Research log

Jamie Scheper - 373689

16 November, 2020

Table of content

1 Original dataset

1.1 Introduction

The article, "Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records", states that hyperglycemia management has a significant impact on outcomes and readmission rates for hospitalized patients. The authors base this conclusion on a comprehensive assessment of 70,000 diabetes patient records retrieved from 140 US hospitals. The results describing the relationship between readmission and HbA1c levels can further strengthen existing diabetes management strategies to reduce readmission rates.[1]

1.2 Dataset and Attributes Information

All information used in this article comes from a database consisting of 41 tables and a total of 117 features, including demographics (such as gender, race, and age), inpatient and outpatient indicators, and in-hospital mortality. The data were collected from 130 hospitals in the USA over more than ten years (1998 to 2008) and contained around 74 million unique visits by 18 million unique patients. This research used encounters that had to meet the following specifications:

1. The record is a hospital admission;
2. The encounter includes a diagnosis of diabetes (any type);
3. The length of stay is at least one day and at most eighteen days;
4. Laboratory results are available; and
5. Medications were administered.

Of these criteria, 101,000 encounters satisfied all specifications. After removing encounters with incomplete data (weight and medical specialty) or potentially biasing cases (discharge to hospice or death), 69,984 encounters remained in the final dataset.

The initial dataset consists of 55 attributes, with each instance representing a single patient encounter. Because there are too many attributes to describe in full, please refer to the codebook for detailed descriptions. Here, we highlight essential attributes, including their types and possible values. Patient age is nominal and grouped into ten-year intervals. Admission type, which describes the reason for hospitalization, has nine distinct values and is nominal. Some attributes are numeric count variables, such as the number of lab procedures, the number of medications, and the number of emergency visits. The database includes three diagnosis attributes, which can take 848, 923, and 954 distinct values, respectively. These values are based on ICD-9 codes and are nominal. Other important attributes include whether a patient changed medications (values: "no change" and "change", nominal) and

whether a patient received diabetes medication (values: "yes" or "no", nominal). Twenty-four additional attributes indicate whether a medication was prescribed and, if so, whether the dosage was increased ("up"), decreased ("down"), or unchanged ("steady") during the encounter. Readmission status is recorded in the nominal variable `Readmitted`, with possible values "<30" for readmission within 30 days, ">30" for readmission after 30 days, and "No" for no readmission. Because the authors aimed to determine whether a relationship exists between readmission and HbA1c measurement, they derived a new attribute `HbA1c` with four categories based on available fields: (1) no HbA1c test performed; (2) HbA1c performed and within the normal range; (3) HbA1c performed and greater than 8% with no change in diabetes medication; and (4) HbA1c performed, greater than 8%, and diabetes medication was changed.

1.3 Research Goal

Is it possible, using machine learning techniques, to predict the duration of an inpatient hospital visit?

2 EDA (Exploratory Data Analysis)

This section performs an exploratory data analysis (EDA) on the dataset described above. Through EDA, we examine distributions and relationships between variables and identify missing values. If missing values are present, we address them by imputing or removing records where appropriate. Additionally, we evaluate variation within and between attributes to determine which variables are most relevant to the research goal.

First, we load the required packages, primarily for data handling and visualization, as well as the dataset itself. We also load a codebook that contains descriptions of every attribute in the initial dataset. The codebook was not retrieved from an external source but was constructed based on interpretation of the dataset.

```r
# Load used packages
library(plyr)
library(dplyr, warn.conflicts = FALSE)
library(tidyr)
library(tidyverse)
library(ggplot2)
library(grid)
library(gridExtra)
library(hexbin)
library(foreign)
library(kableExtra)

# Load in used data
encounter.data <- read.table(
  file ='datasets/data_diabetes_retrieved/diabetic_data.csv',
  sep = ',', header = TRUE)

codebook <- read.csv(file = 'misc/codebook.csv', sep = ';',
                     header = T)
```

```r
# Read codebook via kbl
kable(codebook, "latex",
      caption = "Codebook with information about attributes",
      booktabs = T, longtable = T) %>%
 kable_styling(latex_options = c("repeat_header"),
               repeat_header_continued =
                 "\\textit{(Continued on Next Page...)}") %>%
  column_spec(1, width = "4cm") %>%
  column_spec(2, width = "3cm") %>%
  column_spec(3, width = "3cm") %>%
  column_spec(4, width = "5cm")
```

Table 1: Codebook with information about attributes

Name	Full.name	Data.type	Description.and.valuation
encounter_id	Encounter ID	Int	An unique number to identify an encounter.
patient_nbr	Patient number	Int	An unique number to identify a patient.
race	Race	Factor	The race of a patient with values: Caucasian, Asian, African American, Hispanic, and other.
gender	Gender	Factor	The gender of a patient with values: male, female, and unknown/invalid.
age	Age	Factor	The age of a patient, grouped in ten-year intervals, for example: [0, 10) and [90, 100).
weight	Weight	Factor	Weight in pounds.
adminsion_type_id	Admission type	Int	An integer identifying what a patient's admission type is, which corresponds with a method, for example:1 - emergency and 2 - urgent. Valuation has nine distinct values.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
discharge_disposition_id	Discharge disposition	Int	An integer identifying how a patient was discharged, which corresponds with a method, for example: 1 - discharged to home and 2 - expired. Valuation has 29 distinct values.
admission_source_id	Admission source	Int	An integer identifying how a patient was admitted to hospital, which corresponds with a method, for example: 1 - physician referral and 2 - emergency room. Valuation has 21 distinct values.
time_in_hospital	Time in hospital	Int	Integer number of days between admission and discharge.
payer_code	Payer code	Factor	An integer identifying how a patient is paying, which corresponds with a method, for example: 1 - Blue Cross/Blue Shield and 2 - Medicare. Valuatoon has 23 distinct values.
medical_specialty	Medical specialty	Factor	An integer identifying the specialty of the admitting physician, which corresponds with a method, for example: 1 - cardiology and 2 - internal medicine. Valuation has 84 distinct values.
num_lab_procedures	Number of lab procedures	Int	Number of lab tests performed during the encounter.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
num_procedures	Number of procedures	Int	Number of procedures, other than lab tests, performed during the encounter.
num_medications	Number of medications	Int	Number of distinct generic names administered during the encounter.
number_outpatient	Number of outpatient visits	Int	Number of outpatient visits of the patient in the year preceding the encounter.
number_emergency	Number of emergency visits	Int	Number of emergency visits of the patient in the year preceding the encounter.
number_inpatient	Number of inpatient visits	Int	Number of inpatient visits of the patient in the year preceding the encounter.
diag_1	Diagnosis 1	Factor	The primary diagnosis and is coded as first three digits of ICD9, with 848 distinct values.
diag_2	Diagnosis 2	Factor	The secondary diagnosis and is coded as first three digits of ICD9, with 923 distinct values.
diag_3	Diagnosis 3	Factor	An additional secondary diagnosis and is coded as first three digits of ICD9, with 954 distinct values.
number_diagnoses	Number of diagnoses	Int	Number of diagnoses entered to the system.
max_glu_serum	Glucose serum test results	Factor	Indicates the range of the result or if the test was not taken. Values: >200, >300 and normal if the test is taken, and none if the test is not taken.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
A1Cresult	Alc test results	Factor	Indicates the range of the result or if the test was not taken. Values: >8% if result was greater than 8%, 7% if result was greater than 7% but less than 8%, normal if the test was less than 7%, and none if no test was taken.
metformin	Metformin prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
repaglinide	Repaglinide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
nateglinide	Nateglinide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
chlorpropamide	Chlorpropamide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
glimepiride	Glimepiride prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
acetohexamide	Acetohexamide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
glipizide	Glipizide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
glyburide	Glyburide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
tolbutamide	Tolbutamide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
pioglitazone	Pioglitazone prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
rosiglitazone	Rosiglitazone prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
acarbose	Acarbose prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
miglitol	Miglitol prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
troglitazone	Troglitazone prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
tolazamide	Tolazamide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
examide	Examide prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
citoglipton	Citoglipton prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
insulin	Insulin prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
glyburide-metformin	Glyburide-metformin prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
glipizide-metformin	Glipizide-metaformin prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.

(Continued on Next Page...)

Table 1: Codebook with information about attributes *(continued)*

Name	Full.name	Data.type	Description.and.valuation
glimepiride-pioglitazone	Glimepiride-pioglitazone prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
metformin-pioglitazone	Metformin-pioglitazone prescribed	Factor	Indicates whether the drug was prescribed or there was a change in dosage. Values: up if the dosage was increased, down if the dosage was decreased, steady if there was no change in dosage, and No if the drug was not prescribed.
change	Change of medications	Factor	Indicates if medication was changed, with values: 'Change' or 'No change'.
diabetesMed	Diabetes medication	Factor	Indicates if diabetes medication was prescribed, with values: 'Yes' and 'No'.
readmitted	Readmitted	Factor	Days to inpatient readmission, with values: <30 if the patient was readmitted in less than 30 days, >30 if the patient was readmitted in more than 30 days, and 'NO' for no record of readmission.

To get a clearer picture of the dataset's structure and distribution, we used `glimpse()`. The output shows that the data contains 50 columns (attributes). In addition, `summary()` provides an overview of each variable, including counts for each level/value where applicable.

```r
# Give a glimpse of how the data is distributed
glimpse(encounter.data)
```

```
## Rows: 101,766
## Columns: 50
## $ encounter_id             <int> 2278392, 149190, 64410, 500364, 16680, 35754,~
## $ patient_nbr              <int> 8222157, 55629189, 86047875, 82442376, 425192~
## $ race                     <chr> "Caucasian", "Caucasian", "AfricanAmerican", ~
## $ gender                   <chr> "Female", "Female", "Female", "Male", "Male",~
## $ age                      <chr> "[0-10)", "[10-20)", "[20-30)", "[30-40)", "[~
## $ weight                   <chr> "?", "?", "?", "?", "?", "?", "?", "?", "?", ~
## $ admission_type_id        <int> 6, 1, 1, 1, 1, 2, 3, 1, 2, 3, 1, 2, 1, 1, 3, ~
## $ discharge_disposition_id <int> 25, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 3, 6, 1,~
## $ admission_source_id      <int> 1, 7, 7, 7, 7, 2, 2, 7, 4, 4, 7, 4, 7, 7, 2, ~
## $ time_in_hospital         <int> 1, 3, 2, 2, 1, 3, 4, 5, 13, 12, 9, 7, 7, 10, ~
## $ payer_code               <chr> "?", "?", "?", "?", "?", "?", "?", "?", "?", ~
## $ medical_specialty        <chr> "Pediatrics-Endocrinology", "?", "?", "?", "?~
## $ num_lab_procedures       <int> 41, 59, 11, 44, 51, 31, 70, 73, 68, 33, 47, 6~
## $ num_procedures           <int> 0, 0, 5, 1, 0, 6, 1, 0, 2, 3, 2, 0, 0, 1, 5, ~
## $ num_medications          <int> 1, 18, 13, 16, 8, 16, 21, 12, 28, 18, 17, 11,~
## $ number_outpatient        <int> 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ number_emergency         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ~
## $ number_inpatient         <int> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ diag_1                   <chr> "250.83", "276", "648", "8", "197", "414", "4~
## $ diag_2                   <chr> "?", "250.01", "250", "250.43", "157", "411",~
## $ diag_3                   <chr> "?", "255", "V27", "403", "250", "250", "V45"~
## $ number_diagnoses         <int> 1, 9, 6, 7, 5, 9, 7, 8, 8, 8, 9, 7, 8, 8, 8, ~
## $ max_glu_serum            <chr> "None", "None", "None", "None", "None", "None~
## $ A1Cresult                <chr> "None", "None", "None", "None", "None", "None~
## $ metformin                <chr> "No", "No", "No", "No", "No", "No", "Steady",~
## $ repaglinide              <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ nateglinide              <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ chlorpropamide           <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ glimepiride              <chr> "No", "No", "No", "No", "No", "No", "Steady",~
## $ acetohexamide            <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ glipizide                <chr> "No", "No", "Steady", "No", "Steady", "No", "~
## $ glyburide                <chr> "No", "No", "No", "No", "No", "No", "No", "St~
## $ tolbutamide              <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ pioglitazone             <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ rosiglitazone            <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ acarbose                 <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ miglitol                 <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ troglitazone             <chr> "No", "No", "No", "No", "No", "No", "No", "No~
```

```
## $ tolazamide               <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ examide                  <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ citoglipton              <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ insulin                  <chr> "No", "Up", "No", "Up", "Steady", "Steady", "~
## $ glyburide.metformin      <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ glipizide.metformin      <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ glimepiride.pioglitazone <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ metformin.rosiglitazone  <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ metformin.pioglitazone   <chr> "No", "No", "No", "No", "No", "No", "No", "No~
## $ change                   <chr> "No", "Ch", "No", "Ch", "Ch", "No", "Ch", "No~
## $ diabetesMed              <chr> "No", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes~
## $ readmitted               <chr> "NO", ">30", "NO", "NO", "NO", ">30", "NO", "~
```

```r
# Give a quick summary of the data,
# and give information such as min and max value, median and mean
summary(encounter.data)
```

```
##   encounter_id        patient_nbr            race              gender
##  Min.   :    12522   Min.   :      135   Length:101766      Length:101766
##  1st Qu.: 84961194   1st Qu.: 23413221   Class :character   Class :character
##  Median :152388987   Median : 45505143   Mode  :character   Mode  :character
##  Mean   :165201646   Mean   : 54330401
##  3rd Qu.:230270888   3rd Qu.: 87545950
##  Max.   :443867222   Max.   :189502619
##      age                weight          admission_type_id
##  Length:101766      Length:101766      Min.   :1.000
##  Class :character   Class :character   1st Qu.:1.000
##  Mode  :character   Mode  :character   Median :1.000
##                                        Mean   :2.024
##                                        3rd Qu.:3.000
##                                        Max.   :8.000
##  discharge_disposition_id admission_source_id time_in_hospital
##  Min.   : 1.000           Min.   : 1.000      Min.   : 1.000
##  1st Qu.: 1.000           1st Qu.: 1.000      1st Qu.: 2.000
##  Median : 1.000           Median : 7.000      Median : 4.000
##  Mean   : 3.716           Mean   : 5.754      Mean   : 4.396
##  3rd Qu.: 4.000           3rd Qu.: 7.000      3rd Qu.: 6.000
##  Max.   :28.000           Max.   :25.000      Max.   :14.000
##   payer_code        medical_specialty  num_lab_procedures num_procedures
##  Length:101766      Length:101766      Min.   :  1.0      Min.   :0.00
##  Class :character   Class :character   1st Qu.: 31.0      1st Qu.:0.00
##  Mode  :character   Mode  :character   Median : 44.0      Median :1.00
##                                        Mean   : 43.1      Mean   :1.34
```

```
##                                       3rd Qu.: 57.0      3rd Qu.:2.00
##                                       Max.   :132.0      Max.   :6.00
##  num_medications number_outpatient number_emergency  number_inpatient
##  Min.   : 1.00   Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000
##  1st Qu.:10.00   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000
##  Median :15.00   Median : 0.0000   Median : 0.0000   Median : 0.0000
##  Mean   :16.02   Mean   : 0.3694   Mean   : 0.1978   Mean   : 0.6356
##  3rd Qu.:20.00   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 1.0000
##  Max.   :81.00   Max.   :42.0000   Max.   :76.0000   Max.   :21.0000
##     diag_1             diag_2             diag_3          number_diagnoses
##  Length:101766      Length:101766      Length:101766      Min.   : 1.000
##  Class :character   Class :character   Class :character   1st Qu.: 6.000
##  Mode  :character   Mode  :character   Mode  :character   Median : 8.000
##                                                           Mean   : 7.423
##                                                           3rd Qu.: 9.000
##                                                           Max.   :16.000
##  max_glu_serum       A1Cresult          metformin          repaglinide
##  Length:101766      Length:101766      Length:101766      Length:101766
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  nateglinide        chlorpropamide     glimepiride        acetohexamide
##  Length:101766      Length:101766      Length:101766      Length:101766
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##   glipizide          glyburide         tolbutamide        pioglitazone
##  Length:101766      Length:101766      Length:101766      Length:101766
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  rosiglitazone        acarbose           miglitol         troglitazone
##  Length:101766      Length:101766      Length:101766      Length:101766
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
```

```
##
##   tolazamide          examide          citoglipton          insulin
##  Length:101766      Length:101766      Length:101766      Length:101766
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  glyburide.metformin glipizide.metformin glimepiride.pioglitazone
##  Length:101766       Length:101766       Length:101766
##  Class :character    Class :character    Class :character
##  Mode  :character    Mode  :character    Mode  :character
##
##
##
##  metformin.rosiglitazone metformin.pioglitazone    change
##  Length:101766           Length:101766          Length:101766
##  Class :character        Class :character       Class :character
##  Mode  :character        Mode  :character       Mode  :character
##
##
##
##  diabetesMed         readmitted
##  Length:101766      Length:101766
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

2.1 Missing values

As shown by the output of `glimpse()`, the columns `race`, `weight`, `payer_code`, `gender`, `diag_3`, and `medical_specialty` contain some (or in some cases substantial) missing values. To get a clearer picture, we zoom in on these attributes and quantify both the number and percentage of missing values.

```
myvars <- c('race', 'weight', 'payer_code',
            'medical_specialty', 'gender', 'diag_3')
amountRecords <- length(rownames(encounter.data))
tableMissingValues <- encounter.data %>%
  summarise_each_(funs("TotalMissing" =
                  sum(. %in% c('?', 'Unknown/Invalid'), na.rm = TRUE),
                       "MissingValuesPercentage" =
                 round(sum(. %in% c('?', 'Unknown/Invalid'),
                           na.rm = TRUE)/amountRecords * 100, 2)),
                  myvars) %>%
  t()
tableMissingValues <- data.frame("Missing.Values" =
                                   tableMissingValues[1:6,],
                                 "Missing.Values.Percentage" =
                                   tableMissingValues[7:12,])
rownames(tableMissingValues) <- myvars

kable(tableMissingValues, "latex", booktabs = T,
      caption = "Attributes that are missing values
      and their the amount of missing data" ) %>%
  kable_styling(full_width = F, latex_options = "hold_position") %>%
  column_spec(1, bold = T)
```

Table 2: Attributes that are missing values and their the amount of missing data

	Missing.Values	Missing.Values.Percentage
race	2273	2.23
weight	98569	96.86
payer_code	40256	39.56
medical_specialty	49949	49.08
gender	3	0.00
diag_3	1423	1.40

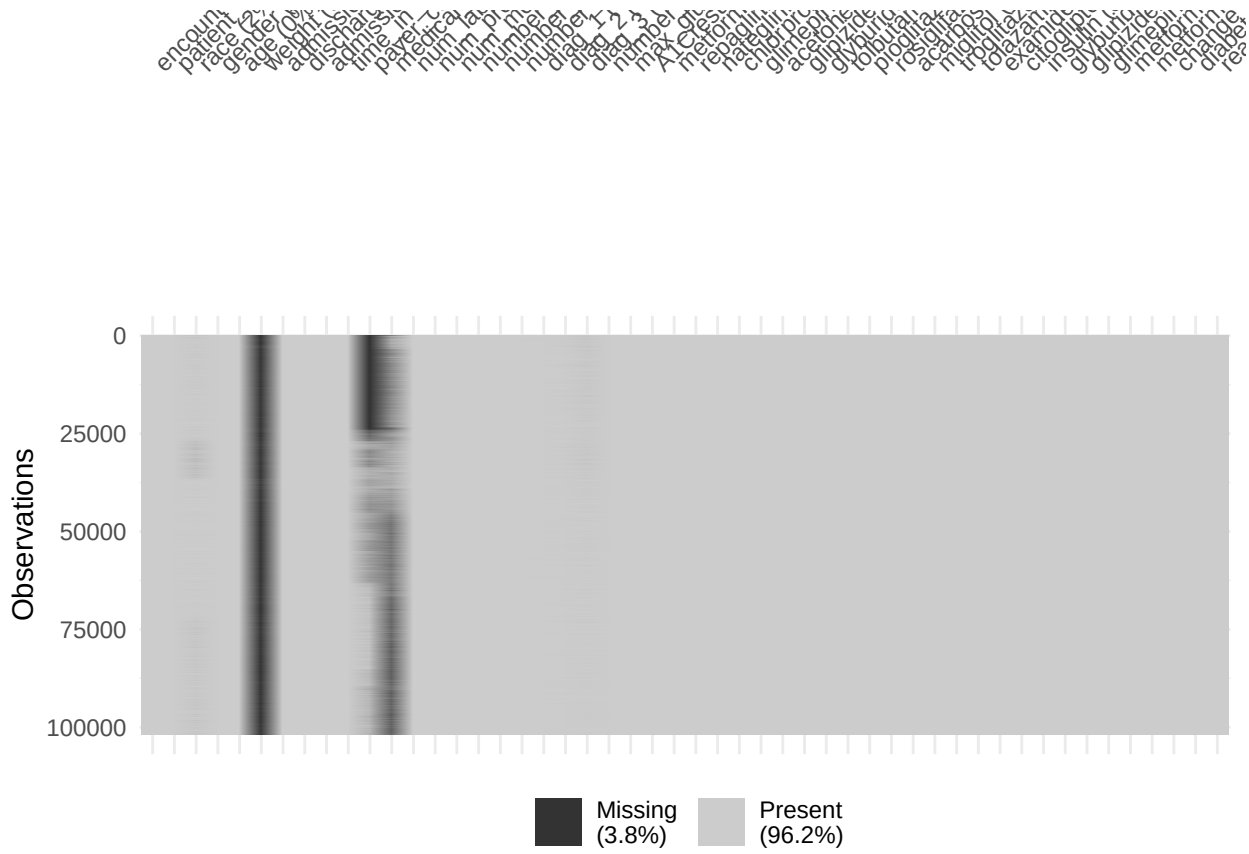


Figure 1: Missing values per attribute (depicted in black).

As shown in Table 2 and Figure 1, the `weight` attribute is almost entirely missing. For that reason, it is a strong candidate for removal. The same applies to `payer_code` and `medical_specialty`, where 39.56% and 49.08% of values are missing, respectively. Because `payer_code` does not add meaningful information for our research question, we remove it.

In contrast, `medical_specialty` may be informative because it captures the clinical context of the encounter. However, imputing or inferring missing values is risky given that nearly half of the entries are missing. Dropping all records with missing `medical_specialty` is also not feasible because it would remove roughly half of the dataset. Therefore, we encode missing values as `"Missing"`.

The attributes `race` and `diag_3` contain relatively small amounts of missing data. Still, removing these variables (or removing rows with missing values) could bias downstream analyses, so we apply the same strategy and encode missing values as `"Missing"`. The `gender` attribute contains only three missing values. Given how small this fraction is, we remove those records; additionally, `gender` is treated as a binary variable in most cases.

In the next section, we introduce a new attribute (`HbA1c` measurement) based on whether an A1C test was performed and whether its result was followed by a change in diabetes medication. The test was performed at the time of hospital admission. We define four encounter groups:

1. No HbA1c test performed;
2. HbA1c performed and within the normal range;
3. HbA1c performed and the result was greater than 8
4. HbA1c performed, the result was greater than 8

```
encounter.data <- encounter.data %>%
  mutate(hba1c_res=ifelse((A1Cresult=='None'), 'one', NA)) %>%
  mutate(hba1c_res=ifelse((A1Cresult %in% c('Norm', '>7'))
                          & is.na(hba1c_res), 'two', hba1c_res)) %>%
  mutate(hba1c_res=ifelse((A1Cresult=='>8') & (change=='No')
                          & is.na(hba1c_res), 'three', hba1c_res)) %>%
  mutate(hba1c_res=ifelse((A1Cresult=='>8')
                          & (change=='Ch')
                          & is.na(hba1c_res), 'four', hba1c_res))
encounter.data$hba1c_res <- as.factor(encounter.data$hba1c_res)
```

Our dataset contains multiple (near) zero-variance variables. For example, the attribute `examide` consists of a single level across all observations and has no missing values. Therefore, it is a zero-variance variable. Because zero-variance variables carry no information and cannot contribute to predicting the outcome, we remove them. We also remove all other (near) zero-variance variables, as keeping them would only increase computation time without improving model performance. In total, we removed the following eighteen attributes:

```
library(caret)
removal.names <- nearZeroVar(encounter.data, names = T)
encounter.data <- encounter.data %>%
  select(-c(removal.names))
```

Before we start analyzing our dataset, it is crucial to check for duplicates because the dataset contains multiple inpatient visits for some patients. These repeated observations are not statistically independent and can introduce noise. Therefore, we keep only one encounter per patient for downstream analyses.

```
duplicateCheck <- encounter.data %>%
  group_by(patient_nbr) %>%
  summarise(count = n()) %>%
  filter(count > 1)
```

```r
table.dup <- data.frame('Initial number of records' = nrow(encounter.data),
                        'Number of Duplicates' = nrow(duplicateCheck),
                        'Difference in percent' = round((
                          nrow(encounter.data) - nrow(duplicateCheck) -
                          nrow(encounter.data)) /
                          nrow(encounter.data) * 100, 2) )

kbl(table.dup, booktabs = T,
    caption = "The effects of removing data") %>%
  kable_styling(full_width = F, latex_options = "hold_position") %>%
  column_spec(1, bold = T)
```

Table 3: The effects of removing data

Initial.number.of.records	Number.of.Duplicates	Difference.in.percent
101766	16773	-16.48

Looking at Table 3, the initial dataset contains 101,766 records, of which 16,773 are duplicates. Removing these records would leave 84993 observations, corresponding to a loss of 16.48%. This is a reasonable trade-off to obtain a dataset in which observations are (approximately) statistically independent.

2.2 Categorical attributes

In this section, we examine variation within and between categorical attributes. The `gender` attribute has three levels (`"Female"`, `"Male"`, and `"Missing/Unknown"`). Because we include this variable in our analyses, we remove the `"Missing/Unknown"` category (i.e., exclude those records).

```
# First, we need to remove all records containing the third option
# of gender: 'Missing/unknown', as this can be lethal in further analysis
encounter.data <- encounter.data %>%
  select(everything()) %>%
  filter(gender != 'Unknown/Invalid') %>%
  droplevels()

# Count specific variables of interest
agg <- count(encounter.data, age, gender, A1Cresult, race)

# Specify a color per value for visualization
ecols <- c(Female = "pink", Male = "blue2")
p1 <- ggplot(agg) +
       geom_col(aes(x = race, y = n, fill = gender)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Race", y = "Total counts")
p2 <- ggplot(agg) +
       geom_col(aes(x = age, y = n, fill = gender)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Age", y = "Total counts")
ecols <- c(">7" = "blue", ">8" = "orange",
           "None" = "green", "Norm" = "yellow")
p3 <- ggplot(agg) +
       geom_col(aes(x = age, y = n, fill = A1Cresult)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Age", y = "Total counts")
```

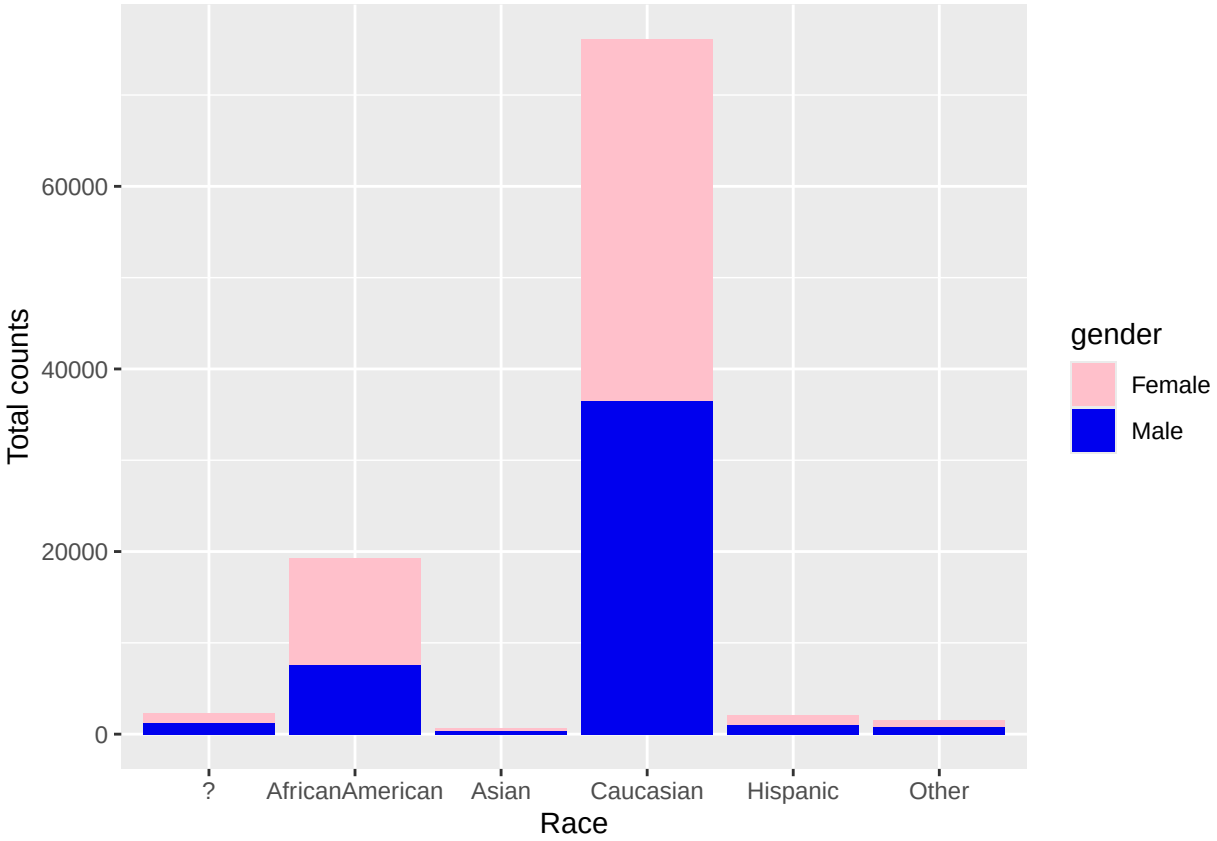


Figure 2: Race distributed with gender

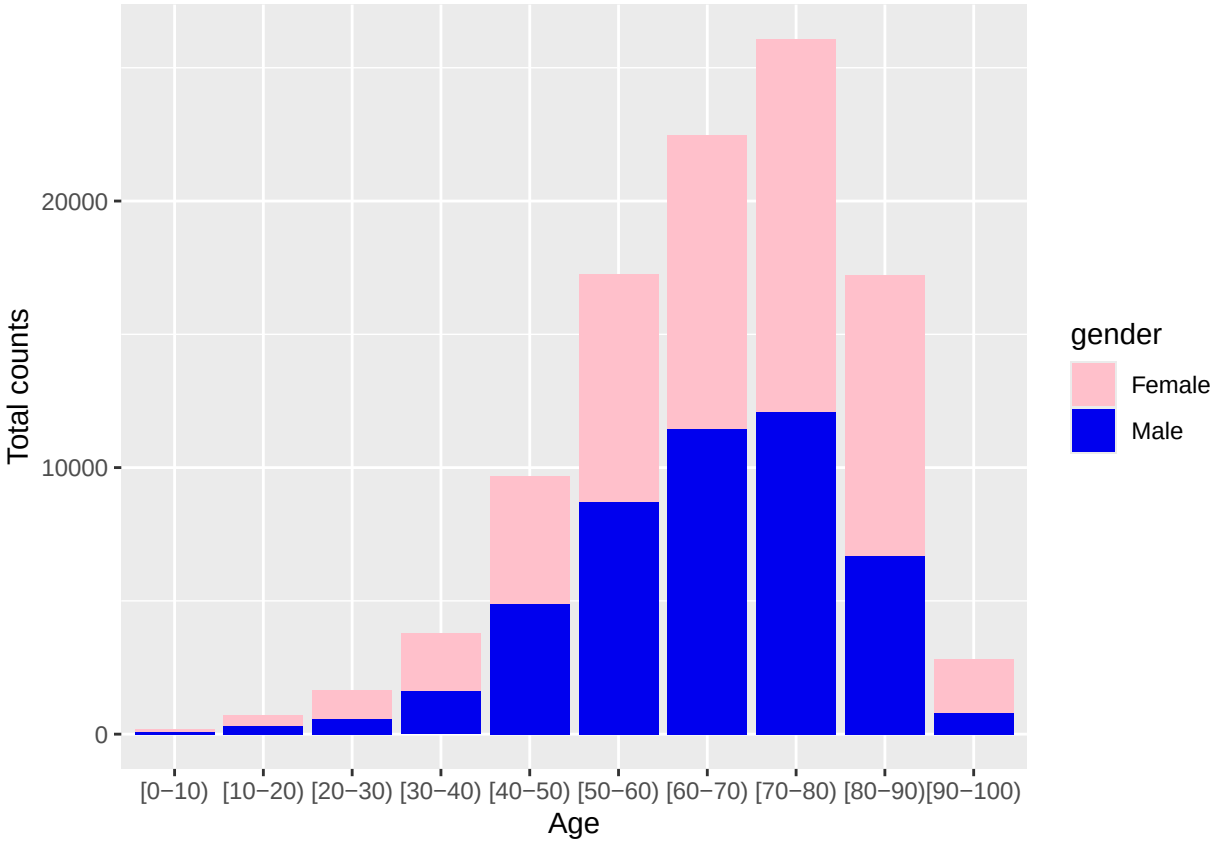


Figure 3: Age distributed with gender

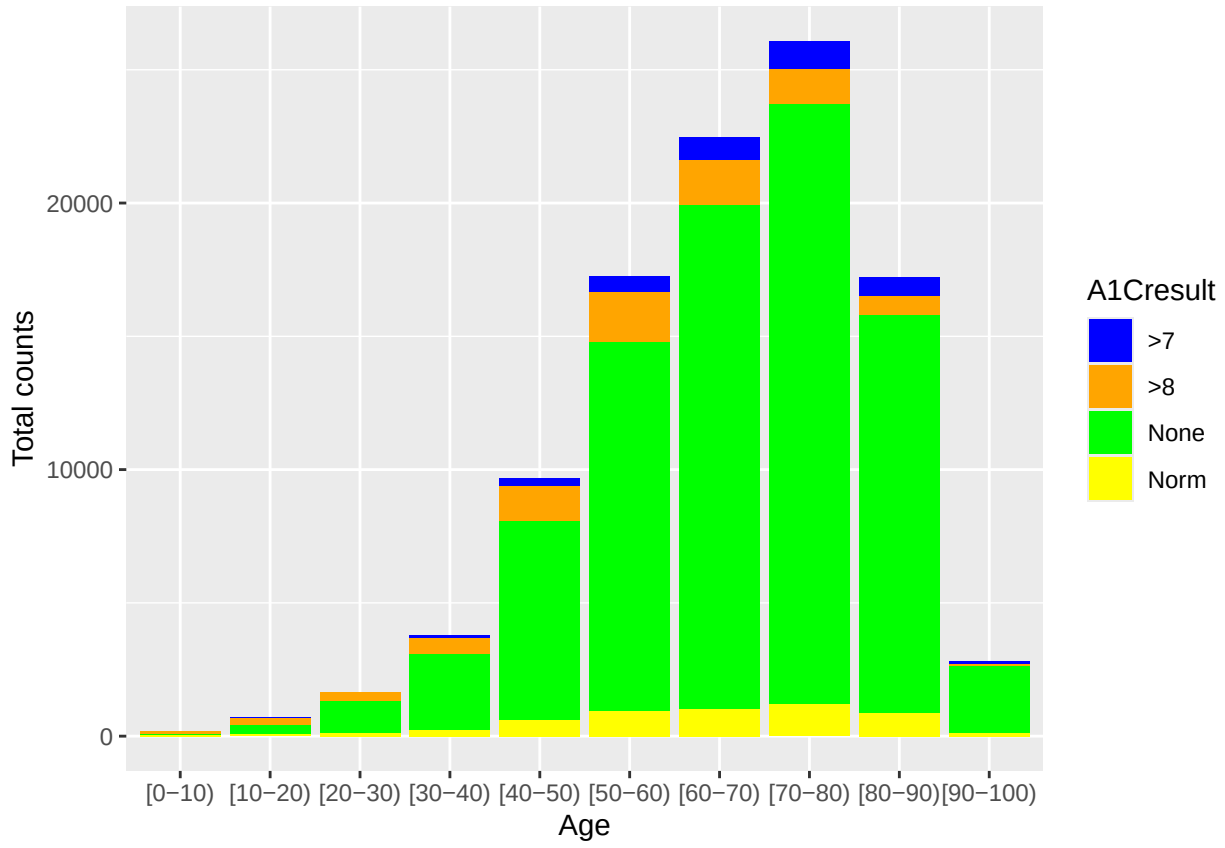


Figure 4: Age distributed with A1C test result

As shown in Figure 2, most patients are recorded as Caucasian, with an approximately balanced distribution between male and female patients. We expect the number of diabetes-related admissions to increase in older age groups. Figure 3 supports this expectation: the distribution is negatively skewed, with more encounters observed in the older age categories. A similar pattern is visible in Figure 4, where HbA1c testing becomes more common in older age groups. Importantly, Figure 4 also shows that for many encounters—most of them—the test was not performed, suggesting that HbA1c testing was not routinely integrated into hospital protocols during the study period.

However, Figure 4 does not distinguish between ICU and non-ICU encounters. The authors of the original analysis noted that ICU departments follow stricter protocols for diabetes-related testing. To compare ICU and non-ICU records, we construct a new attribute, `icu_or_non`, based on `admission_type_id`, `admission_source_id`, and `discharge_disposition_id`, and use it to classify encounters as ICU or non-ICU.

```
encounter.data <- encounter.data %>%
  # Removing dead patients
  filter(discharge_disposition_id != 11) %>%
  # Distinction between ICU and non-ICU patients
  mutate(admission_type_id =
```

```
          ifelse(admission_type_id %in% c(1, 2, 7),
                 'ICU patient', 'Non-ICU patient')) %>%
  mutate(admission_source_id =
          ifelse(admission_source_id %in% c(4, 7, 10, 12, 26),
                 'ICU patient', 'Non-ICU patient')) %>%
  mutate(discharge_disposition_id =
          ifelse(discharge_disposition_id %in% c(13, 14, 19, 20, 21),
                 'ICU patient', 'Non-ICU patient'))
# Now that we changed the labels, let us discover
# how many ICU and non-ICU patients we have
encounter.data <- encounter.data %>%
  # All columns need to be equal to each other
  mutate(icu_or_non = ifelse(admission_source_id == discharge_disposition_id
                             & admission_type_id == discharge_disposition_id,
                             admission_source_id,
                             ifelse(admission_source_id ==
                                    discharge_disposition_id,
                                    admission_source_id,
                                    admission_source_id)))
encounter.data$icu_or_non <- as.factor(encounter.data$icu_or_non)

agg <- count(encounter.data, gender, A1Cresult, icu_or_non, race)
ecols <- c(Female = "pink", Male = "blue2")
p1 <- ggplot(agg) +
       geom_col(aes(x = icu_or_non, y = n, fill = gender)) +
       scale_fill_manual(values = ecols) +
       labs(x ="ICU or non-ICU patient", y = "Total counts")

p2 <- ggplot(agg) +
       geom_col(aes(x = icu_or_non, y = n, fill = race)) +
       labs(x ="ICU or non-ICU patient", y = "Total counts")

ecols <- c('ICU patient' = 'blue', 'Non-ICU patient' = 'red')
p3 <- ggplot(agg) +
       geom_col(aes(x = A1Cresult, y = n, fill = icu_or_non)) +
       scale_fill_manual(values = ecols) +
       labs(x ="ICU or non-ICU patient", y = "Total counts")
```

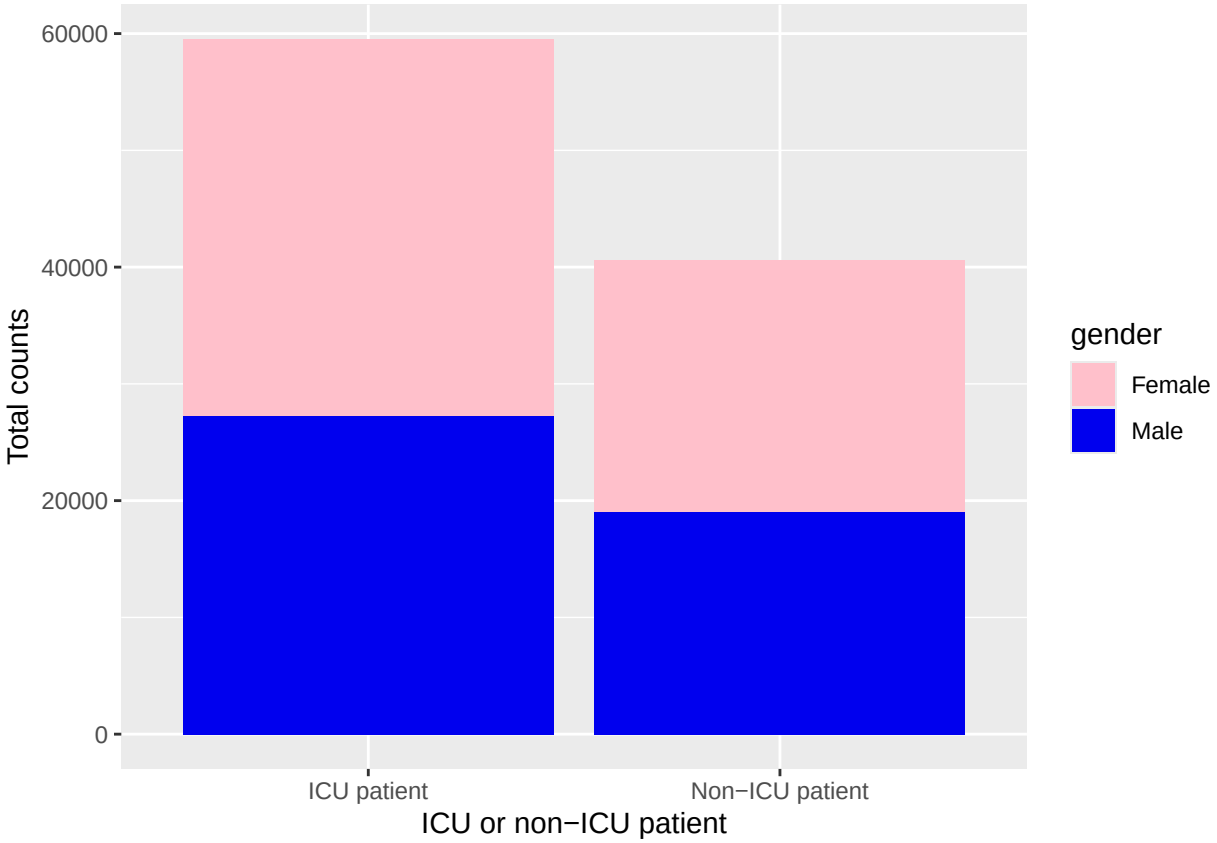


Figure 5: ICU statistics distributed with gender

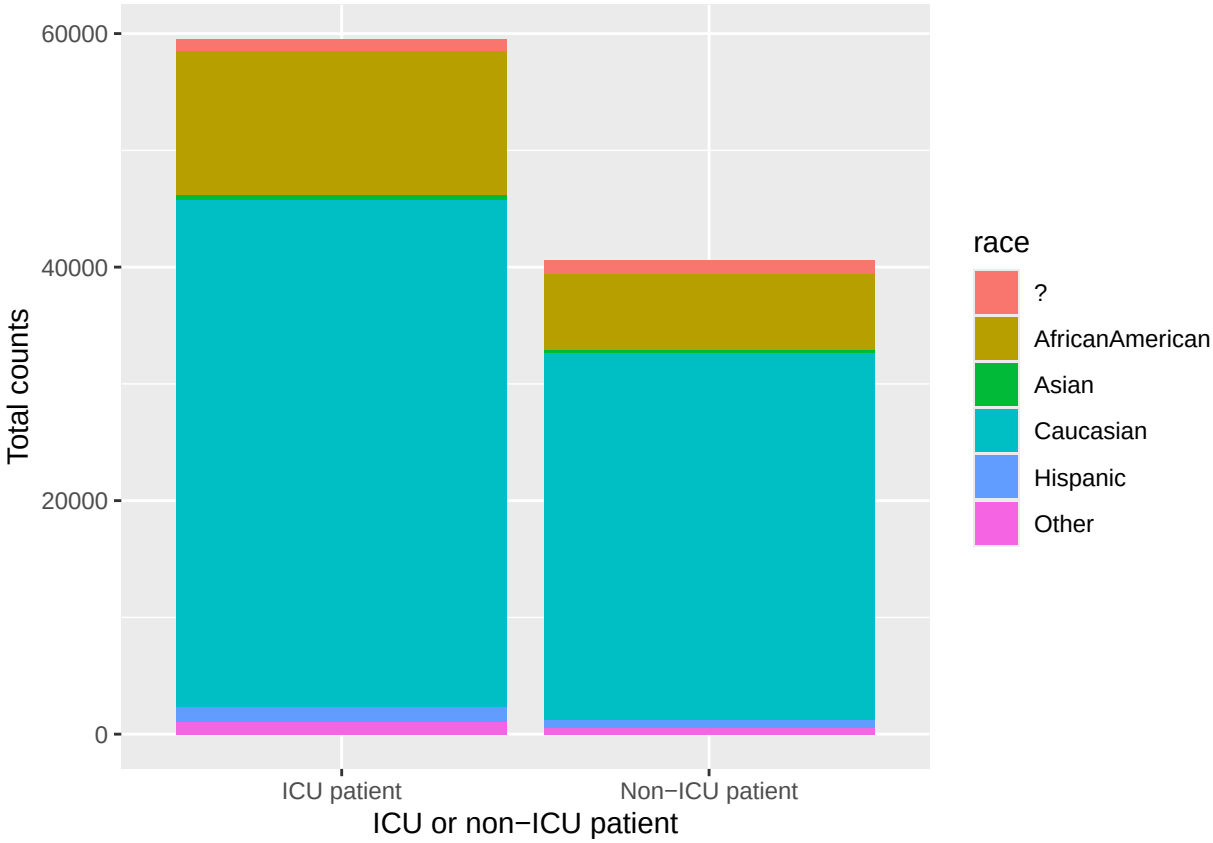


Figure 6: ICU statistics distributed with race

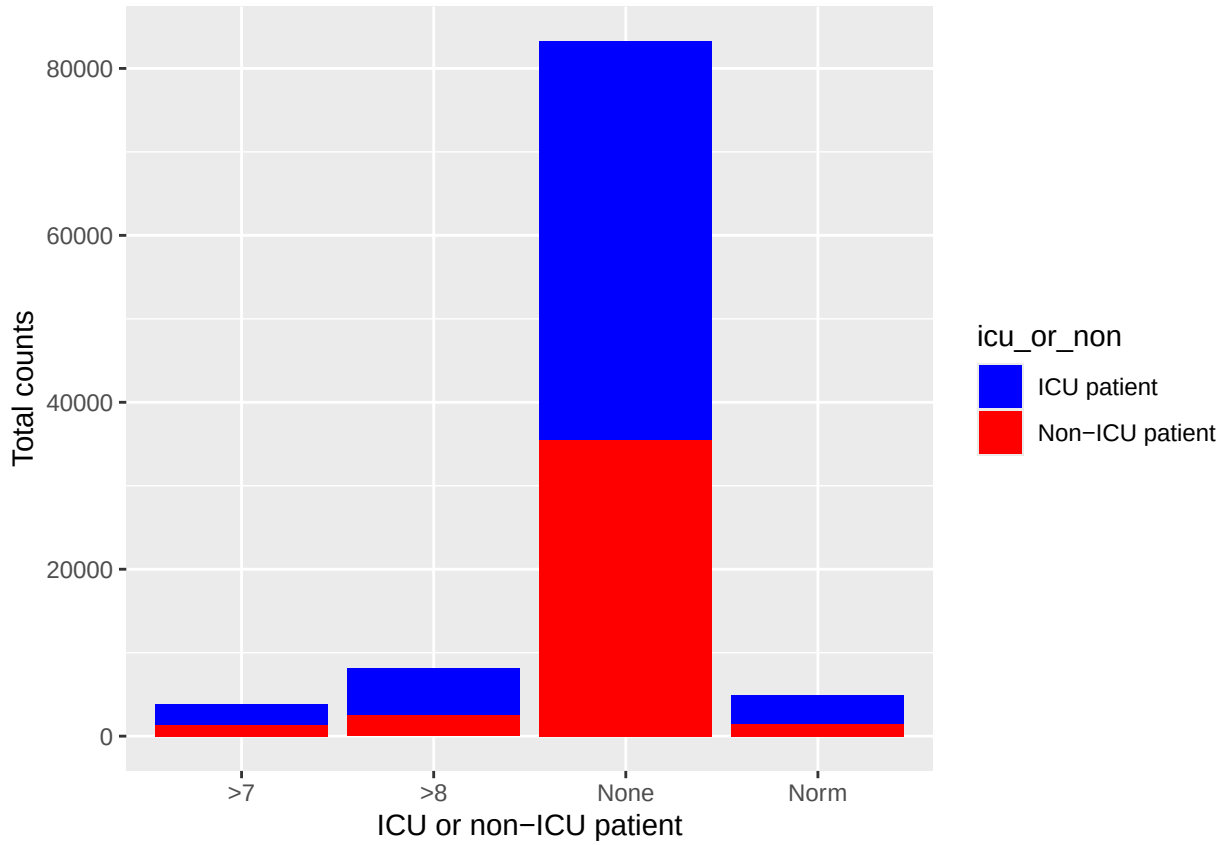


Figure 7: ICU statistics distributed with A1C test result

Figures 5, 6, and 7 compare ICU and non-ICU encounters. Across these figures, the ICU subgroup is smaller than the non-ICU subgroup. Despite this difference in sample size, the relative distributions remain similar: the gender ratio is comparable between groups, and Figure 6 shows a similar pattern for race. Figure 7 indicates that, regardless of encounter type (ICU or non-ICU), HbA1c testing is not consistently performed. This suggests that keeping the ICU vs. non-ICU distinction may not be essential for the descriptive analysis. However, it may still be useful for downstream machine learning because it is a simple binary feature and allows us to collapse three attributes into one.

The attributes `diag_1`, `diag_2`, and `diag_3` contain many ICD-9 diagnosis codes. Many of these codes can be grouped into clinically meaningful categories. In the following section, we construct a more interpretable representation of diagnoses by mapping ICD codes to broader groups. ICD code descriptions were retrieved from [2].

```
encounter.data <- encounter.data %>%
  mutate(diag_1 = case_when(diag_1 %in% c(390:459) ~ 'Circulatory',
                            diag_1 %in% c(460:519) ~ 'Respiratory',
                            diag_1 %in% c(520:579) ~ 'Digestive',
                            diag_1 %in% c(240:279) ~ 'Diabetes',
                            diag_1 %in% c(800:999) ~ 'Injury',
```

```
                              diag_1 %in% c(710:739) ~ 'Muscoloskeletal',
                              diag_1 %in% c(580:629) ~ 'Genitourinary',
                              TRUE ~ 'Others'))

encounter.data <- encounter.data %>%
  mutate(diag_2 = case_when(diag_2 %in% c(390:459) ~ 'Circulatory',
                              diag_2 %in% c(460:519) ~ 'Respiratory',
                              diag_2 %in% c(520:579) ~ 'Digestive',
                              diag_2 %in% c(240:279) ~ 'Diabetes',
                              diag_2 %in% c(800:999) ~ 'Injury',
                              diag_2 %in% c(710:739) ~ 'Muscoloskeletal',
                              diag_2 %in% c(580:629) ~ 'Genitourinary',
                              TRUE ~ 'Others'))
encounter.data <- encounter.data %>%
  mutate(diag_3 = case_when(diag_3 %in% c(390:459) ~ 'Circulatory',
                              diag_3 %in% c(460:519) ~ 'Respiratory',
                              diag_3 %in% c(520:579) ~ 'Digestive',
                              diag_3 %in% c(240:279) ~ 'Diabetes',
                              diag_3 %in% c(800:999) ~ 'Injury',
                              diag_3 %in% c(710:739) ~ 'Muscoloskeletal',
                              diag_3 %in% c(580:629) ~ 'Genitourinary',
                              TRUE ~ 'Others'))

# Convert to factors again
cols <- c('diag_1', 'diag_2', 'diag_3')
encounter.data[cols] <- lapply(encounter.data[cols], factor)
agg <- count(encounter.data, gender, diag_1, diag_2, diag_3)

# Specify a color per value for visualization
ecols <- c(Female = "pink", Male = "blue2")
p1 <- ggplot(agg) +
       geom_col(aes(x = diag_1, y = n, fill = gender)) +
       scale_fill_manual(values = ecols) +
      labs(x ="Primary diagnosis [category]", y = "Total counts")
p2 <- ggplot(agg) +
       geom_col(aes(x = diag_2, y = n, fill = gender)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Secondary diagnosis [category]", y = "Total counts")
p3 <- ggplot(agg) +
       geom_col(aes(x = diag_3, y = n, fill = gender)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Final diagnosis [category]", y = "Total counts")
```

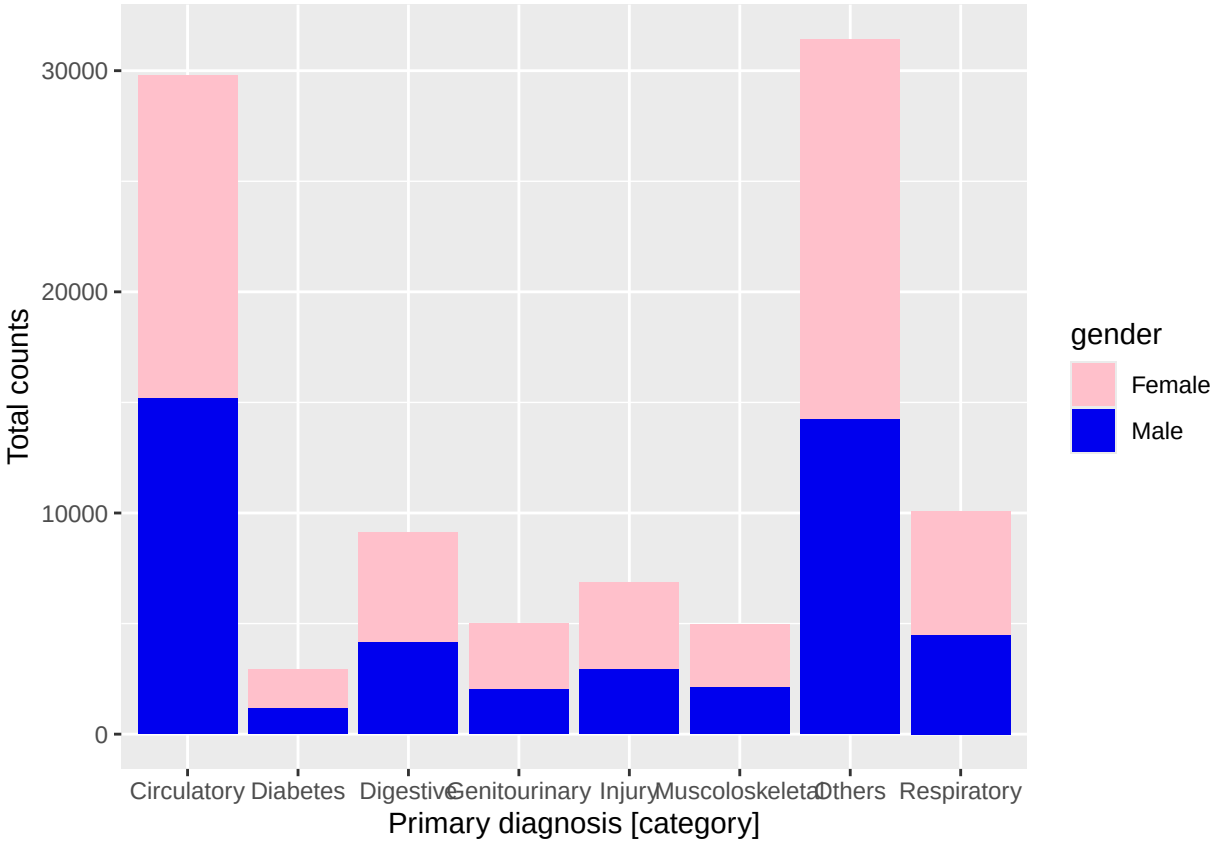


Figure 8: Primary diagnosis distributed with gender

Figure 9: Secondary diagnosis distributed with gender

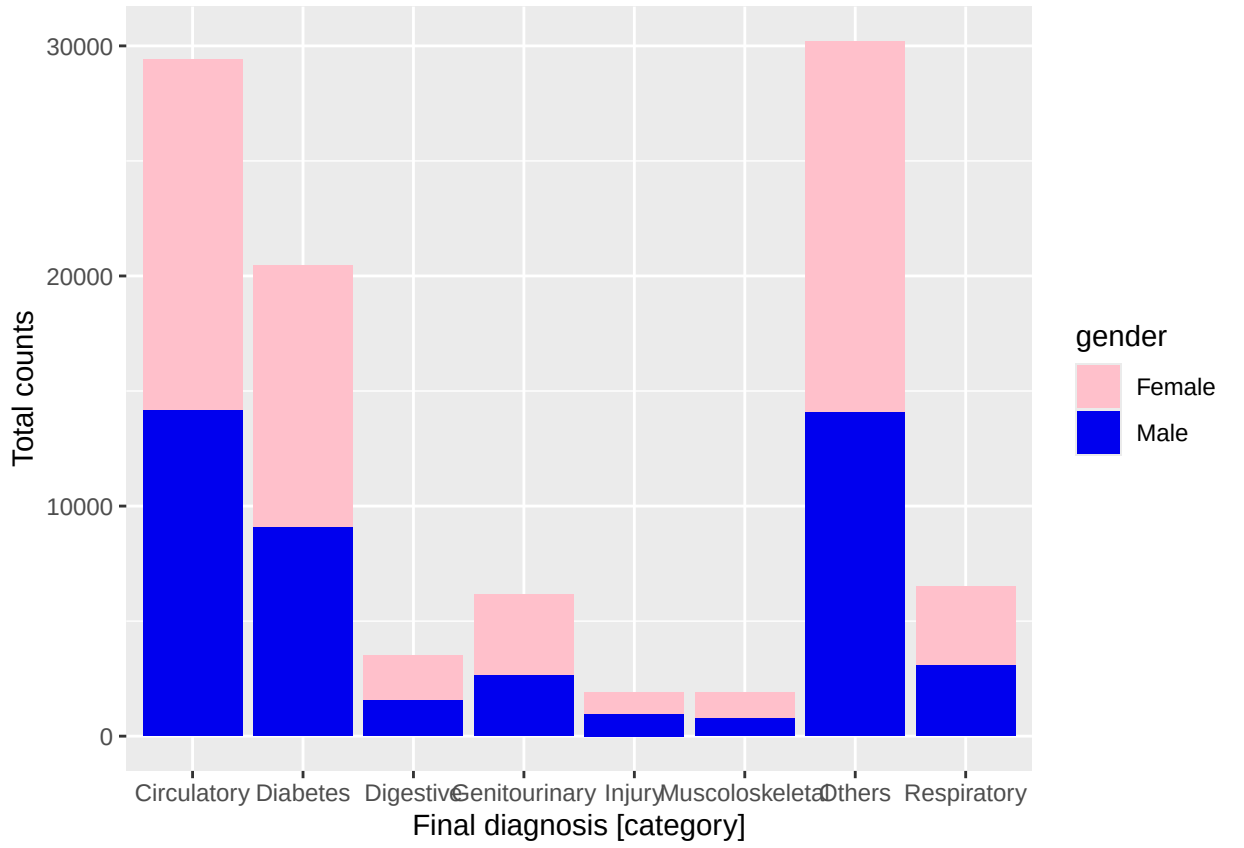


Figure 10: Final diagnosis distributed with gender

Figures 8, 9, and 10 show the results of recoding `diag_1`, `diag_2`, and `diag_3`. These attributes represent the primary, secondary, and tertiary diagnosis for an encounter, respectively. A notable difference across the three figures is the increasing proportion of diabetes-related diagnoses. One possible explanation is that when diabetes is not identified or prioritized during the initial admission, glycemic management may be suboptimal, which could contribute to higher readmission risk. Overall, the recoded diagnosis groups provide a clearer and more interpretable summary of diagnostic patterns and are likely to be useful in the final dataset.

The original authors did not retain `diag_2` and `diag_3`, likely to reduce feature complexity. Whether to remove these two attributes remains an open question. At this stage, the EDA does not provide a clear indication that they should be dropped.

In the next section, we examine `medical_specialty` and evaluate whether it is informative enough—given its high fraction of missing values—to include in the final dataset. We create several plots that focus on the categories and frequencies of this attribute.

```
library(pals)

# Count data we want to compare
```

```r
agg <- count(encounter.data, age, medical_specialty, gender)

# Determine a color scheme for 71 values
ecols <- c(alphabet(26), cols25(25), glasbey(22))

# Order variables from high to low via '-n'
agg_ord <- mutate(agg,
                  age = reorder(age, -n, sum),
                  medical_specialty = reorder(medical_specialty, -n, sum))
# p1: bar plot combined, and p2: per gender
p1 <- ggplot(agg_ord) +
      geom_col(aes(x = age, y = n, fill = medical_specialty)) +
      scale_fill_manual(values = ecols) +
      theme(legend.position = 'none') +
      labs(x ="Age [group]", y = "Total counts")

p1 <- p1 + facet_wrap(~gender) +
      theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
# Do the same as above, only now with pie charts
p2 <- ggplot(agg_ord) +
      geom_col(aes(x = 1, y = n, fill = medical_specialty),
               position = "fill") +
      coord_polar(theta = "y") +
      scale_fill_manual(values = ecols) +
      theme(legend.position = 'none')
p2 <- p2 + facet_wrap(~gender) +
    theme_bw() +
    theme(legend.position = 'none')
```

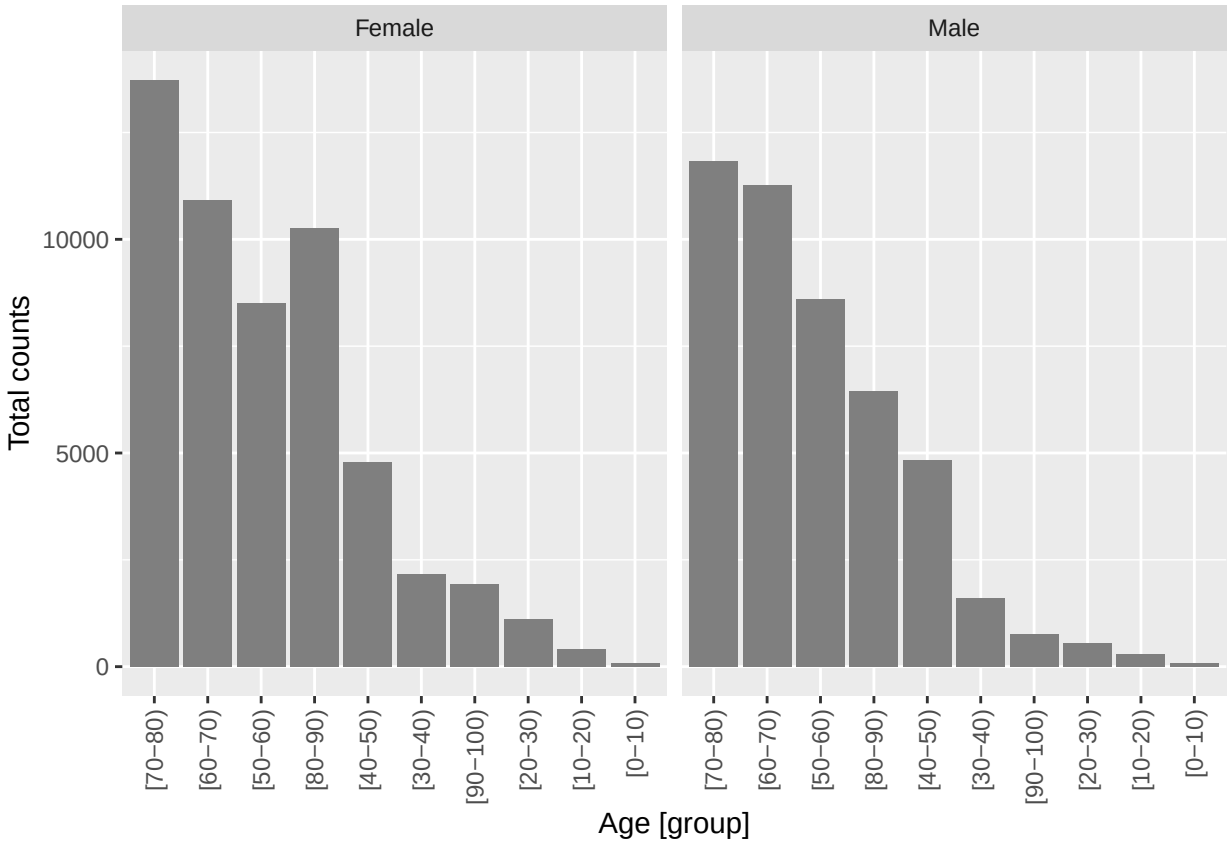


Figure 11: Medical specialty distributed with age groups and divided into gender

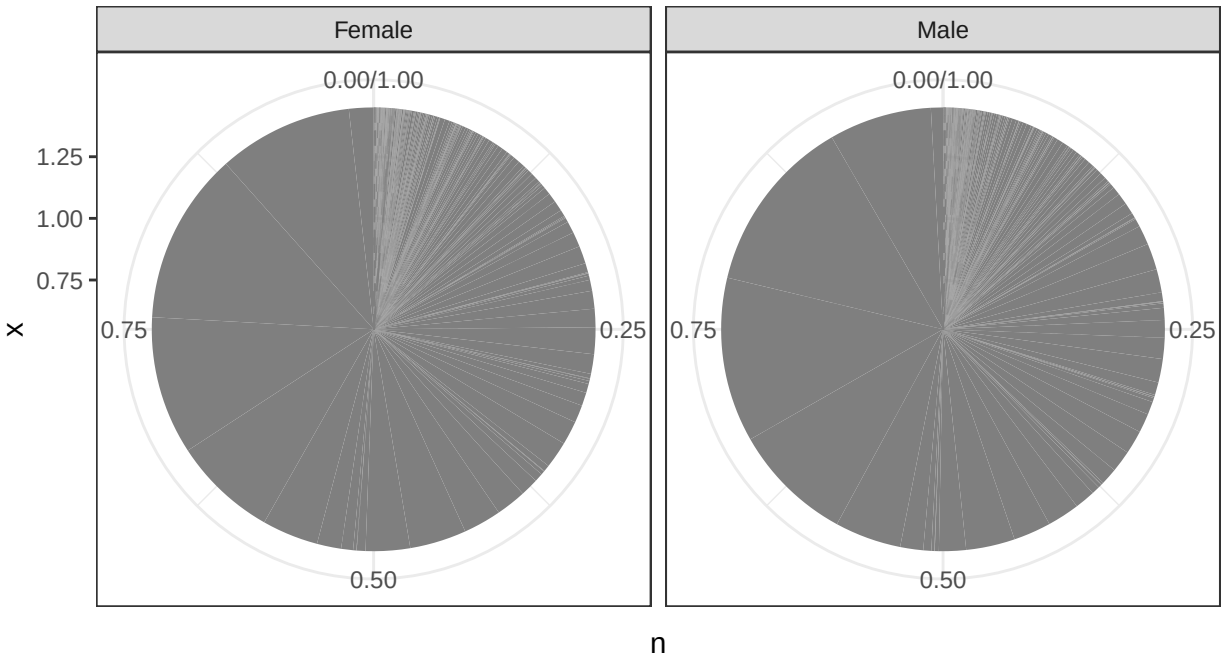


Figure 12: Medical specialty distributed with age groups and divived into gender

Looking at Figures 11 and 12, we observe substantial variation across the demographic attributes. The missing values, shown in purple, account for nearly 50% of all records, as established earlier. Unlike attributes such as `admission_id` (which can be collapsed into a smaller set of categories), `medical_specialty` is difficult to meaningfully simplify without losing a lot of detail. This raises the question of whether the variable adds information beyond what is already captured by the diagnosis codes. Because the diagnosis attributes likely provide equal or even more clinically relevant context for an encounter, removing `medical_specialty` at this stage would probably not be harmful.

As our final categorical attribute—and one of the most influential variables according to the original authors—we discuss `readmitted`. This variable has three levels: `"<30"` for readmission within 30 days after discharge, `">30"` for readmission more than 30 days after discharge, and `"NO"` for no readmission.

```
agg <- count(encounter.data, readmitted, race, gender, age, diag_1, diag_2)
ecols <- c('<30' = 'red', '>30' = 'blue', 'NO' = 'pink')

p1 <- ggplot(agg) +
       geom_col(aes(x = race, y = n, fill = readmitted)) +
       scale_fill_manual(values = ecols) +
```

```
       labs(x ="Race [group]", y = "Total counts")
p1 <- p1 + facet_wrap(~gender) +
       theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
p2 <- ggplot(agg) +
       geom_col(aes(x = age, y = n, fill = readmitted)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Age [group]", y = "Total counts")
p2 <- p2 + facet_wrap(~gender) +
       theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
p3 <- ggplot(agg) +
       geom_col(aes(x = diag_1, y = n, fill = readmitted)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Primary diagnosis [category]", y = "Total counts")
p3 <- p3 + facet_wrap(~gender) +
       theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
p4 <- ggplot(agg) +
       geom_col(aes(x = diag_2, y = n, fill = readmitted)) +
       scale_fill_manual(values = ecols) +
       labs(x ="Secondary diagnosis [category]", y = "Total counts")
p4 <- p4 + facet_wrap(~gender) +
       theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```

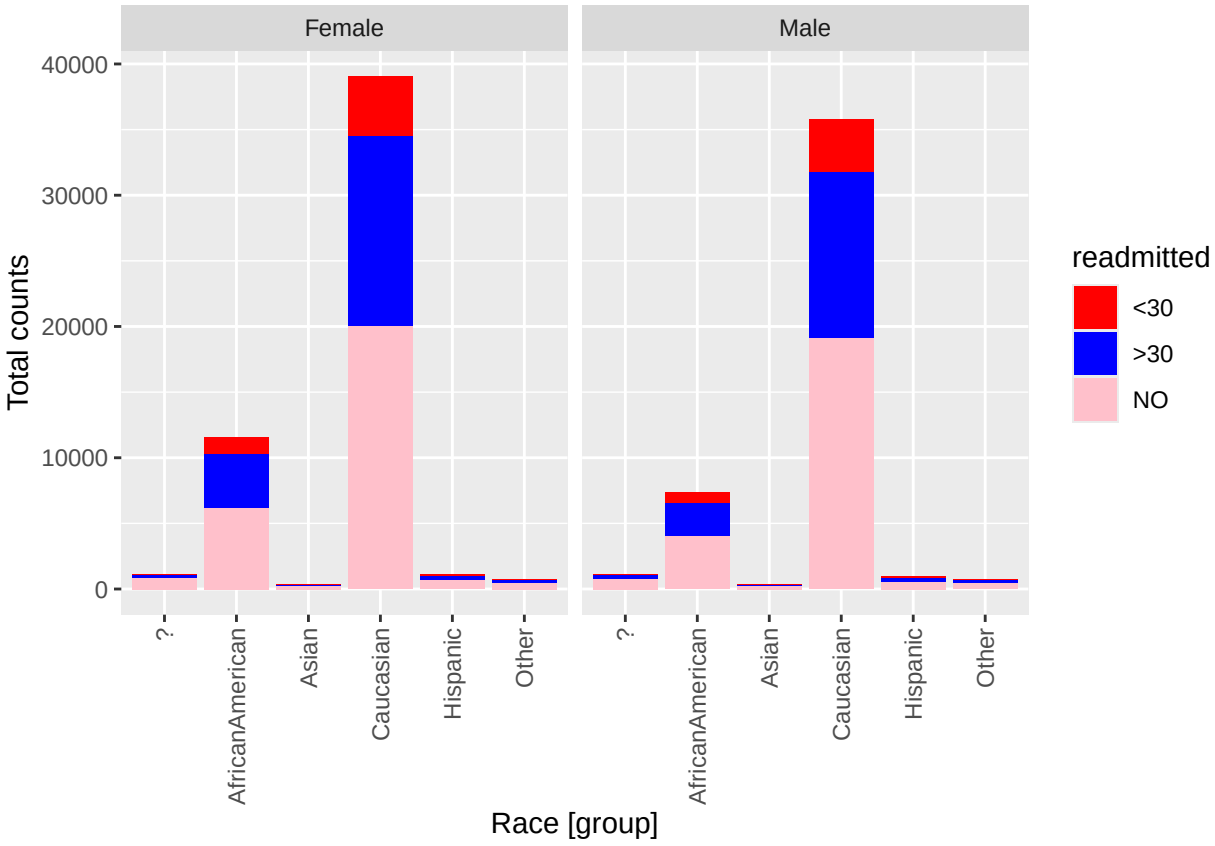


Figure 13: Readmitted distributed with race and divided into gender

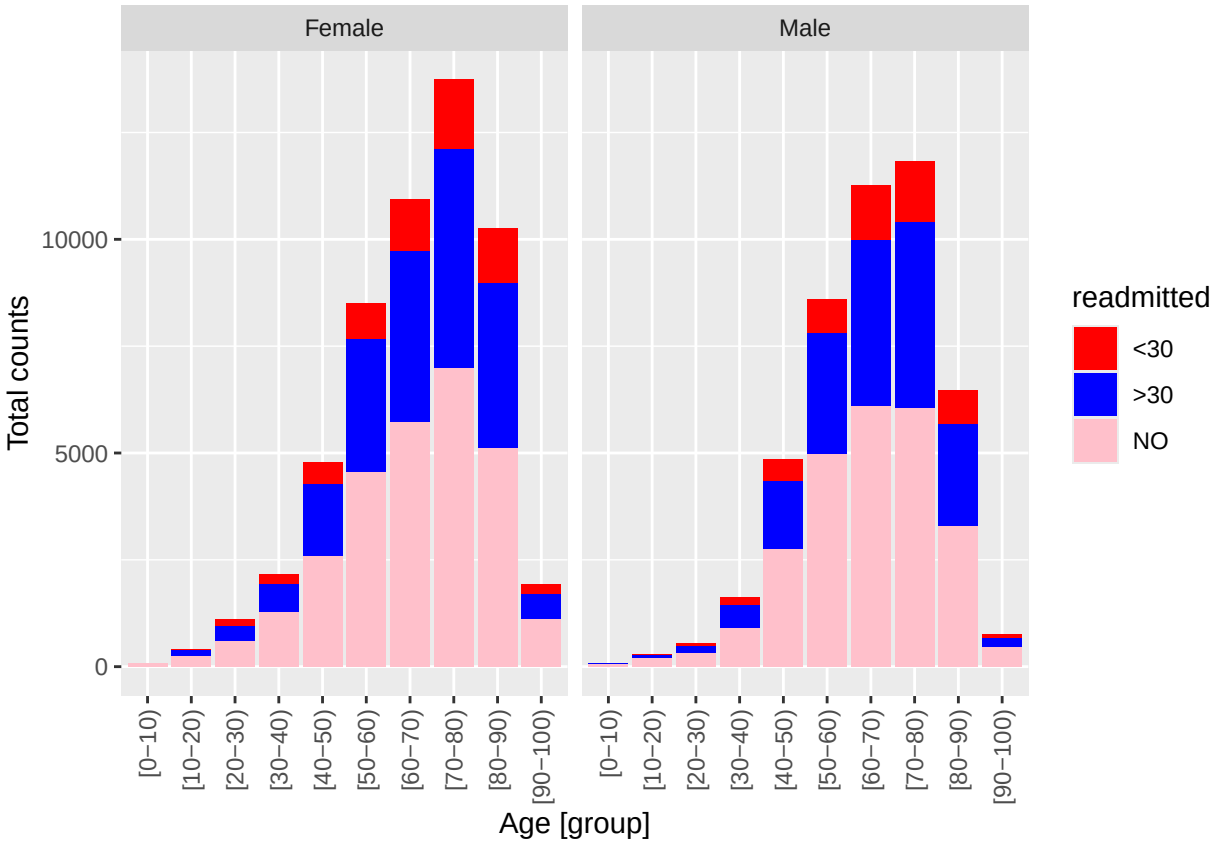


Figure 14: Readmitted distributed with age and divided into gender

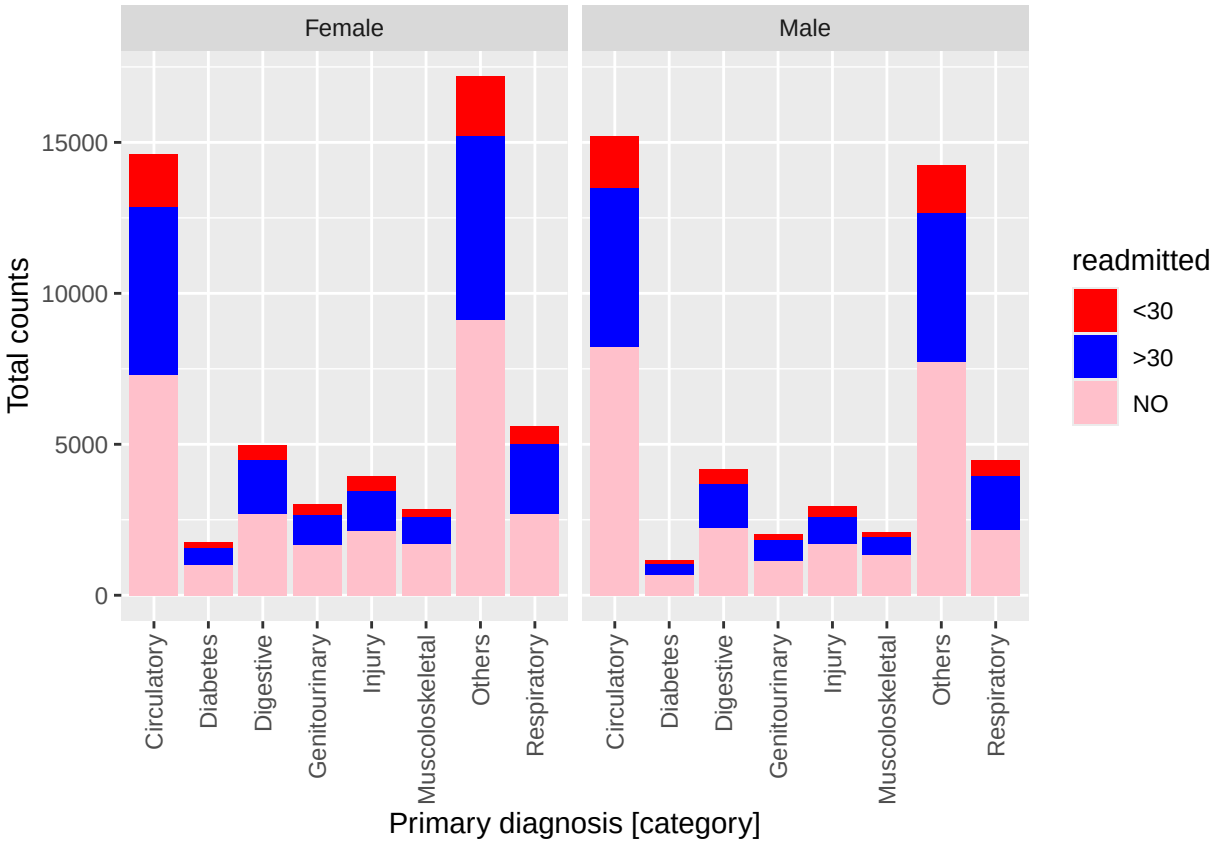


Figure 15: Readmitted distributed with primary diagnosis and divided into gender

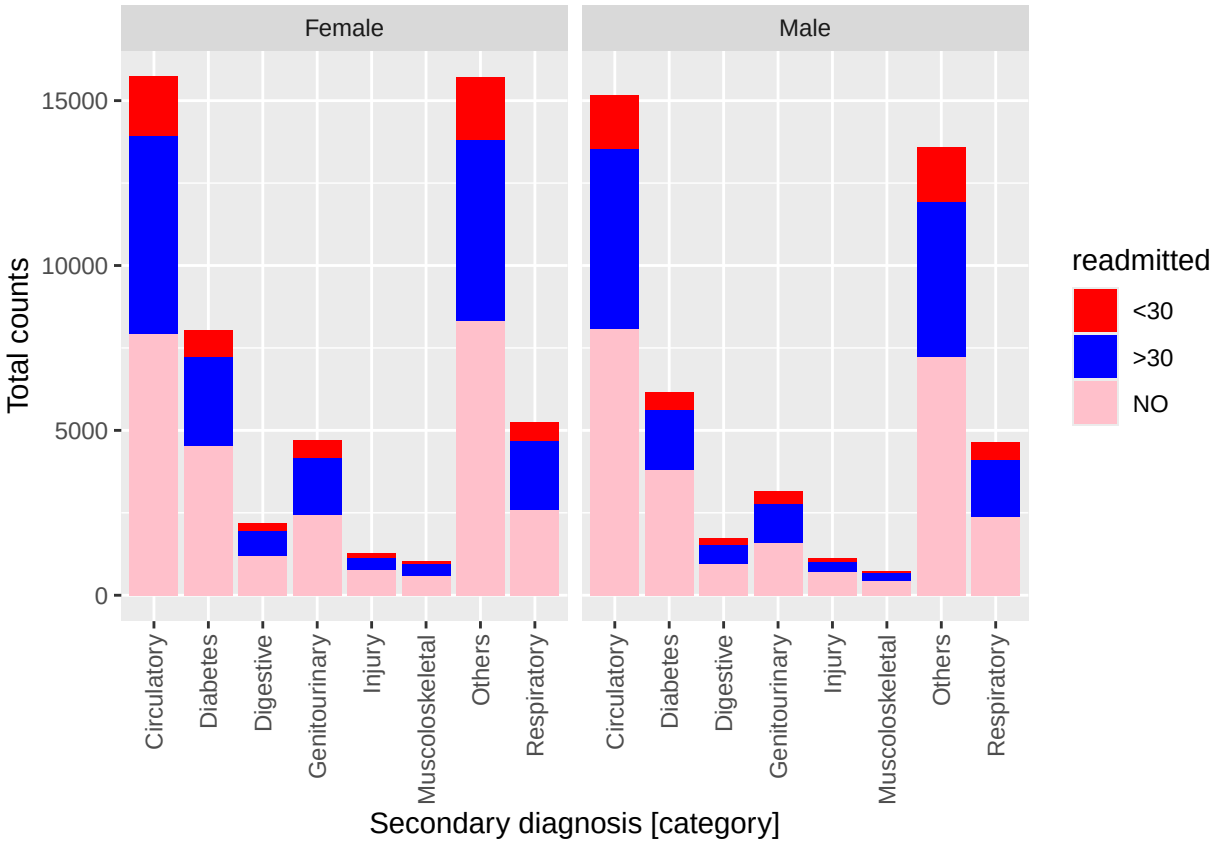


Figure 16: Readmitted distributed with secondary diagnosis and divided into gender

We next analyze the `readmitted` attribute by stratifying it across demographics (gender, age, and race) and two of the three diagnosis attributes. Figure 13 shows that Caucasian patients account for the largest share of readmissions in both gender groups, which likely reflects the underlying class imbalance in the dataset. Figure 14 shows that readmissions generally increase with age for both female and male patients, which is consistent with increased frailty and comorbidity at older ages. Readmission rates appear to decrease after age 80. One possible explanation is increased mortality or discharge to hospice or long-term care facilities in the oldest age groups.

Comparing Figures 15 and 16, we observe a familiar pattern: the proportion of diabetes-related diagnoses increases from the primary to the secondary diagnosis. This may indicate that many patients are admitted for another primary condition, while diabetes is more often recorded as a comorbidity (and therefore appears more frequently in secondary diagnoses).

Because we are primarily interested in diabetes-related effects, one option would be to collapse diagnosis categories into a binary variable (e.g., `"diabetes"` vs. `"other"`). However, this simplification may remove clinically relevant information and could affect downstream analyses.

2.3 Distribution – numeric attributes

After examining categorical variables, we now turn to the numeric attributes. Inspecting their distributions is important, especially to identify variables with extreme ranges or heavy skew, which may require normalization or transformation. We therefore generate a set of histograms using the `histogram.plotter()` function.

```
# Define a histogram plotter to construct multiple figures really quick
histogram.plotter <- function(attribute_1, number, attribute_name){
  histogram <- ggplot() +
                geom_bar(mapping = aes(x=attribute_1)) +
                ggtitle(number) +
                xlab(attribute_name)
}

# Make a smaller dataset so that only the numeric data is collected
numeric.data <- select_if(encounter.data, is.numeric)
# Remove encounter and patient ID
numeric.data <- numeric.data[, -c(1, 2)]
smaller.codebook <- codebook[c(10, 13:18, 22), 2]
numeric.data.log <- log2(numeric.data + 0.1)

wrapper <- function(data){
  p <- list()
  for ( i in c(1:length(data)) ){
    p[[i]] <- histogram.plotter(data[, i], LETTERS[i], smaller.codebook[i])
  }
  return(p)
}
numeric.normal <- wrapper(numeric.data)
numeric.log <- wrapper(numeric.data.log)
```

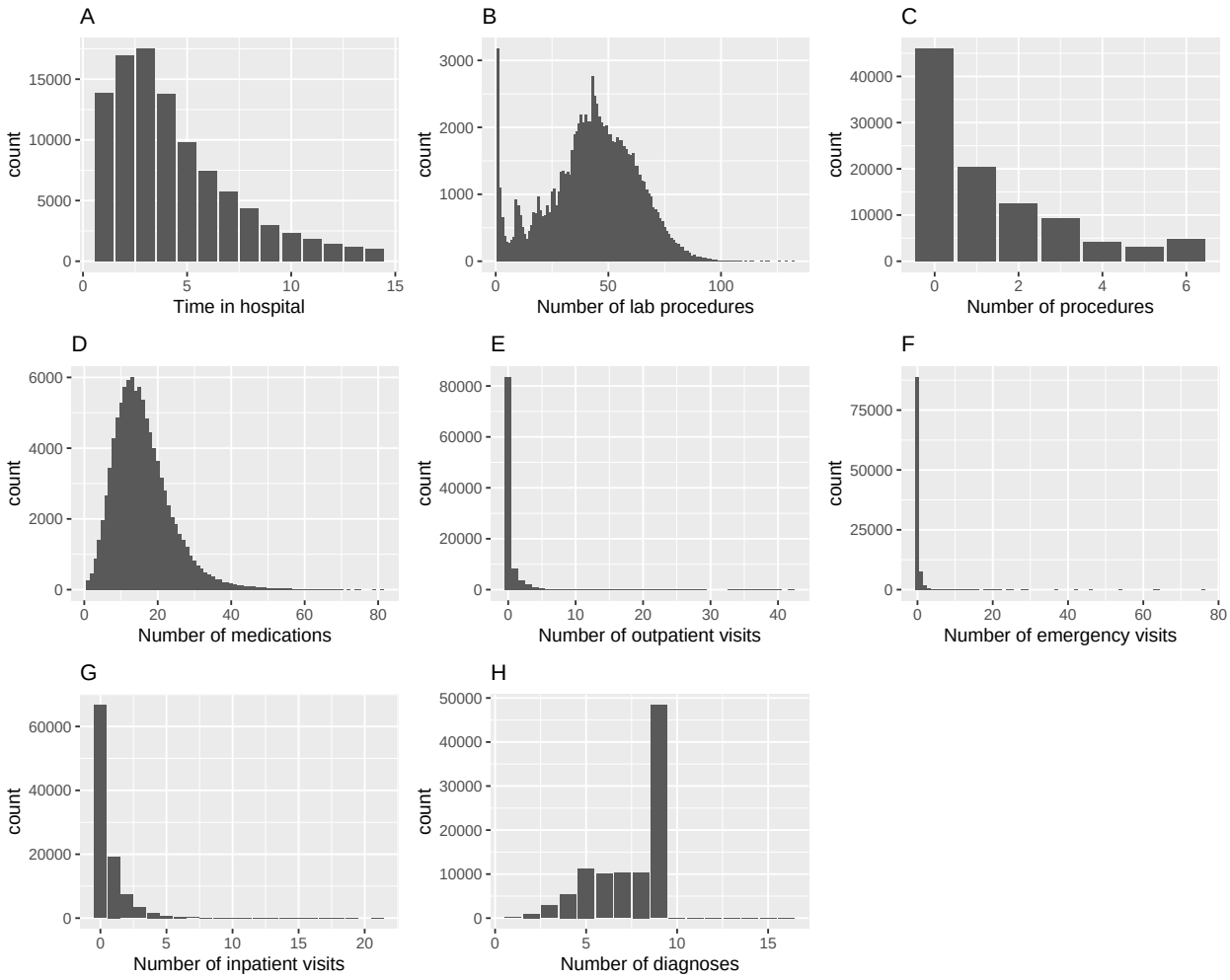


Figure 17: Numeric data presented in histograms without any normalization

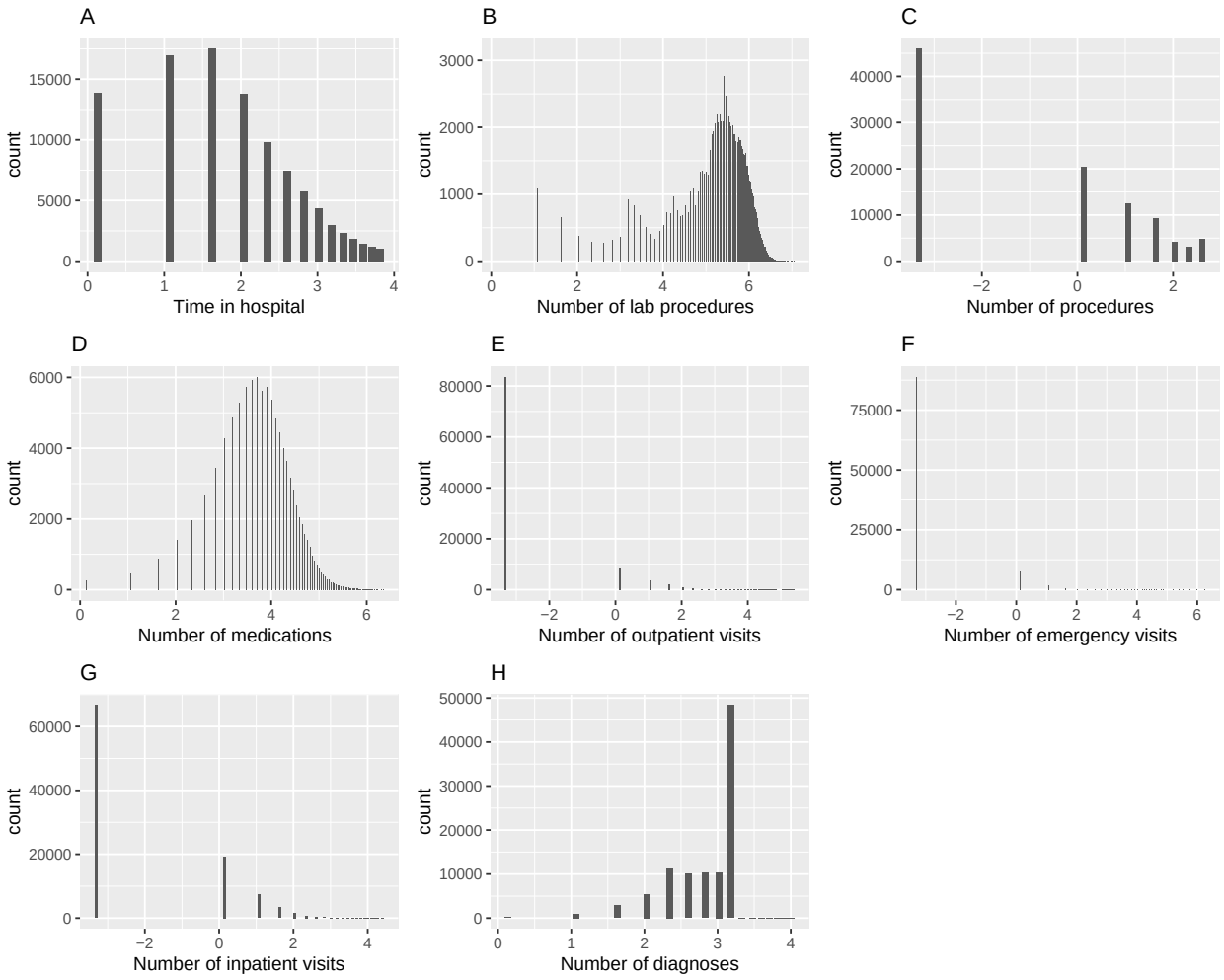


Figure 18: Numeric data presented in histograms on a log-2 scale

The results shown in Figures 17 and 18 indicate that several numeric attributes are not evenly distributed. Some form of normalization or transformation may be required, particularly for variables such as B (which shows several unusual spikes) and E, F, and G, which represent visit counts per encounter. As an alternative (or complement), we may introduce a new attribute, `total_visits`, defined as the total number of visits associated with an encounter (e.g., by summing the relevant visit-count variables). The most appropriate normalization approach is still under discussion and will depend on the modeling strategy.

2.4 Correlation – numeric attributes

To explore correlations among the numeric variables, we constructed the heatmap shown below. In particular, we assess whether age is associated with length of hospital stay, and we examine the relationship between `num_lab_procedures` and `num_medications`.

```
library(reshape2)
melted.data <- numeric.data %>%
  cor() %>%
  melt()
```

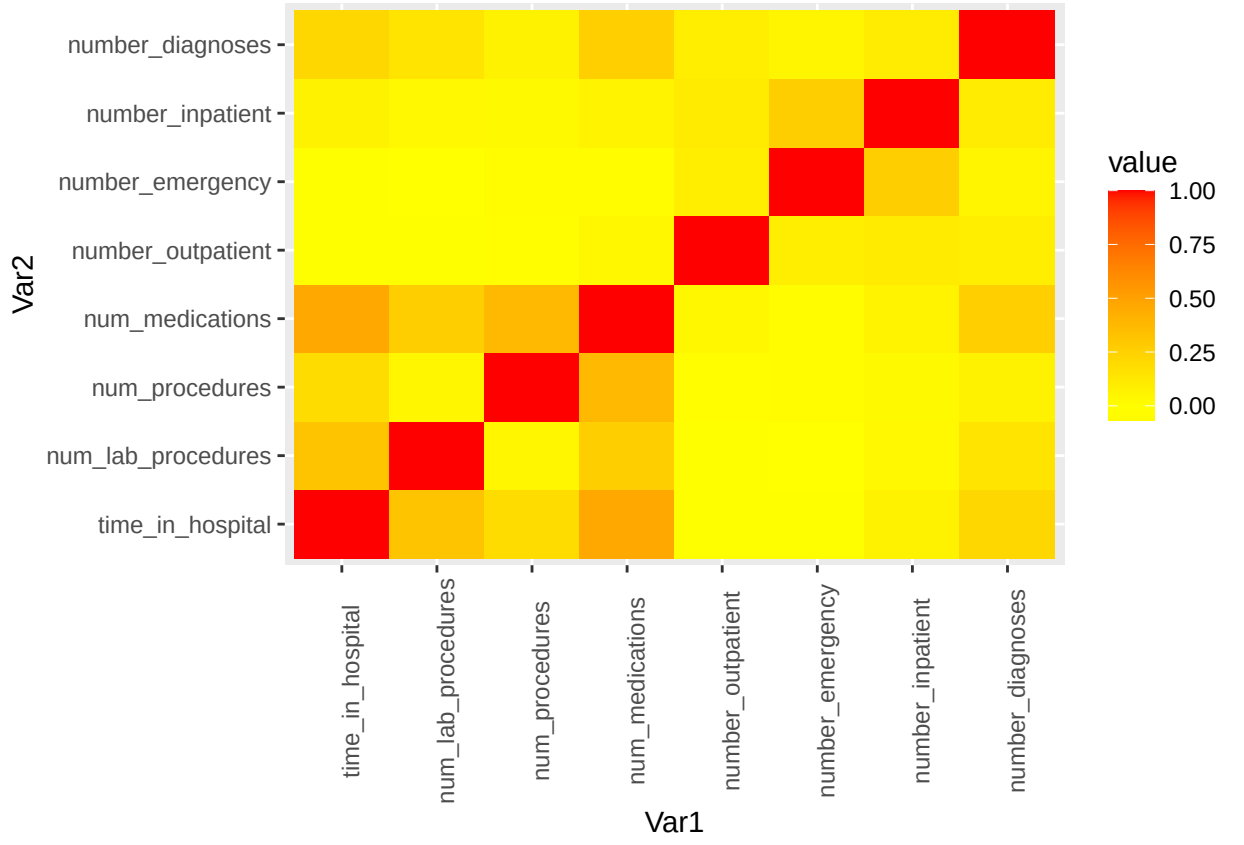


Figure 19: Heatmap of numeric attributes, with red depicted as a strong correlation, orange as a (small) correlation, and yellow as no correlation

Figure 20: Attribute time spent in hospital plotted against age group

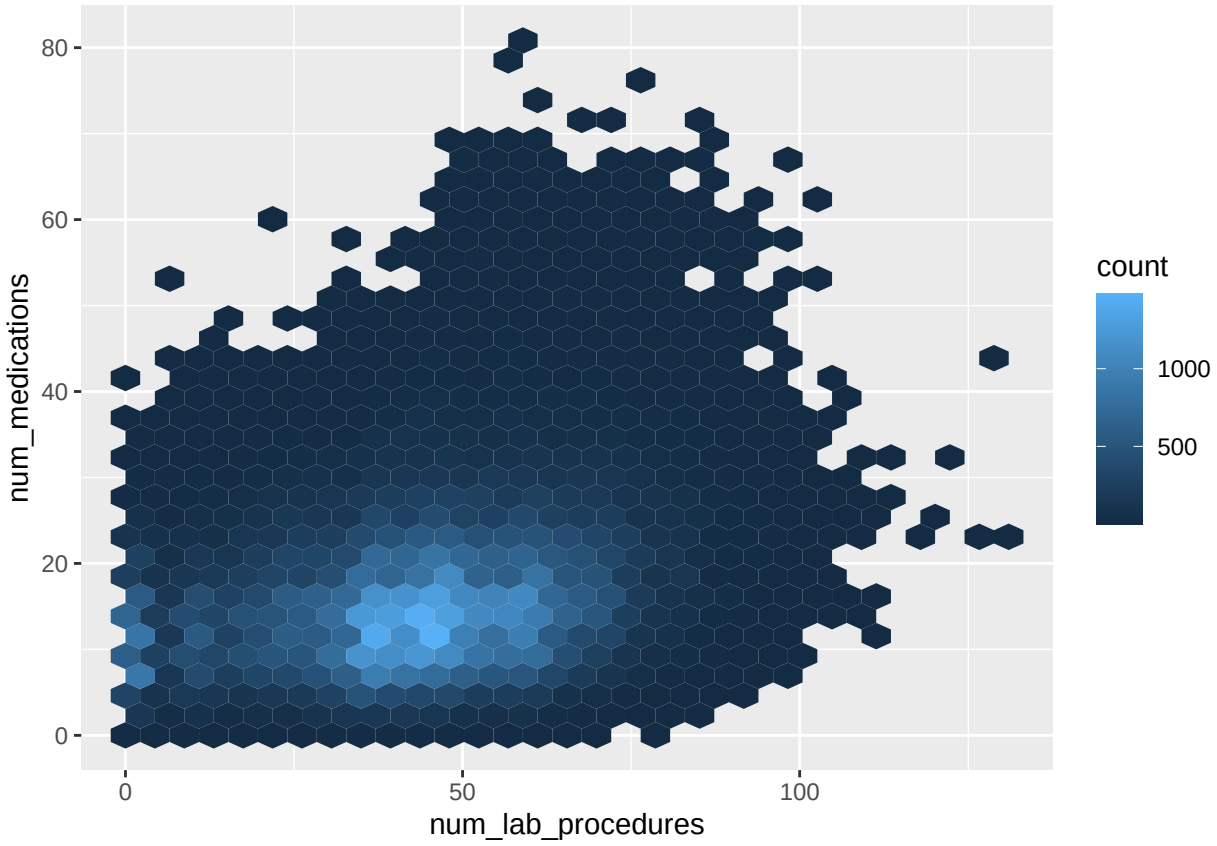


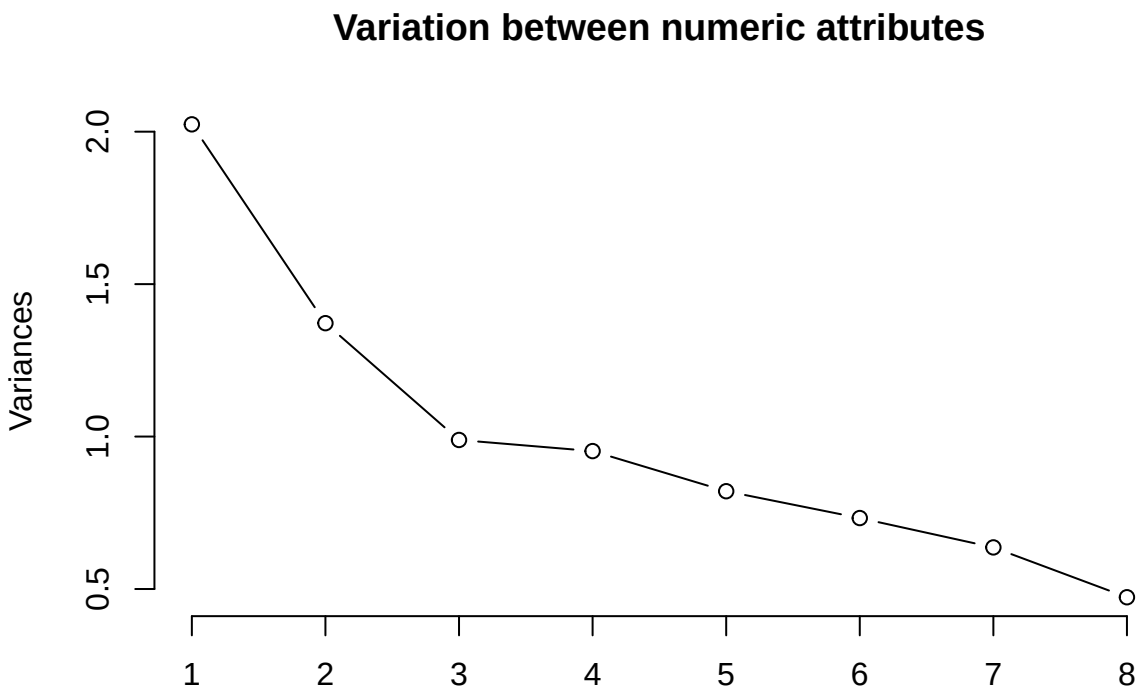
Figure 21: Attribute num_medications and num_lab_procedures plotted against each other

Our heatmap (Figure 19) visualizes strong correlations in red, weak correlations in orange, and little to no correlation in yellow. Several relationships stand out, including `num_medications` vs. `time_in_hospital`, `num_procedures` vs. `num_medications`, and `hba1c_res` vs. `num_medications`. Overall, `num_medications` is the variable that shows the most consistent associations with other numeric features, while most other pairs do not exhibit notable correlations. Based on this heatmap, the majority of numeric attributes appear to be weakly correlated or uncorrelated.

In Figure 20, we observe that age groups above `[40-50)` have higher total counts and, correspondingly, a higher total time spent in the hospital before discharge. This suggests that `time_in_hospital` increases with patient age. Next, we zoom in on a pair of variables with a small correlation in Figure 19. Figure 21 shows `num_medications` against `num_lab_procedures`: most encounters fall around 15–20 medications and 40–60 lab procedures. The scatter also contains many outliers, which may be addressed through transformation or normalization.

Next, we assess whether the class variable forms separable clusters or shows substantial overlap between instances. This matters because it provides an early indication of how predictable the target may be. We first examine how much variance is explained by principal components derived from the numeric variables (Figure 21). As expected, the explained variance decreases with each additional principal component. In the PCA plot (Figure 22), there is substantial overlap between classes, suggesting that separating the outcome using only these numeric features may be challenging and that more complex models and/or additional informative features may be required.

```
cc.pca <- prcomp(numeric.data,
                 center = TRUE,
                 scale. = TRUE)
```


Variation between numeric attributes
Variances
0.5
1.0
1.5
2.0
1
2
3
4
5
6
7
8

```
library(ggbiplot)
ggbiplot(cc.pca, obs.scale = 1, var.scale = 1,
         groups = as.factor(encounter.data$time_in_hospital),
         ellipse = TRUE, circle = TRUE, alpha = 0.5) +
  scale_color_discrete(name = '') +
  theme(legend.direction = 'horizontal', legend.position = 'top') +
  xlim(-5, 20) + ylim(-10, 50)
```

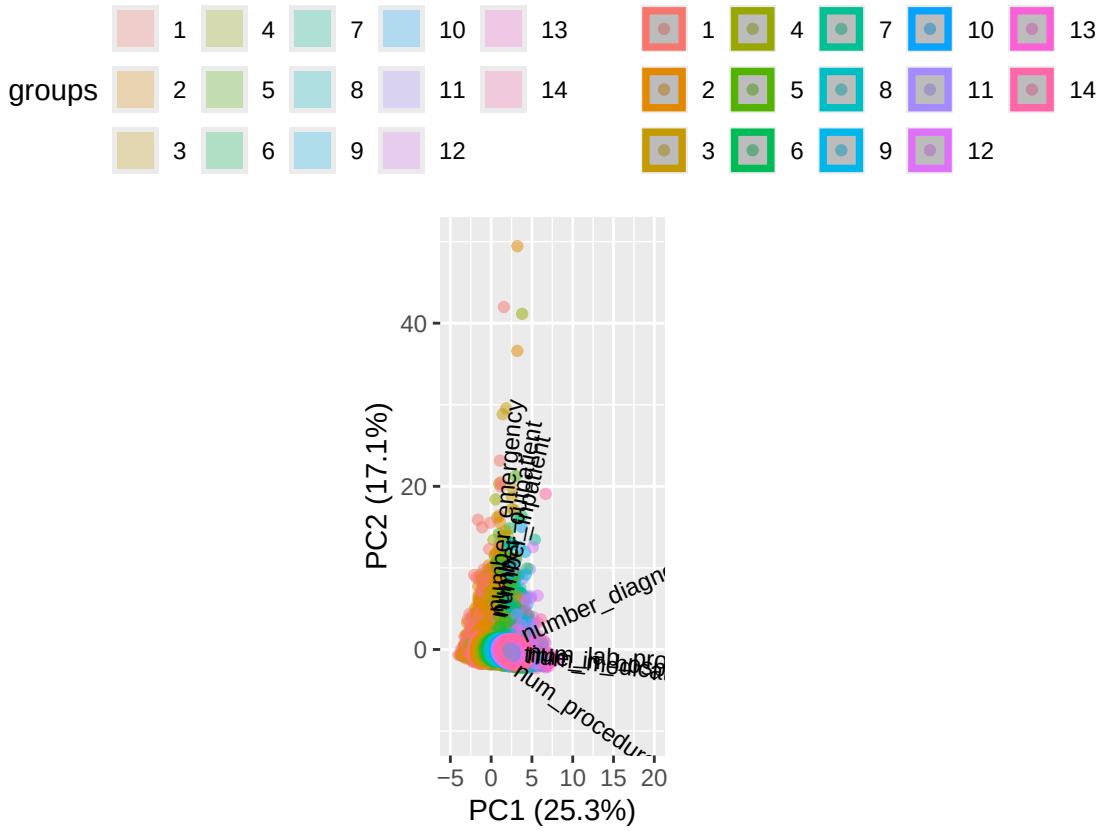


Figure 22: A PCA with all numeric attributes, plotted against the class variable time_spent_hospital, with fourteen different valuations

Because we adjusted the labels of the class variable, we need to rerun the PCA. Figure 22 showed the class variable with 14 levels; we have now reduced this to two. The updated results are shown in Figure 23.

In Figure 23, the two classes still overlap substantially, but the separation is slightly clearer than before. The blue and red groups show some tendency to separate, although they still overlap in the center. This is likely a consequence of collapsing the original multi-level outcome into a binary split (lower vs. higher values), which can increase contrast while still preserving shared structure across encounters.

```
encounter.label <- encounter.data %>%
  mutate(time_in_hospital =
           case_when(time_in_hospital %in%
                       c('1', '2', '3', '4', '5') ~ 'brief',
                       TRUE ~ 'long'))

ggbiplot(cc.pca, obs.scale = 1, var.scale = 1,
         groups = as.factor(encounter.label$time_in_hospital),
         ellipse = TRUE, circle = TRUE, alpha = 0.05) +
  theme(legend.direction = 'horizontal', legend.position = 'top') +
  xlim(-5, 20) + ylim(-10, 50)
```

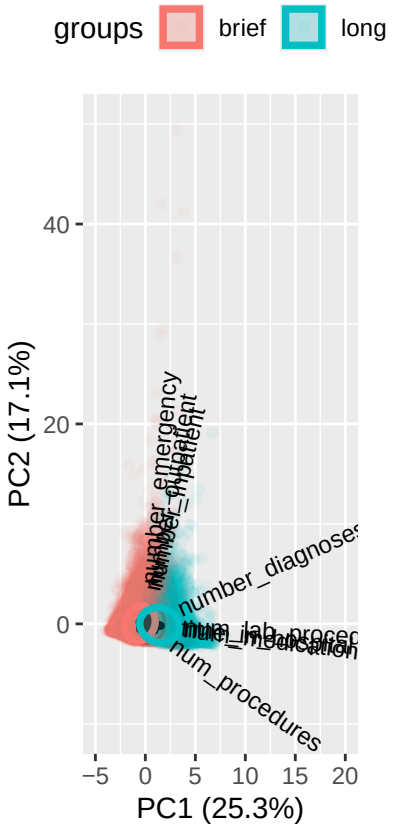


Figure 23: A PCA with all numeric attributes, plotted against the class variable time_spent_hospital, with two different valuations

3 A Clean Dataset

In this section, we apply all final preprocessing steps to construct a clean analysis dataset. While many filters, mutations, and relabeling steps were performed throughout the EDA, they are currently scattered across the notebook. To provide a clear and reproducible overview of the final dataset, we rerun all filtering, transformation, and relabeling steps that are retained for the final analysis in one place.

```
# Load in the data again to perform only filtering that is needed,
# discussed in the EDA
final.data <- read.table(
  file = 'datasets/data_diabetes_retrieved/diabetic_data.csv',
  sep = ',', header = TRUE)

final.data <- final.data %>%
  # Filter out every duplicate, based on patient number
  distinct(patient_nbr, .keep_all = TRUE) %>%
  # Relabel '?' to 'Missing'
  mutate(race=recode(race,'?'='Missing')) %>%
  # Remove instances with a missing labels and patients
  # who died in hospital
  filter(gender != 'Unknown/Invalid' & discharge_disposition_id != 11) %>%
  # Introduce the 'hba1c_res attribute, which is our class attribute
  mutate(hba1c_res=ifelse((A1Cresult=='None'), 'one', NA)) %>%
  mutate(hba1c_res=ifelse((A1Cresult %in% c('Norm', '>7'))
                          & is.na(hba1c_res), 'two', hba1c_res)) %>%
  mutate(hba1c_res=ifelse((A1Cresult=='>8') & (change=='No')
                          & is.na(hba1c_res), 'three', hba1c_res)) %>%
  mutate(hba1c_res=ifelse((A1Cresult=='>8') & (change=='Ch')
                          & is.na(hba1c_res), 'four', hba1c_res)) %>%
  # Introduce the new attribute icu_or_non
  mutate(admission_type_id =
           ifelse(admission_type_id %in% c(1, 2, 7),
                  'ICU patient', 'Non-ICU patient')) %>%
  mutate(admission_source_id =
           ifelse(admission_source_id %in% c(4, 7, 10, 12, 26),
                  'ICU patient', 'Non-ICU patient')) %>%
  mutate(discharge_disposition_id =
           ifelse(discharge_disposition_id %in% c(13, 14, 19, 20, 21),
                  'ICU patient', 'Non-ICU patient')) %>%
  mutate(icu_or_non =
           ifelse(admission_source_id == discharge_disposition_id &
                  admission_type_id   == discharge_disposition_id,
```

```
                  admission_source_id,
                  ifelse(admission_source_id == discharge_disposition_id,
                         admission_source_id, admission_source_id))) %>%
  # Change the label system in attribute diag_1
  mutate(diag_1 = case_when(diag_1 %in% c(390:459) ~ 'Circulatory',
                            diag_1 %in% c(460:519) ~ 'Respiratory',
                            diag_1 %in% c(520:579) ~ 'Digestive',
                            diag_1 %in% c(240:279) ~ 'Diabetes',
                            diag_1 %in% c(800:999) ~ 'Injury',
                            diag_1 %in% c(710:739) ~ 'Muscoloskeletal',
                            diag_1 %in% c(580:629) ~ 'Genitourinary',
                            TRUE ~ 'Others')) %>%
  # Set all character attributes to factor
  mutate_at(vars("time_in_hospital",
                 "hba1c_res",
                 "icu_or_non"), funs(factor)) %>%
  # Introduce a log2 scale on all numeric attributes
  mutate_at(.vars = names(select_if(., is.numeric)), ~ log2(. + 0.1)) %>%
  # Remove all redundant attributes
  select(-c('encounter_id', 'patient_nbr', 'weight', 'payer_code',
            'medical_specialty', 'diag_2', 'diag_3','admission_source_id',
            'discharge_disposition_id', 'admission_type_id',
            removal.names)) %>%
  relocate(time_in_hospital, .after = last_col())

# Write final dataset to datafile
write.csv(final.data, "datasets/base_datasets/normal_dataset.csv",
          row.names = FALSE)
```

3.1 Determine quality metrics relevancy

For the next part of our research, we focus on predicting the time spent in the hospital. In this project, we use Weka, a Java-based data analysis and algorithm workbench. Its interface makes it practical to train classifiers and evaluate performance on a dataset.

Weka offers multiple ways to evaluate the output of a classifier. When predicting a class variable, Weka reports the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). In our context, these counts describe whether the predicted duration category matches the true category or not. For binary classification, these terms are straightforward; for multiclass classification, they are typically computed per class (one-vs-rest) and then summarized.

Using TP, TN, FP, and FN, we can compute quality metrics to determine how well a classifier performs on our dataset. There are many possible quality metrics, but we focus on the most relevant ones for deciding whether a classifier is worth using: accuracy, precision, recall, true-positive and false-positive rates, the F1-score, and the ROC area (AUC). We also inspect the confusion matrix for a detailed view of model behavior.

A confusion matrix (also called an error matrix) summarizes classifier performance by comparing predicted versus actual class labels. The rows represent predicted classes and the columns represent the actual classes. Each cell contains the number of instances that fall into that predicted–actual combination. Although a confusion matrix is informative, it becomes difficult to interpret when the number of class levels is large. For example, a class variable with 14 levels yields a 14×14 confusion matrix (196 cells). Because such a matrix contains a lot of information, we additionally use summary metrics that are easier to compare across models.

3.1.1 Accuracy

Accuracy is the proportion of correctly classified instances:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

Accuracy provides a quick overall impression of performance (that is, how often predictions are correct). However, accuracy can be misleading when class distributions are imbalanced.

3.1.2 Precision

Precision measures the proportion of predicted positives that are truly positive:

$$\text{Precision} = \frac{TP}{TP + FP}.$$

Precision is important because it reflects how many of the positive predictions are actually correct. In a hospital setting, a model with many false positives could lead to unreliable recommendations and unnecessary actions or resource allocation. Therefore, we only want to provide advice to hospitals and institutions if the model can predict the duration of inpatient visits with sufficient confidence.

3.1.3 Recall

Recall (sensitivity) measures the proportion of actual positives that are correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

Recall reflects how many truly positive instances the model successfully captures. Low recall corresponds to many false negatives (missed positives), which could reduce the effectiveness of any procedure or policy decision informed by the model.

3.1.4 F1-score

The F1-score summarizes precision and recall into a single value (harmonic mean):

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

It ranges from 0 to 1, where higher values indicate a better balance between precision and recall.

3.1.5 ROC area (AUC)

The final metric we consider is the ROC area (AUC). A ROC curve plots the true-positive rate against the false-positive rate across decision thresholds. ROC analysis provides a useful way to compare models and discard suboptimal ones, especially when a confusion matrix becomes difficult to interpret due to many class levels. The AUC summarizes how well a model separates classes: the higher the AUC, the better the model can distinguish between different duration categories.

4 Machine learning algorithm performances

We evaluate multiple classification algorithms. Specifically, we train Naïve Bayes, SimpleLogistic, k-Nearest Neighbour (IBk), a decision tree (J48/C4.5), and a Random Tree. In addition, we include ZeroR and OneR as baseline models.

Weka reports performance metrics for each class level (in our case, levels 1 to 14). To enable comparison across classifiers, we use the averaged (summary) performance measures reported by Weka. The results are shown in Table 4.

```
# Results will be loaded in like this,
# but future code block will not be included due to repetitiveness
normal.result <- read.csv("datasets/dataset_results_weka/base/normal.csv",
                          sep = ";", fileEncoding="UTF-8-BOM")
kbl(normal.result, booktabs = T,
    caption = "Results from chosen classifiers on their base settings") %>%
  kable_styling(full_width = F, latex_options = "hold_position") %>%
  column_spec(1, bold = T)
```

Table 4: Results from chosen classifiers on their base settings

Algorithm	Accuracy	TP.Rate	FP.Rate	Precision	Recall	F.Measure	ROC.Area
ZeroR	17.82%	0.18	0.18	0.00	0.18	0.00	0.50
OneR	20.79% (v)	0.21	0.15	0.17	0.21	0.17	0.53
J48/C4.5	19.55% (v)	0.20	0.12	0.18	0.20	0.18	0.55
Naïve Bayes	20.76% (v)	0.21	0.12	0.19	0.21	0.20	0.65
IBk	16.16% (*)	0.16	0.12	0.16	0.16	0.16	0.52
Simple Logistics	23.60% (v)	0.24	0.14	0.20	0.24	0.20	0.68
Random Tree	17.20% (*)	0.17	0.12	0.17	0.17	0.17	0.53

We now examine the results shown in Table 4. Overall, all classifiers perform poorly: accuracy is low across the board, ranging from 16,20% for Nearest Neighbour (IBk) to a highest value of 23,55% for Simple Logistics. This is not ideal because, on average, only about one-fifth of the class labels are predicted correctly. A likely reason is that the class variable is positively skewed, meaning lower values occur much more frequently than higher values. In practice, this reflects that most patients spend only a few days in the hospital rather than, for example, more than a week.

Next, we determine which classifiers are most worthwhile given our research goals. As stated earlier, accuracy and precision are the most important metrics. The best option appears to be Simple Logistics, with an accuracy of 23,60% and a precision of 0,523, which are the best values among the evaluated models.

Selecting a runner-up is less straightforward because J48/C4.5 and Naïve Bayes both perform reasonably well. Naïve Bayes scores better on the presented metrics overall, but J48/C4.5 offers more options for tuning and optimization, whereas Naïve Bayes can be treated more

as a fixed baseline. Additionally, the accuracy difference is only 1,21% (1.21), which is not substantial. For this reason, we include both J48/C4.5 and Naïve Bayes in the following sections.

4.1 Optimizing our classifiers

To determine the best-performing algorithm, we tune the parameter settings of the selected models. To do this, we use Weka's Experimenter mode, which allows systematic exploration of parameter ranges and identifies settings that yield the best performance. In other words, we aim to improve the accuracy of the evaluated algorithms by testing different meta-learning configurations.

We apply the meta-learner `CvParameterSelection` to J48/C4.5 and Simple Logistics to identify optimal parameter sets. Naïve Bayes is tuned manually because it is relatively simple and has only a few adjustable settings. We report the results in the sections below. All classifiers are evaluated using the cost-sensitive setup introduced earlier, with 10-fold cross-validation.

4.1.1 J48/C4.5

We first examine the parameter settings of the decision tree algorithm J48/C4.5. We tune two parameters: the confidence factor (C) and the minimum number of instances per leaf (M).

The confidence factor C controls the degree of pruning: smaller values correspond to heavier pruning, while larger values result in less pruning. The default value is 0.25. We evaluate values from 0.1 to 0.5 in steps of 0.1. We cap the range at 0.5 because values above this tend to produce little to no pruning. [3]

Parameter M has a default value of 2 and specifies the minimum number of instances required per leaf. Increasing M results in larger leaves and a less complex tree. We test values from 2 to 10 in steps of 1. The results are shown in Table 5.

Table 5: Accuracy under different settings regarding J48/C4.5

Confidence.Factor	Accuracy.C	MinNumObject	Accuracy.M	Combination	Accuracy.C.M
0.10	20.95%	**2**	19.55%	**0.1/5**	22.28%
0.15	20.28%	**4**	20.56%	**0.1/10**	23.28%
0.25	19.55%	**5**	20.88%	**0.35/5**	20.57%
0.35	19.32%	**7**	21.34%	**0.35/10**	21.46%
0.45	19.11%	**10**	21.80%	**0.25/2**	19.55%

Looking at Table 5, we report three sets of results: accuracy obtained by varying each parameter individually, and accuracy obtained by combining selected parameter values. We first evaluated the effect of each parameter on its own. We then combined the most promising settings to identify an optimal configuration.

For the confidence factor C, we observe a clear pattern: lower values yield higher accuracy. This indicates that heavier pruning (a smaller tree) improves performance. At the same time, we must be cautious when tuning pruning-related parameters, because overly aggressive

tuning can lead to overfitting to the training folds. Therefore, we interpret improvements in C conservatively.

For parameter M, we see the opposite trend: increasing M leads to higher accuracy. Starting from the default value of 2 (19.55% accuracy), increasing M to 5 improves accuracy by 1.33 percentage points. At M = 10, accuracy reaches its highest value in this range (21.80%). When combining the best-performing values of C and M, the accuracy increases to 23.28%, which is also the highest value reported in the "Combination" column.

Overall, optimizing J48/C4.5 improves accuracy by 3.73 percentage points compared to the default settings. While this improvement is modest, it is consistent. Because the confidence factor was selected using cross-validation and the performance trend is stable across folds, we adopt these tuned settings as the optimized version of J48/C4.5 for the remainder of this study.

4.1.2 Simple Logistics

We now turn to Simple Logistics (SL). SL has the advantage of built-in attribute selection: it stops adding SimpleLinearRegression models once the cross-validation classification error no longer decreases. [4] SL exposes many parameters, but not all are equally relevant for our use case. Here, we focus on the following set of parameters: `useAIC` (A), `heuristicStop` (H), `maxBoostingIterations` (M), `numBoostingIterations` (I), and `weightTrimBeta` (W).

Parameter A (`useAIC`) determines when to stop adding models based on the Akaike Information Criterion (AIC), providing an alternative to cross-validation for model selection. Parameter H (`heuristicStop`) controls early stopping: a run can be terminated if no new minimum error has been reached within the last H iterations. Although Weka recommends using the default value, we test how changing H affects performance.

Parameter M (`maxBoostingIterations`) sets the maximum number of LogitBoost iterations. The default is 500, and Weka recommends using a higher value for larger datasets, which applies in our case. Parameter I (`numBoostingIterations`) sets a fixed number of LogitBoost iterations. If I < 0, the number of iterations is chosen via cross-validation or a stopping criterion on the training set, which is the default behavior.

Finally, W (`weightTrimBeta`) trims low-weight instances during boosting: only instances that account for (1 - W)% of the total weight from the previous iteration are retained for the next iteration. This reduces the number of instances considered in later iterations and may improve accuracy by focusing learning on higher-weight examples. The results of these parameter explorations are shown in Table 6.

Table 6: Accuracy under different settings regarding Simple Logistics

Settings	Accuracy	TP.Rate	FP.Rate	Precision	Recall	F.Measure	ROC.Area
Default	23.60%	0.236	0.138	0.203	0.236	0.205	0.677
A	23.64%	0.236	0.137	0.205	0.236	0.207	0.678
H (10)	23.43%	0.234	0.140	0.199	0.234	0.200	0.676
H (100)	23.58%	0.236	0.138	0.203	0.236	0.205	0.678
I (100)	23.63%	0.236	0.137	0.205	0.236	0.207	0.678
I (250)	23.63%	0.236	0.136	0.207	0.236	0.208	0.678
M (400)	23.59%	0.236	0.138	0.203	0.236	0.205	0.677
M (600)	23.59%	0.236	0.138	0.203	0.236	0.205	0.677
M (1000)	23.59%	0.236	0.138	0.203	0.237	0.205	0.677
W (0.3)	20.44%	0.204	0.170	?	0.204	?	0.635

The explored parameters do not meaningfully improve classifier accuracy. Combining parameters also did not lead to better performance (results not shown). This suggests that SL is already operating close to its optimal configuration on this dataset. As shown earlier in Table 6, SL achieved one of the highest accuracies using the default settings. In addition, SL includes an internal stopping criterion: it stops adding models once cross-validation error no longer decreases, which effectively prevents unnecessary complexity. For these reasons, we keep the default parameter settings for SL in the remainder of this study.

4.1.3 Naïve Bayes

As discussed earlier, Naïve Bayes is a relatively simple classifier with only a few adjustable settings, so we tuned it manually. We evaluate two options and report the results in Table 7:

1. Using a kernel density estimator instead of assuming a normal distribution (K); and

2. Using supervised discretization (D).

Option K replaces the normal-distribution assumption with a kernel density estimate of the probability density function. This provides a smoother, data-driven estimate and can better capture non-normal feature distributions. Option D enables supervised discretization, meaning that binning is performed with knowledge of the class label. In contrast, the default (unsupervised) discretization does not consider the class label. Because supervised discretization aims to create bins that are informative for the target, we expect it to improve performance.

We do not combine K and D because these settings are not compatible and would conflict.

Table 7: Accuracy under different settings regarding Naïve Bayes

Settings	Accuracy	TP.Rate	FP.Rate	Precision	Recall	F.Measure	ROC.Area
Default	20.76%	0.208	0.118	0.192	0.208	0.197	0.647
K	23.25%	0.233	0.131	0.203	0.233	0.208	0.669
M	22.61%	0.226	0.127	0.199	0.226	0.205	0.668

We now examine Table 7. Both tested settings improve accuracy and the other reported metrics. In particular, enabling the kernel density estimator (K) increases accuracy by a few percentage points, while supervised discretization (D) also improves performance, but to a smaller extent. The precision values are very similar between the two settings (0.203 for K versus 0.199 for D). Overall, K provides the best results, and we therefore proceed with the kernel density estimator as the optimized Naïve Bayes configuration.

Even after optimization, Naïve Bayes performs worse across all metrics than SL using its default settings. We revisit the final model choice after additional optimization steps.

4.2 Selecting attributes

Weka provides a meta-learner called `AttributeSelectedClassifier`. This meta-learner evaluates whether a classifier performs better after attribute selection. It operates in two steps: (1) dimensionality reduction through attribute selection, and (2) training and evaluating the chosen classifier on the selected subset of attributes. [5]

Attributes can be ranked based on their contribution to performance (for example, accuracy or other metrics). To assess whether removing attributes improves performance, we evaluate three attribute evaluators: (1) `CorrelationAttributeEval`, (2) `InfoGainAttributeEval`, and (3) `GainRatioAttributeEval`. Before removing any attribute, we examine the potential impact of low-performing features and compare attribute behavior across evaluators. Because different evaluators can rank attributes differently, it is important to consider consistency across methods.

4.2.1 Correlation

We begin with `CorrelationAttributeEval`, which computes the correlation between each attribute and the class variable. Correlation values range from -1 to 1. We are primarily interested in attributes with moderate-to-strong positive or negative correlations. Attributes with near-zero correlation are less likely to contribute predictive signal and may add noise, so they are candidates for removal. We use the `Ranker` search method for this evaluation.

We observe the results in Table 8, and they are not very strong. The highest-scoring attribute is `num_medications` with a correlation of 0.12651, while the lowest-scoring attribute is `rosiglitazone` with a correlation of 0.000389. In many settings, a cutoff of 0.05 is used as a rough threshold for feature relevance. If we applied this cutoff here, only four of the 23 evaluated attributes would remain, which would be too restrictive for our purposes. If we instead used a cutoff of 0.01, we would remove eight attributes rather than 19. Before deciding on any threshold, we first examine the results from additional evaluators.

Table 8: Correlation between all independent attributes

Attribute	Correlation
num_medications	0.12651
num_lab_procedures	0.07422
number_diagnoses	0.06756
num_procedures	0.05114
change	0.03278
insulin	0.02285
diabetesMed	0.02270
hba1c_res	0.02034
diag_1	0.02014
A1Cresult	0.01970
number_inpatient	0.01868
icu_or_non	0.01729
readmitted	0.01707
age	0.01459
gender	0.01257
metformin	0.00947
glyburide	0.00922
glipizide	0.00840
number_outpatient	0.00795
race	0.00472
number_emergency	0.00418
pioglitazone	0.00402
rosiglitazone	0.00389

4.2.2 Info gain

Next, we use `InfoGainAttributeEval`, which measures the information gain (entropy reduction) each attribute provides about the class variable. Values range from 0 (no information gain) to 1 (maximum information gain). Higher values indicate that an attribute reduces uncertainty about the class label more strongly and is therefore more likely to be useful for prediction. As with the correlation-based evaluator, low-scoring attributes may be candidates for removal if their inclusion adds noise without improving performance. We again use the `Ranker` search method. The results are shown in Table 9.

As in the correlation-based analysis, we observe that the same subset of attributes performs poorly compared to the others. Two attributes even receive a score of 0 (`number_emergency` and `number_outpatient`), suggesting that they contribute no measurable information about the class variable under this evaluator and are therefore candidates for removal.

Choosing a cutoff value remains difficult. Using commonly cited thresholds would again remove most attributes. For example, with a cutoff of 0.05 we would retain only two attributes, and with 0.01 we would retain eight. Rather than selecting an arbitrary cutoff upfront, we can evaluate the effect of removing attributes in batches and rerun our classifiers to measure the impact on performance.

Table 9: Info gain regarding each independent attributes

Attribute	Info.Gain
num_medications	0.18362
num_lab_procedures	0.10033
number_diagnoses	0.04923
num_procedures	0.03651
diag_1	0.02377
age	0.01880
insulin	0.01602
change	0.01094
hba1c_res	0.00639
diabetesMed	0.00507
A1Cresult	0.00500
readmitted	0.00452
glipizide	0.00380
number_inpatient	0.00350
glyburide	0.00328
icu_or_non	0.00317
metformin	0.00249
pioglitazone	0.00163
gender	0.00147
rosiglitazone	0.00135
race	0.00131
number_emergency	0.00000
number_outpatient	0.00000

4.2.3 Gain ratio

The final evaluator we use is `GainRatioAttributeEval`. This method is closely related to information gain but applies a normalization that accounts for the intrinsic information of an attribute. In practice, it provides a more balanced view than raw information gain, especially when attributes have many possible values. A higher gain ratio indicates a stronger reduction in entropy with respect to the class variable. As before, we use the `Ranker` search method. The results are shown in Table 10.

Again, we observe a consistent trend: the same five attributes—`num_medications`, `num_lab_procedures`, `num_diagnoses`, `num_procedures`, and `change`—rank highest across all evaluators. The same consistency holds for the lowest end of the rankings: `number_emergency` and `number_outpatient` are among the worst-performing attributes in every evaluation. At a minimum, we remove these two attributes. In the next section, we specify which attributes will be removed and which will be retained.

Table 10: Gain ratio regarding each independent attributes

Attribute	Gain.Ratio
num_medications	0.06346
num_lab_procedures	0.03877
num_diagnoses	0.02652
num_procedures	0.01968
change	0.01102
insulin	0.00945
diag_1	0.00937
age	0.00707
number_inpatient	0.00671
hba1c_res	0.00671
diabetesMed	0.00640
glipizide	0.00605
glyburide	0.00566
A1Cresult	0.00520
pioglitazone	0.00400
rosiglitazone	0.00363
readmitted	0.00353
icu_or_non	0.00322
metformin	0.00292
gender	0.00147
race	0.01120
number_emergency	0.00000
number_outpatient	0.00000

4.2.4 Final decision

As stated above, we remove `number_emergency` and `number_outpatient`. Deciding whether to remove additional attributes is less straightforward. Most attributes score far below commonly used cutoff values such as 0.05 (and often even 0.01), meaning that a strict threshold-based approach would discard the majority of features.

Instead, we also consider the practical and clinical relevance of each attribute. Some variables may score low in these evaluators but still be important to retain for interpretation and subgroup analyses. For example, `gender` scores low across evaluators, but retaining it is important because stratifying outcomes by gender is common in medical research and may be relevant for downstream analyses. In contrast, other low-scoring attributes may be redundant or less informative given the presence of higher-level medication indicators (for example, `diabetesMed`).

Using this combination of evaluator output and domain relevance, we select additional candidates for removal. These are: `metformin`, `diabetesMed`, `rosiglitazone`, `pioglitazone`, `glipizide`, and `glyburide`. This results in a dataset with 16 attributes, including the class variable.

We then rerun all classifiers on this reduced feature set and compare performance using

accuracy. The results are shown in Table 11.

Table 11: Results from removing certain attributes (new) versus not removing them (old)

Metric	Old.NB	New.NB	Old.SL	New.SL	Old.J48	New.J48
Accuracy	20.76%	23.42%	23.60%	23.47%	19.55%	23.42%
TP.Rate	0.21	0.234	0.24	0.235	0.20	0.234
FP.Rate	0.12	0.129	0.14	0.140	0.12	0.134
Precision	0.19	0.207	0.20	0.200	0.18	0.202

We now examine Table 11. Removing the selected attributes improves the accuracy (and other metrics) for Naïve Bayes and J48/C4.5, while Simple Logistics performs slightly worse with the reduced feature set. In practice, it makes sense to use the best-performing version of each classifier. Therefore, if we select Simple Logistics as the final model, we must decide whether to keep the removed attributes (which improve performance for the other models) or remove them to maintain a consistent reduced feature set across models.

The removed attributes contain relatively little useful information compared to the remaining features. Although keeping them benefits Simple Logistics, the improvement is small (0.13 percentage points). For consistency, and because the difference is marginal, we continue with the reduced attribute set for all classifiers.

4.3 Confusion matrices

In this section, we examine the confusion matrices of the best-performing versions of our selected classifiers. Because the class variable has 14 levels, each confusion matrix has a 14×14 structure. Due to this complexity, we previously relied on weighted averages for model comparison. However, because the overall performances of the models are relatively close, examining confusion matrices can provide additional insight and help guide the final model choice.

4.3.1 Naïve Bayes

We begin with Naïve Bayes. The confusion matrix is shown in Table 12. In a confusion matrix, the main diagonal represents correct predictions. Off-diagonal cells represent misclassifications. In general, higher values along the diagonal indicate better performance because more instances are correctly classified. Naïve Bayes performs reasonably well, achieving an accuracy of 23.42%.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<– classified as
5036	2811	1968	450	153	46	21	11	3	2	1	0	0	0	a = 1
3523	4023	3037	931	426	149	50	41	33	4	1	0	2	1	b = 2
2334	3239	4250	1468	730	263	95	111	49	9	4	0	2	1	c = 3
1292	2050	3159	1416	818	261	160	233	63	12	8	0	3	1	d = 4
723	1138	2054	1118	815	314	176	287	76	22	13	0	10	2	e = 5
444	719	1510	825	635	330	186	294	103	27	13	1	6	3	f = 6
273	476	1035	669	523	308	171	321	87	30	15	0	22	7	g = 7
208	260	694	427	392	235	148	298	98	59	28	2	16	7	h = 8
95	155	429	281	255	177	135	248	80	41	22	2	12	9	i = 9
60	109	324	207	184	145	109	224	73	41	30	1	10	6	j = 10
46	81	241	170	155	134	80	170	53	42	20	0	13	14	k = 11
39	54	182	123	104	94	64	139	58	34	21	2	6	6	l = 12
40	36	127	85	96	70	67	138	50	24	24	1	11	2	m = 13
24	31	122	66	69	61	46	106	49	33	26	2	12	4	n = 14

Table 12: Confusion matrix based on the optimal settings regarding Naïve Bayes

When we examine the results, we again observe that the data is positively skewed: the number of instances decreases as length of stay increases. This imbalance is a key challenge when predicting the exact time a patient will spend in the hospital. We have many examples of short stays but relatively few examples of long stays, which limits how well a classifier can learn patterns associated with longer admissions. Addressing this issue would require more instances for the longer-stay classes, which we currently lack.

In addition, accuracy and precision per class generally decrease as length of stay increases. For example, for the 14-day class we obtain only four correct predictions, compared with 5,036 correct predictions for the 1-day class (class a).

4.3.2 Simple Logistics

Next, we examine the confusion matrix from the Simple Logistics classifier, shown in Table 13. Although Simple Logistics achieved a slightly higher accuracy when using the full feature set, we proceed with the reduced feature set. The rationale is that retaining additional low-value attributes for only a small gain in accuracy is not worth the added complexity. With the reduced attribute set, Simple Logistics achieves an accuracy of 23.47%.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<– classified as
4556	3091	2566	217	53	7	11	1	0	0	0	0	0	0	a = 1
3127	4167	4099	579	163	45	32	8	0	1	0	0	0	0	b = 2
2029	3345	5645	1027	325	93	70	20	1	0	0	0	0	0	c = 3
1150	2092	4375	1098	446	150	101	55	4	4	0	0	1	0	d = 4
646	1214	2920	1004	531	149	179	70	18	7	5	0	4	1	e = 5
432	735	2087	848	492	175	180	103	24	11	5	0	2	2	f = 6
256	518	1478	681	422	202	190	124	31	23	9	0	2	1	g = 7
203	316	955	479	341	157	233	115	35	30	5	0	2	1	h = 8
92	163	606	330	261	146	197	96	27	14	4	0	3	2	i = 9
60	125	454	233	218	111	181	74	36	16	8	0	3	4	j = 10
39	86	361	181	181	102	129	72	28	19	8	0	11	2	k = 11
34	57	255	142	133	78	115	58	24	16	5	0	5	4	l = 12
34	43	197	77	122	58	119	54	31	14	12	1	6	3	m = 13
18	35	167	79	85	67	91	43	29	21	6	0	9	1	n = 14

Table 13: Confusion matrix based on the optimal settings regarding Simple Logistics

The confusion matrix for Simple Logistics shows a similar pattern to Naïve Bayes: the class distribution is positively skewed, and there are few correct predictions for the higher-valued length-of-stay classes. It is also noticeable that the number of correct predictions for class a (1 day) is lower than for Naïve Bayes: 4,556 for Simple Logistics versus 5,036 for Naïve Bayes.

In contrast, Simple Logistics produces more correct predictions for several of the other classes, although this advantage becomes smaller as the length of stay increases. This may reflect the way Simple Logistics assigns weights to attributes and models more complex decision boundaries than Naïve Bayes. However, despite this increased model complexity, the overall performance does not improve meaningfully.

4.3.3 J48/C4.5

Finally, we evaluate J48/C4.5. After attribute selection, the classifier achieves an accuracy of 23.42%. The corresponding confusion matrix is shown in Table 14.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<– classified as
5071	2593	2258	315	122	72	39	18	3	2	4	2	1	2	a = 1
3344	3746	3973	570	275	147	86	35	19	9	13	2	2	0	b = 2
2137	3095	5346	922	481	256	159	71	38	20	19	4	4	3	c = 3
1257	1906	4193	882	541	279	204	103	42	26	19	10	8	6	d = 4
643	1181	2787	697	517	367	244	136	60	52	33	9	11	11	e = 5
440	730	1968	573	444	352	269	134	64	42	36	16	16	12	f = 6
284	508	1417	438	395	282	230	169	79	58	43	12	14	8	g = 7
186	358	884	301	304	228	216	181	56	63	55	12	22	6	h = 8
95	191	550	199	205	190	182	132	59	47	45	10	22	14	i = 9
71	138	402	174	169	129	130	126	38	40	48	25	21	12	j = 10
55	92	294	120	141	114	116	107	58	44	35	13	17	13	k = 11
36	73	222	99	107	93	84	80	35	31	33	11	13	9	l = 12
37	49	178	74	73	64	84	78	33	39	29	11	13	9	m = 13
27	46	143	59	59	77	44	69	26	35	36	6	13	11	n = 14

Table 14: Confusion matrix based on the optimal settings regarding J48/C4.5

The overall performance of J48/C4.5 is again very similar to that of the other classifiers. While there are small differences in how individual class levels are predicted, the confusion matrix does not reveal substantial advantages over Naïve Bayes or Simple Logistics. In addition, the resulting decision tree is very large and complex, which limits interpretability. This complexity is likely driven by the number of predictors and the 14-level class variable. Given the strong similarity in performance across models, the lack of interpretability provides an additional reason to deprioritize J48/C4.5.

Inspecting the confusion matrices did not lead to a clear choice of a final model. As noted earlier, the three classifiers perform comparably overall. Although there are minor differences in accuracy across individual class levels, these differences are not sufficient to justify selecting one model over the others. More importantly, an overall accuracy of approximately 23% is not acceptable for a model intended to predict the exact length of hospital stay in days.

A major driver of this limitation is the strong class imbalance in the target variable. The dataset contains many encounters with short stays and relatively few encounters with long stays, which restricts the models' ability to learn patterns associated with longer admissions. In this setting, attempting to predict the exact length of stay as a 14-class problem is likely too granular. Therefore, the next sections focus on alternative scenarios that may yield a more useful and accurate model, as well as methodological changes aimed at improving performance.

5 Change of plans

The results above indicate that the current approach does not produce a model with sufficient predictive performance. The best accuracy achieved is 23.60%, which is too low for practical use in a clinical or operational context. In this section, we explore more substantial strategies to improve predictive performance. These include restructuring the target variable (for example by merging classes), applying more aggressive instance selection, and reconsidering the research question to better match what can be learned reliably from the available data.

5.1 Effects of removing instances

```
count.data <- final.data %>%
  group_by(time_in_hospital) %>%
  tally()

instance.data <- final.data %>%
  filter(!time_in_hospital %in% c(11, 12, 13, 14))

write.csv(instance.data,
          "datasets/base_datasets/less_instances.csv",
          row.names = FALSE)
```

As shown above, there is no straightforward way to increase accuracy without making substantial changes to the dataset. Options include removing high-value classes with few instances, restructuring the target variable, or increasing the weights of rare classes to obtain a more balanced distribution. However, these approaches have drawbacks. We already used a cost-sensitive setup to give more weight to classes with fewer instances, and increasing these weights further could distort the results. Removing high-value classes is also undesirable, because it requires defining an arbitrary cutoff.

To illustrate the effect of removing rare high-value instances, we consider excluding `time_in_hospital` values from 11 to 14. This would remove 3567 instances, corresponding to a loss of around 6%. After removal, we would retain 66,871 encounters.

With this temporary dataset, we return to Weka and evaluate whether the proportion of correctly classified instances improves. In short, accuracy increases, but not enough to justify removing these instances in the final model. We reran J48 and Simple Logistics and observed an improvement of 1,13 percentage points for J48 (from 19,81% to 20,94%). Simple Logistics shows a similar trend, with an improvement of 1,33 percentage points. The results are shown in Table 15.

Given that this requires removing 6% of the dataset for only a small gain in accuracy, this approach is not worthwhile. We therefore conclude that excluding the high-value instances (11–14 days) is not an effective strategy for achieving substantially better performance.

Table 15: Results from chosen classifiers on their base settings and removing extra instances to improve metrics

Algorithm	Accuracy	TP.Rate	FP.Rate	Precision	Recall	F.Measure	ROC.Area
ZeroR	18.77%	0.19	0.19	0.00	0.19	0.00	0.50
OneR	21.94% v	0.22	0.16	0.20	0.22	0.18	0.53
J48/C4.5	21.11 (v)	0.21	0.13	0.20	0.21	0.20	0.55
Naïve Bayes	21.94% (v)	0.22	0.13	0.21	0.22	0.21	0.64
IBk	17.23% (*)	0.17	0.13	0.17	0.17	0.17	0.52
Simple Logistics	24.73% (v)	0.25	0.15	0.22	0.25	0.22	0.67
Random Tree	18.23%	0.18	0.13	0.18	0.18	0.18	0.53

5.2 A new research goal

Another option to improve accuracy is to relabel the target attribute. The advantage of this approach is that all instances are retained. The drawback is that the classes become broader, which removes the possibility of predicting the exact number of days spent in the hospital. Instead, we can predict clinically meaningful categories, such as a brief versus a longer visit. This framing still supports classification, but at a lower resolution.

We consider a binary relabeling with two classes:

1. Time spent in hospital is less than or equal to 5 days (brief visit); and
2. Time spent in hospital is longer than 5 days (long visit).

In the next code section, we mutate `time_in_hospital` according to this definition. We also show the effect of this relabeling in Figure 22. The resulting two classes contain approximately 55,000 and 19,000 instances, respectively.

```
final.data.label <- final.data %>%
  mutate(time_in_hospital =
           case_when(time_in_hospital %in%
                       c('1', '2', '3', '4', '5') ~ 'brief',
                       TRUE ~ 'long')) %>%
# Reshape data so that class variable is the last column
# We did this manually using the Explorer, but now we use the Experimenter,
# and the only way to appoint a class variable is having it be the last column.
  relocate(time_in_hospital, .after = last_col())

write.csv(final.data.label, "datasets/base_datasets/relabeled.csv",
          row.names = FALSE)

# Add legends and introduce a two-in-one plot
p1 <- ggplot() +
```

```
  geom_bar(mapping = aes(x=final.data$time_in_hospital)) +
  ggtitle("Time in hospital (1-14 days)") +
  xlab("Days")

p2 <- ggplot() +
  geom_bar(mapping = aes(x=final.data.label$time_in_hospital)) +
  ggtitle("Time in hospital (two values)") +
  xlab("Classification")
```

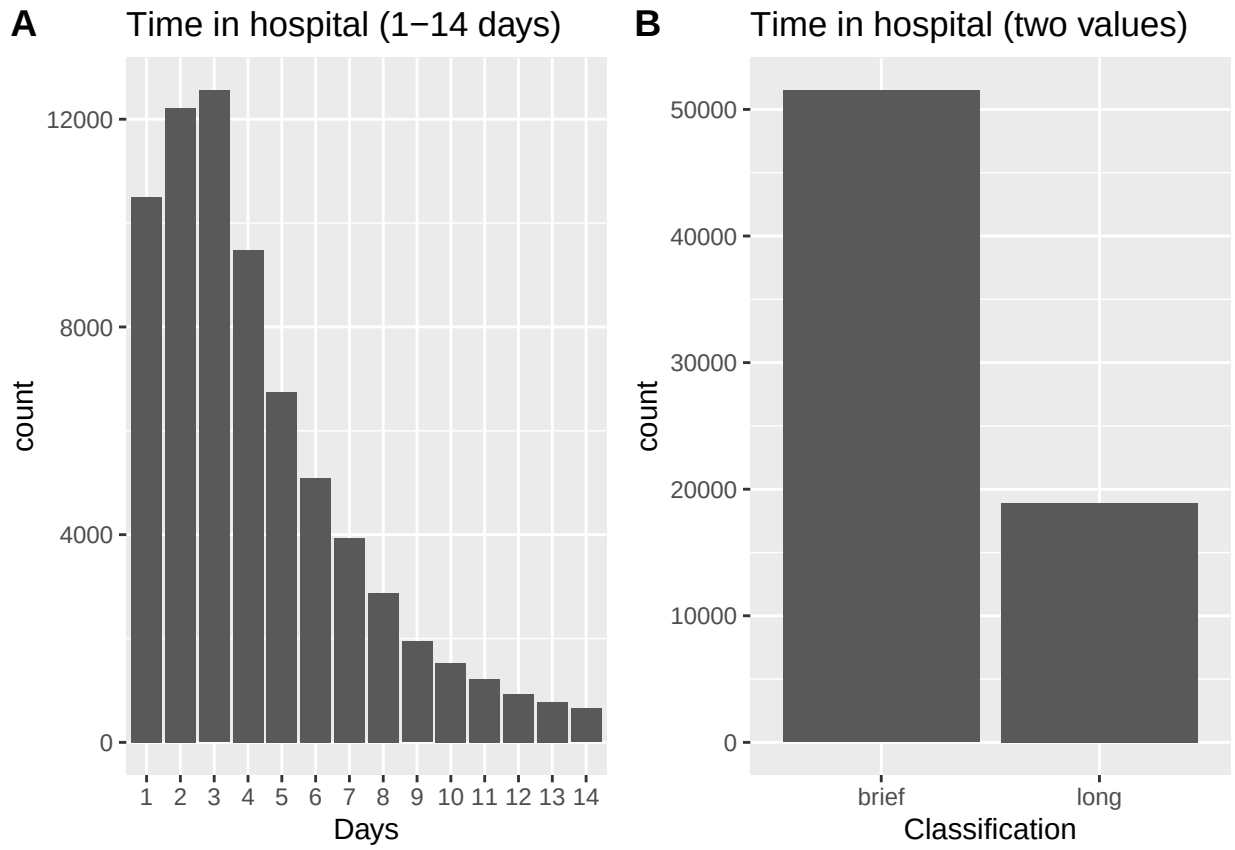


Figure 24: Difference between class variables labeling

With the relabeled dataset, we return to Weka to evaluate classifier performance under the new target definition. The results are shown in Table 16.

Having obtained results for both target definitions, we can compare their performance. As expected, accuracy increases across the full set of classifiers. Most models improve by at least 50 percentage points, with the exception of IBk, which improves by roughly 40 percentage points. IBk also performed worst in the earlier experiments, so it is unsurprising that it does not become competitive after relabeling.

When deciding which dataset to carry forward, we must weigh whether the loss of specificity in the target variable is acceptable in exchange for substantially better predictive

Table 16: Results from chosen classifiers on their base settings and relabeling the class variable to improve metrics

Algorithm	Accuracy	TP.Rate	FP.Rate	Precision	Recall	F.Measure	ROC.Area
ZeroR	73.12%	0.731	0.731	0.000	0.731	0.000	0.500
OneR	76.31% v	0.762	0.540	0.740	0.762	0.792	0.611
J48/C4.5	77.79% v	0.778	0.454	0.761	0.778	0.761	0.718
Naïve Bayes	72.80%	0.731	0.365	0.745	0.731	0.373	0.753
IBk	66.71% *	0.667	0.522	0.664	0.667	0.666	0.573
Simple Logistics	78.55% v	0.786	0.467	0.771	0.786	0.764	0.790
Random Forest	78.72% v	0.789	0.454	0.775	0.789	0.770	0.802

performance. Although we no longer predict the exact length of stay in days, the relabeled outcome still provides useful information about hospital duration (brief versus long stay), and the predictive accuracy improves markedly. If a future approach yields stronger performance on the original 14-class target, we can revisit that formulation. At this stage, the relabeled dataset is the more promising option, and we therefore continue the analysis using this binary outcome.

Among the evaluated classifiers, Simple Logistics (78.55%) and Random Forest (78.72%) perform best and are the most suitable candidates for further development. In the next sections, we attempt to optimize these classifiers to improve accuracy further.

5.3 Attribute selection

We previously performed attribute selection on the original dataset. Because the class variable has now changed, we reassess attribute relevance for the relabeled dataset. We use the same meta-learner and evaluators as before: `AttributeSelectedClassifier` with `CorrelationAttributeEval`, `InfoGainAttributeEval`, and `GainRatioAttributeEval`. We discuss the results of each method in the following subsections.

5.3.1 Correlation

We begin with `CorrelationAttributeEval`, which calculates the correlation between each attribute and the class variable. Correlation values range from -1 to 1. Attributes with near-zero correlation are less likely to carry predictive signal and may add noise, so they are candidates for removal. We use the `Ranker` search method for this evaluation.

Table 17: Correlation between all independent attributes

Attribute	Score
num_medications	0.35966
num_lab_procedures	0.19294
number_diagnoses	0.18630
num_procedures	0.17575
change	0.08199
number_inpatient	0.06257
hba1c_res	0.05615
A1Cresult	0.05428
insulin	0.05356
diabetesMed	0.04432
readmitted	0.04372
age	0.03356
icu_or_non	0.03096
number_outpatient	0.02507
glyburide	0.02295
gender	0.01707
diag_1	0.01495
metformin	0.01390
glipizide	0.01253
number_emergency	0.00685
pioglitazone	0.00422
race	0.00287
rosiglitazone	0.00155

We observe the results in Table 17 and find patterns similar to those from the initial dataset, at least in terms of attribute ranking. However, the absolute scores are higher overall. For example, in the initial dataset the top-ranked attribute (`num_medications`) had a correlation score of 0.12651, whereas under the relabeled outcome it scores 0.35966. With these higher scores, applying a cutoff becomes more feasible: a threshold of 0.01 would remove four

attributes, while 0.05 would remove fourteen. Before choosing a threshold, we first examine the results from the other evaluators.

5.3.2 Information gain

Next, we use `InfoGainAttributeEval`, which measures the information gain (entropy reduction) each attribute provides about the class variable. Values range from 0 (no information gain) to 1 (maximum information gain). Attributes with higher values reduce uncertainty about the class label more strongly and are therefore more likely to be useful for prediction. As before, low-scoring attributes may be candidates for removal if their inclusion adds noise without improving classifier performance. We again use the `Ranker` search method. The results are shown in Table 18.

Table 18: Info gain regarding all independent attributes

Attribute	Score
num_medications	0.113531
num_lab_procedures	0.063789
num_diagnoses	0.030539
num_procedures	0.023768
age	0.009319
insulin	0.008625
diag_1	0.006242
change	0.004833
hba1c_res	0.003411
readmitted	0.002796
number_inpatient	0.002619
A1Cresult	0.002614
glipizide	0.001990
glyburide	0.001785
diabetesMed	0.001446
pioglitazone	0.000959
metformin	0.000920
icu_or_non	0.000694
number_outpatient	0.000512
rosiglitazone	0.000503
race	0.000302
gender	0.000210
number_emergency	0.000000

As in the correlation-based evaluation, we observe that the same subset of attributes performs poorly compared to the others. Some attributes score as low as 0, such as `number_emergency`, indicating that they provide no measurable information about the class variable under this evaluator and are therefore candidates for removal. Selecting a cutoff remains challenging: applying commonly used thresholds would again remove a large fraction of the attributes. For example, with a cutoff of 0.05 we would retain only two attributes, and with 0.01 we would retain eight.

5.3.3 Gain ratio

Next, we use `GainRatioAttributeEval`, which is closely related to information gain but applies a normalization that accounts for the intrinsic information of an attribute. In practice, this can provide a more balanced ranking than raw information gain, particularly for attributes with many possible values. A higher gain ratio indicates a stronger reduction in entropy with respect to the class variable. As before, we use the `Ranker` search method. The results are shown in Table 19.

Table 19: Gain ratio regarding all independent attributes

Attribute	Gain.Ratio
num_medications	0.06346
num_lab_procedures	0.03877
num_diagnoses	0.02652
num_procedures	0.01968
change	0.01102
insulin	0.00945
diag_1	0.00937
age	0.00707
number_inpatient	0.00671
hba1c_res	0.00671
diabetesMed	0.00640
glipizide	0.00605
glyburide	0.00566
A1Cresult	0.00520
pioglitazone	0.00400
rosiglitazone	0.00363
readmitted	0.00353
icu_or_non	0.00322
metformin	0.00292
gender	0.00147
race	0.01120
number_emergency	0.00000
number_outpatient	0.00000

We observe the same pattern seen in the initial dataset and again in this relabeled dataset: the same five attributes—`num_medications`, `num_lab_procedures`, `num_diagnoses`, `num_procedures`, and `change`—rank highest across all evaluators. At the other extreme, `number_emergency` and `number_outpatient` again score 0. A cutoff of 0.05 would leave only one attribute, and a cutoff of 0.01 would retain only the five top-ranked attributes. In the next section, we discuss which attributes to drop, guided by the combined evaluator results and their practical relevance.

5.3.4 Final decision

As stated earlier, we remove `number_emergency` and `number_outpatient` at a minimum. Deciding whether to remove additional attributes is more difficult. The evaluators use different scoring criteria, so their absolute scores are not directly comparable, which complicates a single cutoff-based decision.

For example, `insulin` scores 0.0521 under correlation, which would keep it if we used a 0.05 threshold. However, it scores 0.008625 and 0.00945 under information gain and gain ratio, respectively, which would exclude it under a 0.01 threshold. Despite these differences in scale, the rank ordering is broadly consistent: attributes that score low under one evaluator also tend to score low under the others. This consistency makes correlation a practical choice for applying a cutoff, especially because its scores span a wider range (approximately 0.35 to 0.00155), compared with gain ratio (approximately 0.06 to 0.00112).

We therefore examine the effect of applying correlation-based cutoffs of 0.01 and 0.05. The results are shown in Figure 18.

Table 20: Results from removing certain attributes (new) versus not removing them (old)

Metric	SL.normal	SL.0.01	SL.0.05	RF.normal	RF.0.01	RF.0.05
Accuracy	78.55%	78.59%	77.96%	78.72%	78.45%	74.85%
TP.Rate	0.786	0.786	0.780	0.789	0.785	0.749
FP.Rate	0.467	0.469	0.501	0.454	0.452	0.464
Precision	0.771	0.771	0.764	0.775	0.769	0.732
Recall	0.786	0.786	0.780	0.789	0.785	0.749
F-Measure	0.764	0.764	0.752	0.770	0.766	0.737
ROC-curve	0.790	0.788	0.770	0.802	0.790	0.733

We now examine Table 20. Applying a cutoff of 0.01 would remove five attributes in total (`number_emergency`, `pioglitazone`, `race`, `rosiglitazone`, and `number_outpatient`). Applying a cutoff of 0.05 would remove 14 attributes, leaving only eleven attributes in the dataset.

The two classifiers respond differently to these reductions. For Simple Logistics, applying a 0.01 cutoff yields a very small increase in accuracy, whereas applying a 0.05 cutoff reduces accuracy. However, the improvement at 0.01 is negligible (0.004 percentage points). In contrast, Random Forest does not improve under either cutoff: accuracy decreases by 0.27 percentage points with a 0.01 cutoff and by 3.8 percentage points with a 0.05 cutoff. This indicates that removing 14 attributes is clearly not worthwhile and that even the 0.01 cutoff provides no meaningful benefit overall.

Nevertheless, we will still remove `number_emergency` and `number_outpatient`, because these features consistently score 0 and are not aligned with our goal of predicting time spent in the hospital.

5.4 Optimization

In the next section, we explore further optimization, as the best-performing classifiers differ from those used in the initial dataset. The meta-learner `CvParameterSelection` evaluates multiple parameter configurations and selects the settings that maximize accuracy.

5.4.1 Simple Logistics

We again optimize Simple Logistics. The key parameters and their default values were discussed earlier, and prior experiments showed that several settings have minimal impact on performance. Weka also recommends keeping certain defaults, such as the `heuristicStop` (H) parameter, which we previously tested without observing improvement. Therefore, we focus on the two parameters that showed the most promise in earlier experiments: `maxBoostingIterations` (M) and `numBoostingIterations` (I).

Table 21: Results regarding Simple Logistics's base (old) and improved parameter settings (new)

Metric	Old	New
Accuracy	78.59%	78.62%
TP.Rate	0.786	0.786
FP.Rate	0.467	0.467
Precision	0.771	0.772
Recall	0.786	0.786
F-Measure	0.764	0.765
ROC-Curve	0.788	0.789

Looking at Table 21, we compare performance from the attribute-selection setup (old) with the best configuration identified by `CvParameterSelection` (new). We tested parameter I in steps of 250, ranging from 0 to 750, and parameter M ranging from 0 to 1,500 in six steps. The best result was obtained with M = 750 and I = 0. Under these settings, accuracy increases by 0.03 percentage points and the ROC area increases by 0.01, while the remaining metrics are unchanged. Because this configuration yields the best observed performance, we use these parameter settings for the Simple Logistics model going forward.

5.4.2 Random Forest

Random Forest is a classifier that we have not optimized previously. Although it exposes many parameters, we focus on a single one: `numIterations` (I). This parameter controls the number of trees generated in the ensemble. In general, increasing the number of trees can improve performance by reducing variance and stabilizing predictions, but beyond a certain point additional trees yield diminishing returns while increasing computation time. We therefore explore a range of values for I to identify a practical trade-off between performance and runtime.

Table 22: Results regarding Random Forest's base (old) and improved parameter settings (new)

Metric	Old	New
Accuracy	78.45%	78.77%
TP.Rate	0.785	0.788
FP.Rate	0.452	0.455
Precision	0.769	0.773
Recall	0.785	0.788
F-Measure	0.766	0.768
ROC-Curve	0.790	0.796

We now examine Table 22 and observe that the optimized (new) configuration achieves a higher accuracy than the baseline (old) configuration. We tested `numIterations` (I) in increments of 100, from 0 up to 500. The best performance in Table 22 occurs at I = 200. Several metrics improve under this setting: accuracy increases by 0.32 percentage points, and precision increases from 0.769 to 0.773, which are the two largest improvements. Based on these results, we continue with I = 200 for Random Forest.

5.5 Exploring other meta-learners

Another strategy to improve predictive performance is to combine predictions from multiple classifiers. This can be done using ensemble meta-learners. An ensemble refers to a set of models that are considered jointly rather than individually. [6] We have already explored Random Forest, which is an ensemble method that builds on bagging. In this section, we evaluate additional ensemble approaches: Bagging, Boosting, Voting, and Stacking.

5.5.1 Bagging

Bagging (bootstrap aggregation) is primarily used to reduce variance. As noted earlier, Random Forest already includes bagging-like components. Reducing variance is important because high variance means that model predictions can change substantially in response to noise in the training data, which increases sensitivity to overfitting. Bagging addresses this by repeatedly sampling (with replacement) from the training set to create multiple bootstrap datasets. A separate classifier is trained on each dataset, and the predictions are aggregated (for example by majority vote). This aggregation typically yields more stable and robust predictions than any single model.

Because bagging is already a core component of Random Forest, we do not evaluate Bagging as an additional wrapper for Random Forest. Doing so would be redundant. The Bagging results shown in Table 23 are therefore based on Simple Logistics only.

Table 23: Results regarding meta-learner Bagging on Simple Logistics (new) and without its influence (old)

Metric	Old	New
Accuracy	78.62%	78.62%
TP.Rate	0.786	0.786
FP.Rate	0.467	0.467
Precision	0.772	0.772
Recall	0.786	0.786
F-Measure	0.765	0.765
ROC-curve	0.789	0.789

Looking at the results in Table 23, we compare the best-performing model from earlier (old) with the Bagging results (new). Bagging does not improve performance relative to the older model. One possible explanation is that the base model does not exhibit substantial variance, leaving limited room for bagging to provide gains. Based on these results, we retain the older model and do not continue with the Bagging configuration.

5.5.2 Boosting

Boosting methods are primarily used to reduce bias. Several boosting algorithms exist, but in this section we use AdaBoost, which is a widely used meta-classifier for boosting. High bias can lead to underfitting, where a model fails to capture relevant structure or relationships between the predictors and the class variable. Boosting addresses this by training a sequence of classifiers. It starts with a base classifier trained on the training set. Subsequent classifiers then place increased emphasis on instances that were misclassified in earlier rounds. This process continues until performance stabilizes or a predefined limit on the number of models is reached.

Because Random Forest is already an ensemble approach, we do not include it in this boosting experiment. [7]

Table 24: Results regarding meta-learner Boosting on Simple Logistics (new) and without its influence (old)

Metric	Old	New
Accuracy	78.62%	78.62%
TP.Rate	0.786	0.786
FP.Rate	0.467	0.467
Precision	0.772	0.772
Recall	0.786	0.786
F-Measure	0.765	0.765
ROC-curve	0.789	0.789

We now examine Table 24, which reports the results of AdaBoost. As with Bagging, AdaBoost does not improve accuracy relative to the best-performing single model. One possible explanation is that AdaBoost is less effective when the base learner is already relatively strong, leaving limited room for bias reduction. In contrast, boosting often provides larger gains when starting from a weaker learner. For these reasons, we do not continue with the AdaBoost configuration.

5.5.3 Voting

Voting is one of the simplest ensemble approaches, but it can still be effective. The method combines two or more classifiers and aggregates their predictions using a predefined combination rule. While the default rule is often the average of predicted class probabilities, we use majority voting. Under majority voting, each classifier "votes" for a class label, and the final prediction is the class receiving the most votes.

Table 25: Results regarding meta-learner Vote on Simple Logistics and Random Forest (new) and without its influence (old)

Metric	ZeroR	OneR	J48	Naïve.Bayes	Simple.Logistics	IBk	Random.Forest	Vote
Accuracy	73.12%	76.31%	77.79%	72.80%	78.62%	66.71%	78.77%	78.66%
TP.Rate	0.731	0.762	0.778	0.731	0.786	0.667	0.788	0.787
FP.Rate	0.731	0.540	0.454	0.363	0.467	0.522	0.455	0.498
Precision	0	0.740	0.761	0.745	0.772	0.664	0.773	0.776
Recall	0.731	0.762	0.778	0.731	0.786	0.667	0.788	0.787
F-Measure	0	0.792	0.761	0.733	0.765	0.666	0.768	0.757
ROC-curve	0.500	0.611	0.718	0.753	0.789	0.573	0.796	0.644

Looking at Table 25, we summarize the best results obtained with the Vote classifier. We tested voting across all available classifiers, independent of their earlier performance. The Simple Logistics and Random Forest models used in this experiment were configured with the optimized parameter settings described above. The Vote classifier achieves relatively high accuracy, precision, and recall. It performs as the runner-up after Random Forest, with only a 0.11 percentage point difference in accuracy.

However, the false-positive rate of the Vote classifier is higher than that of Simple Logistics, which ranks third. Overall, Random Forest remains the strongest individual contender. For that reason, we do not carry the Vote classifier forward as a candidate for the final model.

5.5.4 Stacking

Stacking is similar to voting in the sense that it combines multiple classifiers, but the underlying mechanism is different. Instead of aggregating predictions through a fixed rule (such as majority vote), stacking trains a meta-learner that learns how to best combine the predictions of the base models. This can improve performance if the base classifiers make complementary errors.

We evaluate stacking using Random Forest and Simple Logistics as base learners, with logistic regression as the meta-learner, which is commonly used for stacking. As before, we use the optimized parameter settings for the base classifiers.

Table 26: Results regarding meta-learner Stacking on Simple Logistics and Random Forest (new) and without its influence (old)

Metric	Simple.Logistics	Random.Forest	Stacking
Accuracy	78.62%	78.77%	79.04%
TP.Rate	0.786	0.788	0.790
FP.Rate	0.467	0.455	0.442
Precision	0.772	0.773	0.777
Recall	0.786	0.788	0.790
F-Measure	0.765	0.768	0.773
ROC-curve	0.789	0.796	0.803

We now examine Table 26 and observe that stacking increases accuracy to a new highest value. The Stacking classifier achieves 79.04% accuracy, outperforming both Simple Logistics and Random Forest. Although the performance differences between the models are small, selecting the best-performing approach remains preferable. We therefore carry the stacking configuration forward to the next section as a potential final model.

5.6 Confusion matrices

This section examines the confusion matrices of the best-performing classifiers: Random Forest and the Stacking ensemble. Because the relabeled class variable has only two levels, each confusion matrix has a 2×2 structure. Given that overall model performances are relatively close, inspecting the confusion matrices can provide additional detail and help guide the final model choice.

5.6.1 Random Forest

We first consider Table 27, which presents the confusion matrix for the optimized Random Forest model. Random Forest achieves an accuracy of 78.77% under its best-performing parameter settings, and the confusion matrix is based on these results.

a	b	<– classified as
47830	3672	a = 1
11283	7653	b = 2

Table 27: Confusion matrix based on the optimal settings regarding Random Forest

We examine the confusion matrix in Table 27. The relabeled outcome remains positively skewed: 5.9113×10^4 instances belong to class a and 1.1325×10^4 to class b. Performance also differs between the two classes. In particular, class b is predicted less accurately than class a. For example, the true-positive rate for class a is 0.929, whereas for class b it is 0.404, indicating that the model still favors class a.

Despite this imbalance, the number of correctly classified instances is substantially higher than in the original 14-class setting, and the binary formulation allows a more precise comparison between the two outcome groups. Although class b remains more difficult to predict, the model output is still informative and can support analyses aimed at understanding and potentially reducing longer hospital stays.

5.6.2 Stacking

Next, we evaluate the stacking classifier, which achieves the best overall performance observed so far. This model combines the two strongest base classifiers, Random Forest and Simple Logistics, using the stacking meta-learner. The resulting accuracy is 79.04%, and the corresponding confusion matrix is shown in Table 28.

a	b	<– classified as
47669	3833	a = 1
10928	8008	b = 2

Table 28: Confusion matrix based on the combining of Random Forest and Simple Logistics with the meta-learner Stacking

Looking at the results, we observe a pattern that is complementary to Random Forest. The number of predicted positives is slightly higher (by 194 instances). This difference reflects a trade-off: compared with Random Forest, stacking yields fewer true positives for class a but more true positives for class b. In other words, stacking shifts some predictions toward the minority class, which increases the number of correctly identified long-stay cases.

5.6.3 Conclusion

With a clearer view of the prediction distributions, we can select a final model. As discussed earlier, stacking achieved slightly better overall performance than Random Forest alone. Inspecting the confusion matrices confirms where the difference arises. Although stacking produces slightly fewer correct predictions for class a (47,669 versus 47,830), it predicts class b more accurately than Random Forest (8,008 versus 7,653). As a result, the total number of correctly classified instances (true positives plus true negatives) is higher for stacking. For this reason, we select stacking as the final model for the remainder of this study.

5.7 A ROC curve

Before finalizing the model choice, we also examine the receiver operating characteristic (ROC) curve. The ROC curve visualizes the trade-off between the true-positive rate (TPR) and the false-positive rate (FPR) across decision thresholds. It can be interpreted as a threshold-dependent extension of the confusion matrix.

In ROC space, FPR is plotted on the x-axis and TPR on the y-axis. The diagonal line represents random performance: points above the diagonal correspond to better-than-random classification (higher TPR at lower FPR), whereas points below indicate worse-than-random performance. In this section, we inspect the ROC curve for the stacking model and interpret its results.

```
roc_data <- read.table(
  "datasets/dataset_results_weka/roc_curve/roc-curve_stacking.arff",
                   sep = ",",
                   comment.char = "@")
names(roc_data) <- c("Instance_number",
                  "True_Positives",
                  "False_Negatives",
                  "False_Positives",
                  "True_Negatives",
                  "False_Positive_Rate",
                  "True_Positive_Rate",
                  "Precision",
                  "Recall",
                  "Fallout",
                  "FMeasure",
                  "Sample_Size",
                  "Lift",
                  "Threshold")

library(ggpubr)
colors <- c(classifier = "red", threshold = "blue")
plt <- ggplot(data = roc_data,
       mapping = aes(x = False_Positive_Rate, y = True_Positive_Rate)) +
    geom_point(mapping = aes(color = "classifier")) +
    geom_abline(aes(color = "threshold",
                    slope = 1,
                    intercept = 0)) +
    scale_color_manual(values = colors) +
    xlab("True Positive Rate") +
    ylab("False Positive Rate") +
    theme_pubr() +
```

```
    theme(legend.title = element_blank())
```

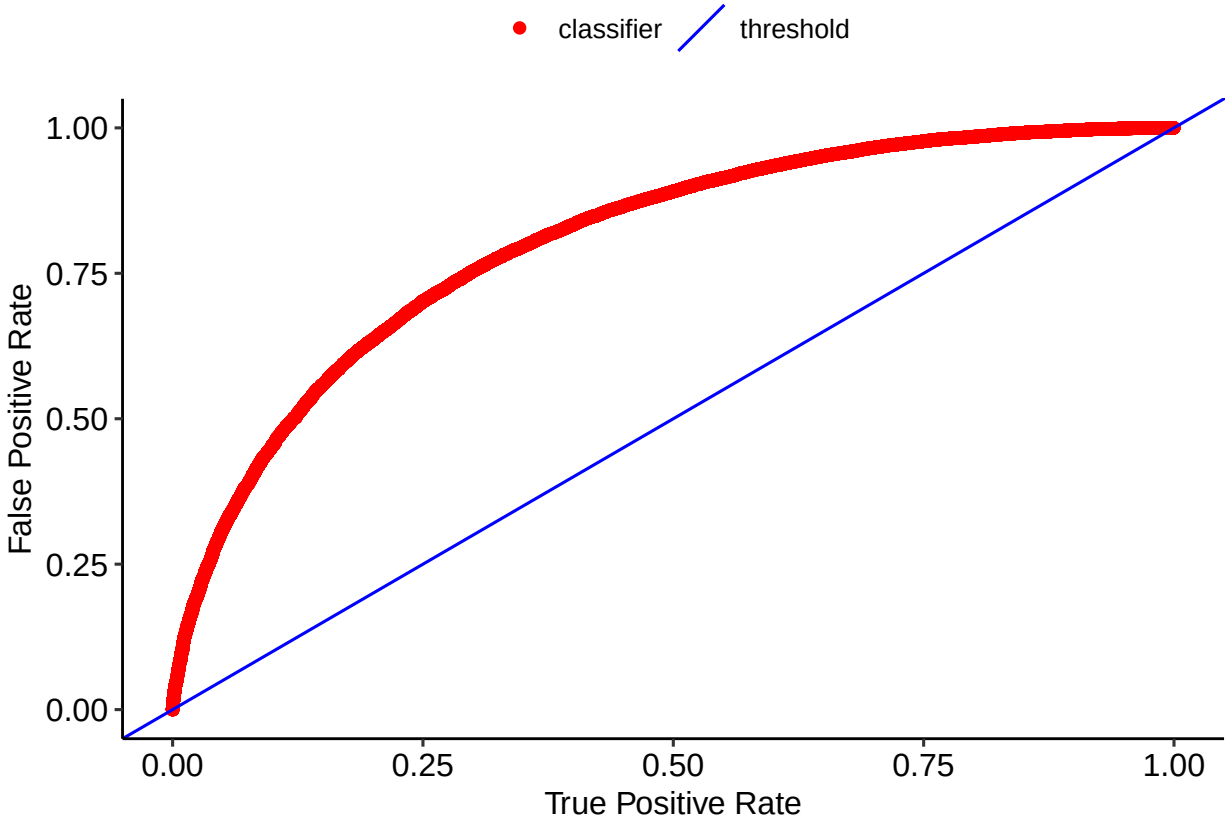


Figure 25: ROC Curve based on the combining of Random Forest and Simple Logistics with the meta-learner Stacking

Looking at Figure 23, we observe the ROC curve for the stacking model. The red curve represents the classifier's performance, and the diagonal reference line (blue) represents random classification. Because the ROC curve lies well above the diagonal, the model shows meaningful discriminative ability.

A reasonable operating region appears around a true-positive rate (TPR) of approximately 0.65 at a false-positive rate (FPR) of approximately 0.25. This indicates a sensitivity (recall) of about 0.65 at that threshold, while accepting a moderate false-positive rate. In practice, the preferred threshold depends on the relative cost of false positives versus false negatives in the intended use case.

The area under the ROC curve (AUC) is 0.803. AUC summarizes how well the model separates the two classes across all thresholds: higher values indicate better separability. An AUC of 0.803 suggests that the final model performs sufficiently well to be considered useful under this binary formulation.

Our next goal is to implement a Java wrapper that accepts new instances and generates predictions using the trained model. The wrapper is available in the project's Github repository.

6 References

References

[1] Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios, and John N. Clore, Impact of HbA1c Measurement on Hospital Readmission Rates: Analysis of 70,000 Clinical Database Patient Records, BioMed Research International, vol. 2014, Article ID 781670, 11 pages, 2014. DOI: https://doi.org/10.1155/2014/781670

[2] Centers for Disease Control and Prevention, National Center for Health Statistics, ICD-9, https://www.cdc.gov/nchs/icd/icd9.htm, November 6, 2015.

[3] Stiglic, G., Kocbek S., Pernek I., Kokol P., Comprehensive Decision Tree Models in Bioinformatics. DOI: 10.1371/journal.pone.0033812

[4] Eibe Frank, Administrator; Logistic VS Simple Logistic, https://weka.8497.n7.nabble.com/Logistic-VS-Simple-Logistic-td31410.html

[5] Hall, M, Class AttributeSelectedClassifier, https://weka.sourceforge.io/doc.dev/weka/classifiers/meta/AttributeSelectedClassifier.html.

[6] Thornton, C.; Hutter, F.; Hoos, H.H.; Leyton-Brown, K.: Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms, https://arxiv.org/pdf/1208.3719.pdf.

[7] Boinee, P; De Angelis, A; and Foresti, G.L.: Meta Random Forests, International Journal of Computational Intelligence Volume 2 Number 3, https://www.researchgate.net/publication/242416336_Meta_Random_Forests.