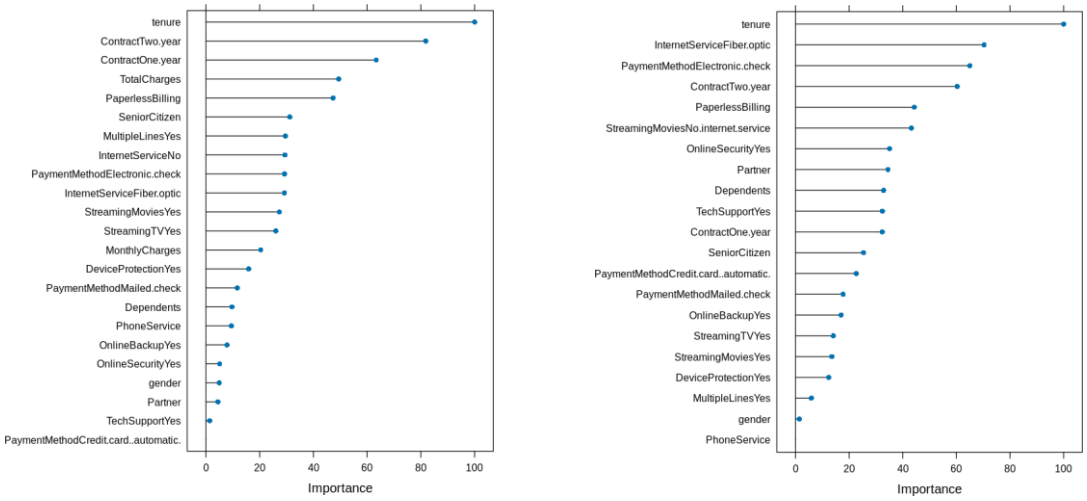
The best models for predicting churn are Logistic Regression and Mixture Discriminant Analysis (MDA). Logistic Regression achieved the highest performance with a testing Kappa of 0.5034, indicating its strong ability to differentiate between churn and non-churn customers based on the given predictors. This model's simplicity, interpretability, and effectiveness in handling binary outcomes make it an excellent choice for churn prediction. On the other hand, MDA with subclasses set to 2 also performed well, achieving a testing Kappa of 0.4876. MDA's performance reflects its capability to manage the complex relationships between predictors and churn, providing a more nuanced separation of classes. This model is particularly useful in handling imbalanced datasets, such as the one used here, by focusing on maximizing class separability. The combination of these models offers a balanced approach to churn prediction, leveraging both the straightforward predictive power of Logistic Regression and the nuanced class separation provided by MDA.

| **Best Models** | **Best Tuning Parameter** | **Training Kappa** | **Testing Kappa** |
|---|---|---|---|
| Logistic Regression | N/A | 0.4627 | 0.5034 |
| MDA | subclasses = 2 | 0.4620 | 0.4876 |

Table 4: Performance of Best Models

# 7 Important Predictors of Best Models

As we can infer from the plots below for the two best models, Logistic Regression and Mixture Discriminant Analysis (MDA), the predictor `tenure` consistently emerges as the top predictor for both models. This suggests that the length of time a customer has been with the service provider is a crucial indicator of churn. In both models, `tenure` significantly influences the likelihood of a customer leaving, highlighting its importance in churn prediction. The clear and consistent impact of `tenure` across these models indicates its strong predictive power, making it a key feature in identifying customers at risk of churn.




(a) Important Predictors for LR (b) Important Predictors for MDA

Figure 3: Important Predictors for Best Models

# 1 Appendix 1: Supplemental Material for Linear Models

**Logistic Regression:** The LR model was trained without pre-processing steps. The re- sampling technique used was 10-fold cross-validation. Accuracy was observed at 80.15%, and the Kappa statistic was 0.4674.

**Linear Discriminant Analysis (LDA):** Pre-processed with centering and scaling for all 21 predictors, removed highly correlated predictors, the optimal model achieved 79.90% accuracy and a Kappa value of 0.4616.

**Partial Least Squares (PLS):** Pre-processing included centering and scaling for all 30 predictors. Highly correlated predictors were not removed. The model was tuned across the 30 number of components ($n_{comp}$), with the optimal value selected based on the highest Kappa statistic. The final model used $n_{comp}$ = 25 components, which achieved a Kappa of 0.4531.

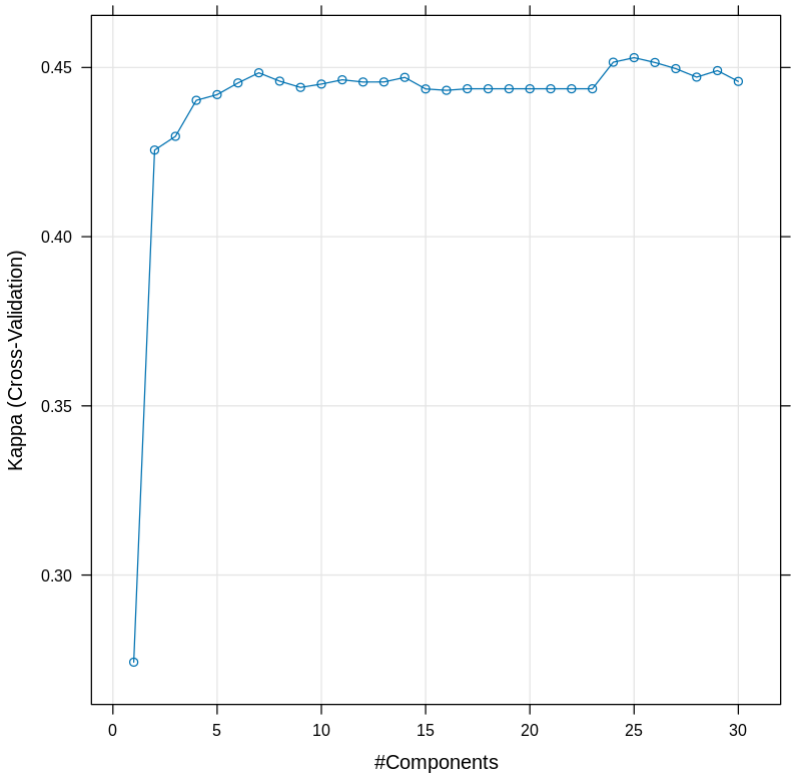



Figure 4: Partial Least Squares Tuning Parameters Plot

**glmnet (Elastic Net):** The Elastic Net model was pre-processed using centering and scaling. Multiple values of alpha and lambda were tested, with the optimal parameters being $\alpha$ = 0.1 and $\lambda$ = 0.01. This model yielded 80.56% accuracy and a Kappa statistic of 0.4674.

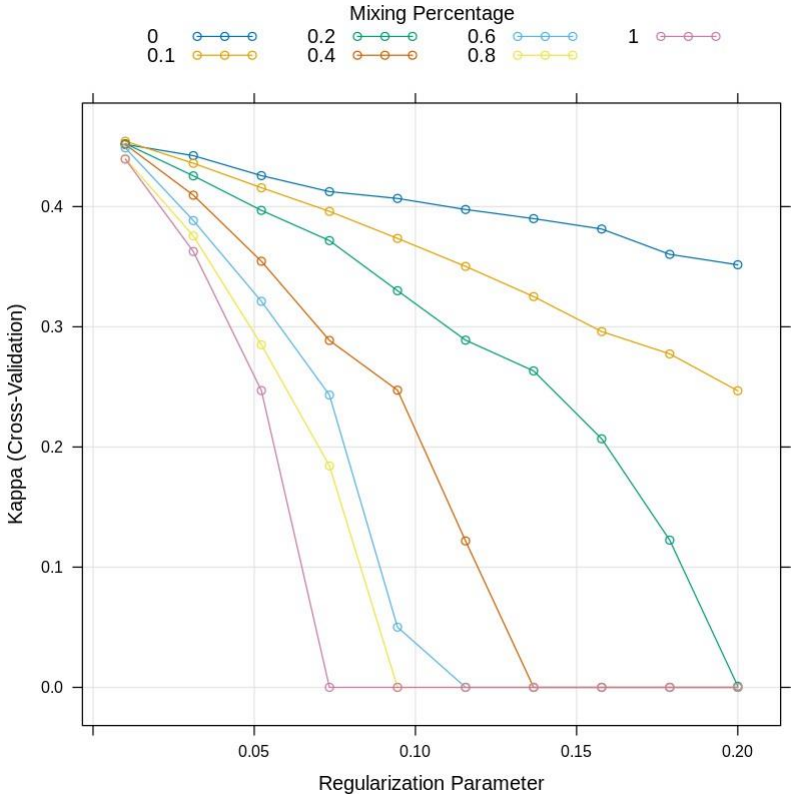



Figure 5: Glmnet Tuning Parameters Plot

# 2 Appendix 2: Supplemental Material for Non-Linear Models

**Quadratic Discriminant Analysis (QDA):** Pre-processed with centering and scal- ing, QDA achieved an accuracy of 76.61% with a Kappa of 0.471.

**Regularized Discriminant Analysis (RDA):** The RDA model utilized centered and scaled predictors, optimizing gamma and lambda parameters. The optimal values selected were $\gamma$ = 0.01 and $\lambda$ = 0.6352, resulting in an accuracy of 79.24% and a Kappa statistic of 0.4902.

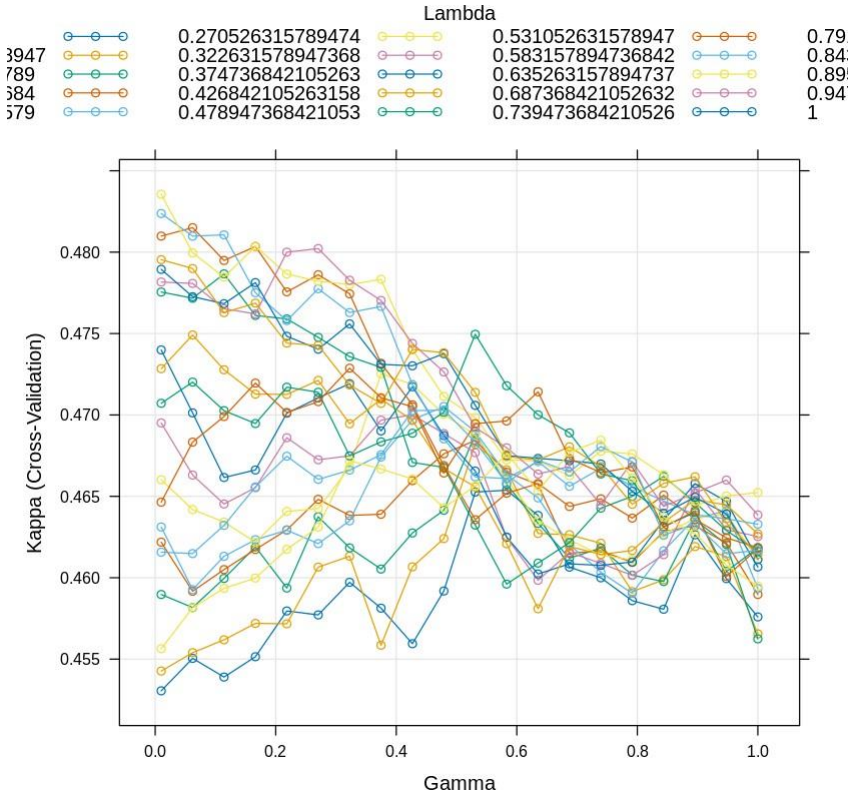



Figure 6: Regularized Discriminant Analysis Tuning Parameters plot

**Mixture Discriminant Analysis (MDA):** Pre-processing steps included centering and scaling for all 21 predictors. The model was tuned across the number of subclasses (*subclasses*) ranging from 1 to 30, with the optimal value selected based on the highest Kappa statistic. The final model used *subclasses* = 2, achieving an accuracy of 78.73% and a Kappa value of 0.4738.

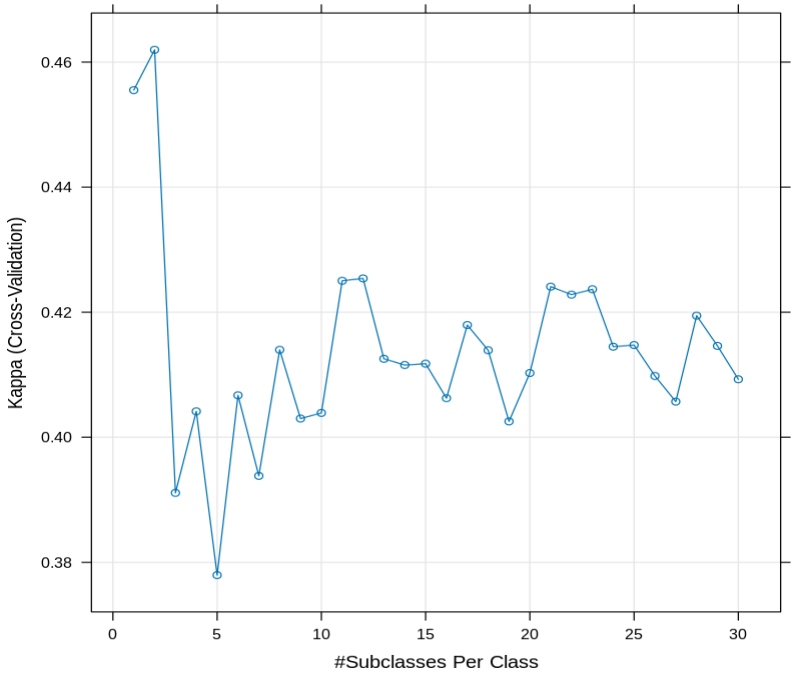



Figure 7: Mixture Discriminant Analysis Tuning Parameters Plot

**Neural Network:** The Neural Network model underwent pre-processing steps that included centering, scaling, and spatial sign transformation for all 21 predictors. The model was tuned across combinations of *size* (number of hidden units) and *decay* (regularization parameter), with the optimal values selected based on the highest Kappa statistic. The final model used *size* = 1 and *decay* = 0.1, achieving an accuracy of 80.29% and a Kappa value of 0.468.

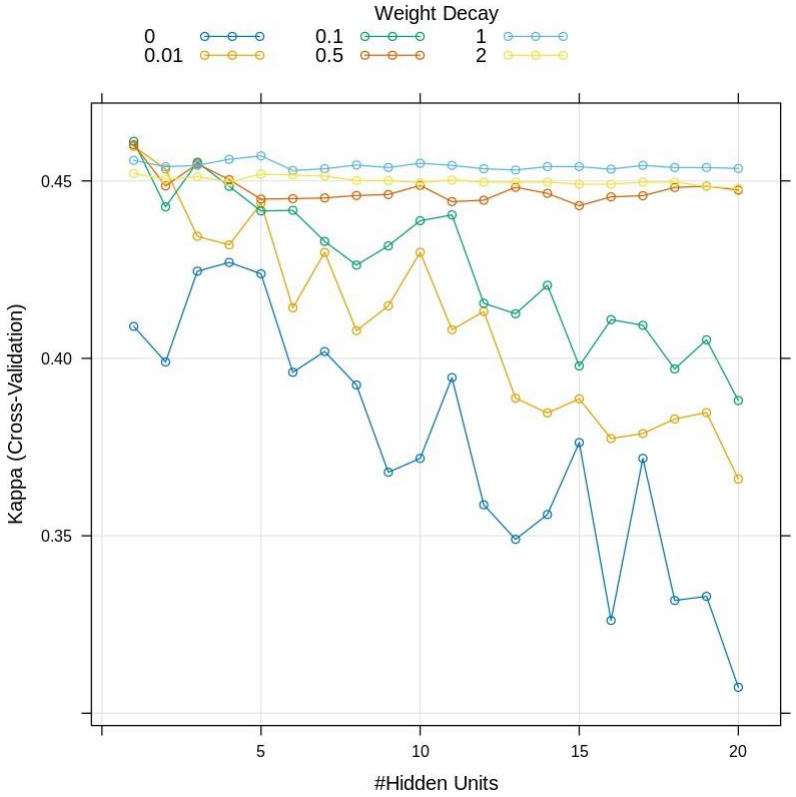



Figure 8: Neural Network Tuning Parameters Plot

**Flexible Discriminant Analysis (FDA):** The Flexible Discriminant Analysis model underwent pre-processing steps including centering and scaling for all 21 predictors. The model was tuned over combinations of *degree* (polynomial degree) and *nprune* (number of terms in the model), with the optimal values selected based on the highest Kappa statistic. The final model used *degree* = 2 and *nprune* = 21, achieving an accuracy of 80.49% and a Kappa value of 0.4713.

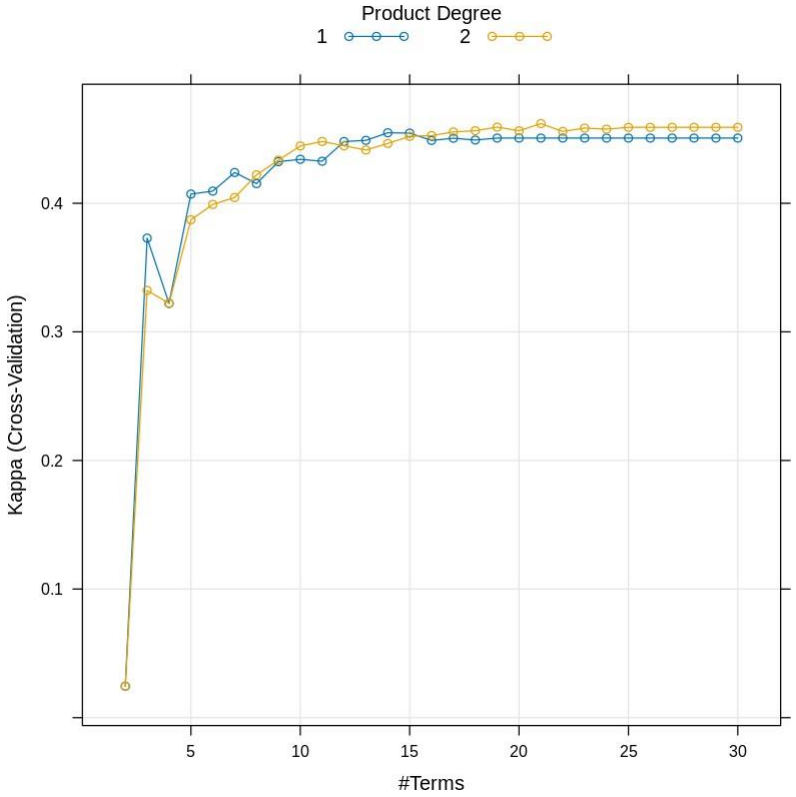



Figure 9: Flexible Discriminant Analysis Tuning Parameters Plot

**Support Vector Machines with Radial Basis Function Kernel (SVM-RBF):** The Support Vector Machines model with a Radial Basis Function (RBF) kernel under- went pre-processing steps including centering and scaling for all 30 predictors. The model was tuned across values of the regularization parameter (*C* ), while the *sigma* parameter was held constant at a value of 0.01763468. The optimal model was selected based on the highest Kappa statistic, with final values of *sigma* = 0.01763468 and *C* = 2. This model achieved an accuracy of 80.98% and a Kappa value of 0.4584.

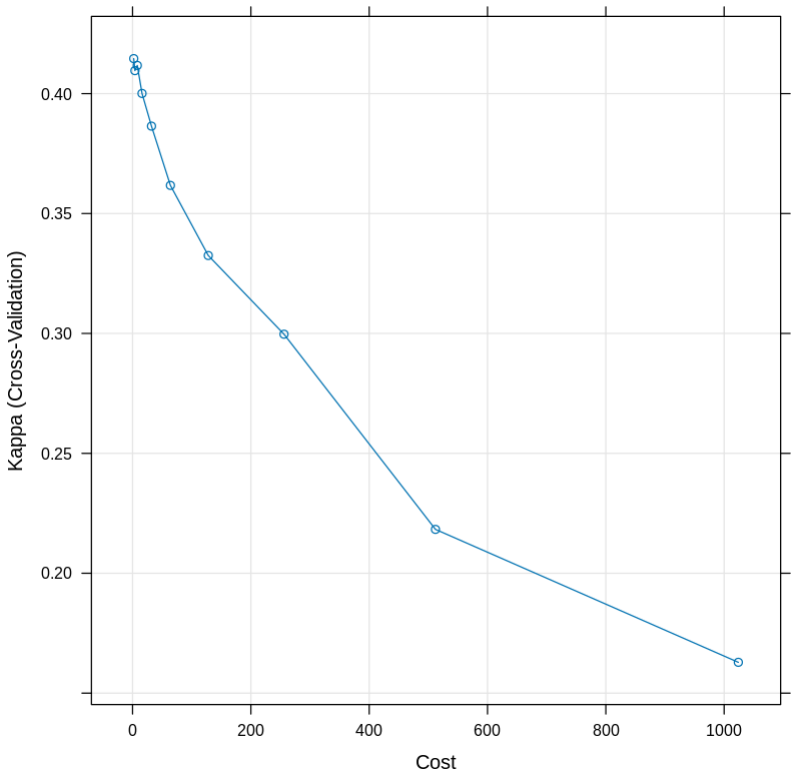



Figure 10: Support Vector Machines Tuning Paratmeters Plot

**k-Nearest Neighbors (k-NN):** The k-Nearest Neighbors (k-NN) model applied pre-processing steps, including centering and scaling for all 30 predictors. The model was tuned across different values of $k$, with the optimal value of $k = 49$ selected based on the highest Kappa statistic. The final model achieved an accuracy of 79.90% and a Kappa value of 0.4774.

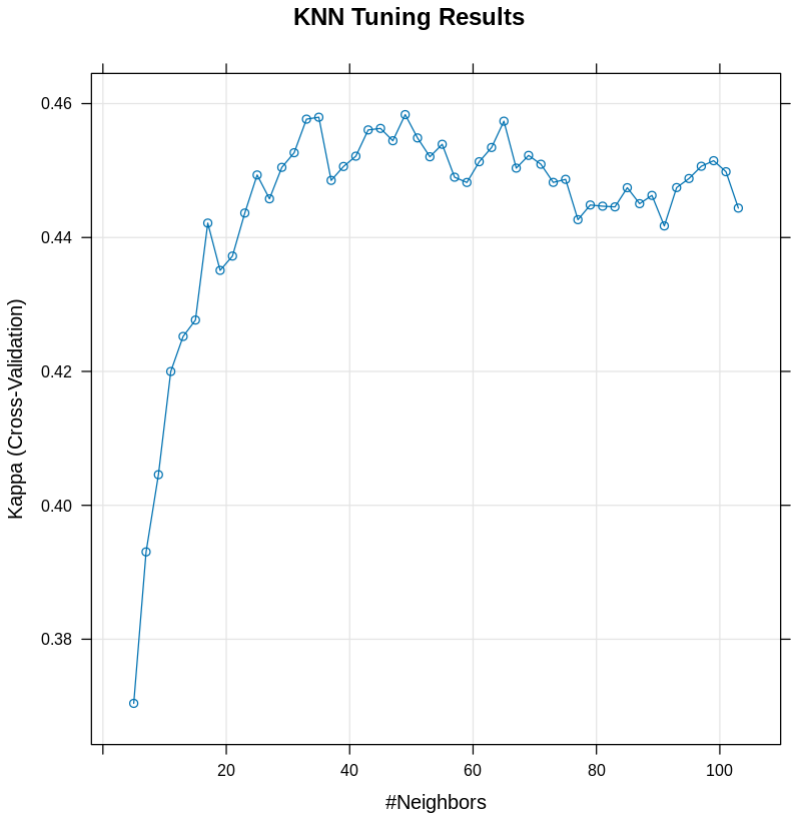



Figure 11: K-Nearest Neighbours Tuning Parameters Plot

**Naive Bayes:** The Naive Bayes model included a Box-Cox transformation as a pre- processing step for one predictor. The model was evaluated using 10-fold cross-validation, and tuning parameters were held constant with the following values: $fL = 1$, `usekernel = TRUE`, and `adjust = 1`. The final model achieved an accuracy of 78.91% and a Kappa value of 0.4694.

# Bibliography

[1] SIU, “Modelling time statistics for customer churn prediction,” in *28th Signal Pro- cessing and Communications Applications Conference*, 2020.

[2] IEEE, “An intelligent customer churn prediction and response framework,” in *14th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 2019.

[3] “Customer - churn prediction using machine learning,” in *ICICT Conference*, 2020.

[4] IEEE, “Customer churn reasoning in telecommunication domain,” in *2020 IEEE Conference*, 2020.

listings xcolor appendix

# Appendix A

# R Code for Telco Churn Analysis

The following R code was used for preprocessing, analysis, and modeling in the Telco Churn Analysis project.

## Full R Code

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

#load and explore dataset
telco_data <- read.csv("Customer-Churn.csv")
str(telco_data)
summary(telco_data)

#remove customer ID because it has no useful information
churn_data <-telco_data[,-1]
str(churn_data)

numeric_predictors <- churn_data[sapply(churn_data, is.numeric)]
num_cols <- 3
num_rows <- ceiling(length(colnames(numeric_predictors)) / num_cols)
par(mfrow = c(num_rows, num_cols))

#Plot histograms for each numeric predictor
for (predictor in colnames(numeric_predictors)) {
  hist(
    numeric_predictors[[predictor]],
    main = paste("Histogram of", predictor),
    xlab = predictor,
    col = "skyblue",
    border = "black",
    breaks = 20
  )
}

par(mfrow = c(1, 1))



# missing values
#colSums(is.na(churn_data))
# checking the column with missing value
missing_total_charges <- sum(is.na(churn_data$TotalCharges))
missing_total_charges

#Since we have only 11 missing values we remove the 11 rows.
data <- na.omit(churn_data)

#checking the missing values after removing the rows with missing values
missing_total_charges <- sum(is.na(data))
```

```
missing_total_charges

#checking the dimension of the data after removing the rows with null values
dim(data)
str(data)

#check response variable balance
barplot(table(data$Churn),
      main = "Distribution of Customer Churn",
      xlab = "Churn",
      ylab = "Frequency",
      col = c("lightblue", "salmon"),
      border = "black")
table(data$Churn)

#Separate predictors and target variable
predictors <- data[, -which(names(data) == "Churn")]
target <- data$Churn

sapply(predictors, function(x) if (is.character(x) | is.factor(x)) unique(x))


# Binary encoding for categorical variables with two levels
predictors$gender <- ifelse(predictors$gender == "Female", 1, 0)
predictors$Partner <- ifelse(predictors$Partner == "Yes", 1, 0)
predictors$Dependents <- ifelse(predictors$Dependents == "Yes", 1, 0)
predictors$PhoneService <- ifelse(predictors$PhoneService == "Yes", 1, 0)
predictors$PaperlessBilling <- ifelse(predictors$PaperlessBilling == "Yes", 1, 0)

# List of multi-level categorical variables
multi_level_vars <- c("MultipleLines", "InternetService", "OnlineSecurity",
                "OnlineBackup", "DeviceProtection", "TechSupport",
                "StreamingTV", "StreamingMovies", "Contract", "PaymentMethod")

# Select only multi-level variables from predictors
multi_level_data <- predictors[, multi_level_vars]

# Create dummy variables for the multi-level variables
dummy_model <- dummyVars(~ ., data = multi_level_data, fullRank = TRUE)
multi_level_encoded <- data.frame(predict(dummy_model, newdata = multi_level_data))
multi_level_encoded

# Remove the original multi-level columns from predictors
predictors <- predictors[, !names(predictors) %in% multi_level_vars]


# Add the dummy-encoded multi-level variables back to predictors
predictors <- cbind(predictors, multi_level_encoded)
str(predictors)

library(caret)
# Analyze near-zero variance
nzv <- nearZeroVar(predictors, saveMetrics = TRUE)
nzv_count <- sum(nzv$nzv)
nzv_count

install.packages("corrplot")
library(corrplot)

# Extract numeric predictors
num_predictors <- predictors[sapply(predictors, is.numeric)]
correlation <- cor(num_predictors, use = "complete.obs")
```

```
tooHigh <- findCorrelation(correlation, cutoff = 0.75)
num_high_corr <- length(tooHigh)
cat("Number of Highly Correlated Predictors:", num_high_corr, "\n")

# Plot the correlation matrix
corrplot(correlation, method = "circle", order = "hclust")

# Set a seed for reproducibility
set.seed(123)

# Create an 80-20 split
train_index <- createDataPartition(target, p = 0.8, list = FALSE)

# Split the data
train_data <- predictors[train_index, ]
test_data <- predictors[-train_index, ]
train_target <- target[train_index]
test_target <- target[-train_index]

# Verify dimensions
cat("Training Set Dimensions (Predictors):", dim(train_data), "\n")
cat("Test Set Dimensions (Predictors):", dim(test_data), "\n")
cat("Training Set Dimensions (Target):", length(train_target), "\n")
cat("Test Set Dimensions (Target):", length(test_target), "\n")

# Remove highly correlated predictors from training and test sets
train_data_reduced <- train_data[, -tooHigh]
test_data_reduced <- test_data[, -tooHigh]

# Verify dimensions after removal
cat("Training Set Dimensions (After Removal):", dim(train_data_reduced), "\n")
cat("Test Set Dimensions (After Removal):", dim(test_data_reduced), "\n")


train_target <- as.factor(train_target)
test_target <- as.factor(test_target)

# Check levels of train_predictions and train_target
print("Levels of test_target:")
print(levels(test_target))

print("Levels of train_target:")
print(levels(train_target))


## Logistic Regression

library(caret)

# Set a seed for reproducibility
set.seed(123)

# Resampling using 10-fold cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Training the model
logistic_model <- train(
```

```
  x = train_data,
  y = train_target,
  method = "glm",
  metric = "Kappa",
  family = binomial,
  trControl = ctrl
)

# Printing and plotting the results
print(logistic_model)

# Predicting on the training set
train_predictions <- predict(logistic_model, newdata = train_data)

# Creating Confusion Matrix for training predictions
train_conf_matrix <- confusionMatrix(
  data = train_predictions,
  reference = train_target,
  positive = "Yes" )
print(train_conf_matrix)

# Predicting on the test set
test_predictions <- predict(logistic_model, newdata = test_data)

# Creating Confusion Matrix for test predictions
test_conf_matrix <- confusionMatrix(
  data = test_predictions,
  reference = test_target,
  positive = "Yes" )
print(test_conf_matrix)

## Linear Discriminant Analysis

library(caret)

# Resampling using 10-fold cross-validation
lda_ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Set seed for reproducibility
set.seed(123)

# Train the LDA Model
lda_model <- train(
  x = train_data_reduced,
  y = train_target,
  method = "lda",
  metric = "Kappa",
  preProcess = c("center", "scale"),
  trControl = lda_ctrl
)

# Print model details
print(lda_model)

# Predict on the training set
lda_train_predictions <- predict(lda_model, newdata = train_data_reduced)
```

```
# Confusion Matrix for the Training Set
lda_train_conf_matrix <- confusionMatrix(
  data = lda_train_predictions,
  reference = train_target,
  positive = "Yes"  )
print(lda_train_conf_matrix)


# Predict on the testing set
lda_test_predictions <- predict(lda_model, newdata = test_data_reduced)

# Confusion Matrix for the Testing Set
lda_test_conf_matrix <- confusionMatrix(
  data = lda_test_predictions,
  reference = test_target,
  positive = "Yes")
print(lda_test_conf_matrix)


## Partial Least Squar (PLS)

install.packages("pls")
library(pls)

# Resampling using 10-fold cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Train the PLSDA model
set.seed(123)
plsda_model <- train(
  x = train_data,
  y = train_target,
  method = "pls",
  tuneGrid = expand.grid(.ncomp = 1:30),
  preProcess = c("center", "scale"),
  metric = "Kappa",
  trControl = ctrl
)

# Print model summary
print(plsda_model)

# Plot model performance
plot(plsda_model)

# Predict on the training set
plsda_train_predictions <- predict(plsda_model, newdata = train_data)

# Confusion Matrix for the Training Set
plsda_train_conf_matrix <- confusionMatrix(
  data = plsda_train_predictions,
  reference = train_target,
  positive = "Yes" )
print(plsda_train_conf_matrix)
```

```
# Predict on the testing set
plsda_test_predictions <- predict(plsda_model, newdata = test_data)

# Confusion Matrix for the Testing Set
plsda_test_conf_matrix <- confusionMatrix(
  data = plsda_test_predictions,
  reference = test_target,
  positive = "Yes")
print(plsda_test_conf_matrix)


## Penalized Model

install.packages("glmnet")
library(glmnet)

# Resampling using 10-fold cross-validation
ctrl <- trainControl(
  method = "cv",
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Define grid for hyperparameters
glmnet_grid <- expand.grid(
  .alpha = c(0, 0.1, 0.2, 0.4, 0.6, 0.8, 1),
  .lambda = seq(0.01, 0.2, length = 10)
)

# Set seed for reproducibility
set.seed(123)

# Train the glmnet model
glmnet_model <- train(
  x = train_data,
  y = train_target,
  method = "glmnet",
  tuneGrid = glmnet_grid,
  preProcess = c("center", "scale"),
  metric = "Kappa",
  trControl = ctrl
)

# Print the trained model
print(glmnet_model)

# Plot
plot(glmnet_model)

# Predict on the training set
train_predictions <- predict(glmnet_model, newdata = train_data)

# Confusion Matrix for the Training Set
train_conf_matrix <- confusionMatrix(
  data = train_predictions,
  reference = train_target,
  positive = "Yes"  )
print(train_conf_matrix)


# Predict on the testing set
test_predictions <- predict(glmnet_model, newdata = test_data)
```

```
# Confusion Matrix for the Test Set
test_conf_matrix <- confusionMatrix(
  data = test_predictions,
  reference = test_target,
  positive = "Yes"  )
print(test_conf_matrix)


## Quadratic Discriminant Analysis (QDA)

# Resampling using 10-fold cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Set seed for reproducibility
set.seed(476)

# Train the QDA model
qdaFit <- train(
  x = train_data_reduced,
  y = train_target,
  method = "qda",
  metric = "Kappa",
  preProcess = c("center", "scale"),
  trControl = ctrl
)

# Print model results
print(qdaFit)

# Predict on the training set
qda_train_pred <- predict(qdaFit, newdata = train_data_reduced)

# Confusion Matrix for the Training Set
qda_train_conf <- confusionMatrix(
  data = qda_train_pred,
  reference = train_target,
  positive = "Yes")
print(qda_train_conf)


# Predict on the testing set
qda_test_pred <- predict(qdaFit, newdata = test_data_reduced)

# Confusion Matrix for the Test Set
qda_test_conf <- confusionMatrix(
  data = qda_test_pred,
  reference = test_target,
  positive = "Yes")
print(qda_test_conf)


## Regularized Discriminant Analysis (RDA)

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

```
# Resampling using 10-fold cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Set seed for reproducibility
set.seed(476)

# Train the RDA model
rdaFit <- train(
  x = train_data_reduced,
  y = train_target,
  method = "rda",
  metric = "Kappa",
  tuneGrid = expand.grid(
    .gamma = seq(0.01, 1, length = 20),
    .lambda = seq(0.01, 1, length = 20)
  ),
  preProcess = c("center", "scale"),
  trControl = ctrl
)

# Print model results
print(rdaFit)

# Plot model performance
plot(rdaFit)

# Predict on the training set
rda_train_pred <- predict(rdaFit, newdata = train_data_reduced)

#Confusion Matrix for the Training Set
rda_train_conf <- confusionMatrix(
  data = rda_train_pred,
  reference = train_target,
  positive = "Yes"
)
print(rda_train_conf)



#Predict on the testing set
rda_test_pred <- predict(rdaFit, newdata = test_data_reduced)

#Confusion Matrix for the Test Set
rda_test_conf <- confusionMatrix(
  data = rda_test_pred,
  reference = test_target,
  positive = "Yes")
print(rda_test_conf)


### Mixture Discriminant Analysis

install.packages("mda")
library(mda)

# Resampling using 10-fold cross-validation
ctrl <- trainControl(
```

```
  method = "cv",
  number = 10,
  classProbs = TRUE,
  summaryFunction = defaultSummary
)

# Set seed for reproducibility
set.seed(476)

# Train the MDA model
mdaFit <- train(
  x = train_data_reduced,
  y = train_target,
  method = "mda",
  metric = "Kappa",
  tuneGrid = expand.grid(.subclasses = 1:30),
  preProcess = c("center", "scale"),
  trControl = ctrl
)

# Print model results
print(mdaFit)

# Plot
plot(mdaFit)

# Predict on the training set
mda_train_pred <- predict(mdaFit, newdata = train_data_reduced)

# Confusion Matrix for the Training Set
mda_train_conf <- confusionMatrix(
  data = mda_train_pred,
  reference = train_target,
  positive = "Yes" )
print(mda_train_conf)


# Predict on the testing set
mda_test_pred <- predict(mdaFit, newdata = test_data_reduced)

# Confusion Matrix for the Test Set
mda_test_conf <- confusionMatrix(
  data = mda_test_pred,
  reference = test_target,
  positive = "Yes")
print(mda_test_conf)


## Neural Networks

install.packages("nnet")
library(nnet)

#grid for tuning parameters
nnetGrid <- expand.grid(
  .size = 1:25,
  .decay = c(0, 0.01, 0.1, 0.5, 1, 2)
)

#maximum weights for neural network
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (ncol(train_data_reduced) + 1) + (maxSize + 1) * length(unique(train_target)))
```

```
#cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE,
  savePredictions = "final"
)

# Set seed for reproducibility
set.seed(123)

# Train the neural network model
nnetFit <- train(
  x = train_data_reduced,
  y = train_target,
  method = "nnet",
  metric = "Kappa",
  preProcess = c("center", "scale")
  tuneGrid = nnetGrid,
  trace = FALSE,
  maxit = 2000,
  trControl = ctrl
)

# Print and plot model results
print(nnetFit)
plot(nnetFit)

# Predict on the training set
train_predictions <- predict(nnetFit, newdata = train_data_reduced)
train_probabilities <- predict(nnetFit, newdata = train_data_reduced, type = "prob")

# Confusion Matrix for the Training Set
train_conf_matrix <- confusionMatrix(
  data = train_predictions,
  reference = train_target,
  positive = "Yes")
print(train_conf_matrix)



# Predict on the testing set
test_predictions <- predict(nnetFit, newdata = test_data_reduced)
test_probabilities <- predict(nnetFit, newdata = test_data_reduced, type = "prob")

# Confusion Matrix for the Testing Set
test_conf_matrix <- confusionMatrix(
  data = test_predictions,
  reference = test_target,
  positive = "Yes")
print(test_conf_matrix)

## Flexible Discriminant Analysis

#install the necessary packages
install.packages("MASS")
install.packages("mda")
install.packages("earth")
install.packages("themis")
```

```
#load the library
library(MASS)
library(mda)
library(earth)
library(themis)

#grid for hyperparameter tuning
marsGrid <- expand.grid(
  .degree = 1:2,
  .nprune = seq(2, 30)
)

# Set up cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Set seed for reproducibility
set.seed(123)

# Train the FDA model
fdaTuned <- train(
  x = train_data_reduced,
  y = train_target,
  method = "fda",
  tuneGrid = marsGrid,
  metric = "Kappa",
  preProcess = c("center", "scale"),
  trControl = ctrl
)

# Print and plot model results
print(fdaTuned)
plot(fdaTuned)

# Predict on the training set
fda_train_pred <- predict(fdaTuned, newdata = train_data_reduced)
fda_train_prob <- predict(fdaTuned, newdata = train_data_reduced, type = "prob")

# Confusion Matrix for the Training Set
fda_train_conf <- confusionMatrix(
  data = fda_train_pred,
  reference = train_target,
  positive = "Yes")
print(fda_train_conf)

# Predict on the testing set
fda_test_pred <- predict(fdaTuned, newdata = test_data_reduced)
fda_test_prob <- predict(fdaTuned, newdata = test_data_reduced, type = "prob")

# Confusion Matrix for the Testing Set
fda_test_conf <- confusionMatrix(
  data = fda_test_pred,
  reference = test_target,
  positive = "Yes" )
print(fda_test_conf)


## Support Vector Machines
```

```
# Install and load necessary package
install.packages("kernlab")
library(kernlab)

#sigma
sigmaValue <- sigest(as.matrix(train_data), frac = 0.9)[2]
#range of values for the Cost parameter
CValues <- 2^(seq(1, 10, by = 1))

# Create the tuning grid
svmRGrid <- expand.grid(.sigma = sigmaValue, .C = CValues)
# Set up 10-fold cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE,
  savePredictions = "final"
)

# Set seed for reproducibility
set.seed(123)

# Train the SVM model
svmRModel <- train(
  x = train_data,
  y = train_target,
  method = "svmRadial",
  metric = "Kappa",
  preProcess = c("center", "scale"),
  tuneGrid = svmRGrid,
  trControl = ctrl
)

# Print and plot model results
print(svmRModel)
plot(svmRModel)

# Predict on the training set
train_pred <- predict(svmRModel, newdata = train_data)
train_prob <- predict(svmRModel, newdata = train_data, type = "prob")

# Confusion Matrix for the Training Set
train_conf <- confusionMatrix(
  data = train_pred,
  reference = train_target,
  positive = "Yes")
print(train_conf)

# Predict on the testing set
test_pred <- predict(svmRModel, newdata = test_data)
test_prob <- predict(svmRModel, newdata = test_data, type = "prob")

# Confusion Matrix for the Testing Set
test_conf <- confusionMatrix(
  data = test_pred,
  reference = test_target,
  positive = "Yes")
print(test_conf)
```

```
## K-Nearest Neighbors

library(caret)

# Set up 10-fold cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE
)

# Set seed for reproducibility
set.seed(123)

# Train the KNN model
knnFit <- train(
  x = train_data,
  y = train_target,
  method = "knn",
  metric = "Kappa",
  preProcess = c("center", "scale"),
  tuneLength = 50,
  trControl = ctrl
)

# Print and plot model results
print(knnFit)
plot(knnFit, main = "KNN Tuning Results")

# Predict on the training set
train_pred <- predict(knnFit, newdata = train_data)
train_prob <- predict(knnFit, newdata = train_data, type = "prob")

# Confusion Matrix for the Training Set
train_conf <- confusionMatrix(
  data = train_pred,
  reference = train_target,
  positive = "Yes")
print(train_conf)


# Predict on the testing set
test_pred <- predict(knnFit, newdata = test_data)
test_prob <- predict(knnFit, newdata = test_data, type = "prob")

#Confusion Matrix for the Testing Set
test_conf <- confusionMatrix(
  data = test_pred,
  reference = test_target,
  positive = "Yes")
print(test_conf)

## Naive Bayes

#install necessary packages
install.packages("klaR")

#Load the library
library(klaR)

# Define the tuning grid for Naive Bayes
```

```
nbGrid_simple <- expand.grid(
  fL = c(1),
  usekernel = c(TRUE),
  adjust = c(1)
)

#Uisng 10 fold cross-validation
ctrl <- trainControl(
  method = "cv",
  number = 10,
  summaryFunction = defaultSummary,
  classProbs = TRUE,
  savePredictions = "final"
)

# Train the Naive Bayes model
set.seed(123)
nbFit <- train(
  x = train_data_reduced,
  y = train_target,
  method = "nb",
  metric = "Kappa",
  preProcess = c("BoxCox"),
  tuneGrid = nbGrid_simple,
  trControl = ctrl
)

# Print and visualize results
print(nbFit)
#plot(nbFit)

# Predict on the training set
train_pred <- predict(nbFit, newdata = train_data_reduced)
train_prob <- predict(nbFit, newdata = train_data_reduced, type = "prob")

# Confusion Matrix for the Training Set
train_conf <- confusionMatrix(
  data = train_pred,
  reference = train_target,
  positive = "Yes")
print(train_conf)


# Predict on the testing set
test_pred <- predict(nbFit, newdata = test_data_reduced)
test_prob <- predict(nbFit, newdata = test_data_reduced, type = "prob")

# Confusion Matrix for the Testing Set
test_conf <- confusionMatrix(
  data = test_pred,
  reference = test_target,
  positive = "Yes" )
print(test_conf)

## Important Predictors

#Install Necessary Libraraies
install.packages("caret")
library(caret)

#Install necessary Libraraies
install.packages("vip")
```

```
library(vip)

#Logistic Regression Predcitor Importance
variable_importance=varImp(logistic_model)
variable_importance
plot(variable_importance)

#MDA Predictor Importance
importance <- varImp(mdaFit)
importance
plot(importance)
```