Data deidentification and modification

ETC5512 Assignment 3, Master of Business Analytics

Prepared by Tuguldur Ganbat, 33762740, tgan0022@student.monash.edu (mailto:tgan0022@student.monash.edu)

2024-05-11

A Data Ethics Checklist

A data ethics checklist is crucial before delving into data as it ensures legal compliance, protection of privacy, maintenance of trust with stakeholders, avoidance of bias and discrimination, promotion of fairness, minimization of harm, promotion of accountability, and enhancement of reputation. By systematically assessing ethical considerations before collecting, processing, and utilizing data, organizations can mitigate risks, prioritize individuals' rights and well-being, and foster trust and credibility in their data practices, ultimately leading to more responsible, equitable, and beneficial outcomes for all involved parties. Below I evaluated the Data Science Ethics Checklist provided in the assignment in the context of this data set:

Necessary Steps:

Limit PII exposure (A.3): Given that the dataset contains personal information such as name, age, and demographic variables, minimizing exposure of personally identifiable information (PII) is critical. Anonymization or excluding irrelevant PII for analysis is necessary to protect individuals' privacy.

Data security (B.1): Ensuring data security is paramount to prevent unauthorized access or breaches. Implementing encryption protocols, access controls, and keeping software up-to-date are essential measures to safeguard the data.

Right to be forgotten (B.2): Providing individuals with the ability to request the removal of their personal information is crucial for compliance with data protection regulations such as GDPR. Establishing mechanisms for data deletion upon request is necessary.

Data retention plan (B.3): Having a clear plan for data retention is essential to avoid retaining data longer than necessary. Establishing a schedule for deleting data once it's no longer needed helps mitigate privacy risks.

Missing perspectives (C.1): Engaging with relevant stakeholders to address blind spots in the analysis ensures a comprehensive understanding of the implications and impacts of the data. Seeking input from affected communities and subject matter experts helps mitigate biases and ensures ethical analysis.

Dataset bias (C.2): Examining the dataset for possible biases and taking steps to mitigate them is crucial for fair and unbiased analysis. Addressing issues such as stereotype perpetuation, confirmation bias, and imbalanced classes helps ensure the integrity of the analysis.

Privacy in analysis (C.4): Ensuring that personally identifiable information (PII) is not used or displayed unless necessary for the analysis helps protect individuals' privacy. Minimizing the exposure of sensitive information in visualizations and reports is essential.

Auditability (C.5): Documenting the process of generating the analysis ensures transparency and reproducibility. Having comprehensive documentation allows for the identification and resolution of issues in the future.

Explainability (D.4): Ensuring that decisions made by models can be explained in understandable terms is crucial for transparency and accountability. Providing explanations for model decisions helps build trust and understanding among stakeholders.

Not Necessary:

Informed consent (A.1): Since the dataset is simulated and anonymized for teaching purposes, obtaining informed consent may not be applicable in this context. However, ensuring data privacy and security remains essential.

Collection bias (A.2): While considering sources of bias during data collection is generally important, since the dataset is simulated, biases related to data collection may not be directly relevant in this scenario.

Downstream bias mitigation (A.4): While testing downstream results for biased outcomes is important in real-world scenarios, since the dataset is simulated for teaching purposes, downstream bias mitigation may not be directly applicable.

Fairness across groups (D.2): Testing model results for fairness across different affected groups is crucial in real-world applications to prevent discrimination. However, since the dataset is simulated, fairness testing may not be directly relevant in this context.

Metric selection (D.3): While considering additional metrics and their effects is important for model evaluation, since the dataset is simulated for teaching purposes, specific metric selection may not be directly relevant in this scenario.

Removal of the direct identifiers

Direct identifiers refer to the factors that can precisely identify a person within a data set. Typically, these factors contain personal data that could be exploited for harmful purposes if not eliminated prior to releasing the data to the public. The subsequent list outlines the variables functioning as direct identifiers that are eradicated from the original data set.

First name (first_name)

Last name (last_name)

Loan ID (loan_id)

```
loandata <- read_rds(here::here("raw_data/loanData.rds"))
loandata_directid_filtered <- loandata %>%
  select(-c(first_name,last_name, loan_id))
```

De-identification strategies

Removal of variables(Quasi-identifiers)

Quasi-identifiers refer to bits of data that, on their own, can't pinpoint a specific person but can be pieced together to potentially reveal identifying details.

Below is the roster of variables to be omitted to mitigate the risk of re-identification:

- Income (**income**)
- Metropolitan Statistical Area (**msa**)
- Property state (**state**)
- Seller name (**seller**)
- Servicer name (**servicer**)
- Zip code short (**zip_3**)

```
loandata_quasi_filtered <- loandata_directid_filtered %>% select(-c('msa','state','seller','servicer','z
ip_3', 'income'))
```

Customer Age

Aggregating customer age is vital for safeguarding individual privacy by consolidating individuals into wider groupings. This process obscures precise identifying details, thereby decreasing the likelihood of re-identification. Utilizing age brackets, such as 30-34, 35-39, etc., maintains general demographic insights while mitigating the possibility of pinpointing specific individuals.

```
loandata_age_grouped <- loandata_quasi_filtered %>%
  mutate(cus_age_group = case_when(
    cus_age >= 30 & cus_age < 35 ~ "30-34 years",
    cus_age >= 35 & cus_age < 40 ~ "35-39 years",
    cus_age >= 40 & cus_age < 45 ~ "40-44 years",
    cus_age >= 45 ~ "45+ years"
  ))
age_groups <- c("30-34 years", "35-39 years", "40-44 years", "45+ years")

age_group_counts <- loandata_age_grouped %>%
  count(cus_age_group) %>%
  rename(`Age Group` = cus_age_group, `Number of Borrowers` = n)

# After Aggregation
print(age_group_counts)
```

```
##     Age Group Number of Borrowers
## 1 30-34 years             1319644
## 2 35-39 years             1319286
## 3 40-44 years             1320434
## 4   45+ years              263821
```

Number of dependents

The count of dependents might not directly correlate with a bank's loan performance, yet it can offer crucial insights into borrowers' capacity to repay loans without defaulting. However, to mitigate potential misuse, this data could be encoded into a binary indicator: 1 for borrowers with dependents and 0 for those without.

```
loan_data_new <- loandata_age_grouped %>%
  mutate(have_dependents = case_when( no_depend == 0 ~ "None",
                                      no_depend == 1 ~ "Single",
                                      no_depend > 1 ~ "Multiple"))
dependent_counts <- loan_data_new %>%
  count(have_dependents) %>%
  rename(`Dependent Status` = have_dependents, `Number of Borrowers` = n)

print(dependent_counts)
```

```
##   Dependent Status Number of Borrowers
## 1         Multiple             2046302
## 2             None              870014
## 3           Single             1306869
```

Original Interest Rate

Customers often readily recognize the initial interest rates, which can lead to their easy identification within a dataset, facilitating re-identification. A significant portion of these interest rates falls within the range of 2.5% to 6.5%. Therefore, aggregating the interest rates would be a prudent choice.

```
loan_data_new <- loan_data_new %>%
  mutate(orig_rt_grouped = case_when((orig_rt <= 2.5) ~ "0-2.5",
                                     (orig_rt > 2.5 & orig_rt <= 3.5) ~ "2.5-3.5",
                                     (orig_rt > 3.5 & orig_rt <= 4.5) ~ "3.5-4.5",
                                     (orig_rt > 4.5 & orig_rt <= 5.5) ~ "4.5-5.5",
                                     (orig_rt > 5.5 & orig_rt <= 6.5) ~ "5.5-6.5",
                                     (orig_rt > 6.5 ) ~ "> 6.5"))

loan_breaks <- c("0-2.5","2.5-3.5","3.5-4.5","4.5-5.5","5.5-6.5","> 6.5")
orig_rate_counts <- loan_data_new %>%
  count(orig_rt_grouped) %>%
  rename(`Original Interest Rate (in %)` = orig_rt_grouped, `Number of Borrowers` = n)
#After Aggregation
print(orig_rate_counts)
```

```
##   Original Interest Rate (in %) Number of Borrowers
## 1                         0-2.5                 984
## 2                       2.5-3.5              308597
## 3                       3.5-4.5             2144898
## 4                       4.5-5.5             1623667
## 5                       5.5-6.5              143737
## 6                         > 6.5                1302
```

Last loan rate

Individuals can easily identify the ultimate loan rate, much like the concept discussed in this section. Hence, a similar method of aggregation will be employed to aid in de-identifying the data. The following table showcases how the loan rate can be aggregated.

```
loan_data_new <- loan_data_new %>%
  mutate(last_rt_grouped = case_when((last_rt <= 2.5) ~ "0-2.5",
                                     (last_rt > 2.5 & last_rt <= 3.5) ~ "2.5-3.5",
                                     (last_rt > 3.5 & last_rt <= 4.5) ~ "3.5-4.5",
                                     (last_rt > 4.5 & last_rt <= 5.5) ~ "4.5-5.5",
                                     (last_rt > 5.5 & last_rt <= 6.5) ~ "5.5-6.5",
                                     (last_rt > 6.5 ) ~ "> 6.5"))

loan_breaks <- c("0-2.5","2.5-3.5","3.5-4.5","4.5-5.5","5.5-6.5","> 6.5")

interest_rate_counts <- loan_data_new %>%
  count(last_rt_grouped) %>%
  rename(`Last Interest Rate (in %)` = last_rt_grouped, `Number of Borrowers` = n)
#After Aggregation
print(interest_rate_counts)
```

```
##   Last Interest Rate (in %) Number of Borrowers
## 1                     0-2.5                 984
## 2                   2.5-3.5              312080
## 3                   3.5-4.5             2146973
## 4                   4.5-5.5             1619045
## 5                   5.5-6.5              142818
## 6                     > 6.5                1285
```

Credit score of borrower

The credit score of borrowers should be aggregated because it plays a crucial role in determining their creditworthiness, particularly in the US credit industry, where FICO scores are widely used by financial institutions like banks to assess borrowers' ability to repay loans. Also, credit scores are categorized into distinct groups, making them suitable candidates for aggregation. This aggregation process offers the advantage of de-identifying individual credit score values, thus enhancing privacy protection. By grouping credit scores into broader categories, the risk of re-identification is minimized, aligning with ethical data practices and regulatory requirements.

```
Scores <- c("0-580","580-670","670-740","740-800","800-850")
Ratings <- c("Poor","Fair","Good","Very Good","Excellent")

df_ratings <- data.frame(Scores, Ratings)
#Credit Score Ratings
print(df_ratings)
```

```
##    Scores   Ratings
## 1   0-580      Poor
## 2 580-670      Fair
## 3 670-740      Good
## 4 740-800 Very Good
## 5 800-850 Excellent
```

```
loan_data_new <- loan_data_new %>%
  mutate(cscore_b_rate = case_when( cscore_b > 0 & cscore_b <580 ~ "Poor",
                                    cscore_b >= 580 & cscore_b <670 ~ "Fair",
                                    cscore_b >=670  & cscore_b <740 ~ "Good",
                                    cscore_b >= 740 & cscore_b < 800 ~ "Very Good",
                                    cscore_b >= 800 ~ "Excellent"
  ))

credit_score_counts <- loan_data_new %>%
  count(cscore_b_rate)

credit_score_table <- as.data.frame(credit_score_counts)
colnames(credit_score_table) <- c("Credit Score Rating", "Number of Borrowers")
#Aggregated Borrower Credit Score
print(credit_score_table)
```

```
##   Credit Score Rating Number of Borrowers
## 1           Excellent              597747
## 2                Fair              318407
## 3                Good             1323649
## 4                Poor                   1
## 5           Very Good             1979623
## 6                <NA>                3758
```

As indicated above, it’s evident that there’s merely one person falling within the 0-580 credit score range (Poor category). Considering the risk of potential re-identification, we can implement value suppression by nullifying the borrower’s credit score, effectively preventing any inadvertent identification, as illustrated above.

```
loan_data_new <- loan_data_new %>%
  mutate(cscore_b_rate = ifelse(cscore_b_rate == "Poor", NA, cscore_b_rate))

credit_score_counts <- loan_data_new %>%
  count(cscore_b_rate)

credit_score_table <- as.data.frame(credit_score_counts)
colnames(credit_score_table) <- c("Credit Score Rating", "Count")
#After Suppression
print(credit_score_table)
```

```
##   Credit Score Rating   Count
## 1           Excellent  597747
## 2                Fair  318407
## 3                Good 1323649
## 4           Very Good 1979623
## 5                <NA>    3759
```

Number of borrowers

Since customers typically have a good grasp of the number of borrowers, it's important to use top coding to conceal any outliers.

```
orig_borrowers <- loan_data_new %>% count(num_bo)

orig_borrowers_table <- as.data.frame(orig_borrowers)
colnames(orig_borrowers_table) <- c("Number of Borrowers", "Count")
#Before Top Encoding
print(orig_borrowers_table)
```

```
##   Number of Borrowers   Count
## 1                   1 2279927
## 2                   2 1901820
## 3                   3   35583
## 4                   4    5839
## 5                   5      10
## 6                   6       5
## 7                   7       1
```

In the table above, it's evident that there are only five instances involving six borrowers for the loan, and just one instance with seven borrowers. Because of this limited number, these individuals might still be identifiable. Hence, to mitigate this risk, a technique known as the "top encoding strategy" has been implemented to mask all the outliers, as illustrated in the subsequent table, aiding in the de-identification process.

```
loan_data_new <- loan_data_new %>%
  mutate(num_bo_new = ifelse(num_bo >= 5, ">4", as.character(num_bo)))

borrowers_new <- c("1", "2", "3", "4", ">4")

borrowers_counts <- loan_data_new %>%
  count(num_bo_new)

borrowers_table <- as.data.frame(borrowers_counts)
colnames(borrowers_table) <- c("Number of Borrowers", "Count")
#After Top Encoding
print(borrowers_table)
```

```
##   Number of Borrowers   Count
## 1                   1 2279927
## 2                   2 1901820
## 3                   3   35583
## 4                   4    5839
## 5                 > 4      16
```

Debt to income ratio

Individual borrowers possess specific data regarding their debt and income, enabling potential re-identification within a dataset. Presently, the dataset contains three distinct values, which will undergo de-identification through a top encoding approach. The table below illustrates the implementation of the top-coded term "> 50" for debt-to-income ratio, effectively safeguarding against re-identification attempts.

```
loan_data_new <- loan_data_new %>% mutate(dti_new = ifelse(dti > 50, "> 50", dti))

dti_out <- loan_data_new %>%
  count(dti_new) %>%
  arrange(n) %>%
  head(5) %>%
  rename("Debt to income ratio" = dti_new, "Count" = n)

dti_table <- as.data.frame(dti_out)
#After Top Encoding
print(dti_table)
```

```
##   Debt to income ratio Count
## 1                 > 50    24
## 2                    1   270
## 3                    2   299
## 4                    3   421
## 5                    4   601
```

First paid installment date

The customer will readily identify the initial payment date through their bank's payment summary, facilitating comparison with previous records for verification. The dataset contains a specific number of unique dates, as highlighted in Table 1. To enhance de-identification, we introduce random noise to the data by adjusting the dates in months. Table 2 displays the original first payment dates alongside the new dates generated after introducing this random noise.

```
df_unique <- loan_data_new %>%
  count(frst_dte) %>%
  filter(n == 1)

unique_dates <- as.Date(df_unique$frst_dte)

df_unique <- data.frame("Unique First Payment Dates" = unique_dates)
#Original First Payment Dates
print(df_unique)
```

```
##   Unique.First.Payment.Dates
## 1                 2013-01-01
## 2                 2013-09-01
## 3                 2014-12-01
## 4                 2015-02-01
```

```
loan_data_new$frst_dte <- as.Date(loan_data_new$frst_dte)

loan_data_new <- loan_data_new %>%
  mutate(frst_dte_new = ifelse(frst_dte %in% unique_dates, frst_dte + months(floor(runif(1, min=
2, max=3))), frst_dte))

loan_data_new$frst_dte_new <- as.Date(loan_data_new$frst_dte_new, origin="1970-1-1")

df_new_dates <- loan_data_new %>%
  filter(frst_dte %in% unique_dates) %>%
  select(frst_dte, frst_dte_new)

df_new_dates <- df_new_dates %>%
  rename("Original First Payment Date" = frst_dte, "New First Payment Date" = frst_dte_new)
#Original & New first payment dates after adding noise
print(df_new_dates)
```

```
##   Original First Payment Date New First Payment Date
## 1                  2015-02-01             2015-04-01
## 2                  2013-09-01             2013-11-01
## 3                  2014-12-01             2015-02-01
## 4                  2013-01-01             2013-03-01
```

Last paid installment date

The customer’s last paid installment date is easily accessible to them through their bank’s payment summary, making it readily comparable with past records for re-identification purposes. The raw dataset includes five dates that can be uniquely identified, as noted in table below. To address this, we introduce random noise to the data in the form of months, which helps in the de-identification process.

```
df_unique <- loan_data_new %>%
  count(lpi_dte) %>%
  filter(n == 1)

unique_dates <- as.Date(df_unique$lpi_dte)

df_unique <- data.frame("Unique Last Payment Dates" = unique_dates)

#Last payment date
print(df_unique)
```

```
##   Unique.Last.Payment.Dates
## 1                2017-01-01
## 2                2017-03-01
## 3                2020-11-01
## 4                2021-02-01
## 5                2021-03-01
```

```
loan_data_new$lpi_dte <- as.Date(loan_data_new$lpi_dte)

loan_data_new <- loan_data_new %>%
  mutate(lpi_dte_new = ifelse(lpi_dte %in% unique_dates, lpi_dte + months(floor(runif(1, min=1, max=3))), lpi_dte))

loan_data_new$lpi_dte_new <- as.Date(loan_data_new$lpi_dte_new, origin="1970-01-01")

df_new_dates <- loan_data_new %>%
  filter(lpi_dte %in% unique_dates) %>%
  select(lpi_dte, lpi_dte_new) %>%
  rename("Original Last Payment Date" = lpi_dte, "New Last Payment Date" = lpi_dte_new)

# Original and new last payment dates after adding noise
print(df_new_dates)
```

```
##   Original Last Payment Date New Last Payment Date
## 1                 2017-03-01            2017-04-01
## 2                 2017-01-01            2017-02-01
## 3                 2020-11-01            2020-12-01
## 4                 2021-02-01            2021-03-01
## 5                 2021-03-01            2021-04-01
```

Saving the data as a .csv file.

```
omitted_fields <- c("last_upb","no_depend","cus_age","orig_rt",
                    "last_rt","cscore_b","num_bo","dti"," frst_dte ",
                    "lpi_dte")

loan_data_new <- loan_data_new %>% select(-one_of(omitted_fields))

write_csv(loan_data_new, here::here("data/release-data-Tuguldur-Ganbat.csv"))
```

References

RStudio : RStudio Team (2020). RStudio: Integrated Development for R. RStudio, PBC, Boston, MA URL http://www.rstudio.com/ (http://www.rstudio.com/).

Tidyverse : Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). “Welcome to the tidyverse.” *Journal of Open Source Software*, *4*(43), 1686. doi:10.21105/joss.01686 (doi:10.21105/joss.01686) https://doi.org/10.21105/joss.01686 (https://doi.org/10.21105/joss.01686).

ggplot2 : H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

dplyr : Wickham H, François R, Henry L, Müller K, Vaughan D (2023). *dplyr: A Grammar of Data Manipulation*. R package version 1.1.4, https://CRAN.R-project.org/package=dplyr (https://CRAN.R-project.org/package=dplyr).

Checklist : Deon.drivendata.org (2024). “An ethics checklist for data scientists”. https://deon.drivendata.org/#default-checklist (https://deon.drivendata.org/#default-checklist)