**Dataset Description**

A credit-risk dataset where each row is one borrower and one loan application. It combines basic demographics, finances, and loan details, plus two binary outcome fields.

All columns:

**person\_age** — borrower’s age (years)

**person\_gender** — gender

**person\_education** — highest education level

**person\_income** — annual income

**person\_emp\_exp** — years of employment experience

**person\_home\_ownership** — housing status (e.g., RENT, OWN, MORTGAGE)

**loan\_amnt** — requested loan amount

**loan\_intent** — purpose of the loan (e.g., PERSONAL, MEDICAL)

**loan\_int\_rate** — interest rate (%)

**loan\_percent\_income** — loan payment as a share of income (0–1)

**cb\_person\_cred\_hist\_length** — credit history length (years)

**credit\_score** — credit score (approx. 300–850)

**previous\_loan\_defaults\_on\_file** — whether the borrower ever defaulted before (0/1)

**loan\_status** — outcome of this loan application or performance (0/1)

**1.Handle missing value:**

**Description:**

* **Drop missing rows (complete cases).**

We first list rows that contain any NA to inspect what’s missing, then create a complete-case view by removing those rows.

* **Replace with mean (numeric).**

For a numeric column that is roughly symmetric (here, person\_income), we fill NA values with the column mean calculated from the available (non-missing) values.

* **Replace with median (numeric).**

For a numeric column that may be skewed or affected by outliers (here, person\_age), we fill NA values with the median, which is more robust than the mean.

* **Replace with mode (categorical).**

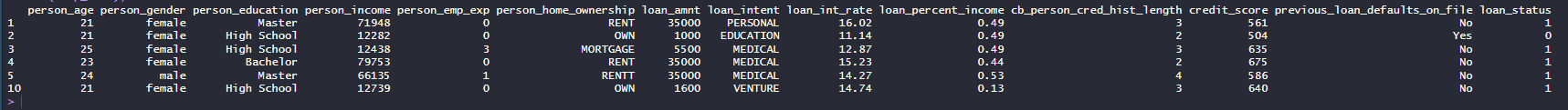
For a categorical/label column (here, loan\_status), we fill NA values with the mode (the most frequent category). The mlv(..., method="mfv") function returns that most frequent value, which we use to impute the missing entries.

**Code:**

* Drop missing value:

|  |
| --- |
| missing\_rows <- data[!complete.cases(data), ] head(missing\_rows) |

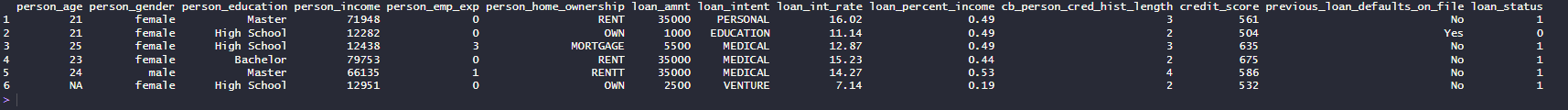
**Output:**



* Replace with mean

|  |
| --- |
| df\_mean <-data  df\_mean$person\_income[is.na(df\_mean$person\_income)] <- mean(df\_mean$person\_income,na.rm=TRUE)  head(df\_mean) |

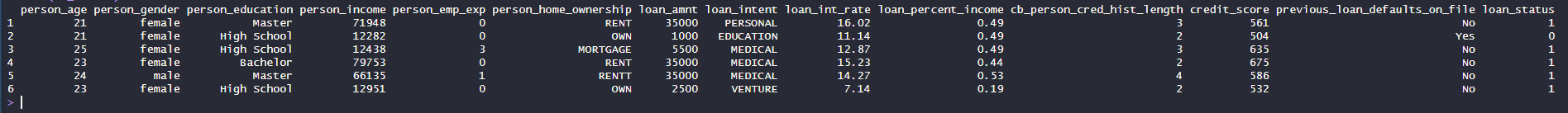
**Output:**



* Replace with median

|  |
| --- |
| df\_median <- df\_mean#replace by median  df\_median$person\_age[is.na(df\_median$person\_age)] <- median(df\_median$person\_age,na.rm = TRUE);  head(df\_median) |

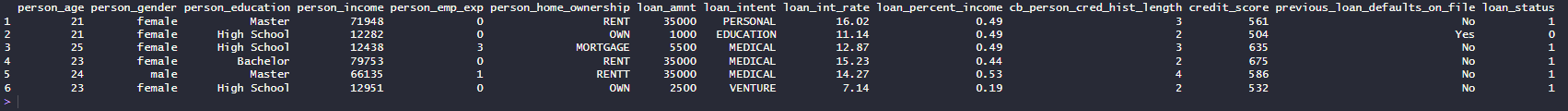
**Output:**



* Replace with mode

|  |
| --- |
| mode\_val <- mlv(df\_mode$loan\_status,method = "mfv",na.rm = TRUE);  df\_mode$loan\_status[is.na(df\_mode$loan\_status)] <- mode\_val  head(as.data.frame(df\_mode)) |

**Output:**



**2. see missing values on a graph:**

**Description:**In this task we will use the vis\_miss() function from the *naniar* package to visualize missing values in the dataset with a graph. The graph will display variables as columns and records as rows, where shaded blocks indicate missing entries. This makes it easy to see which columns contain missing data and how much is missing, giving us a clear picture of the dataset’s quality.

|  |
| --- |
| vis\_miss(data) |

A graph with text on it

AI-generated content may be incorrect.

**3. convert attributes from numeric to categorical or categorical to numeric:**

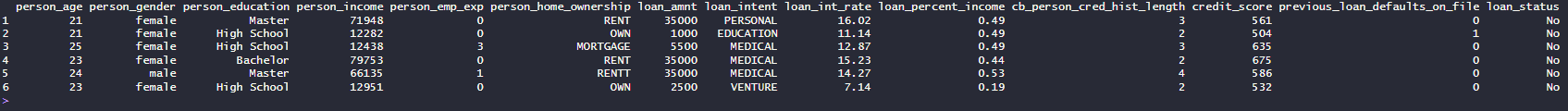
**Description:**

In this task we will convert attributes from numeric to categorical or from categorical to numeric depending on how we need them for analysis. For example, the column loan\_status is stored as numbers (0/1), but for easier interpretation we convert it into labels "Yes" and "No". On the other hand, the column previous\_loan\_defaults\_on\_file may be stored as text ("Yes"/"No"), so we convert it back into numeric values (1/0) for modeling.

We solve this by using simple ifelse() statements:

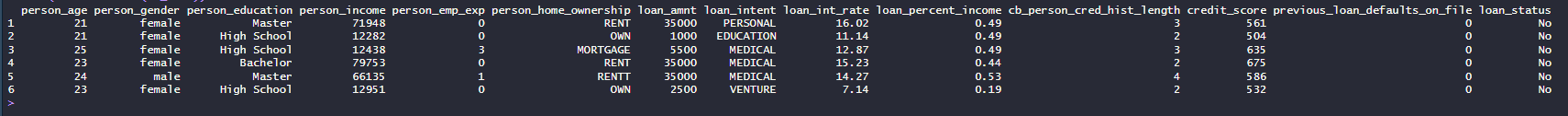
|  |
| --- |
| df\_mode$loan\_status <-ifelse(df\_mode$loan\_status == 1 ,"Yes","No");  head(df\_mode) |

**Output:**



|  |
| --- |
| df\_mode$previous\_loan\_defaults\_on\_file <- ifelse(df\_mode$previous\_loan\_defaults\_on\_file == "Yes", 1,0);  head(df\_mode) |

**Output:**

****

**4. Detect outliers in the data set and use the appropriate approach to handle those values:**

**Description:**

In this task we detect and handle extreme values in numeric columns using the Interquartile Range (IQR) method. First, we calculate Q1 (25th percentile), Q3 (75th percentile), and the IQR (Q3 − Q1). Any value smaller than Q1 − 1.5 × IQR is treated as a lower outlier, and any value greater than Q3 + 1.5 × IQR is treated as an upper outlier.

The get\_outliers() function lists these values so we can see which records fall outside the normal range. Then the fix\_outliers() function corrects them:

* If a value is less than the lower bound, it is replaced by the lower bound.
* If a value is greater than the upper bound, it is replaced by the upper bound.

This way, the dataset does not lose any rows, but all values are adjusted to stay within reasonable limits, reducing the influence of extreme numbers on further analysis.

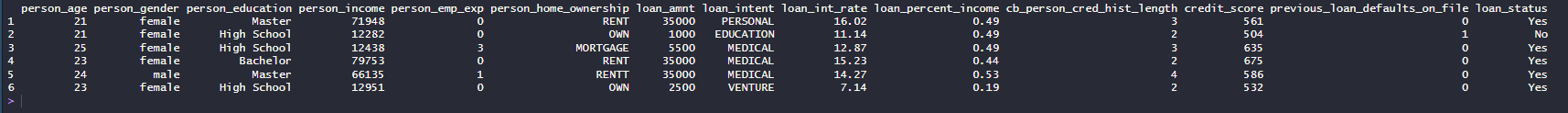
|  |
| --- |
| get\_outliers <- function(data, colname) {  if (!colname %in% names(data)) stop("Column not found.")  if (!is.numeric(data[[colname]])) stop("Column must be numeric.")    Q1 <- quantile(data[[colname]], 0.25, na.rm = TRUE)  Q3 <- quantile(data[[colname]], 0.75, na.rm = TRUE)  IQR <- Q3 - Q1    lower <- Q1 - 1.5 \* IQR  upper <- Q3 + 1.5 \* IQR    # return only the outlier values from dataset  outliers <- data[[colname]][data[[colname]] < lower | data[[colname]] > upper]    return(outliers)  }  outlier\_detect<-get\_outliers(df, "person\_age")  head(outlier\_detect) |

**Output:**



|  |
| --- |
| fix\_outliers <- function(data, colname) {  if (!colname %in% names(data)) stop("Column not found.")  if (!is.numeric(data[[colname]])) stop("Column must be numeric.")    Q1 <- quantile(data[[colname]], 0.25, na.rm = TRUE)  Q3 <- quantile(data[[colname]], 0.75, na.rm = TRUE)  IQR <- Q3 - Q1    lower <- Q1 - 1.5 \* IQR  upper <- Q3 + 1.5 \* IQR    is\_outlier <- data[[colname]] < lower | data[[colname]] > upper  n\_out <- sum(is\_outlier, na.rm = TRUE)    if (n\_out > 0) {  data[[colname]] <- pmin(pmax(data[[colname]], lower), upper)  message("Fixed ", n\_out, " outliers in ", colname,  " (capped to [", round(lower, 2), ", ", round(upper, 2), "])")  } else {  message("No outliers to fix in ", colname)  }    return(data)  }  df\_clean <- fix\_outliers(df, "person\_age")  head(as.data.frame(df\_clean)) |

**Output:**

****

**Task 5: Normalization method for any continuous attribute:**

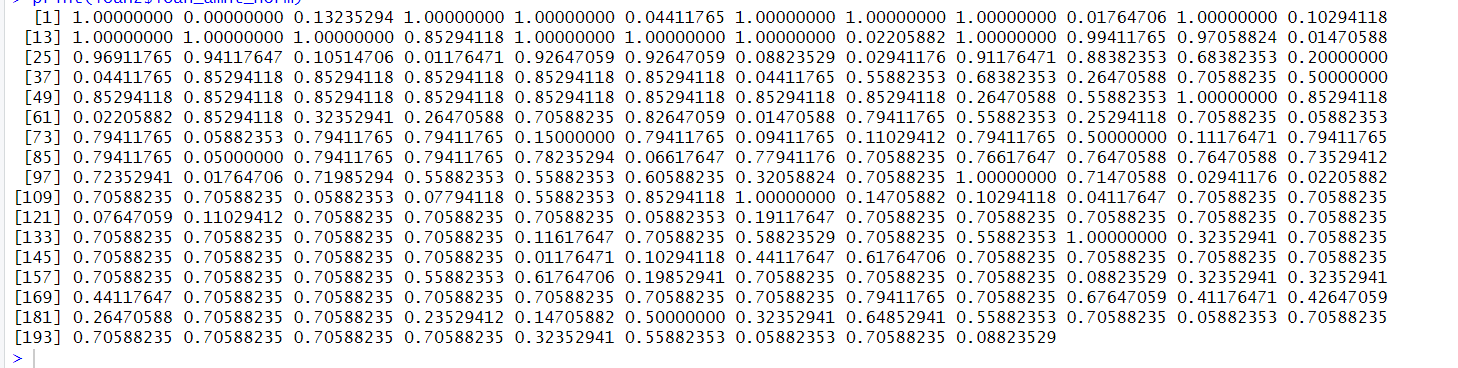
**Description:** In this task we apply normalization to a continuous column so that all its values are scaled into the range 0 to 1. We use the min–max normalization formula:

Here we normalize the column loan\_amnt. The smallest loan amount becomes 0, the largest loan amount becomes 1, and all other values are scaled proportionally between 0 and 1.

This step ensures that attributes measured on different scales can be compared fairly

|  |
| --- |
| loan2 <- Loan\_Datas  loan2$loan\_amnt\_norm <- (loan2$loan\_amnt - min(loan2$loan\_amnt, na.rm = TRUE)) /  (max(loan2$loan\_amnt, na.rm = TRUE) - min(loan2$loan\_amnt, na.rm = TRUE))  print(loan2$loan\_amnt\_norm) |

**Output:**



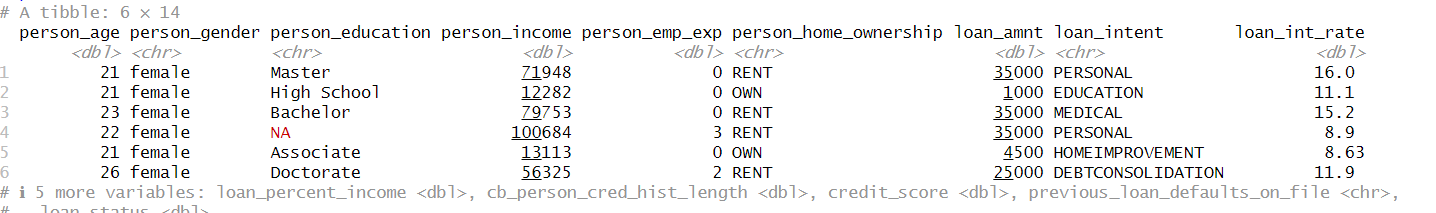
**Task 6: Find and remove duplicate rows to reduce congestion:**

**Description:**

In this task we look for duplicate rows in the dataset and remove them to make the data cleaner and less congested. We use the distinct() function from the dplyr package. This function checks for duplicates based on a chosen column (here person\_education) and keeps only the first occurrence, while the option .keep\_all = TRUE ensures that all other columns in that row are also preserved.

|  |
| --- |
| loan3 <- Loan\_Datas  library(dplyr)  distinct\_data <- distinct(loan3, person\_education, .keep\_all = TRUE)  print(distinct\_data) |

**Output:**



**Description:**

**Task 7: Filter Data to find specific numerical value , row, categorical value:**

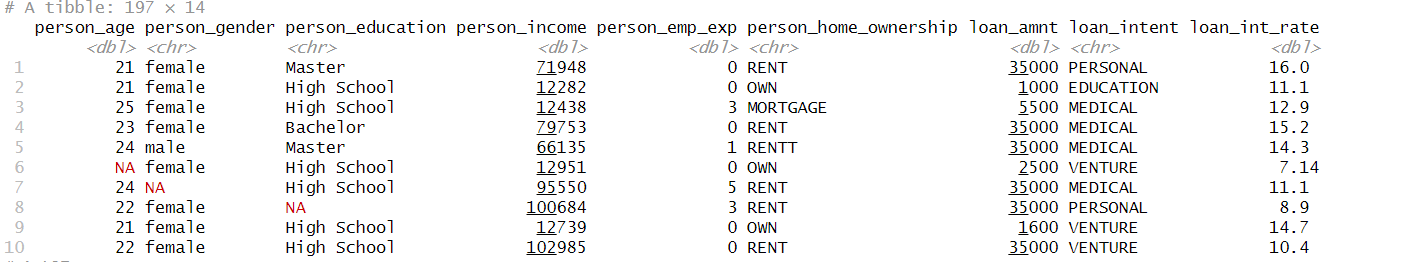
**Description:**

There are many ways to filter data in R, but here we are using the filter() function from the dplyr package. This allows us to select rows that match certain conditions. For example:

* Filtering numerical values such as loan\_amnt > 10000 to get only loans greater than 10,000.
* Filtering based on both missingness and value, such as keeping rows where person\_income is not missing and greater than 20,000.
* Filtering categorical values, such as selecting rows where person\_education is "Bachelor" or "Master".

|  |
| --- |
| loan4<-Loan\_Datas  filtered\_data <- filter(loan3, loan\_amnt > 10000)  print(filtered\_data)  filtered\_data <- filter(loan3, !is.na(person\_income) & person\_income > 20000)  print(filtered\_data)  filtered\_data <- filter(loan3, person\_education %in% c("Bachelor", "Master"))  print(filtered\_data) |

**Output:**



**Description:**

We used filter method we can use multiple operation for filter that is more than 10000 value or whether the value is null or not If is null then delete this row. Then filter only master, bachelor row of person\_income column. Also filter more than 20000 of person\_income and without null value.

**Task8: Detect invalid value then remove this row or replace mean value:**

**Description:**

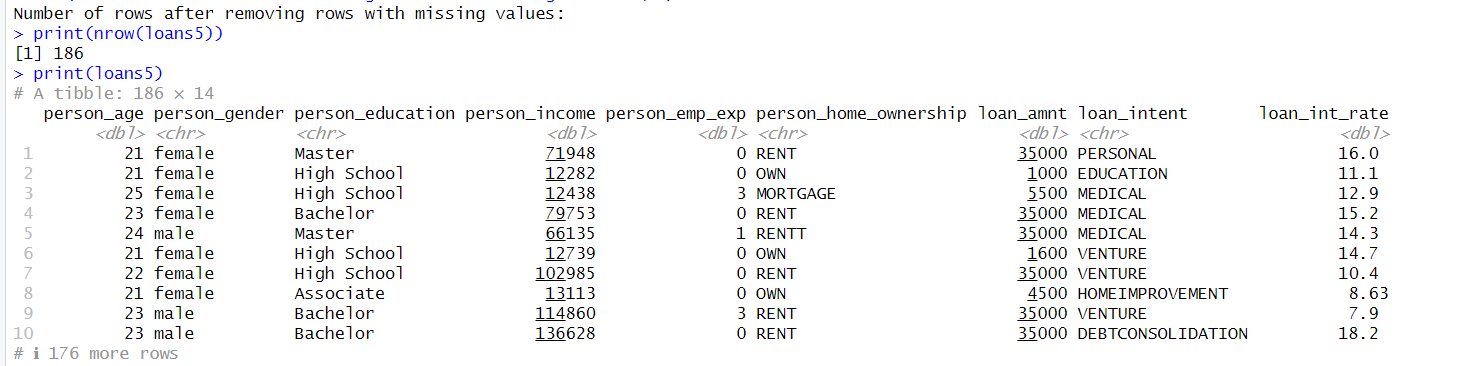
There are many ways to handle invalid/missing values; here we use two simple approaches:

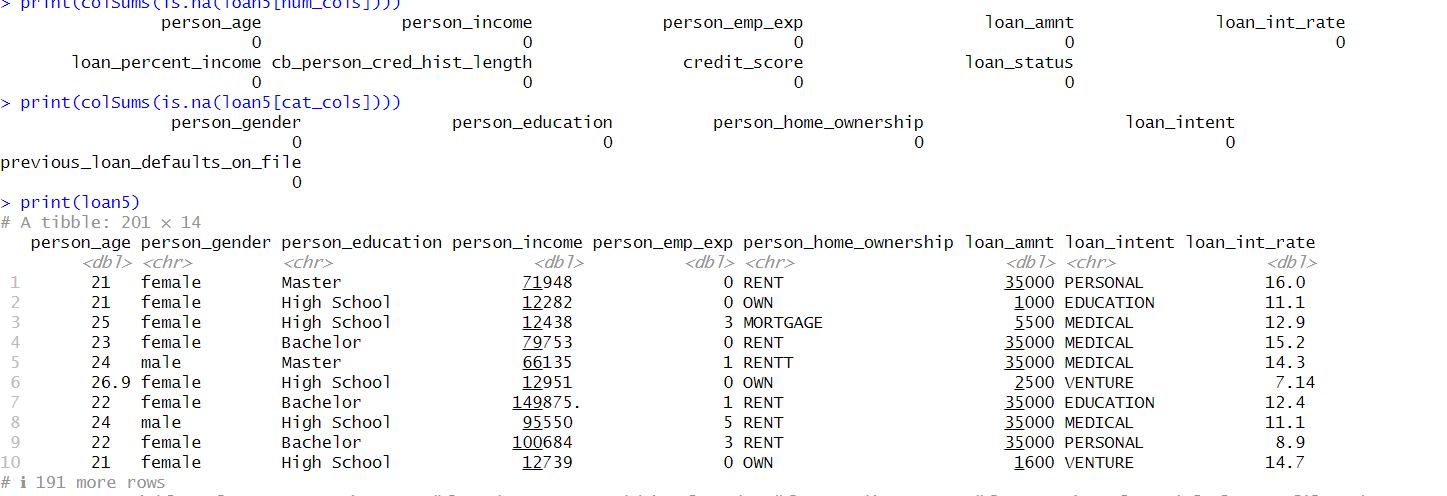
* Remove rows with any missing values using na.omit() to get a clean subset and check how many rows remain.
* Impute (fill) missing values in the full dataset:
  + For numeric columns, replace NA with the mean of that column.
  + For categorical columns, replace NA with the mode (most frequent category).

After imputation, we verify the result by printing the count of remaining NAs per column group and previewing the cleaned data.

|  |
| --- |
| loans5<-Loan\_Datas  loans5 <- na.omit(loans5)  print(nrow(loans5))  print(loans5)  loan5 <- Loan\_Datas  loan3\_clean <- na.omit(loan5)  print(nrow(loan3\_clean))  loan3\_clean  num\_cols <- sapply(loan5, is.numeric)  cat\_cols <- sapply(loan5, is.character)  loan5[num\_cols] <- lapply(loan5[num\_cols], function(x) {  x[is.na(x)] <- mean(x, na.rm = TRUE)  return(x)  })  loan5[cat\_cols] <- lapply(loan5[cat\_cols], function(x) {  mode\_val <- names(sort(table(x), decreasing = TRUE))[1]  x[is.na(x)] <- mode\_val  return(x)  })  print(colSums(is.na(loan5[num\_cols])))  print(colSums(is.na(loan5[cat\_cols])))  print(loan5) |

**Output:**





**Task 9. We can convert the imbalanced data set into the balanced data set :**

* **Undersampling:**

|  |
| --- |
| library(ROSE)  df$loan\_status <- factor(df$loan\_status, levels = c(0,1))  table(df$loan\_status)  N\_under <- 2 \* min(table(df$loan\_status))  set.seed(199)  under\_df <- ovun.sample(loan\_status ~ ., data = df,method = "under", N = N\_under, seed = 199)$data  table(under\_df$loan\_status)  head(under\_df) |

**Output:**

A group of small squares

AI-generated content may be incorrect.

* **Oversampling:**

|  |
| --- |
| library(ROSE)  df$loan\_status <- factor(df$loan\_status, levels = c(0,1))  table(df$loan\_status)  N\_over <- 2 \* max(table(df$loan\_status))  set.seed(199)  over\_df <- ovun.sample(loan\_status ~ ., data = df,  method = "over", N = N\_over, seed = 199)$data  table(over\_df$loan\_status)  head(over\_df) |

**Output:**

**A black background with white text

AI-generated content may be incorrect.**

* **Smote**

|  |
| --- |
| library(ROSE)  set.seed(199)  df\_smote$previous\_loan\_defaults\_on\_file <- factor(df\_smote$previous\_loan\_defaults\_on\_file, levels = c(0,1))  df\_smote[sapply(df\_smote, is.character)] <- lapply(df\_smote[sapply(df\_smote, is.character)], factor)  df\_smote <- df\_smote[complete.cases(df\_smote), ]  table(df\_smote$previous\_loan\_defaults\_on\_file)  rose\_df <- ROSE(previous\_loan\_defaults\_on\_file ~ ., data = df\_smote, N = 2000, p = 0.5)$data  table(rose\_df$previous\_loan\_defaults\_on\_file)  head(rose\_df) |

**Output:**

**A group of colorful squares

AI-generated content may be incorrect.**

**Task 10: Split the dataset for Training and Testing , 70% row for Training data and 30% for Testing data:**

**Description:**

In this task we divide the dataset into two parts: 70% for training and 30% for testing. The function initial\_split() from the rsample package is used to create the split. From this split object, the training() function extracts the training set, and the testing() function extracts the testing set.

The training set is used to fit and build the model, while the testing set is kept aside to check how well the model performs on new, unseen data. Finally, the dim() function shows the number of rows and columns in each set to confirm that the split was done correctly (about 70% of rows in training and 30% in testing).

|  |
| --- |
| set.seed(123)  split <- initial\_split(df, prop = 0.7)  train\_data <- training(split)  test\_data <- testing(split)  dim(train\_data)  dim(test\_data) |

Output:

A computer screen shot of a code

AI-generated content may be incorrect.

**Description:**

**11. statistics and interpret the results for the following numerical:**

variables between two target classes (loan\_status = 1 and loan\_status = 0)

➢ person\_age

➢ Person Income:  
**Description:**In this task we calculate summary statistics of two numerical variables — person\_age and person\_income — separately for the two target classes of loan\_status (0 and 1).

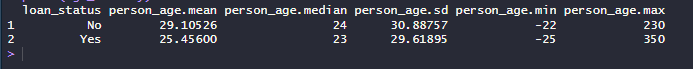
We use the aggregate() function with a custom summary that computes the mean, median, standard deviation, minimum, and maximum for each variable within each class. This produces grouped summaries:

* For loan\_status = 0 (non-default customers)
* For loan\_status = 1 (default customers)

The results help us interpret how age and income differ between the two classes. For example, we can see whether defaulters tend to be younger or older, and whether their average income is higher or lower compared to non-defaulters. This comparison gives useful insights into which demographic or financial factors may be linked to loan outcomes.

|  |
| --- |
| age\_summary <- aggregate(person\_age ~ loan\_status, data = df,  FUN = function(x) c(mean = mean(x), median = median(x),  sd = sd(x), min = min(x), max = max(x)))  age\_summary <- do.call(data.frame, age\_summary)  # Create summary table for Income  income\_summary <- aggregate(person\_income ~ loan\_status, data = df,  FUN = function(x) c(mean = mean(x), median = median(x),  sd = sd(x), min = min(x), max = max(x)))  income\_summary <- do.call(data.frame, income\_summary)  print(age\_summary) |

**Output:**



|  |
| --- |
| print(income\_summary) |

**Output:**

A screenshot of a computer

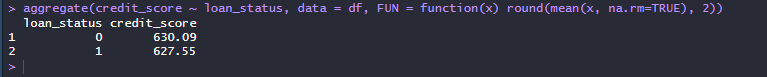
AI-generated content may be incorrect.

**Task 12 .Compare average credit\_score between customers with loan\_status 1 and with loan\_status 0:  
  
Description:**To compare the average credit score between customers with different loan statuses, we applied the aggregate() function in R. The dataset was grouped by loan\_status, where 0 represents customers without default (good status) and 1 represents customers with default (bad status). The mean credit score was then calculated for each group, rounded to two decimal places, and missing values were ignored (na.rm=TRUE).

The results show that customers with loan\_status = 0 have an average credit score of 630.09, while customers with loan\_status = 1 have an average credit score of 627.55. This indicates that, on average, customers without default (0) have slightly higher credit scores than those with default (1).

|  |
| --- |
| aggregate(credit\_score ~ loan\_status, data = df, FUN = function(x) round(mean(x, na.rm=TRUE), 2)) |

Output:



**Description:**

Here group by function used for find out all value such as mean, median, mode, max, min for each group. Here has 0 and 1 group. Summarise function use computing statistics like mean, median, sum, count etc.

**13.** **Compare spread in person\_emp\_exp for customers with different levels of**

**person\_education:**

**Description:**

In this task we are analyzing how the variable person\_emp\_exp (employment experience in years) is distributed across different categories of person\_education (such as High School, Bachelor, Master, etc.).

We created a custom function compare\_spread(), which groups the dataset by the specified categorical variable (person\_education) and then calculates descriptive spread measures for the numerical variable (person\_emp\_exp). Specifically, it computes:

* Count: Number of records in each education level.
* Mean: The average employment experience.
* Standard Deviation (SD): How much the experience varies within that group.
* Minimum and Maximum: The range of values observed.
* Interquartile Range (IQR): The spread of the middle 50% of the data.

By comparing these statistics, we can see whether people with higher education levels tend to have more or less work experience, and whether the variation (spread) in experience differs between groups. For example, graduates may have higher average experience but also larger variation compared to high school educated individuals.

**CODE:**

|  |
| --- |
| compare\_spread <- function(data, group\_col, value\_col) {  library(dplyr)    if (!group\_col %in% names(data)) stop("Group column not found.")  if (!value\_col %in% names(data)) stop("Value column not found.")    data %>%  group\_by(.data[[group\_col]]) %>%  summarise(  count = n(),  mean = mean(.data[[value\_col]], na.rm = TRUE),  sd = sd(.data[[value\_col]], na.rm = TRUE),  min = min(.data[[value\_col]], na.rm = TRUE),  max = max(.data[[value\_col]], na.rm = TRUE),  IQR = IQR(.data[[value\_col]], na.rm = TRUE),  .groups = "drop"  )  }  aggregate(credit\_score ~ loan\_status, data = data, mean, na.rm = TRUE) |

**Output:**

A screenshot of a computer code

AI-generated content may be incorrect.

|  |
| --- |
| compare <- compare\_spread(df\_clean, "person\_education", "person\_emp\_exp")  head(as.data.frame(compare)) |

**Output:**

A screenshot of a computer screen

AI-generated content may be incorrect.