**Dataset Description:**

The given dataset is a modified version of the “Loan Approval Classification Dataset” available in [Kaggle](https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data). The dataset contains information about individuals seeking loan approval. This is a supervised dataset with the target variable being “loan\_status” containing 2 categorical values: 0 (Rejected) and 1 (Accepted).

Apart from the target variable, this dataset contains 13 attributes (excluding the target) with a mixture of numeric and categorical types. The total dataset contains 201 instances. A brief overview of the dataset attribute types (with the total class values for categorical) is given below:

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Description** | **Type** |
| person\_age | Age of the person | Numeric (integer) |
| person\_gender | Gender of the person | Categorical  (male, female) |
| person\_eduation | Highest education level of the person | Categorical  (Associate, Bachelor, Doctorate, High School, Master) |
| person\_income | Annual income | Numeric (Continuous) |
| person\_emp\_exp | Years of employment experience | Numeric (integer) |
| person\_home\_ownership | Home ownership status | Categorical  (MORTGAGE, OWN, OTHER, RENT) |
| loan\_amnt | Loan amount request | Numeric (Continuous) |
| loan\_intent | Purpose of the loan | Categorical  (PERSONAL, EDUCATION, MEDICAL, VENTURE, DEBTCONSOLIDATION) |
| loan\_int\_rate | Loan interest rate | Numeric (Continuous) |
| loan\_percent\_income | Loan amount as a percentage of annual income | Numeric (Continuous) |
| cb\_person\_cred\_hist\_length | Length of credit history in years | Numeric (integer) |
| credit\_score | Credit score of the person | Numeric (Continuous) |
| previous\_loan\_defaults\_on\_file | Indicator of previous loan defaults | Categorical  (YES, NO) |
| loan\_status | Loan approval status | Categorical (accepted, rejected) |

**The primary questions explored in this project were:**

* What are the key demographic and financial factors that influence whether a loan gets approved or rejected?
* How do categorical attributes (such as education level, gender, and home ownership) impact loan approval?
* What relationships or correlations exist among the numeric variables (income, loan amount, credit score, etc.)?
* How can missing values, outliers, and class imbalance be effectively handled to ensure the dataset is suitable for modeling?
* After preprocessing, does the dataset show a more balanced, reliable structure for future predictive analysis?

|  |
| --- |
| 1. **Load all the libraries needed:** |
| **Code:**  install.packages("dplyr")  install.packages('openxlsx')  install.packages("stringdist")  install.packages("corrplot")  library(dplyr)  library(openxlsx)  library(stringdist)  library(corrplot) |
| **Description:**  **dplyr**: To manipulate the column & row contents of dataframes.  **openxlsx**: Open, Read & Write to an Excel file.  **stringdist**: Matching strings with predefined valid values.  **Corrplot**: used to visualize correlation matrices in a graphical and easy-to-interpret way**.** |

|  |
| --- |
| 1. **Load the data:** |
| **Code:**  data <- read.xlsx("Midterm\_Dataset\_Section(C).xlsx") |
| **Description:**  By using the ‘openxlsx’ library, the Excel file contents were converted to an R data frame. |

|  |
| --- |
| 1. **Check the data summary:** |
| **Code:**  str(data)  summary(data) |
| **Description:**  **str(data)** shows a small overview of the columns in ‘data’. And **summary(data)** shows a short summary of each column (minimum and maximum values, mean, median, 1st quartile, and 3rd quartile values for numeric attributes, and the instance count for categorical attributes and the number of missing values for all attributes). |
| **Screenshot:** |

|  |
| --- |
| 1. **Check and Remove Duplicate Rows:** |
| **Code:**  nrow(data)  nrow(distinct(data))  distinct\_data <- distinct(data) |
| **Description:**  **nrow()** returns the number of instances of the dataframe.  **distinct()** returns another dataset with only the unique instances.  Finally, the dataset returned using **distinct()** has been saved to a new dataframe named ‘distinct\_data’. |
| **Screenshot:** |

|  |  |
| --- | --- |
| 1. **Annotating Target Attribute:** | |
| **Code:**  annotated <- distinct\_data  annotated$loan\_status <- factor(annotated$loan\_status,  levels = c(0, 1),  labels = c("rejected", "accepted")) | |
| **Description:**  **factor()** is used to rename/annotate the current attribute values to a new one. ‘levels’ parameter contains the current attribute values in ‘loan\_status’ column and it maps them in the following way:  0 -> “rejected” & 1: “accepted” | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Visualizing the class distribution for categorical columns:** | |
| **Code:**  categorical\_cols <- names(annotated)[sapply(annotated, function(x) is.factor(x) | is.character(x))]  categorical\_cols  plotCategoricalCols <- function(data = annotated, col)  {  counts <- table(data[[col]])  bar\_positions <- barplot(counts,  main = paste("Distribution of ", col),  col = "orange",  xlab = "",  ylab = "Frequency",  cex.lab = 0.8,  cex.names = 0.9,  las = 2)    text(bar\_positions, counts, labels = counts, pos = 1, cex = 1)  }  plotCategoricalCols(annotated, "person\_gender")  plotCategoricalCols(annotated, "person\_education")  plotCategoricalCols(annotated, "person\_home\_ownership")  plotCategoricalCols(annotated, "loan\_intent")  plotCategoricalCols(annotated, "previous\_loan\_defaults\_on\_file")  plotCategoricalCols(annotated, "loan\_status") | |
| **Description:**  Returns the frequency of unique values of a specific column and visualizes them into a bar plot using the **barplot()** function. The ‘categorical\_cols’ array holds the values of the names of all the columns that are holding character or factor type data. | |
| **Screenshots:**    **A graph with numbers and a bar  AI-generated content may be incorrect.**  **A graph showing the number of objects  AI-generated content may be incorrect.** | A graph of a bar chart  AI-generated content may be incorrect.  A graph showing a number of different colored rectangular objects  AI-generated content may be incorrect. |

|  |  |
| --- | --- |
| 1. **Fixing Invalid Values in the Categorical Columns:** | |
| **Code:**  fixed\_invalid\_categorical <- annotated  valid\_values <- c("MORTGAGE", "OWN", "OTHER", "RENT")  fix\_values <- function(column, valid\_values) {  sapply(column, function(value) {  closest <- valid\_values[which.min(stringdist::stringdist(value, valid\_values))]  return(closest)  })  }  fixed\_invalid\_categorical <- fixed\_invalid\_categorical %>%  mutate(person\_home\_ownership = fix\_values(person\_home\_ownership, valid\_values))  plotCategoricalCols(fixed\_invalid\_categorical, "person\_home\_ownership") | |
| **Description:**  Fixing the invalid attribute values of a column by matching them with the valid values. If the values are valid, then keep them as they are, but if the values are invalid, replace them with the closest matching valid value.  This line: **closest <- valid\_values[which.min(stringdist::stringdist(value, valid\_values))]**  Matches a ‘value’ with the list of values in ‘valid\_values’ using the function from ‘stringdist’ library. If the ‘value’ is already valid, it will match with the ‘valid\_values’ and if it’s an invalid value then it will match with the list of ‘valid values’ and the closest matching ‘valid\_value’ is returned and replaced from the invalid. | |
| **Screenshot (Before correction):** | **Screenshot (After correction):** |

|  |  |
| --- | --- |
| 1. **Convert the values of categorical columns into lowercase letters.** | |
| **Code:**  categorical\_cols  lowered <- fixed\_invalid\_categorical  for (col in categorical\_cols) {  lowered[[col]] <- tolower(lowered[[col]])  } | |
| **Description:**  The attribute values of categorical columns were in both capital lettersand small letters, which is a critical aspect while mapping them to numeric values. To overcome this issue, all the attributes of all the categorical columns have been converted to lowercase letters using the **tolower()** method. | |
| **Screenshots:** | |
| **Before:** | **After:** |

|  |
| --- |
| 1. **Visualizing the missing values** |
| **Code:**  **1.**  colSums(is.na(lowered[categorical\_cols]))  for (col\_name in categorical\_cols)  {  cat(col\_name, " -> ", which(is.na(lowered[col\_name])), "\n")  } |
| **2.**  barplot(colSums(is.na(lowered[categorical\_cols])), las = 2, col = "blue",  main = "Missing Values per Categorical Column",  xlab = "", ylab = "Count of missing Values",  cex.lab = 0.9,  cex.names = 0.9) |
| **Description:**  The 1st code snippet returns the number of missing values for every categorical column using the **colSums()** function that sums all the occurrences of ‘TRUE’ values returned by **is.na()** function.  The 2nd snippet shows the instance index where the missing values are present for each of the categorical columns. For this **which()** function is used which returns the indexes of instances where at least one attribute value is ‘NA’.  The 3rd code snippet returns a bar plot showing the missing values in all the categorical columns using the **barplot()** function. |
| **Screenshot:**  **1st & 2nd code output:** |
|  |

|  |  |
| --- | --- |
| 1. **Discard rows with NULL values for Categorical Columns** | |
| **Code:**  discraded\_null <- lowered  discraded\_null <- na.omit(discraded\_null) | |
| **Description:**  This is one of the techniques to handle null values. This technique removes all the instances containing NULL or missing values. The **na.omit(discraded\_null)** function returns the dataset with all of its null values removed. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Handle NULL values with Top down Approach for Categorical Columns.** | |
| **Code:**  top\_down <- lowered  categorical\_cols <- names(top\_down)[sapply(top\_down, function(x) is.factor(x) | is.character(x))]  for (col in categorical\_cols) {  for (i in seq\_len(nrow(top\_down))[-1]) {  if (is.na(top\_down[[col]][i])) {  top\_down[[col]][i] <- top\_down[[col]][i - 1]  }  }  } | |
| **Description:**  This technique replaces the NULL values with the previous instance value of the same column.  A loop is running till the end of the column and checking if any instance value is null or not; if the condition finds any null values then replace them with the previous value. Here is the condition: **if (is.na(bottom\_up[[col]][i])).**  **top\_down[[col]][i] <- top\_down[[col]][i - 1]** this line replacing the previous instance value with the NULL value. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Handle NULL values with Bottom Up Approach for Categorical Columns.** | |
| **Code:**  bottom\_up <- lowered  categorical\_cols <- names(bottom\_up)[sapply(bottom\_up, function(x) is.factor(x) | is.character(x))]  for (col in categorical\_cols) {  for (i in seq\_len(nrow(bottom\_up) - 1)) {  if (is.na(bottom\_up[[col]][i])) {  bottom\_up[[col]][i] <- bottom\_up[[col]][i + 1]  }  }  } | |
| **Description:**  This technique handles the NULL values by replacing them with the value of the next instance of the same column.  A loop runs till the end of a column and checks if any instance is NULL or not. If the condition finds any NULL value then replace the value with the next instance value .Here is the condition: **if (is.na(bottom\_up[[col]][i]))**  **bottom\_up[[col]][i] <- bottom\_up[[col]][i + 1]** this line replacing the NULL value with the next instance value. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Replace NULL values with MODE for categorical columns** | |
| **Code:**  most\_frequent\_data <- lowered  categorical\_cols <- names(most\_frequent\_data)[sapply(most\_frequent\_data, function(x) is.factor(x) | is.character(x))]  for (col in categorical\_cols) {  most\_frequent <- names(sort(table(most\_frequent\_data[[col]]), decreasing = TRUE))[1]    most\_frequent\_data[[col]][which(is.na(most\_frequent\_data[[col]]))] <- most\_frequent  } | |
| **Description:**  Mode is the most frequent value of the whole column, and for handling null values for categorical columns, the mode value (the attribute value with more instances) was used.  A loop runs till the end of the column for finding Null values. If the condition finds a null value then replace them with mode value.  **names(sort(table(most\_frequent\_data[[col]]), decreasing = TRUE))[1]**, this line of code returns the mode value of a column.  **most\_frequent\_data[[col]][which(is.na(most\_frequent\_data[[col]]))] <- most\_frequent,** this line of code replacing the null values with the mode value. | |
| **Screenshot:** | |
| **Before:** | **After:** |

For handling the missing values of categorical columns, Mean and Median cannot be used. Generally, the most frequent value is used to handle the missing values. So, Mode has been used to replace all the “NA” values.

|  |
| --- |
| 1. **Label Encoding for the Categorical Columns** |
| **Code:**  label\_encoded <- most\_frequent\_data  gender <- unique(label\_encoded$person\_gender)  label\_encoded$person\_gender <- factor(label\_encoded$person\_gender,  levels = gender,  labels = 0:(length(gender) - 1))  education <- unique(label\_encoded$person\_education)  label\_encoded$person\_education <- factor(label\_encoded$person\_education,  levels = education,  labels = 0:(length(education) - 1))  home\_ownership <- unique(label\_encoded$person\_home\_ownership)  label\_encoded$person\_home\_ownership <- factor(label\_encoded$person\_home\_ownership,  levels = home\_ownership,  labels = 0:(length(home\_ownership) - 1))  intent <- unique(label\_encoded$loan\_intent)  label\_encoded$loan\_intent <- factor(label\_encoded$loan\_intent,  levels = intent,  labels = 0:(length(intent) - 1))  loan\_defaults\_on\_file <- unique(label\_encoded$previous\_loan\_defaults\_on\_file)  label\_encoded$previous\_loan\_defaults\_on\_file <- factor(label\_encoded$previous\_loan\_defaults\_on\_file,  levels = loan\_defaults\_on\_file,  labels = 0:(length(loan\_defaults\_on\_file) - 1)) |
| **Description:**  This technique maps all the attributes of a categorical column to a numeric value.  **gender <- unique(label\_encoded$person\_gender)** , this line of code takes all the unique values of a categorical column.  **label\_encoded$person\_gender <- factor(label\_encoded$person\_gender,** this line of code maps all the unique values to a numeric value. |
| **Screenshot:** |
| **Before:** |
| **After:** |

|  |
| --- |
| 1. **Summary of the Numeric Columns** |
| **Code:**  str(label\_encoded)  numeric\_cols <- names(label\_encoded)[sapply(label\_encoded, is.numeric)]  summary(label\_encoded[numeric\_cols]) |
| **Description:**  **str(data)** shows a small overview of the numeric columns. And **summary()** shows a short summary of each numeric column (minimum and maximum values, mean, median, 1st quartile, and 3rd quartile values for numeric attributes and the number of missing values for all attributes). |
| **Screenshot:** |
|  |

|  |  |
| --- | --- |
| 1. **Plotting the Numeric Columns** | |
| **Code:**  plotFreq <- function(col\_name)  {  # Create bar plot  barplot(table(data[[col\_name]]),  main = paste("Mean value for ", col\_name, ": ", mean(data[[col\_name]], na.rm = TRUE)),  col = "skyblue",  xlab = col\_name,  ylab = "Frequency",  las = 2)  }  plotFreq("person\_age")  plotFreq("person\_income")  plotFreq("person\_emp\_exp")  plotFreq("loan\_amnt")  plotFreq("loan\_int\_rate")  plotFreq("loan\_percent\_income")  plotFreq("cb\_person\_cred\_hist\_length")  plotFreq("credit\_score") | |
| **Description:**  This code snippet returns the frequency of the values in all the numeric columns. Then the frequency is shown using a **barplot().** | |
| **Screenshot:** | |
|  | **A graph of blue and black lines  AI-generated content may be incorrect.** |

|  |
| --- |
| 1. **Plotting the NULL values of Numeric Columns** |
| **Code:**  colSums(is.na(label\_encoded))  for (col\_name in numeric\_cols)  {  cat(col\_name, " -> ", which(is.na(label\_encoded[col\_name])), "\n")  }  barplot(colSums(is.na(label\_encoded[numeric\_cols])), las = 2, col = "blue",  main = "Missing Values per Numeric Column",  xlab = "", ylab = "Count of missing Values",  cex.lab = 0.9,  cex.names = 0.9) |
| **Description:**  This code snippet returns a plot with all the numeric columns containing missing or NULL values. |
| **Screenshot:** |

|  |  |
| --- | --- |
| 1. **Discard numeric entries with missing or NULL values for Numerical Columns.** | |
| **Code:**  discraded\_null\_numeric <- label\_encoded  discraded\_null\_numeric <- na.omit(discraded\_null\_numeric) | |
| **Description:**  This is one of the techniques to handle null values. This technique removes all the instances containing NULL or missing values. The **na.omit(discraded\_null\_numeric)** function returns the dataset with all of its null values removed from the numeric columns. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Handling NULL values with Top Down Approach for Numerical Columns.** | |
| **Code:**  top\_down\_numeric\_null <- label\_encoded  for (col in numeric\_cols) {  for (i in seq\_len(nrow(top\_down\_numeric\_null))[-1]) {  if (is.na(top\_down\_numeric\_null[[col]][i])) {  top\_down\_numeric\_null[[col]][i] <- top\_down\_numeric\_null[[col]][i - 1]  }  }  } | |
| **Description:**  This approach is for replacing NULL values using the previous value of the column. This is a similar technique with 11 no technique. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Handling NULL values with Bottom Up Approach for Numerical Columns.** | |
| **Code:**  bottom\_up\_numeric\_null <- label\_encoded  for (col in numeric\_cols) {  for (i in seq\_len(nrow(bottom\_up\_numeric\_null) - 1)) {  if (is.na(bottom\_up\_numeric\_null[[col]][i])) {  bottom\_up\_numeric\_null[[col]][i] <- bottom\_up\_numeric\_null[[col]][i + 1]  }  }  } | |
| **Description:**  This approach is for replacing NULL values using the next value of the column. This is a similar technique with 12 no technique. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Handling NULL values with MODE Approach for Numerical Columns.** | |
| **Code:**  mode\_replaced\_numeric\_null <- label\_encoded  for (col in numeric\_cols) {  most\_frequent <- names(sort(table(mode\_replaced\_numeric\_null[[col]]), decreasing = TRUE))[1]  mode\_replaced\_numeric\_null[[col]][which(is.na(mode\_replaced\_numeric\_null[[col]]))] <- most\_frequent  } | |
| **Description:**  This approach is for replacing NULL values using the MODE value of the column. This is a similar technique with 13 no technique. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Handling NULL values with MEAN Approach for Numerical Columns.** | |
| **Code:**  mean\_replaced\_numeric\_null <- label\_encoded  for (col in numeric\_cols) {  if (any(is.na(mean\_replaced\_numeric\_null[[col]]))) {  mean\_value <- round(mean(mean\_replaced\_numeric\_null[[col]], na.rm = TRUE))  mean\_replaced\_numeric\_null[[col]][which(is.na(mean\_replaced\_numeric\_null[[col]]))] <- mean\_value  }  } | |
| **Description:**  This approach is for replacing NULL values with the MEAN value of the column.  A loop is running until the last attribute of a column to check for NULL values.  **if (any(is.na(mean\_replaced\_numeric\_null[[col]]))),** this is the condition to check for NULL values and if the condition finds a NULL value it replace it with the MEAN value.  **mean\_replaced\_numeric\_null[[col]][which(is.na(mean\_replaced\_numeric\_null[[col]]))] <- mean\_value,** this is the code for replacing the NULL value with MEAN value. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Handling NULL values with MEDIAN Approach for Numerical Columns.** | |
| **Code:**  median\_replaced\_numeric\_null <- label\_encoded  for (col in numeric\_cols) {  if (any(is.na(median\_replaced\_numeric\_null[[col]]))) {  median\_value <- median(median\_replaced\_numeric\_null[[col]], na.rm = TRUE)  median\_replaced\_numeric\_null[[col]][which(is.na(median\_replaced\_numeric\_null[[col]]))] <- median\_value  }  } | |
| **Description:**  This technique replaces the NULL values with the median value of a column.  A loop is running in a column until the column's last attribute to find the NULL value from the column.  **if (any(is.na(median\_replaced\_numeric\_null[[col]]))),** this is the line of the condition of checking NULL values.  **median\_replaced\_numeric\_null[[col]][which(is.na(median\_replaced\_numeric\_null[[col]]))] <- median\_value,** this replace the median value with the NULL value | |
| **Screenshot:** | |
| **Before:** | **After:** |

For handling ‘NA’ values there are several techniques, but Median has been used because the dataset contains outlier values and mean does not work well when the dataset contains outliers. Aside from the missing values, the distribution of the dataset is skewed. So, the mode value is not suitable for numeric columns. That’s why the median has been used.

|  |
| --- |
| 1. **Finding the Standard Deviation for all the Numeric Columns** |
| **Code:**  median\_replaced\_numeric\_null %>% summarise\_if(is.numeric, sd) |
| **Description:**  This returns the standard Deviation of all the numeric columns. |
| **Screenshot:** |

|  |  |
| --- | --- |
| 1. **Applying Z score to check and handle all the Outlier in the Numerical Columns** | |
| **Code:**  z\_score\_outlier\_handeled <- median\_replaced\_numeric\_null  for (col in numeric\_cols) {  z\_scores <- scale(z\_score\_outlier\_handeled[[col]])  z\_score\_outlier\_handeled <- z\_score\_outlier\_handeled[abs(z\_scores) <= 3, ]  } | |
| **Description:**  This technique helps to check all the outliers in the numeric columns. This identifies and handles all the outliers from the numeric values.  **z\_scores <- scale(z\_score\_outlier\_handeled[[col]]),** this line returns all the outliers.  **z\_score\_outlier\_handeled <- z\_score\_outlier\_handeled[abs(z\_scores) <= 3,** by this line of code all the outliers have been handled in between the z-score 3. If the value of z is greater than 3, it marks those values as outliers and discard them. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Using IQR to check and handle the outliers** | |
| **Code:**  iqr\_outlier\_handled <- median\_replaced\_numeric\_null  for (col in numeric\_cols) {  Q1 <- quantile(iqr\_outlier\_handled[[col]], 0.25, na.rm = TRUE)  Q3 <- quantile(iqr\_outlier\_handled[[col]], 0.75, na.rm = TRUE)  IQR <- Q3 - Q1    lower\_bound <- Q1 - 1.5 \* IQR  upper\_bound <- Q3 + 1.5 \* IQR    iqr\_outlier\_handled <- iqr\_outlier\_handled[iqr\_outlier\_handled[[col]] >= lower\_bound & iqr\_outlier\_handled[[col]] <= upper\_bound, ]  } | |
| **Description:**  This technique finds and handles outliers. **IQR <- Q3 - Q1,** by this line of code, it detects the outliers. If the value is greater than Q3 and less than Q1, then the value is marked as an outlier.  **iqr\_outlier\_handled <- iqr\_outlier\_handled[iqr\_outlier\_handled[[col]] >= lower\_bound & iqr\_outlier\_handled[[col]] <= upper\_bound,** this code is removing all the values that are higher than **upper\_bound** and lower than **lower\_bound.** | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Applying CHI square to get the range of Numeric columns** | |
| **Code:**  sapply(z\_score\_outlier\_handeled[numeric\_cols], function(x) if(is.numeric(x)) sd(x, na.rm = TRUE))  chi\_squared <- z\_score\_outlier\_handeled  person\_income\_bins <- cut(chi\_squared$person\_income, breaks = 4)  levels(person\_income\_bins)  levels(person\_income\_bins) <- c("Low", "Lower Middle", "Upper Middle", "High")  chi\_squared$person\_income <- person\_income\_bins  amount <- cut(chi\_squared$loan\_amnt, breaks = 3)  levels(amount)  levels(amount) <- c("Small", "Medium", "Large")  chi\_squared$loan\_amnt <- amount  str(chi\_squared) | |
| **Description:**  This code helps to convert numeric column to categorical column and helps to find the perfect range for doing so.  **cut(chi\_squared$person\_income, breaks = 4),** By this code, the person\_income\_bins column has been partitioned into 4 categories.  **chi\_squared$person\_income <- person\_income\_bins,** this line of code replacing the numeric values to the categorical values. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Normalizing the numeric values** | |
| **Code:**  normalized\_numeric <- chi\_squared  col\_min <- min(normalized\_numeric[["credit\_score"]])  col\_max <- max(normalized\_numeric[["credit\_score"]])  normalized\_numeric[["credit\_score"]] <- (normalized\_numeric[["credit\_score"]] - col\_min) / (col\_max - col\_min)  normalized\_numeric[["loan\_int\_rate"]] <- (normalized\_numeric[["loan\_int\_rate"]] / 100) | |
| **Description:**  This technique helps to normalize the numeric value and convert every value in a range on 0-1. The large numbers are squeezed between 0-1 for easy representation.  **normalized\_numeric[["credit\_score"]] <- (normalized\_numeric[["credit\_score"]] - col\_min) / (col\_max - col\_min),** Min-max algorithm has been used to normalize the numeric data.  **normalized\_numeric[["loan\_int\_rate"]] <- (normalized\_numeric[["loan\_int\_rate"]] / 100,**  this line of code convert all the values to 0 - 1. | |
| **Screenshot:** | |
| **Before:** | **After:** |
|  | |

|  |  |
| --- | --- |
| 1. **Filtering the numeric values** | |
| **Code:**  normalized\_numeric\_filtered <- median\_replaced\_numeric\_null %>% filter(person\_age < 80) | |
| **Description:**  This code is filtering the outliers and replacing those with median values. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Using upsampling in the numeric columns to balance the dataset** | |
| **Code:**  balanced\_data <- normalized\_numeric\_filtered  table(balanced\_data$loan\_status)  plotCategoricalCols(balanced\_data, "loan\_status")  minority\_class <- filter(balanced\_data, loan\_status == "rejected")  majority\_class <- filter(balanced\_data, loan\_status == "accepted")  num\_to\_add <- nrow(majority\_class) - nrow(minority\_class)  num\_to\_add <- num\_to\_add + 20  upsampled\_minority <- slice\_sample(minority\_class, n = num\_to\_add, replace = FALSE)  balanced\_data <- bind\_rows(majority\_class, minority\_class, upsampled\_minority)  table(balanced\_data$loan\_status)  plotCategoricalCols(balanced\_data, "loan\_status") | |
| **Description:**  This technique has been used to make the dataset balanced. By doing upsampling, the minor category increases its instance numbers.  **upsampled\_minority <- slice\_sample(minority\_class, n = num\_to\_add, replace = FALSE),** by this line of code, the minor category has been increased. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |  |
| --- | --- |
| 1. **Applying Downsampling to balance the dataset** | |
| **Code:**  minority\_class <- filter(balanced\_data, loan\_status == "accepted")  majority\_class <- filter(balanced\_data, loan\_status == "rejected")  downsampled\_majority\_class <- majority\_class %>% sample\_n(nrow(minority\_class))  balanced\_data <- bind\_rows(downsampled\_majority\_class, minority\_class)  table(balanced\_data$loan\_status)  plotCategoricalCols(balanced\_data, "loan\_status") | |
| **Description:**  This technique helps to reduce the size of instances of the major class of a column.  **downsampled\_majority\_class <- majority\_class %>% sample\_n(nrow(minority\_class)),** this code helps to reduce the number of instances of the majority class in a column. | |
| **Screenshot:** | |
| **Before:** | **After:** |

|  |
| --- |
| 1. **Summary after the preprocessed dataset** |
| **Code:**  str(balanced\_data)  summary(balanced\_data) |
| **Description:**  This shows the summary of the dataset after preprocessing |
| **Screenshot:**  **Before Preprocessing:** |
| **After Preprocessing:** |

From the before and after summaries of the dataset, it can be seen that handling the missing values & outliers, the overall measures of central tendencies as well as the spread have decreased, which was the initial goal of data preprocessing. The target attribute was encoded, as well as some of the numeric columns with high standard deviation (person\_income, loan\_amnt) which were also encoded.

|  |
| --- |
| 1. **Correlation Matrix** |
| numeric\_data <- dplyr::select\_if(balanced\_data, is.numeric)  cor\_matrix <- cor(numeric\_data, use = "pairwise.complete.obs")  library(corrplot)  corrplot(cor\_matrix,  method = "color",  type = "upper",  addCoef.col = "black",  tl.col = "black",  tl.srt = 45,  col = colorRampPalette(c("red", "white", "blue"))(200)) |
| **Description:**  This technique demonstrates **downsampling**, which is used to handle class imbalance in a dataset. When one class (the *majority class*) has far more instances than another (the *minority class*), models may become biased toward the majority. |
| **Screenshot:** |

|  |
| --- |
| 1. **Export preprocessed dataset** |
| **Code:**  write.xlsx(balanced\_data, "data\_preprocessed.xlsx") |
| **Description:**  This code exported the preprocessed dataset named “data\_preprocessed” in .xlsx format. |
| **Screenshot:**  **Before preprocess** |
| **After preprocess:** |

**Conclusion:**

Through this project, the dataset was carefully examined and preprocessed to handle issues such as missing values, invalid categorical entries, skewed distributions, and outliers. Multiple strategies (including top-down, bottom-up, mean, median, and mode imputation) were tested for missing data, while outlier detection techniques such as Z-score and IQR were applied. Class imbalance was addressed using both upsampling and downsampling to create a more balanced dataset.

The main findings indicate that variables such as credit score, income, loan amount, and loan percent income play a significant role in influencing loan approval decisions. Correlation analysis further revealed important relationships among numeric features that could guide predictive modeling.

However, some limitations were encountered:

* The dataset size (201 instances) is relatively small, which may restrict generalizability.
* Imputation methods for missing values may introduce bias, especially when replacing with median or mode.
* Even after balancing, some degree of information loss occurs due to downsampling the majority class.

Overall, the preprocessing steps improved the dataset’s quality and readiness for further machine learning applications, ensuring more reliable insights in loan approval prediction tasks.