TSAR Project Assignment Part 2

Data Wrangling with dplyr and tidyr

Philipp Altenbach, Taoufik Brinis, Ronny Grieder, Ryan Kreis

April 24, 2025

1 Introduction

The purpose of this report is to demonstrate data wrangling techniques using `dplyr` and `tidyr` within the R environment. This assignment builds upon the previously created private data set from Project 1 and applies additional steps to intentionally "dirtyfy" the data. Through exploratory data analysis (EDA) and systematic detection of special values (missing values, NaN, and outliers), this report aims to visualize data quality issues present within our data set.

2 Task 1

2.1 Creating a report using the DataExplorer's function `create_report()` .

```
# create_report(myLCdata_dirty) # Generate EDA Report
```

Do not execute `create_report()` within the Quarto document. Use a direct execution in the console instead.

2.2 Screenshot of the `Missing Data Profile` overview

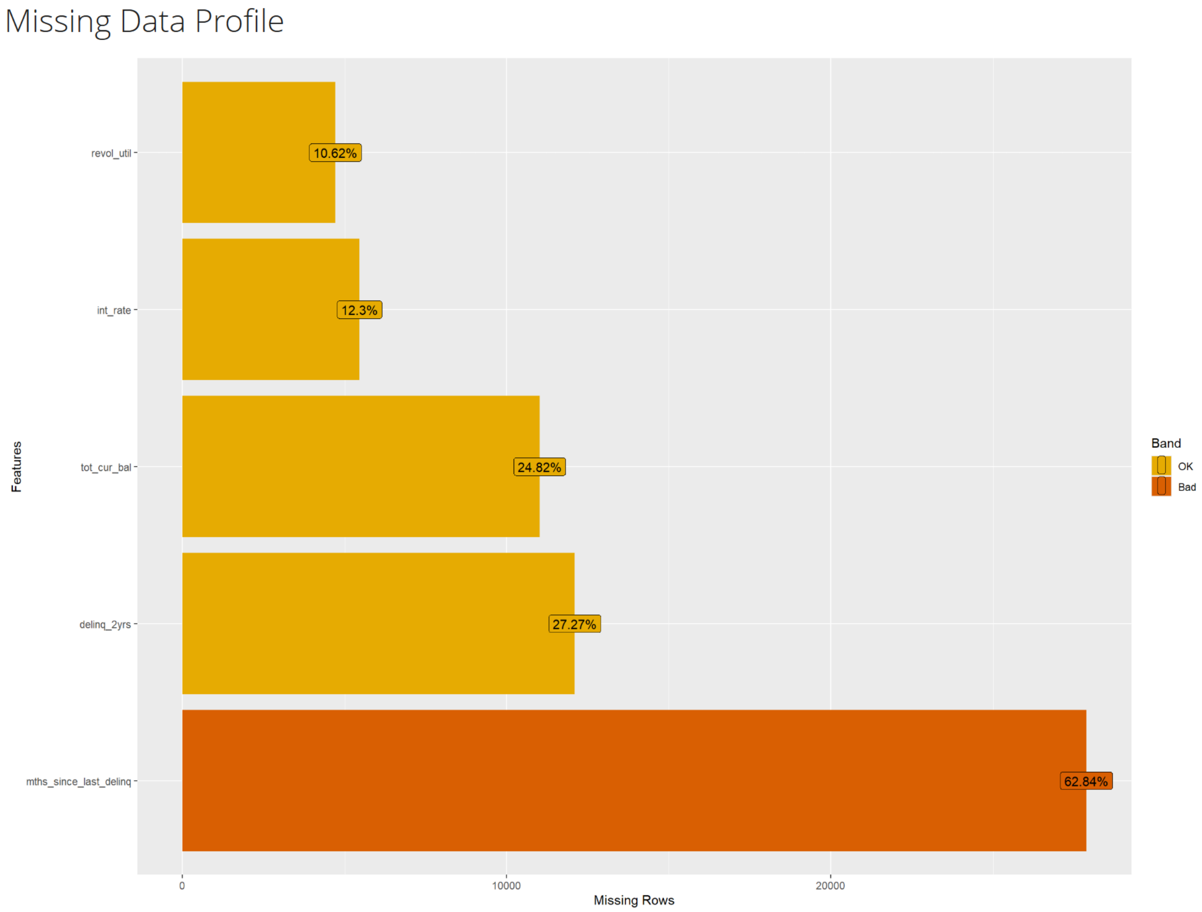

Figure 1: Missing Data Profile generated by DataExplorer. The plot shows the percentage of missing values (NAs) per attribute.

As shown in Figure 1, the Missing Data Profile generated by DataExplorer provides an overview of the percentage of missing values across the attributes. This visualization helps us to quickly identify variables with significant amounts of missing data.

3 Task 2

We decided to split Task 2 into four steps for better readability and an increased learning experience.

3.1 Counting specials

Our first goal was to create a new data frame object called `char_specials`. `char_specials` is a **data frame** with **one row**, where each **column** contains the **count** of a specific type of special value, as indicated in the project description (e.g., `NA`, `"n/a"`, etc.), for each **character column** in the original `myLCdata_dirty`. However, since our data frame from project one, `myLCdata,` only contains numerical values/attributes, we expect the counts to be zero or empty for the special character columns. We found out that the result will be a empty **tibble,** or in other words, an **empty data frame.**

```r
char_specials <- myLCdata_dirty %>%
  summarise(
    across(
      where(is.character),                    # Apply to all character columns
      list(
        na_count = ~sum(is.na(.)),            # Count NAs
        n_a_count = ~sum(. == "n/a"),         # Count "n/a" strings
        space_count = ~sum(. == " "),         # Count single spaces
        empty_count = ~sum(. == "")           # Count empty strings
      ),
      .names = "{.col}_{.fn}"                 # Create clear output column names
    )
  )

# Testing what char_specials really is now
# print(typeof(char_specials))
# ncol(char_specials)
# nrow(char_specials)
# is.data.frame(char_specials)
str(char_specials)
```

```
'data.frame':   1 obs. of  0 variables
```

Following that, we applied the same approach for the numerical values contained within the `myLCdata_dirty` data frame.

```r
num_specials <- myLCdata_dirty %>%
  summarise(
    across(
      where(is.numeric),
      list(
        na_count = ~sum(is.na(.)),   # Count NAs
        nan_count = ~sum(is.nan(.))  # Count NaNs
      ),
      .names = "{.col}_{.fn}"
    )
  )

# print(typeof(num_specials))
# ncol(num_specials)
# nrow(num_specials)
# is.data.frame(num_specials)
str(num_specials) # df structure
```

```
'data.frame':   1 obs. of  10 variables:
 $ int_rate_na_count                : int 5458
 $ int_rate_nan_count               : int 3035
 $ mths_since_last_delinq_na_count : int 27880
 $ mths_since_last_delinq_nan_count: int 3346
```

```
 $ revol_util_na_count             : int 4713
 $ revol_util_nan_count            : int 3074
 $ tot_cur_bal_na_count            : int 11014
 $ tot_cur_bal_nan_count           : int 3906
 $ delinq_2yrs_na_count            : int 12098
 $ delinq_2yrs_nan_count           : int 4745
```

3.2 Identifying outliers and relative calculation

Secondly, we focused on the outliers. For that, we could recycle the logic or approach we did in step one. Since we already proofed again that our the underlying data set does not hold any character values, we solely focused on the `is.numeric` analysis.

```
outlier_specials <- myLCdata_dirty %>%
  summarise(
    across(
      where(is.numeric), # Across all numeric values
      ~length(boxplot.stats(.)$out), # Get number of outliers of each variable
      .names = "{.col}_outlier_count"
    )
  )
```

The above code created a new data frame called `outlier_specials` that holds the following values.

```
head(outlier_specials)
```

```
  int_rate_outlier_count mths_since_last_delinq_outlier_count
1                    237                                    9
  revol_util_outlier_count tot_cur_bal_outlier_count delinq_2yrs_outlier_count
1                        1                      1138                      6034
```

```
is.data.frame(outlier_specials)
```

```
[1] TRUE
```

For each attribute the **count of outliers** was created, based on the `boxplot.stats(col)$out`logic, and stored in the above mentioned df.

When comparing the above values with our plots created for Task 1, the numbers appear to be reasonable.

3.3 Tidying data

Our next goals was to create a **tidy** data frame showing the **percentage** of:

- `NA`
- `NaN`

- `outliers`

for each **numeric column** in `myLCdata_dirty`.

First, we can bind our two previously created data frames `num_specials` and `outlier_specials` together. Again, as justified earlier, we do not have to do an analysis for characteristic values.

```
# creating a new data frame that binds the data together
num_all_specials <- bind_cols(num_specials, outlier_specials)
# head(num_all_specials) # Might lead to overflow in PDF
```

Subsequently, we needed to tidy our data frame `num_all_specials` using the `pivot_longer()` function from **tidyr.** This step was actually quite helpful to understand the `pivot_longer()` logic in a "real" use case.

```
num_all_specials_long <- num_all_specials %>%
  pivot_longer(
    cols = everything(),
    names_to = c("variable", "type"),
    names_sep = "_(?=[^_]+$)",  # Split at last underscore
    values_to = "count"
  )

# df is now restructured with columns now as rows (longer logic)
print(num_all_specials_long)
```

```
# A tibble: 15 x 3
   variable                      type  count
   <chr>                         <chr> <int>
 1 int_rate_na                   count  5458
 2 int_rate_nan                  count  3035
 3 mths_since_last_delinq_na     count 27880
 4 mths_since_last_delinq_nan    count  3346
 5 revol_util_na                 count  4713
 6 revol_util_nan                count  3074
 7 tot_cur_bal_na                count 11014
 8 tot_cur_bal_nan               count  3906
 9 delinq_2yrs_na                count 12098
10 delinq_2yrs_nan               count  4745
11 int_rate_outlier              count   237
12 mths_since_last_delinq_outlier count     9
13 revol_util_outlier            count     1
14 tot_cur_bal_outlier           count  1138
15 delinq_2yrs_outlier           count  6034
```

Next, we now needed to convert the **counts to percentage**. For this step we made use of the introduced `mutate()` function.

```r
num_all_specials_long <- num_all_specials_long %>% # Apply following functions to df
  mutate(
    percentage = count / nrow(myLCdata_dirty) # First, compute percentage using `count`
  ) %>%
  separate(variable, into = c("attribute", "type"), sep = "_(?=[^_]+$)") %>%
  select(attribute, type, percentage)  # Then drop `count` to keep it clean

# print(nrow(myLCdata_dirty))
print(num_all_specials_long)
```

```
# A tibble: 15 x 3
   attribute              type    percentage
   <chr>                  <chr>        <dbl>
 1 int_rate               na       0.123
 2 int_rate               nan      0.0684
 3 mths_since_last_delinq na       0.628
 4 mths_since_last_delinq nan      0.0754
 5 revol_util             na       0.106
 6 revol_util             nan      0.0693
 7 tot_cur_bal            na       0.248
 8 tot_cur_bal            nan      0.0880
 9 delinq_2yrs            na       0.273
10 delinq_2yrs            nan      0.107
11 int_rate               outlier  0.00534
12 mths_since_last_delinq outlier  0.000203
13 revol_util             outlier  0.0000225
14 tot_cur_bal            outlier  0.0256
15 delinq_2yrs            outlier  0.136
```

At this point, our data frame is **not fully tidy**, as it contains multiple separate rows representing different attributes of the same variable. Therefore, as a final step before visualization, we need to ensure the data conforms to a **wide tidy structure**. For example, the `int_rate` variable currently appears in two distinct rows, each representing the counts and percentages of missing values (i.e., NAs and NaNs) separately.

```r
# Pivot to wide format to get one row per attribute
num_all_specials_tidy <- num_all_specials_long %>%
  pivot_wider(
    names_from = type,
    values_from = percentage,
    names_prefix = "perc_"
  )

print(num_all_specials_tidy)
```

```
# A tibble: 5 x 4
  attribute              perc_na perc_nan perc_outlier
  <chr>                    <dbl>    <dbl>        <dbl>
1 int_rate                 0.123   0.0684      0.00534
```

2	mths_since_last_delinq	0.628	0.0754	0.000203
3	revol_util	0.106	0.0693	0.0000225
4	tot_cur_bal	0.248	0.0880	0.0256
5	delinq_2yrs	0.273	0.107	0.136

3.4 Plotting the results

We now wanted to bring our tidy data frame to life with a plot using **ggplot2.**

```
ggplot(num_all_specials_long, aes(x = attribute, y = percentage)) +
  geom_col(width = 0.7, fill = "steelblue") +
  geom_text(
    aes(label = scales::percent(percentage, accuracy = 0.1)),
    hjust = -0.2,
    size = 3.5,                        # Font size of the labels
    color = "red"
  ) +
  coord_flip() +
  scale_y_continuous(
    labels = percent_format(),
    breaks = seq(0, 1, by = 0.1),
    limits = c(0, 0.7)
  ) +
  facet_wrap(~type, ncol = 1, scales = "free_y") +
  labs(
    title = "Percentage of special values in the numerical attributes of 'myLCdata'",
    x = NULL,
    y = NULL
  ) +
  theme_minimal(base_size = 13) +
  theme(
    plot.title = element_text(size = 12, face = "bold", hjust = 0.5),
    strip.text = element_text(face = "bold", size = 11),
    panel.grid.major.y = element_line(color = "white"),  # Re-enable Y grid lines
    panel.grid.major.x = element_line(color = "white"),  # Subtle X grid lines
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "grey90", color = NA)
  )
```

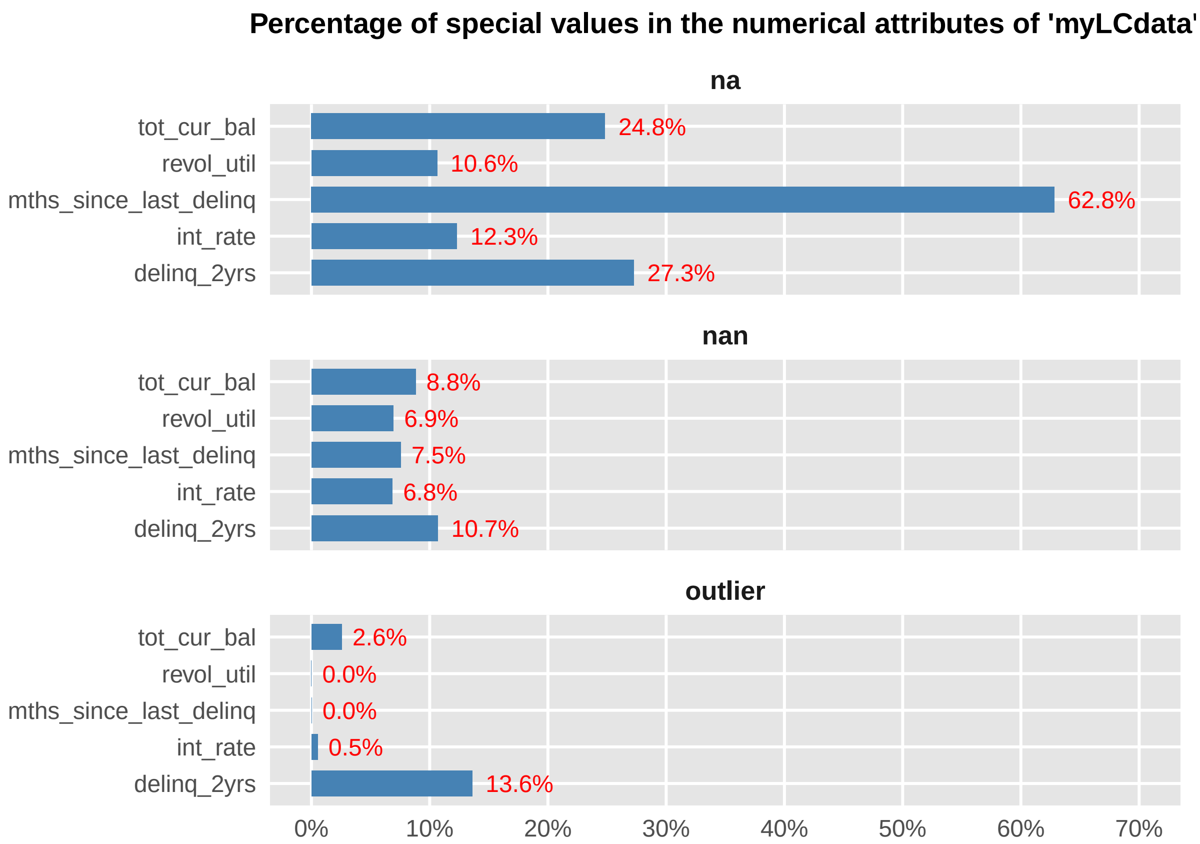

Figure 2: Percentage of special values (NA, NaN, outliers) per numerical attribute in `myLCdata_dirty`. The plot displays the proportion of each special value type across the attributes using `geom_col`. Facetting is applied to separate the categories, allowing for clear comparison between the different types of special values.

Figure 2 illustrates the proportion of special values across the numerical attributes of the dirty data set.

4 Conclusion

In this assignment, data wrangling techniques were successfully applied to identify and quantify special values such as missing values, NaNs, and outliers in the modified version of our data set. The systematic use of `dplyr`, `tidyr`, and `ggplot2` enabled efficient data summarization and visualization. The results highlight the importance of thoroughly inspecting data quality before conducting further analysis.