American International University-Bangladesh (AIUB)

Faculty of Science & Technology (FST)

Department of Computer Science

Introduction to Data Science

Mid-Term Project Report

Summer 2024-2025

Section: F

Group: 02

SL #	Student Name	Student ID	Contribution (%)
1.	Md. Showkat Islam Sakib	22-49858-3	25%
2.	Zeba Tahsin Renaissance	22-49549-3	25%
3.	Aria Rahman	22-49494-3	25%
4.	Md. Sharjeel Hasan Khan	22-48915-3	25%

Dataset Description

This dataset contains 201 observations(rows) and 14 variables(columns) related to loan applications, combining both numerical and categorical features. It is a supervised learning dataset where the target variable, loan_status, indicates whether a loan was approved (1) or rejected (0).

The attributes cover applicant demographics, financial details, and loan-specific information. Key features include person_age, person_gender, person_income, loan_amnt, loan_intent, and credit_score. The dataset contains both numerical and categorical variables, some missing values, and a few extreme outliers (e.g., unusually high ages, incomes, and employment experience).

However, this suggests that data cleaning is necessary before analysis, making the dataset suitable for statistical or machine learning tasks. This dataset is well-suited for loan approval classification tasks**,** exploratory data analysis (EDA), and data preprocessing demonstrations, including handling missing data, encoding categorical variables, and detecting/removing outliers.

Project Implementation Details

1. Read the Excel File

Description of task:

The task is to load an Excel file named *DataSet.xlsx* into RStudio for analysis. This will be done using the readxl package, which allows Excel files to be imported directly into R. The file path will be specified to locate and import the dataset.

RStudio Code:

```
library(readxl)
mydata<-read_excel("C:\\Users\\User\\Documents\\Data Science\\DataSet.xlsx")
mydata
```

Output:

```
Console  Terminal ×  Background Jobs ×
R 4.5.1 · ~/Data Science/
> library(readxl)
> library(dplyr)
>
> mydata<-read_excel("C:\\Users\\User\\Documents\\Data Science\\DataSet.xlsx")
> mydata
# A tibble: 201 x 14
   person_age person_gender person_education person_income person_emp_exp person_home_ownership loan_amnt loan_intent
        <dbl> <chr>         <chr>                    <dbl>          <dbl> <chr>                     <dbl> <chr>
 1         21 female        Master                   71948              0 RENT                      35000 PERSONAL
 2         21 female        High School              12282              0 OWN                        1000 EDUCATION
 3         25 female        High School              12438              3 MORTGAGE                   5500 MEDICAL
 4         23 female        Bachelor                 79753              0 RENT                      35000 MEDICAL
 5         24 male          Master                   66135              1 RENTT                     35000 MEDICAL
 6         NA female        High School              12951              0 OWN                        2500 VENTURE
 7         22 female        Bachelor                    NA              1 RENT                      35000 EDUCATION
 8         24 NA            High School              95550              5 RENT                      35000 MEDICAL
 9         22 female        NA                      100684              3 RENT                      35000 PERSONAL
10         21 female        High School              12739              0 OWN                        1600 VENTURE
# ℹ 191 more rows
# ℹ 6 more variables: loan_int_rate <dbl>, loan_percent_income <dbl>, cb_person_cred_hist_length <dbl>,
#   credit_score <dbl>, previous_loan_defaults_on_file <chr>, loan_status <dbl>
# ℹ Use `print(n = ...)` to see more rows
```

Description of code:

First, library(readxl) is used to load the readxl package for reading Excel files. Then, read_excel() is called with the file path to import the dataset and store it in the mydata. Finally, typing mydata prints the data in the console so the user can confirm it has been loaded successfully.

2. Missing Value Detection

Description of task:

The task is to detect missing values in the dataset so they can be addressed before analysis or model building. Missing values can reduce data quality, cause calculation errors, and lead to inaccurate results if not handled properly. This step identifies which columns and rows contain NA values, allowing for appropriate handling strategies such as imputation or removal.

RStudio Code:

```
is.na(mydata)
colSums(is.na(mydata))

which(is.na(mydata$person_age))
which(is.na(mydata$person_gender))
which(is.na(mydata$person_education))
which(is.na(mydata$person_income))
which(is.na(mydata$loan_percent_income))
which(is.na(mydata$loan_status))
```

Output:

```
> is.na(mydata)
      person_age person_gender person_education person_income person_emp_exp person_home_ownership
 [1,]      FALSE         FALSE            FALSE         FALSE          FALSE                 FALSE
 [2,]      FALSE         FALSE            FALSE         FALSE          FALSE                 FALSE
 [3,]      FALSE         FALSE            FALSE         FALSE          FALSE                 FALSE
 [4,]      FALSE         FALSE            FALSE         FALSE          FALSE                 FALSE
 [5,]      FALSE         FALSE            FALSE         FALSE          FALSE                 FALSE
 [6,]       TRUE         FALSE            FALSE         FALSE          FALSE                 FALSE
 [7,]      FALSE         FALSE            FALSE          TRUE          FALSE                 FALSE
 [8,]      FALSE          TRUE            FALSE         FALSE          FALSE                 FALSE
 [9,]      FALSE         FALSE             TRUE         FALSE          FALSE                 FALSE
```

```
> colSums(is.na(mydata))
                    person_age                  person_gender               person_education
                             4                              4                              2
                 person_income                 person_emp_exp          person_home_ownership
                             4                              0                              0
                     loan_amnt                    loan_intent                  loan_int_rate
                             0                              0                              0
           loan_percent_income     cb_person_cred_hist_length                   credit_score
                             1                              0                              0
previous_loan_defaults_on_file                    loan_status
                             0                              3
```

```
> which(is.na(mydata$person_age))
[1]   6 14 28 35
> which(is.na(mydata$person_gender))
[1]   8  17 190 198
> which(is.na(mydata$person_education))
[1]   9 16
> which(is.na(mydata$person_income))
[1]   7 16 32 40
> which(is.na(mydata$loan_percent_income))
[1] 2
> which(is.na(mydata$loan_status))
[1]   9 15 18
```

Description of code:

is.na(mydata)

- Returns a logical matrix showing TRUE for every missing cell and FALSE for non-missing values.

colSums(is.na(mydata))

- Counts the total number of missing values in each column.

which(is.na(mydata$column_name))

- Finds the row positions where a specific column contains missing values.
- This is repeated for each column (person_age, person_gender, person_education, etc.) to pinpoint exactly where the missing data occurs.

3. Missing Value Handle

Description of task:

The task is to handle missing values in the dataset by replacing them with appropriate statistical estimates. Numerical missing values will be filled using the mean, while categorical missing values will be replaced with the mode. This ensures the dataset is complete and suitable for analysis without losing data integrity.

RStudio Code:

```
mean_age<- round(mean(mydata$person_age,na.rm =TRUE))
mean_age
mydata$person_age[is.na(mydata$person_age)] <- mean_age

mean_income<- round(mean(mydata$person_income,na.rm =TRUE))
mean_income
mydata$person_income[is.na(mydata$person_income)] <- mean_income

mean_loanpercentincome<-round(mean(mydata$loan_percent_income,na.rm =TRUE))
mean_loanpercentincome
mydata$loan_percent_income[is.na(mydata$loan_percent_income)] <-
mean_loanpercentincome

mean_loanstatus<-round(mean(mydata$loan_status,na.rm =TRUE))
mean_loanstatus
mydata$loan_status[is.na(mydata$loan_status)] <- mean_loanstatus

library(modeest)
mode_gender<- mfv(mydata$person_gender)
mode_gender
mydata$person_gender[is.na(mydata$person_gender)] <- mode_gender

mode_education<- mfv(mydata$person_education)
mode_education
mydata$person_education[is.na(mydata$person_education)] <- mode_education
```

Output:

```
> mean_age<- round(mean(mydata$person_age,na.rm =TRUE))
> mean_age
[1] 27
> mydata$person_age[is.na(mydata$person_age)] <- mean_age
>
> mean_income<- round(mean(mydata$person_income,na.rm =TRUE))
> mean_income
[1] 149875
> mydata$person_income[is.na(mydata$person_income)] <- mean_income
>
> mean_loanpercentincome<-round(mean(mydata$loan_percent_income,na.rm =TRUE))
> mean_loanpercentincome
[1] 0
> mydata$loan_percent_income[is.na(mydata$loan_percent_income)] <- mean_loanpercentincome
>
> mean_loanstatus<-round(mean(mydata$loan_status,na.rm =TRUE))
> mean_loanstatus
[1] 1
> mydata$loan_status[is.na(mydata$loan_status)] <- mean_loanstatus
```

```
> library(modeest)
> mode_gender<- mfv(mydata$person_gender)
> mode_gender
[1] "male"
> mydata$person_gender[is.na(mydata$person_gender)] <- mode_gender
> mode_education<- mfv(mydata$person_education)
> mode_education
[1] "Bachelor"
> mydata$person_education[is.na(mydata$person_education)] <- mode_education
```

Description of code:

The code calculates the mean of numeric columns like person_age , person_income, loan_percent_income and loan_status as these has missing values, then replaces the NAs in those columns with the rounded mean. For categorical columns such as person_gender and person_education, it uses the mfv() function from the modeest package to find the mode and replaces missing values with this most frequent category. Each update is applied directly to the dataset to impute missing data.

4. Noisy Value Handle

Description of task:

The task is to identify and correct noisy values in specific columns of the dataset mydata. This will be done by detecting typing error which doesn't fit as input in an specific column and replacing incorrect values with accurate ones.

RStudio Code:

```
Detect_NoisyValue <- levels(factor(mydata$person_home_ownership))
Detect_NoisyValue

mydata$person_home_ownership[which(mydata$person_home_ownership == "OOWN")] <-
"OWN"
mydata$person_home_ownership[which(mydata$person_home_ownership == "RENTT")] <-
"RENT"

library(dplyr)
mydata %>%
  select(person_home_ownership)
```

Output:

```
> Detect_NoisyValue <- levels(factor(mydata$person_home_ownership))
> Detect_NoisyValue
[1] "MORTGAGE" "OOWN"     "OTHER"    "OWN"      "RENT"     "RENTT"
> mydata$person_home_ownership[which(mydata$person_home_ownership == "OOWN")] <- "OWN"
> mydata$person_home_ownership[which(mydata$person_home_ownership == "RENTT")] <- "RENT"
> library(dplyr)
> mydata %>%
+   select(person_home_ownership)
# A tibble: 201 × 1
   person_home_ownership
   <chr>
 1 RENT
 2 OWN
 3 MORTGAGE
 4 RENT
 5 RENT
 6 OWN
 7 RENT
 8 RENT
```

Description of code:

The code uses factor () to get all unique values from the person_home_ownership column. By using which () function, it replaced the two noisy values “OWNN” & “RENTT” with correct values. To verify the data handling, select() is called to check the updated values for the specific column, for instance in row 5, the “RENTT” was a noisy value which is replaced by RENT.

5. Invalid Value Handle

Description of task:

The task is to handle an invalid value in the dataset by recalculating and updating the value for a specific row. Invalid datas can be detected easily as it’s not possible to exist. For instance, the age range is mostly possible within 100 and 0. So, the ages in person_age column which are above 100 and below 0, considered as invalid data.

RStudio Code:

```
library(dplyr)
invalid <- mydata$person_age < 0 | mydata$person_age > 100
invalid_ages <- mydata %>%
  filter(invalid)
invalid_ages
age_median <-median(mydata$person_age, na.rm = TRUE)
mydata$person_age[invalid] <- NA
age_median
mydata$person_age[is.na(mydata$person_age)] <- age_median
factor(mydata$person_age)
```

Output:

```
> invalid <- mydata$person_age < 0 | mydata$person_age > 100
> invalid_ages <- mydata %>%
+   filter(invalid)
> invalid_ages
# A tibble: 4 × 14
  person_age person_gender person_education person_income person_emp_exp person_home_ownership
       <dbl> <chr>         <chr>                    <dbl>          <dbl> <chr>
1        230 male          Bachelor                144855              1 RENT
2        350 male          Associate                15229              1 RENT
3        144 male          Bachelor                300616            125 RENT
4        144 male          Associate               241424            121 RENT
# ℹ 8 more variables: loan_amnt <dbl>, loan_intent <chr>, loan_int_rate <dbl>,
#   loan_percent_income <dbl>, cb_person_cred_hist_length <dbl>, credit_score <dbl>,
#   previous_loan_defaults_on_file <chr>, loan_status <dbl>
> age_median <-median(mydata$person_age, na.rm = TRUE)
> mydata$person_age[invalid] <- NA
> age_median
[1] 23
> mydata$person_age[is.na(mydata$person_age)] <- age_median

> factor(mydata$person_age)
  [1] 21 21 25 23 24 27 22 24 22 21 22 21 23 27 23 23 23 23 23 24 25 25 22 24 22 24 21 27 22 22 21
 [32] 21 23 26 27 26 21 22 24 25 23 26 24 26 22 26 26 25 26 22 26 24 23 23 23 25 26 26 23 25 22 21
 [63] 22 26 25 22 22 26 25 24 25 25 22 21 23 22 23 22 21 22 24 24 23 25 26 26 22 23 24 21 22 22 26
 [94] 22 24 23 22 22 22 25 24 26 25 26 25 23 22 22 21 24 25 24 26 25 26 24 24 23 22 26 25 22 25 21
[125] 22 22 23 26 22 23 25 24 23 22 22 22 23 24 23 25 26 23 22 23 26 22 26 23 23 21 22 22 26 22 24
[156] 23 23 24 26 24 23 23 22 24 26 23 24 24 21 25 23 23 21 25 22 23 21 24 22 25 24 24 22 22 23 25
[187] 22 25 24 22 21 26 23 22 24 25 22 26 26 25 22
Levels: 21 22 23 24 25 26 27
```

Description of code:

First, the code declares an attribute called invalid to store a condition for the person_age column. Filter () function selects all rows where the condition is TRUE and stored it in invalid_ages. Now, invalid_ages contains only rows with invalid ages. Then, median is calculated and na.rm = TRUE ensures that NA values are ignored in the calculation. Invalid values are now set as null values and then replaced with median_age. To ensure no invalid values exist in the person_age column, simply checked the unique values with filter() lastly.

6. Duplicate Value Detection and Handle

Description of task:

The task is to identify and remove duplicate rows from the dataset to maintain data quality and prevent redundancy in analysis. This will be done by detecting repeated records and then creating a cleaned version of the dataset without duplicates.

RStudio Code:

```
duplicated(mydata)
sum(duplicated(mydata))
which(duplicated(mydata))
fixed_mydata <- distinct(mydata)
fixed_mydata
```

Output:

```
> duplicated(mydata)
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [16] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
 [31] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [46] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> sum(duplicated(mydata))
[1] 1
> which(duplicated(mydata))
[1] 30
> fixed_mydata <- distinct(mydata)
> fixed_mydata
# A tibble: 200 × 14
   person_age person_gender person_education person_income person_emp_exp person_home_ownership
        <dbl> <chr>         <chr>                    <dbl>          <dbl> <chr>
 1         21 female        Master                   71948              0 RENT
 2         21 female        High School              12282              0 OWN
 3         25 female        High School              12438              3 MORTGAGE
 4         23 female        Bachelor                 79753              0 RENT
```

Description of code:

The code first uses duplicated(mydata) to check for duplicate rows, returning TRUE for any row that repeats a previous one. sum(duplicated(mydata)) counts the total number of such duplicates, while which(duplicated(mydata)) returns their row positions. Finally, distinct(mydata) from the **dplyr** package removes duplicate rows, keeping only the first occurrence, and stores the cleaned dataset in fixed_mydata, which is then displayed.

7. Outliers Detection

Description of task:

The task is to detect outliers in different numerical variables of the dataset. Outliers are extreme values that deviate significantly from most of the data and can distort analysis. This is done using the **Interquartile Range (IQR) method**, which defines outliers as values lying below the lower bound (Q1 − 1.5 × IQR) or above the upper bound (Q3 + 1.5 × IQR).

RStudio Code:

```
q1_exp <- quantile(fixed_mydata$person_emp_exp, 0.25, na.rm = TRUE)
q3_exp <- quantile(fixed_mydata$person_emp_exp, 0.75, na.rm = TRUE)
iqr_exp <- q3_exp - q1_exp
lower_bound_exp <- q1_exp - 1.5 * iqr_exp
upper_bound_exp <- q3_exp + 1.5 * iqr_exp
outliers_exp <- fixed_mydata$person_emp_exp[fixed_mydata$person_emp_exp <
                                   lower_bound_exp | fixed_mydata$person_emp_exp >
upper_bound_exp]
outliers_exp

q1_age <- quantile(fixed_mydata$person_age, 0.25, na.rm = TRUE)
q3_age <- quantile(fixed_mydata$person_age, 0.75, na.rm = TRUE)
iqr_age <- q3_age - q1_age
lower_bound_age <- q1_age - 1.5 * iqr_age
upper_bound_age <- q3_age + 1.5 * iqr_age
outliers_age <- fixed_mydata$person_age[fixed_mydata$person_age <lower_bound_age
                              | fixed_mydata$person_age > upper_bound_age]
outliers_age

q1_cre_score <- quantile(fixed_mydata$credit_score, 0.25, na.rm = TRUE)
q3_cre_score <- quantile(fixed_mydata$credit_score, 0.75, na.rm = TRUE)
iqr_cre_score <- q3_cre_score- q1_cre_score
lower_bound_cre_score <- q1_cre_score- 1.5 * iqr_cre_score
upper_bound_cre_score <- q3_cre_score + 1.5 * iqr_cre_score
outliers_cre_score <- fixed_mydata$credit_score[fixed_mydata$credit_score <
                                   lower_bound_cre_score | fixed_mydata$credit_score >
upper_bound_cre_score]
outliers_cre_score

q1_loan_amnt <- quantile(fixed_mydata$loan_amnt, 0.25, na.rm = TRUE)
q3_loan_amnt <- quantile(fixed_mydata$loan_amnt, 0.75, na.rm = TRUE)
iqr_loan_amnt <- q3_loan_amnt - q1_loan_amnt
lower_bound_loan_amnt <- q1_loan_amnt - 1.5 * iqr_loan_amnt
upper_bound_loan_amnt <- q3_loan_amnt+ 1.5 * iqr_loan_amnt
outliers_loan_amnt <- fixed_mydata$loan_amnt[fixed_mydata$loan_amnt <
                                   lower_bound_loan_amnt | fixed_mydata$loan_amnt >
upper_bound_loan_amnt]
outliers_loan_amnt
```

Output for column person_income:

```
> q1_exp <- quantile(fixed_mydata$person_emp_exp, 0.25, na.rm = TRUE)
> q3_exp <- quantile(fixed_mydata$person_emp_exp, 0.75, na.rm = TRUE)
> iqr_exp <- q3_exp - q1_exp
> lower_bound_exp <- q1_exp - 1.5 * iqr_exp
> upper_bound_exp <- q3_exp + 1.5 * iqr_exp
> outliers_exp <- fixed_mydata$person_emp_exp[fixed_mydata$person_emp_exp <
+                                               lower_bound_exp | fixed_mydata$person_emp_exp > upper_bound_exp]
> outliers_exp
[1] 125   8 121
> q1_age <- quantile(fixed_mydata$person_age, 0.25, na.rm = TRUE)
> q3_age <- quantile(fixed_mydata$person_age, 0.75, na.rm = TRUE)
> iqr_age <- q3_age - q1_age
> lower_bound_age <- q1_age - 1.5 * iqr_age
> upper_bound_age <- q3_age + 1.5 * iqr_age
> outliers_age <- fixed_mydata$person_age[fixed_mydata$person_age <lower_bound_age
+                                          | fixed_mydata$person_age > upper_bound_age]
> outliers_age
numeric(0)
> q1_cre_score <- quantile(fixed_mydata$credit_score, 0.25, na.rm = TRUE)
> q3_cre_score <- quantile(fixed_mydata$credit_score, 0.75, na.rm = TRUE)
> iqr_cre_score <- q3_cre_score- q1_cre_score
> lower_bound_cre_score <- q1_cre_score- 1.5 * iqr_cre_score
> upper_bound_cre_score <- q3_cre_score + 1.5 * iqr_cre_score
> outliers_cre_score <- fixed_mydata$credit_score[fixed_mydata$credit_score <
+                                                     lower_bound_cre_score | fixed_mydata$credit_score > upper_bound_cre_score]
> outliers_cre_score
[1] 789 484 807
> q1_loan_amnt <- quantile(fixed_mydata$loan_amnt, 0.25, na.rm = TRUE)
> q3_loan_amnt <- quantile(fixed_mydata$loan_amnt, 0.75, na.rm = TRUE)
> iqr_loan_amnt <- q3_loan_amnt - q1_loan_amnt
> lower_bound_loan_amnt <- q1_loan_amnt - 1.5 * iqr_loan_amnt
> upper_bound_loan_amnt <- q3_loan_amnt+ 1.5 * iqr_loan_amnt
> outliers_loan_amnt <- fixed_mydata$loan_amnt[fixed_mydata$loan_amnt <
+                                                 lower_bound_loan_amnt | fixed_mydata$loan_amnt > upper_bound_loan_amnt]
> outliers_loan_amnt
numeric(0)
```

Description of code:

The code calculates the first quartile (Q1), third quartile (Q3), and Interquartile Range (IQR) for each selected numeric column using the quantile() function with na.rm = TRUE to ignore missing values.
Lower bound is computed as Q1 − 1.5 × IQR.
Upper bound is computed as Q3 + 1.5 × IQR.

Any values less than the lower bound or greater than the upper bound are extracted as outliers and stored in separate variables (outliers_age, outliers_loan_amnt, etc.). This process is repeated individually to multiple numeric columns, including person_age, person_emp_exp, loan_amnt and credit_score, ensuring comprehensive outlier detection across the dataset.

OUTLIER DETECTION with summary() for column (person_income)

```
summary(fixed_mydata$person_income)
s <- summary(fixed_mydata$person_income)

Q1 <- as.numeric(s["1st Qu."])
Q3 <- as.numeric(s["3rd Qu."])
IQR_val <- Q3 - Q1
lower_bound <- Q1 - 1.5 * IQR_val
upper_bound <- Q3 + 1.5 * IQR_val

outlier_values <- fixed_mydata$person_income[
        fixed_mydata$person_income < lower_bound | fixed_mydata$person_income >
upper_bound]
outlier_values
```

Output for column person_income:

```
> summary(fixed_mydata$person_income)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  12282   60686   89012  150229  241051 3138998 
> s <- summary(fixed_mydata$person_income)
> Q1 <- as.numeric(s["1st Qu."])
> Q3 <- as.numeric(s["3rd Qu."])
> IQR_val <- Q3 - Q1
> lower_bound <- Q1 - 1.5 * IQR_val
> upper_bound <- Q3 + 1.5 * IQR_val
> outlier_values <- fixed_mydata$person_income[
+            fixed_mydata$person_income < lower_bound | fixed_mydata$person_income > upper_bound]
> outlier_values
[1] 3138998
```

Description of code:

The code utilized the summary () function to get descriptive statistics for column person_income. Then, stored it in s variable, also extracts Q1 and Q3 and converted them to a numeric value. After that, calculated Interquartile Range (IQR) for selected numeric column where,
Lower bound is computed as Q1 − 1.5 × IQR_val.
Upper bound is computed as Q3 + 1.5 × IQR_val.

Any values less than the lower bound or greater than the upper bound are extracted as outliers and stored in separate variable called outliers_values , ensuring comprehensive outlier detection across the dataset.

8. Outliers Handle

Description of task:

The task is to handle outliers in the dataset to minimize their negative impact on analysis. Instead of removing these extreme values, the approach involves capping them at calculated boundary values for certain variables and normalizing others. This preserves the dataset's integrity while reducing the influence of extreme data points.

RStudio Code:

```
fixed_mydata$person_emp_exp <- ifelse(
  fixed_mydata$person_emp_exp < lower_bound_exp, round(lower_bound_exp),
  ifelse(fixed_mydata$person_emp_exp > upper_bound_exp, round(upper_bound_exp),
         fixed_mydata$person_emp_exp)
)
fixed_mydata$credit_score <- ifelse(

  fixed_mydata$credit_score < lower_bound_cre_score, round(lower_bound_cre_score),

  ifelse(fixed_mydata$credit_score > upper_bound_cre_score,
round(upper_bound_cre_score),

         fixed_mydata$credit_score)
)
fixed_mydata$person_income <- ifelse(

  fixed_mydata$person_income < lower_bound, round(lower_bound),

  ifelse(fixed_mydata$person_income > upper_bound, round(upper_bound),

         fixed_mydata$person_income)
)
fixed_mydata$person_emp_exp
fixed_mydata$credit_score
fixed_mydata$person_income
```

Output:

```
> fixed_mydata$person_emp_exp
 [1] 0 0 3 0 1 0 1 5 3 0 0 0 3 0 0 5 0 0 0 1 0 4 0 0 1 0 0 0 1 0 0 1 2 3 1 0 0 3 4 0 5 0 5 0 5 5 2 2 0
[50] 2 2 1 3 0 3 7 2 1 1 1 0 0 6 0 0 0 3 0 5 0 1 0 0 0 4 1 0 0 3 1 2 8 1 1 0 0 0 3 0 0 2 4 2 3 1 0 1 3

> fixed_mydata$credit_score
 [1] 561 504 635 675 586 532 701 585 544 640 621 651 573 708 583 670 663 694 709 679 684 662 691 600
[25] 691 654 626 607 700 553 589 586 681 567 669 600 606 582 649 602 616 631 684 637 649 695 620 622
[49] 645 654 624 570 648 652 559 623 602 609 573 631 623 602 579 582 700 688 607 661 562 664 564 598
[73] 557 677 690 599 604 663 601 634 671 768 622 538 587 683 634 690 518 583 617 668 673 706 536 557

> fixed_mydata$person_income
 [1]  71948  12282  12438  79753  66135  12951 149875  95550 100684  12739 102985  13113 114860
[14] 130713 511599 149875 144943 111369 136628  14283 195718 165792  79255  13866  97420  82443
[27]  14288  14293  79054  14988 149875 144855 114645 368115 361076  15150  58868  78026 149875
[40]  86811  75503  15082 361293 361547 360680 360977 361244  98230  80838 107957  94649  94550
[53] 111153 117250 144985 337133 333566 333399 154793  15229 158338 331034 316466 267671  85191
```

Description of code:

Capping for person_emp_exp, credit_score, person_income

- Uses ifelse() to replace any outlier value lower than the lower_bound with the rounded lower bound and any value higher than the upper_bound with the rounded upper bound.
- Values within the range remain unchanged.

9. Normalization

Description of task:

Normalization for person_income

- Applies min–max normalization to scale all credit scores between 0 and 1.
- Formula:

$$\text{Normalized value} = \frac{\text{value} - \text{min}}{\text{max} - \text{min}}$$

- Result is rounded to 2 decimal places to keep it concise.

RStudio Code:

```
normalizeddata_income <- fixed_mydata
normalizeddata_income$person_income<- round((normalizeddata_income$person_income -
                                min(normalizeddata_income$person_income, na.rm = TRUE)) /
                               (max(normalizeddata_income$person_income, na.rm = TRUE) -
                                min(normalizeddata_income$person_income, na.rm = TRUE)),2)
normalizeddata_income$person_income
```

Output:

```
> normalizeddata_income$person_income
  [1] 0.02 0.00 0.00 0.02 0.02 0.00 0.04 0.03 0.03 0.00 0.03 0.00 0.03 0.04 1.00 0.04 0.04 0.03 0.04
 [20] 0.00 0.06 0.05 0.02 0.00 0.03 0.02 0.00 0.00 0.02 0.00 0.04 0.04 0.03 0.11 0.11 0.00 0.01 0.02
 [39] 0.04 0.02 0.02 0.00 0.11 0.11 0.11 0.11 0.11 0.03 0.02 0.03 0.03 0.03 0.03 0.03 0.04 0.10 0.10
 [58] 0.10 0.05 0.00 0.05 0.10 0.10 0.08 0.02 0.00 0.10 0.10 0.09 0.09 0.00 0.02 0.00 0.02 0.03 0.00
 [77] 0.03 0.00 0.00 0.03 0.09 0.09 0.03 0.04 0.00 0.04 0.05 0.02 0.00 0.02 0.09 0.02 0.03 0.02 0.02
```

Description of code:

- created a copy of fixed_mydata into normalizeddata_income.
- Normalization formula were applied to the dataset.
- rounded the result to 2 decimal places for cleaner representation.

10. Data Types and Conversion (numerical to categorical)

Description of task:

The task is to convert the data type of a variable in the dataset to a more suitable format for analysis. Specifically, a numeric variable indicating previous loan defaults is converted into a categorical factor with clearly defined levels ("No" and "Yes"), making the data more interpretable for analysis and visualization while preserving its categorical nature.

RStudio Code:

```
fixed_mydata$loan_status <- factor(

  fixed_mydata$loan_status,

  levels = c(0, 1),

  labels = c("No", "Yes")
)
fixed_mydata$loan_status
```

Output:

```
> fixed_mydata$loan_status
  [1] Yes No  Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes No  No  Yes Yes Yes No  No  Yes No
 [25] Yes Yes Yes No  Yes Yes Yes No  Yes No  No  Yes Yes Yes Yes Yes Yes Yes No  No  No  No  No  Yes
 [49] Yes Yes Yes Yes Yes Yes No  No  No  No  No  No  No  No  Yes Yes Yes No  No  No  No  No  Yes Yes
 [73] Yes Yes Yes Yes Yes Yes Yes Yes No  No  No  No  Yes Yes No  Yes Yes Yes No  Yes Yes Yes Yes No
 [97] No  Yes No  No  No  Yes No  No  Yes Yes No  Yes Yes Yes Yes No  No  Yes No  Yes Yes Yes Yes Yes
```

Description of code:

The code uses the factor() function to recode the loan_status variable in the distinct_mydata dataset. The levels argument specifies the original numeric values (0 for no default, 1 for default), and the labels argument assigns meaningful category names ("No" for 0, "Yes" for 1). After conversion, the variable is stored back into the dataset, replacing the original numeric representation with the factor version, which is easier to interpret and suitable for categorical analysis.

11. Data Types and Conversion (categorical to numerical)

Description of task:

The task is to convert the data type of a variable in the dataset to a more suitable format for analysis. Specifically, a numeric variable indicating previous loan defaults is converted into a categorical factor with clearly defined levels (“No” and “Yes”), making the data more interpretable for analysis and visualization while preserving its categorical nature.

RStudio Code:

```
fixed_mydata$previous_loan_defaults_on_file <- ifelse(

  fixed_mydata$previous_loan_defaults_on_file == "Yes", 1, 0

)
fixed_mydata$previous_loan_defaults_on_file
```

Output:

```
> fixed_mydata$previous_loan_defaults_on_file
  [1] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0
 [50] 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0
 [99] 1 1 1 0 1 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0
```

Description of code:

This code transforms the previous_loan_defaults_on_file column from a text-based categorical variable into a numeric binary variable by using ifelse(): each value is checked, and if it is "Yes", it is replaced with 1; otherwise (including "No" or any other value), it is replaced with 0. After execution, the column contains only 0s and 1s, making it suitable for numerical analysis or machine learning models.

12. Filter the Data

Description of task:

The task is to filter the dataset to include only rows where a specific condition is met. In this case, the dataset is filtered to retain only the records where person_education equals Master and loan_status equals to yes. This helps focus the analysis on a particular subset of interest.

RStudio Code:

```
filtered_data <- filter(fixed_mydata, person_education == "Master")
head(filtered_data)

filtered_data <- filter(fixed_mydata, loan_status == "Yes")
filtered_data[, c("person_age", "person_gender", "loan_status")]
```

Output:

```
> head(filtered_data)
# A tibble: 6 × 14
  person_age person_gender person_education person_income person_emp_exp person_home_ownership
       <dbl> <chr>         <chr>                    <dbl>          <dbl> <chr>
1         21 female        Master                   71948              0 RENT
2         24 male          Master                   66135              1 RENT
3         27 male          Master                  130713              0 RENT
4         24 female        Master                   14283              1 MORTGAGE
5         22 female        Master                   79255              0 RENT
6         26 male          Master                  360680              5 RENT
```

```
> filtered_data <- filter(fixed_mydata, loan_status == "Yes")
> filtered_data[, c("person_age", "person_gender", "loan_status")]
# A tibble: 124 × 3
   person_age person_gender loan_status
        <dbl> <chr>         <fct>
 1         21 female        Yes
 2         25 female        Yes
 3         23 female        Yes
 4         24 male          Yes
 5         27 female        Yes
 6         22 female        Yes
```

Description of code:

The code uses the filter() function from the dplyr package to select rows from fixed_mydata where person_education is equal to Master. The filtered subset is stored in filtered_data. Finally, head() is used to display the filtered data for verification. Then, this code first filters fixed_mydata to include only rows where loan_status is "Yes". Then, it selects and displays only the columns person_age, person_gender, and loan_status from the filtered data.

13. Convert the Imbalanced Dataset into a Balanced Dataset

Description of task:

This task focuses on detecting and addressing class imbalance in the loan_status variable of the fixed_mydata dataset. Class imbalance occurs when one category is significantly more frequent than another, which can bias predictive models. To improve balance, the dataset is augmented by adding 200 new observations, where the target variable is loan_status.The updated class distribution is then evaluated to assess the impact of the augmentation.

RStudio Code:

```
str(fixed_mydata)
imbalanced_data <- fixed_mydata %>%
  mutate(across(where(is.character), as.factor))
str(imbalanced_data)

library(ROSE)
set.seed(199)


table(imbalanced_data$loan_status)

balanced_data <- ROSE(loan_status ~ .,
                data = imbalanced_data,
                N = 400,
                p = 0.5)$data

table(balanced_data$loan_status)
balanced_data
```

Output:

OUTPUT 1: dataset structure

```
> str(fixed_mydata)
tibble [200 x 14] (S3: tbl_df/tbl/data.frame)
 $ person_age                    : num [1:200] 21 21 25 23 24 27 22 24 22 21 ...
 $ person_gender                 : chr [1:200] "female" "female" "female" "female" ...
 $ person_education              : chr [1:200] "Master" "High School" "High School" "Bachelor" ...
 $ person_income                 : num [1:200] 71948 12282 12438 79753 66135 ...
 $ person_emp_exp                : num [1:200] 0 0 3 0 1 0 1 5 3 0 ...
 $ person_home_ownership         : chr [1:200] "RENT" "OWN" "MORTGAGE" "RENT" ...
 $ loan_amnt                     : num [1:200] 35000 1000 5500 35000 35000 2500 35000 35000 35000 1600
...
 $ loan_intent                   : chr [1:200] "PERSONAL" "EDUCATION" "MEDICAL" "MEDICAL" ...
 $ loan_int_rate                 : num [1:200] 16 11.1 12.9 15.2 14.3 ...
 $ loan_percent_income           : num [1:200] 0.49 0 0 0.44 0.53 0.19 0.37 0.37 0.35 0.13 ...
 $ cb_person_cred_hist_length    : num [1:200] 3 2 3 2 4 2 3 4 2 3 ...
 $ credit_score                  : num [1:200] 561 504 635 675 586 532 701 585 544 640 ...
 $ previous_loan_defaults_on_file: num [1:200] 0 1 0 0 0 0 0 0 0 0 ...
 $ loan_status                   : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 2 2 2 2 2 ...
```

OUTPUT 2: Conversion of columns (character to factors)

```
> str(imbalanced_data)
tibble [200 x 14] (S3: tbl_df/tbl/data.frame)
 $ person_age                    : num [1:200] 21 21 25 23 24 27 22 24 22 21 ...
 $ person_gender                 : Factor w/ 2 levels "female","male": 1 1 1 1 2 1 1 2 1 1 ...
 $ person_education              : Factor w/ 5 levels "Associate","Bachelor",..: 5 4 4 2 5 4 2 4 2 4 ..
 $ person_income                 : num [1:200] 71948 12282 12438 79753 66135 ...
 $ person_emp_exp                : num [1:200] 0 0 3 0 1 0 1 5 3 0 ...
 $ person_home_ownership         : Factor w/ 4 levels "MORTGAGE","OTHER",..: 4 3 1 4 4 3 4 4 4 3 ...
 $ loan_amnt                     : num [1:200] 35000 1000 5500 35000 35000 2500 35000 35000 35000 1600
...
 $ loan_intent                   : Factor w/ 6 levels "DEBTCONSOLIDATION",..: 5 2 4 4 4 6 2 4 5 6 ...
 $ loan_int_rate                 : num [1:200] 16 11.1 12.9 15.2 14.3 ...
 $ loan_percent_income           : num [1:200] 0.49 0 0 0.44 0.53 0.19 0.37 0.37 0.35 0.13 ...
 $ cb_person_cred_hist_length    : num [1:200] 3 2 3 2 4 2 3 4 2 3 ...
 $ credit_score                  : num [1:200] 561 504 635 675 586 532 701 585 544 640 ...
 $ previous_loan_defaults_on_file: num [1:200] 0 1 0 0 0 0 0 0 0 0 ...
 $ loan_status                   : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 2 2 2 2 2 ...
```

OUTPUT 3: implementation of SMOTE to balance the dataset

```
> library(ROSE)
> set.seed(199)
> table(imbalanced_data$loan_status)

 No Yes
 76 124
> balanced_data <- ROSE(loan_status ~ .,
+                    data = imbalanced_data,
+                    N = 400,
+                    p = 0.5)$data
> # Check new class distribution
> table(balanced_data$loan_status)

Yes  No
213 187
```

OUTPUT 4: Displaying the balanced dataset

```
> balanced_data
  person_age person_gender person_education person_income person_emp_exp person_home_ownership
1   23.22828          male         Bachelor    148875.694     1.85062645                  RENT
2   22.25557        female      High School     33127.059    -0.30740362                  RENT
3   21.35975          male        Associate     -7950.102    -0.07010960                  RENT
4   23.69638          male      High School     93201.063     3.93568825                  RENT
5   26.36838        female      High School    306235.345     5.45659786                  RENT
6   27.46571          male        Associate    286847.429     2.21136790                  RENT
```

<u>Description of code:</u>

For OUTPUT 1:

- str(fixed_mydata) helped to identify which columns are numeric, character, or factor.

For OUTPUT 2:

- Converted all character columns in fixed_mydata to factor type.
- Prepared the dataset for ROSE, which requires categorical variables to be factors.
- Stored the dataset in imbalanced_data.
- verfied the dataset again to ensure that character columns are now factors.

For OUTPUT 3:

- Loaded the ROSE package for generating synthetic data to handle imbalanced datasets.
- Ensured reproducibility by setting a random seed.
- table(imbalanced_data$loan_status) displayed the original distribution of the target variable loan_status which revealed that the dataset is imbalanced.
- applied SMOTE to create synthetic data through ROSE function and specified the target column with loan_status ~ . and double the rows number as 400 samples with 50/50 portion (p=0.5).
- now balanced_data contains original + synthetic data points.
- after completeing the oversampling, showed a more equal class distribution with table(balanced_data$loan_status).

For OUTPUT 4:

- Lastly, displayed the balanced dataset.

14. Splitting the Dataset for Training and Testing

Description of task:

The task of splitting the dataset for training and testing involves dividing the balanced dataset into two subsets: a training set, typically 70% of the data, used to train the machine learning model, and a testing set, the remaining 30%, used to evaluate the model's performance on unseen data. This random split ensures that the model learns patterns from the majority and minority classes while providing a separate set to measure how well it generalizes. Checking the class distribution in each subset helps confirm that both classes are represented, ensuring reliable training and evaluation.

RStudio Code:

```
set.seed(199)
index <- sample(1:nrow(balanced_data), 0.7 * nrow(balanced_data))
train_data <- balanced_data[index, ]
test_data  <- balanced_data[-index, ]

train_data
test_data
table(train_data$loan_status)
table(test_data$loan_status)
```

Output:

OUTPUT 1: train_data

```
> train_data
    person_age person_gender person_education person_income person_emp_exp person_home_ownership
307         23        female           Master   191350.8609    2.264950319                  RENT
39          21        female         Bachelor    27267.5324   -1.307966898                  RENT
362         23          male      High School   217220.6096    1.870087984                  RENT
67          23        female      High School   137840.0460    1.587705135                  RENT
125         22        female        Associate   474086.9795    0.150011085                  RENT
223         22        female         Bachelor    35001.6995    0.331972386                   OWN
```

OUTPUT 2: test_data

```
> test_data
   person_age person_gender person_education person_income person_emp_exp person_home_ownership
1          23          male         Bachelor    148875.694    1.850626455                  RENT
4          24          male      High School     93201.063    3.935688253                  RENT
6          27          male        Associate    286847.429    2.211367902                  RENT
10         21        female         Bachelor    -24462.255   -2.758282624                  RENT
11         23          male         Bachelor    130647.803    3.460120457                  RENT
```

OUTPUT 3: Distribution of train_data & test_data

```
> table(train_data$loan_status)

Yes  No 
139 141 
> table(test_data$loan_status)

Yes  No 
 74  46 
```

Description of code:

For OUTPUT 1:

- Contained **70% of the rows** from balanced_data selected randomly using sample().
- This subset is used to **train models**, allowing it to learn patterns from both the majority and minority classes.
- All features and the target variable (loan_status) are included.

For OUTPUT 2:

- Contained the remaining 30% of rows from balanced_data not selected for training.
- This subset is kept aside and used to evaluate the model's performance on unseen data.
- It helped measure generalization, avoiding overfitting to the training data.

For OUTPUT 3:

- Showed the class distribution in the training and testing sets.
- Ensured that both classes are present in each set, which is crucial for proper model training and evaluation.

15. Descriptive Statistics: Central tendencies (mean, median, mode)

Description of task:

The task is to calculate measures of central tendency for various variables in the dataset. These statistics summarize key aspects of the data distribution, including the mean, median, and mode, which provide insights into the typical or most common values within the dataset.

RStudio Code:

```
mean_loan_int_rate   <- mean(fixed_mydata$loan_int_rate)
median_loan_int_rate <- median(fixed_mydata$loan_int_rate)
mode_loan_int_rate   <- mfv(fixed_mydata$loan_int_rate)
mean_loan_percent_income   <- mean(fixed_mydata$loan_percent_income)
median_loan_percent_income <- median(fixed_mydata$loan_percent_income)
mode_loan_percent_income   <- mfv(fixed_mydata$loan_percent_income)
mode_home_ownership <- mfv(fixed_mydata$person_home_ownership)
mode_loan_intent <- mfv(fixed_mydata$loan_intent)
```

Output:

OUTPUT 1: Calculated Central Tendency for Numerical Column

```
> mean_loan_int_rate
[1] 12.3012
> median_loan_int_rate
[1] 11.845
> mode_loan_int_rate
[1] 11.01
```

```
> mean_loan_percent_income
[1] 0.2273
> median_loan_percent_income
[1] 0.23
> mode_loan_percent_income
[1] 0.34
```

OUTPUT 2: Calculated Central Tendency for Categorical Column

```
> mode_home_ownership
[1] "RENT"
> mode_loan_intent
[1] "EDUCATION"
```

Description of code:

For OUTPUT 1:

Loan Interest Rate:

- Mean = 12.3012, means on average the loan interest rate is 12.3%.
- Median = 11.845 means half of the loans have an interest rate below 11.85%, and half above.
- Mode = 11.01 means the most common interest rate in the dataset is 11.01%.

Loan Percent Income:

- Mean = 0.2273 (22.7%), meanson average people spend about 22.7% of their income on loan repayment.
- Median = 0.23 (23%), means half of people spend less than 23% and half spend more.
- Mode = 0.34 (34%), meansthe most common repayment ratio is 34% of income.
-

For OUTPUT 2:

. Person Home Ownership

- Mode = RENT, means most people in the dataset live in rented houses.

Loan Intent

- Mode = EDUCATION, means the most common purpose for taking loans is education.

16. Descriptive Statistics: Measure of Spread

Description of task:

The task is to calculate various measures of spread including range, interquartile range (IQR), variance, and standard deviation for different numeric variables in the dataset. These measures help to understand the dispersion and variability of the data, which is essential for comprehensive statistical analysis.

RStudio Code:

```
range_income <- range(fixed_mydata$person_income)
iqr_income   <- IQR(fixed_mydata$person_income)
var_income   <- var(fixed_mydata$person_income)
sd_income    <- sd(fixed_mydata$person_income)

range_loan   <- range(fixed_mydata$loan_amnt)
iqr_loan     <- IQR(fixed_mydata$loan_amnt)
var_loan     <- var(fixed_mydata$loan_amnt)
sd_loan      <- sd(fixed_mydata$loan_amnt)
```

Output:

person_income

```
> range_income
[1]  12282 511599
> iqr_income
[1] 180365.5
> var_income
[1] 11812133282
> sd_income
[1] 108683.6
```

loan_amnt

```
> range_loan
[1]  1000 35000
> iqr_loan
[1] 18000
> var_loan
[1] 115363387
> sd_loan
[1] 10740.73
```

Description of code:

Person Income

- Range = 12282 to 511599, means Lowest income is 12,282, highest 511,599. It seems there is a huge gap in between.
- IQR = 180365.5, means middle 50% of incomes fall within a spread of 180,365.
- Variance = 11812133282, which means there is very high variability in incomes.
- Standard Deviation = 108683.6, which means on average, each income differs from the mean by about 108,684.

Loan Amount

- Range = 1000 to 35000, which means the smallest loan is 1,000 and the largest is 35,000.
- IQR = 18000, which means the middle 50% of loans fall within a spread of 18,000.
- Variance = 115363387, which means there is very high variability in loan amounts.
- Standard Deviation = 10740.73, which means on average, loan amounts differ from the mean by about 10,741.

PROJECT CODE

```
library(readxl)

library(dplyr)

library(modeest)

library(ROSE)

mydata<-read_excel("D:/9TH SEMESTER/Data Science/MID PROJECT/IDS Sec-F Midterm Summer 24-25 Loan Approval Classification Dataset - modified.xlsx")
mydata

is.na(mydata)
colSums(is.na(mydata))
which(is.na(mydata$person_age))

which(is.na(mydata$person_gender))

which(is.na(mydata$person_education))

which(is.na(mydata$person_income))

which(is.na(mydata$loan_percent_income))

which(is.na(mydata$loan_status))

mean_age<- round(mean(mydata$person_age,na.rm =TRUE))
mydata$person_age[is.na(mydata$person_age)] <- mean_age

mean_income<- round(mean(mydata$person_income,na.rm =TRUE))

mydata$person_income[is.na(mydata$person_income)] <- mean_income

mean_loanpercentincome<-round(mean(mydata$loan_percent_income,na.rm =TRUE))

mydata$loan_percent_income[is.na(mydata$loan_percent_income)]<-
mean_loanpercentincome

mean_loanstatus<-round(mean(mydata$loan_status,na.rm =TRUE))

mydata$loan_status[is.na(mydata$loan_status)] <- mean_loanstatus

mode_gender<- mfv(mydata$person_gender)

mydata$person_gender[is.na(mydata$person_gender)] <- mode_gender

mode_education<- mfv(mydata$person_education)

mydata$person_education[is.na(mydata$person_education)] <- mode_education
```

```
Detect_NoisyValue <- levels(factor(mydata$person_home_ownership))

Detect_NoisyValue

mydata$person_home_ownership[which(mydata$person_home_ownership == "OOWN")] <-
"OWN"

mydata$person_home_ownership[which(mydata$person_home_ownership == "RENTT")] <-
"RENT"

mydata %>%

  select(person_home_ownership)

invalid <- mydata$person_age < 0 | mydata$person_age > 100

invalid_ages <- mydata %>%

  filter(invalid)

invalid_ages

age_median <-median(mydata$person_age, na.rm = TRUE)

mydata$person_age[invalid] <- NA

mydata$person_age[is.na(mydata$person_age)] <- age_median

factor(mydata$person_age)


duplicated(mydata)

sum(duplicated(mydata))

which(duplicated(mydata))

fixed_mydata <- distinct(mydata)

fixed_mydata

which(duplicated(fixed_mydata))

duplicated(fixed_mydata)


summary(fixed_mydata$person_income)

s <- summary(fixed_mydata$person_income)

Q1 <- as.numeric(s["1st Qu."])

Q3 <- as.numeric(s["3rd Qu."])

IQR_val <- Q3 - Q1
```

```
lower_bound <- Q1 - 1.5 * IQR_val
upper_bound <- Q3 + 1.5 * IQR_val
outlier_values <- fixed_mydata$person_income[
    fixed_mydata$person_income < lower_bound | fixed_mydata$person_income >
upper_bound]
outlier_values

q1_exp <- quantile(fixed_mydata$person_emp_exp, 0.25, na.rm = TRUE)
q3_exp <- quantile(fixed_mydata$person_emp_exp, 0.75, na.rm = TRUE)
iqr_exp <- q3_exp - q1_exp
lower_bound_exp <- q1_exp - 1.5 * iqr_exp
upper_bound_exp <- q3_exp + 1.5 * iqr_exp
outliers_exp <- fixed_mydata$person_emp_exp[fixed_mydata$person_emp_exp <
                    lower_bound_exp | fixed_mydata$person_emp_exp >
upper_bound_exp]
outliers_exp
q1_age <- quantile(fixed_mydata$person_age, 0.25, na.rm = TRUE)
q3_age <- quantile(fixed_mydata$person_age, 0.75, na.rm = TRUE)
iqr_age <- q3_age - q1_age
lower_bound_age <- q1_age - 1.5 * iqr_age
upper_bound_age <- q3_age + 1.5 * iqr_age
outliers_age <- fixed_mydata$person_age[fixed_mydata$person_age <lower_bound_age
                    | fixed_mydata$person_age > upper_bound_age]
q1_cre_score <- quantile(fixed_mydata$credit_score, 0.25, na.rm = TRUE)
q3_cre_score <- quantile(fixed_mydata$credit_score, 0.75, na.rm = TRUE)
iqr_cre_score <- q3_cre_score- q1_cre_score
lower_bound_cre_score <- q1_cre_score- 1.5 * iqr_cre_score
upper_bound_cre_score <- q3_cre_score + 1.5 * iqr_cre_score
outliers_cre_score <- fixed_mydata$credit_score[fixed_mydata$credit_score <
                    lower_bound_cre_score | fixed_mydata$credit_score >
upper_bound_cre_score]
outliers_cre_score
```

```
q1_loan_amnt <- quantile(fixed_mydata$loan_amnt, 0.25, na.rm = TRUE)
q3_loan_amnt <- quantile(fixed_mydata$loan_amnt, 0.75, na.rm = TRUE)
iqr_loan_amnt <- q3_loan_amnt - q1_loan_amnt
lower_bound_loan_amnt <- q1_loan_amnt - 1.5 * iqr_loan_amnt
upper_bound_loan_amnt <- q3_loan_amnt+ 1.5 * iqr_loan_amnt
outliers_loan_amnt <- fixed_mydata$loan_amnt[fixed_mydata$loan_amnt <
                                lower_bound_loan_amnt | fixed_mydata$loan_amnt > upper_bound_loan_amnt]
outliers_loan_amnt


fixed_mydata$person_emp_exp <- ifelse(
  fixed_mydata$person_emp_exp < lower_bound_exp, round(lower_bound_exp),
  ifelse(fixed_mydata$person_emp_exp > upper_bound_exp, round(upper_bound_exp),
         fixed_mydata$person_emp_exp)
)
fixed_mydata$credit_score <- ifelse(
  fixed_mydata$credit_score < lower_bound_cre_score, round(lower_bound_cre_score),
  ifelse(fixed_mydata$credit_score > upper_bound_cre_score, round(upper_bound_cre_score),
         fixed_mydata$credit_score)
)
fixed_mydata$person_income <- ifelse(
  fixed_mydata$person_income < lower_bound, round(lower_bound),
  ifelse(fixed_mydata$person_income > upper_bound, round(upper_bound),
         fixed_mydata$person_income)
)
fixed_mydata$person_emp_exp
fixed_mydata$credit_score
fixed_mydata$person_income


normalizeddata_income <- fixed_mydata
```

```
normalizeddata_income$person_income<- round((normalizeddata_income$person_income -
                              min(normalizeddata_income$person_income, na.rm = TRUE)) /
                             (max(normalizeddata_income$person_income, na.rm = TRUE) -
                              min(normalizeddata_income$person_income, na.rm = TRUE)),2)
normalizeddata_income$person_income


fixed_mydata$loan_status <- factor(
  fixed_mydata$loan_status,
  levels = c(0, 1), labels = c("No", "Yes") )
fixed_mydata$loan_status
fixed_mydata$previous_loan_defaults_on_file <- ifelse(
  fixed_mydata$previous_loan_defaults_on_file == "Yes", 1, 0)
fixed_mydata$previous_loan_defaults_on_file
filtered_data <- filter(fixed_mydata, person_education == "Master")
head(filtered_data)
filtered_data <- filter(fixed_mydata, loan_status == "Yes")
filtered_data[, c("person_age", "person_gender", "loan_status")]
str(fixed_mydata)
imbalanced_data <- fixed_mydata %>%
  mutate(across(where(is.character), as.factor))
str(imbalanced_data)
set.seed(199)
table(imbalanced_data$loan_status)
balanced_data <- ROSE(loan_status ~ .,
                  data = imbalanced_data,
                  N = 400,
                  p = 0.5)$data
table(balanced_data$loan_status)
balanced_data$person_age <- round(balanced_data$person_age)
```

```
set.seed(199)
index <- sample(1:nrow(balanced_data), 0.7 * nrow(balanced_data))
train_data <- balanced_data[index, ]
test_data  <- balanced_data[-index, ]
train_data
test_data
table(train_data$loan_status)
table(test_data$loan_status)

mean_loan_int_rate   <- mean(fixed_mydata$loan_int_rate)
median_loan_int_rate <- median(fixed_mydata$loan_int_rate)
mode_loan_int_rate   <- mfv(fixed_mydata$loan_int_rate)
mean_loan_percent_income   <- mean(fixed_mydata$loan_percent_income)
median_loan_percent_income <- median(fixed_mydata$loan_percent_income)
mode_loan_percent_income   <- mfv(fixed_mydata$loan_percent_income)
mode_home_ownership <- mfv(fixed_mydata$person_home_ownership)
mode_loan_intent <- mfv(fixed_mydata$loan_intent)

range_income <- range(fixed_mydata$person_income)
iqr_income   <- IQR(fixed_mydata$person_income)
var_income   <- var(fixed_mydata$person_income)
sd_income    <- sd(fixed_mydata$person_income)
range_loan   <- range(fixed_mydata$loan_amnt)
iqr_loan     <- IQR(fixed_mydata$loan_amnt)
var_loan     <- var(fixed_mydata$loan_amnt)
sd_loan      <- sd(fixed_mydata$loan_amnt)
```