# Mini Projects R Programming Skills

Marc Stadler 20-712-535

20. December 2023

# Project 1 - OOP & SQL

In this first mini project I used object oriented programming as well as SQL. Specifically I write classes that could be used for keeping track of student's grades and the courses they book for the semester. My code first creates a class "student_factory" which is responsible for creating new instances of students as well as storing grades. If a student is created the information is automatically stores in the data base. When new information get added to a student as a new grade for example, the information in the data base need to be updated by using the update function on the student.

Further there is a class "courses" which creates the courses the students can book. This class stores the property of a course including the requirements for booking it. It further stores the information on which students are currently enrolled in it and if there is still free places. Like with the students class new instances of courses are directly saved into the SQL data base. New information such as the information on the free spaces can be saved using the update function. The list of enrolled students is saved into a separate table in the SQL data base.

My process for this mini project was as follows: I first created the students class and solved the different issues appearing a long the way. For example I had to figure out how to implement an attribute (grades) which can hold two different values (the actual grade itself as well as the number of credits the class gave). At this point I did not et implement any SQL code. As a next step I created instances of students and tried to read and change attributes to see if my getters and setters work correctly.

In a second step I created the "courses" class. There I had to figure out how to access the information from my created students and how to handle an attribute that can sometimes have only one value and for other courses two (the targeted study level which can be only BA or MA for some courses and for others it's open for either). During these process I also created instances of courses to test the current implementation. I also encounters some issues when adding students to courses since at first it was possible to add a student more than once to a course which I corrected by looking at the matriculate number as a unique identifier.

In a last step I implemented the SQL part. This was the most tricky part since I encountered several issues when trying to automatically save the students and courses to the SQL table. At first I used a non-parameterized version trying to use string interpolation, however this did not run as intended. Even if it would have ran this approach would have been susceptible to SQL injection. I managed to get the code to run with a parametized form that does not directly take the variables as SQL code but treats them as parameters.

Below you can see some examples of how instances of students and courses get created and how they are saved into the SQL tables. (1) In the first step I create three SQL tables, one for the students, one for the courses and one for the enrollments. The SQL tables are created in my local memory and not saved permanently to a data base. If this would be implemented for an actual purpose this could be changed to a proper data base. For the students table the matricule numbers are the primary key since they will never be identical. For the courses class the primary key is the unique course id while the enrollment table has its own enrollment id as primary key.

```r
library(DBI)
library(RSQLite)
con = dbConnect(RSQLite::SQLite(), ":memory:")
dbListTables(con)
```

```
character(0)
```

```r
dbSendQuery(con, "CREATE TABLE students (
              name TEXT, sex TEXT, birthday DATE, level TEXT, semester INTEGER,
            major TEXT, matricule TEXT PRIMARY KEY, grades TEXT DEFAULT NULL)")
```

```
<SQLiteResult>
  SQL  CREATE TABLE students (
              name TEXT, sex TEXT, birthday DATE, level TEXT, semester INTEGER,
            major TEXT, matricule TEXT PRIMARY KEY, grades TEXT DEFAULT NULL)
  ROWS Fetched: 0 [complete]
       Changed: 0
```

```r
sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM students"))
sub.df
```

```
[1] name      sex       birthday  level     semester  major     matricule
[8] grades
<0 Zeilen> (oder row.names mit Länge 0)
```

```r
#create table courses
dbSendQuery(con, "CREATE TABLE courses (
              name TEXT, capacity INTEGER, min_gpa REAL,
            target_level TEXT, course_id TEXT PRIMARY KEY,
            remaining_capacity INTEGER)")
```

```
<SQLiteResult>
  SQL  CREATE TABLE courses (
              name TEXT, capacity INTEGER, min_gpa REAL,
            target_level TEXT, course_id TEXT PRIMARY KEY,
            remaining_capacity INTEGER)
  ROWS Fetched: 0 [complete]
       Changed: 0
```

```r
sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM courses"))
sub.df
```

```
[1] name                 capacity            min_gpa             target_level
[5] course_id            remaining_capacity
<0 Zeilen> (oder row.names mit Länge 0)
```

```r
#create table enrollments
dbSendQuery(con, "CREATE TABLE enrollments (
              enrollment_id INTEGER PRIMARY KEY, course_id INTEGER,
            student_matricule TEXT,
            FOREIGN KEY (course_id) REFERENCES courses(course_id),
            FOREIGN KEY (student_matricule) REFERENCES students(matricule))")
```

```
<SQLiteResult>
  SQL  CREATE TABLE enrollments (
              enrollment_id INTEGER PRIMARY KEY, course_id INTEGER,
            student_matricule TEXT,
            FOREIGN KEY (course_id) REFERENCES courses(course_id),
            FOREIGN KEY (student_matricule) REFERENCES students(matricule))
  ROWS Fetched: 0 [complete]
       Changed: 0
```

```r
sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM enrollments"))
sub.df
```

```
[1] enrollment_id     course_id         student_matricule
<0 Zeilen> (oder row.names mit Länge 0)
```

(2) Next I create the student factory. The factory is responsible for creating new instances of students. A student needs to have following attributes to get created: a name, a birthday, a sex (male / female) , a study level (Bachelor or Master), a number of the current semester, a major and a unique matriculate number. The column grades in the SQL table has already been created previously but it does not get assigned a value yet when one created a new student.

```r
library(R6)
library(assertive.types)
student_factory = R6Class(
  "Student",
  private = list(
    ..name = NULL,
    ..birthday = NULL,
    ..sex = NULL,
    ..level = NULL,
    ..semester = NULL,
    ..major = NULL,
    ..matricule = NULL,
    ..grades = data.frame(grade = numeric(), credits = numeric())
  ),
  public = list(
    initialize = function(name, sex, birthday, level, semester, major, matricule){
      if(missing(name)){
        stop("Name not specified")
      }
      if(missing(sex)){
        stop("Sex not specified")
      }
      if(missing(birthday)){
        stop("Birthday not specified")
      }
      if(missing(level)){
        stop("Level of studies not specified")
      }
      if(missing(major)){
        stop("Major not specified")
      }
      if(missing(matricule)){
        stop("Matricule number not specified")
      }
      assert_is_character(name)
      assert_is_character(sex)
      assert_is_date(birthday)
      assert_is_character(level)
      assert_is_numeric(semester)
      assert_is_character(major)
      assert_is_character(matricule)
      private$..name = name
      private$..sex = sex
      private$..birthday = birthday
      private$..level = level
      private$..semester = semester
      private$..major = major
      private$..matricule = matricule
      self$insertIntoDB()
    },
    insertIntoDB = function() {
      sql <- sprintf("INSERT INTO students(name, sex, birthday, level, semester,
                     major, matricule) VALUES ('%s', '%s', '%s', '%s', '%s',
                     '%s', '%s')",
                     private$..name, private$..sex, private$..birthday,
                     private$..level, private$..semester, private$..major,
                     private$..matricule)
      dbExecute(con, sql)
    },
    update = function() {
      sql <- sprintf("UPDATE students SET name = '%s', sex = '%s',
      birthday = '%s', level = '%s', semester = '%s',
      major = '%s' WHERE matricule = '%s'",
      private$..name, private$..sex, private$..birthday, private$..level,
      private$..semester, private$..major, private$..matricule)
      dbExecute(con, sql)
      if (nrow(private$..grades) > 0) {
        grades_str <- paste(sprintf("grades = '%s'",
        toString(private$..grades)), collapse = ", ")
        sql <- sprintf("UPDATE students SET %s WHERE matricule = '%s'",
                       grades_str, private$..matricule)
        dbExecute(con, sql)
      }
    },
    new_grade = function(grade, credits){
      assert_is_numeric(grade)
      assert_is_numeric(credits)
      private$..grades <- rbind(private$..grades,
      data.frame(grade = grade, credits = credits))
    },
    calculate_gpa = function() {
      if (nrow(private$..grades) == 0) {
        stop("There are no grades yet")
      }
      weighted_grades <- private$..grades$grade * private$..grades$credits
      total_credits <- sum(private$..grades$credits)
      gpa <- sum(weighted_grades) / total_credits
      return(gpa)
    }
  ),
  active = list(
    name = function(value){
      if(missing(value)){
        private$..name
      }else{
        assert_is_character(value)
        private$..name = value
      }
    },
    age = function(){
      as.integer(floor((as.Date(Sys.time()) - private$..birthday)  /365))
    },
    sex = function(){
      private$..sex
    },
    level = function(){
      private$..level
    },
    semester = function(){
      private$..semester
    },
    major = function(value){
      if(missing(value)){
        private$..major
      }else{
        assert_is_character(value)
        private$..major = value
      }
    },
    grades = function(){
      private$..grades
    },
    matricule = function(){
      private$..matricule
    }
  )
)
```

(3) Now that the students class has been created I initialize five different students. They will automatically get saved into the SQL table. I further also assign them grades to be stored into the SQL table. I will use the update function to save the changes. You also see some getters and setters in action to see that they work.

```
marc = student_factory$new("Marc", "male", as.Date("2000-09-24"),
                           "Master", 1, "Political Science","30-783-829")

raphael = student_factory$new("Raphael", "male", as.Date("2003-05-02"),
                              "Bachelor", 3, "Political Science","30-643-098")

clara = student_factory$new("Clara", "female", as.Date("2000-08-07"),
                            "Bachelor", 2, "Political Science","30-829-920")

nicole = student_factory$new("Nicole", "female", as.Date("2000-08-07"),
                             "Bachelor", 2, "Political Science","30-432-421")

fatima = student_factory$new("Fatima", "female", as.Date("1999-02-14"),
                             "Master", 3, "Political Science","29-937-019")

leandro = student_factory$new("Leandro", "male", as.Date("2002-12-01"),
                              "Bachelor", 1, "Political Science","31-347-982")

marc$age
```

```
[1] 23
```

```r
marc$sex
```

```
[1] "male"
```

```r
marc$semester
```

```
[1] 1
```

```r
marc$major
```

```
[1] "Political Science"
```

```r
marc$level
```

```
[1] "Master"
```

```r
sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM students"))
sub.df
```

```
     name    sex   birthday    level semester             major  matricule
1    Marc   male 2000-09-24   Master        1 Political Science 30-783-829
2 Raphael   male 2003-05-02 Bachelor        3 Political Science 30-643-098
3   Clara female 2000-08-07 Bachelor        2 Political Science 30-829-920
4  Nicole female 2000-08-07 Bachelor        2 Political Science 30-432-421
5  Fatima female 1999-02-14   Master        3 Political Science 29-937-019
6 Leandro   male 2002-12-01 Bachelor        1 Political Science 31-347-982
  grades
1   <NA>
2   <NA>
3   <NA>
4   <NA>
5   <NA>
6   <NA>
```

```r
marc$major = "Political Data Journalism"
marc$major
```

```
[1] "Political Data Journalism"
```

```r
marc$update()
sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM students"))
sub.df
```

```
     name    sex   birthday    level semester                      major
1    Marc   male 2000-09-24   Master        1 Political Data Journalism
2 Raphael   male 2003-05-02 Bachelor        3         Political Science
3   Clara female 2000-08-07 Bachelor        2         Political Science
4  Nicole female 2000-08-07 Bachelor        2         Political Science
5  Fatima female 1999-02-14   Master        3         Political Science
6 Leandro   male 2002-12-01 Bachelor        1         Political Science
  matricule grades
1 30-783-829   <NA>
2 30-643-098   <NA>
3 30-829-920   <NA>
4 30-432-421   <NA>
5 29-937-019   <NA>
6 31-347-982   <NA>
```

```r
marc$new_grade(grade=5,credits=6)
marc$new_grade(grade=6,credits=3)
marc$new_grade(grade=6,credits=12)
marc$grades
```

```
  grade credits
1     5       6
2     6       3
3     6      12
```

```r
marc$calculate_gpa()
```

```
[1] 5.714286
```

```r
raphael$new_grade(grade=4,credits=6)
raphael$new_grade(grade=4,credits=3)
raphael$new_grade(grade=4,credits=12)
raphael$grades
```

```
  grade credits
1     4       6
2     4       3
3     4      12
```

```r
raphael$calculate_gpa()
```

```
[1] 4
```

```r
clara$new_grade(grade=4,credits=6)
clara$new_grade(grade=5.5,credits=3)
clara$new_grade(grade=4.3,credits=12)
clara$new_grade(grade=6,credits=9)
clara$grades
```

```
  grade credits
1   4.0       6
2   5.5       3
3   4.3      12
4   6.0       9
```

```r
clara$calculate_gpa()
```

```
[1] 4.87
```

```r
leandro$new_grade(grade=6,credits=9)
leandro$new_grade(grade=4.5,credits=1)
leandro$new_grade(grade=5.3,credits=6)
leandro$grades
```

```
  grade credits
1   6.0       9
2   4.5       1
3   5.3       6
```

```r
leandro$calculate_gpa()
```

```
[1] 5.64375
```

```r
fatima$new_grade(grade=6,credits=9)
fatima$new_grade(grade=5.75,credits=1)
fatima$new_grade(grade=4.8,credits=6)
fatima$grades
```

```
  grade credits
1  6.00       9
2  5.75       1
3  4.80       6
```

```r
fatima$calculate_gpa()
```

```
[1] 5.534375
```

```r
nicole$new_grade(grade=4,credits=9)
nicole$new_grade(grade=4.75,credits=1)
nicole$new_grade(grade=6,credits=6)
nicole$new_grade(grade=5.5,credits=12)
nicole$grades
```

```
  grade credits
1  4.00       9
2  4.75       1
3  6.00       6
4  5.50      12
```

```r
nicole$calculate_gpa()
```

```
[1] 5.098214
```

```r
marc$update()
```

```
[1] 1
```

```r
clara$update()
```

```
[1] 1
```

```r
raphael$update()
```

```
[1] 1
```

```r
leandro$update()
```

```
[1] 1
```

```r
fatima$update()
```

```
[1] 1
```

```r
nicole$update()
```

```
[1] 1
```

```r
sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM students"))
sub.df
```

```
     name    sex   birthday    level semester                      major
1    Marc   male 2000-09-24   Master        1 Political Data Journalism
2 Raphael   male 2003-05-02 Bachelor        3         Political Science
3   Clara female 2000-08-07 Bachelor        2         Political Science
4  Nicole female 2000-08-07 Bachelor        2         Political Science
5  Fatima female 1999-02-14   Master        3         Political Science
6 Leandro   male 2002-12-01 Bachelor        1         Political Science
   matricule                             grades
1 30-783-829            c(5, 6, 6), c(6, 3, 12)
2 30-643-098            c(4, 4, 4), c(6, 3, 12)
3 30-829-920  c(4, 5.5, 4.3, 6), c(6, 3, 12, 9)
4 30-432-421 c(4, 4.75, 6, 5.5), c(9, 1, 6, 12)
5 29-937-019        c(6, 5.75, 4.8), c(9, 1, 6)
6 31-347-982         c(6, 4.5, 5.3), c(9, 1, 6)
```

(4) Now I create the second class called courses_factory. A course needs to have the following attributes to get created: a name, a maximum capacity, a minimum weighted grade requirement to join the course, a target study level (either BA or MA or either) and a unique course ID.

```r
courses_factory = R6Class(
  "Courses",
  private = list(
    ..name = NULL,
    ..capacity = NULL,
    ..remaining_capacity = NULL,
    ..min_gpa = NULL,
    ..target_level = NULL,
    ..course_id = NULL,
    ..students = list()
  ),
  public = list(
    initialize = function(name, capacity, min_gpa, target_level, course_id){
      if(missing(name)){
        stop("Name not specified")
      }
      if(missing(capacity)){
        stop("Maximum capacity not specified")
      }
      if(missing(min_gpa)){
        stop("Minimum GPA not specified")
      }
      if(missing(target_level)){
        stop("Targeted study level not specified")
      }
      if(missing(course_id)){
        stop("Course ID not specified")
      }
      assert_is_character(name)
      assert_is_numeric(capacity)
      assert_is_numeric(min_gpa)
      assert_is_character(target_level)
      assert_is_character(course_id)
      private$..name = name
      private$..capacity = capacity
      private$..remaining_capacity = capacity
      private$..min_gpa = min_gpa
      private$..target_level = strsplit(target_level, ",")[[1]]
      private$..course_id = course_id
      self$insertIntoDB()
    },
    insertIntoDB = function() {
      target_levels <- toString(private$..target_level)
      sql <- sprintf("INSERT INTO courses(name, capacity, min_gpa,
      target_level, course_id, remaining_capacity) VALUES ('%s', '%s',
      '%s', '%s', '%s', '%s')",
      private$..name, private$..capacity, private$..min_gpa,
      target_levels, private$..course_id,
      private$..remaining_capacity)
      dbExecute(con, sql)
    },
    update = function() {
      target_levels <- paste(private$..target_level, collapse = ',')
      sql <- sprintf("UPDATE courses SET name = '%s', capacity = '%s',
                     min_gpa = '%s', target_level = '%s',
                     remaining_capacity = '%s' WHERE course_id = '%s'",
                     private$..name, private$..capacity, private$..min_gpa,
                     target_levels, private$..remaining_capacity,
                     private$..course_id)
      dbExecute(con, sql)
     },
    add_student = function(student) {
      if (length(private$..students) >= private$..capacity) {
        stop("Class is already fully booked")
      }
      if (any(sapply(private$..students,
        function(s) s$matricule == student$matricule))) {
```

```r
        stop("Student is already in the class")
      }
      if (!any(student$level %in% private$..target_level)) {
        stop("This class is not designated for your level of study Expected level: ",
             private$..target_level)
      }
      student_gpa <- student$calculate_gpa()
      if (!is.na(student_gpa) && student_gpa < private$..min_gpa) {
        stop("The minimum GPA for this class is ", private$..min_gpa,
             " while you have a GPA of ", student_gpa)
      }
      private$..students[[length(private$..students) + 1]] <- student
      private$..remaining_capacity <- private$..remaining_capacity - 1

      # Insert enrollment record into the enrollments table
      enroll_sql <- sprintf("INSERT INTO enrollments(course_id,
                            student_matricule) VALUES ('%s', '%s')",
                            private$..course_id, student$matricule)
      dbExecute(con, enroll_sql)
    },
    get_students = function() {
      return(sapply(private$..students, function(student) student$name))
    }
  ),
  active = list(
    name = function(value){
      if(missing(value)){
        private$..name
      }else{
        assert_is_character(value)
        private$..name = value
      }
    },
    capacity = function(){
      private$..capacity
    },
    min_gpa = function(){
      private$..min_gpa
    },
    target_level = function(){
      private$..target_level
    },
    remaining_capacity = function(){
      private$..remaining_capacity
    },
    course_id = function(){
      private$..course_id
    }
    )
  )
```

(5) Now that both classes have been created I create three instances of courses and fill them with students. Below you can see this process along with some error messages indicating that a student does not have a sufficiently high grade or the wrong study level to join the class as well as the case of a class that is already fully booked.

```
math1 = courses_factory$new("Mathematics I", 5, 4.5, "Bachelor","2030-PS203")
math2 = courses_factory$new("Mathematics II", 5, 5, "Master","2030-PS204")
acwri = courses_factory$new("Academic Writting", 5, 1, "Bachelor,Master","2030-PS205")

sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM courses"))
sub.df
```

```
               name capacity min_gpa     target_level  course_id
1     Mathematics I        5     4.5         Bachelor 2030-PS203
2    Mathematics II        5     5.0           Master 2030-PS204
3 Academic Writting        5     1.0 Bachelor, Master 2030-PS205
  remaining_capacity
1                  5
2                  5
3                  5
```

```
math1$add_student(marc)
```

```
Error in math1$add_student(marc): This class is not designated for your level of study E
```

```
math1$get_students()
```

```
list()
```

```
math1$remaining_capacity
```

```
[1] 5
```

```
math1$add_student(clara)
```

```
[1] 1
```

```
math1$get_students()
```

```
[1] "Clara"
```

```r
math1$remaining_capacity
```

```
[1] 4
```

```r
math1$add_student(raphael)
```

```
Error in math1$add_student(raphael): The minimum GPA for this class is 4.5 while you hav
```

```r
math1$get_students()
```

```
[1] "Clara"
```

```r
math1$remaining_capacity
```

```
[1] 4
```

```r
math2$add_student(marc)
```

```
[1] 1
```

```r
acwri$add_student(marc)
```

```
[1] 1
```

```r
acwri$add_student(raphael)
```

```
[1] 1
```

```r
acwri$add_student(clara)
```

```
[1] 1
```

```r
acwri$add_student(fatima)
```

```
[1] 1
```

```r
acwri$add_student(leandro)
```

```
[1] 1
```

```r
acwri$add_student(nicole)
```

```
Error in acwri$add_student(nicole): Class is already fully booked
```

```r
acwri$get_students()
```

```
[1] "Marc"    "Raphael" "Clara"   "Fatima"  "Leandro"
```

```r
acwri$remaining_capacity
```

```
[1] 0
```

```r
math1$update()
```

```
[1] 1
```

```r
math2$update()
```

```
[1] 1
```

```r
acwri$update()
```

```
[1] 1
```

```r
sub.df = dbFetch(dbSendQuery(con, "SELECT * FROM courses"))
sub.df
```

```
               name capacity min_gpa    target_level  course_id
1     Mathematics I        5     4.5        Bachelor 2030-PS203
2    Mathematics II        5     5.0          Master 2030-PS204
3 Academic Writting        5     1.0 Bachelor,Master 2030-PS205
  remaining_capacity
1                  4
2                  4
3                  0
```

# Project 2 - parallel and efficient code

For my bachelor thesis I have recoded a lot of variables and further used some packages to calculate predicted probabilities. For this second mini project I will look at how I recoded variables and see if there is a way to make this process more efficient. Further I will have a look at the predicts function form glm.predict and see how parallel computing can make a difference in calculating the predicted probabilities and discrete changes.

## recoding variables

For many of the variables I recoded I used a step for step approach which is not that efficient for variables with a lot of categories that should be created, both in terms of the time it takes to write the command, the time it takes to run the code and the readability for someone who looks at the code for the first time.

As an example I include how I recoded the variable containing the level of education a person has:

```
data.NWB <- readstata13::read.dta13("1179_Selects2019_PES_data_v1.1.0.dta",
                                    convert.factors = FALSE)
data.NWB$edu[data.NWB$f21310<=4] <- 1
data.NWB$edu[data.NWB$f21310>=5&data.NWB$f21310<=10] <- 2
data.NWB$edu[data.NWB$f21310>=11&data.NWB$f21310<=13] <- 3
```

This code could be written more efficiently in one statement using the cut function. This approach is also easier to understand for a third party looking at the code.

```
data.NWB$edu <- cut(data.NWB$f21310,
                    breaks = c(-Inf, 4, 10, 13),
                    labels = c(1, 2, 3),
                    include.lowest = TRUE)
```

To test if my rewritten code is not only more precise but also more time efficient, I will test the time it takes to do this task 10'000 times for both versions and compare the outcome.

```
library(microbenchmark)

original_recode <- function() {
  data.NWB$edu[data.NWB$f21310<=4] <- 1
  data.NWB$edu[data.NWB$f21310>=5&data.NWB$f21310<=10] <- 2
  data.NWB$edu[data.NWB$f21310>=11&data.NWB$f21310<=13] <- 3
}


cut_function = function() {
  data.NWB$edu <- cut(data.NWB$f21310,
                      breaks = c(-Inf, 4, 10, 13),
                      labels = c(1, 2, 3),
                      include.lowest = TRUE)
}


microbenchmark(original_recode(),cut_function(), times = 10000)
```

```
Unit: microseconds
              expr   min    lq     mean median    uq     max neval
 original_recode() 302.4 383.4 541.6263  390.9 404.0 73840.1 10000
    cut_function() 266.1 325.8 397.6347  333.5 346.8 72819.2 10000
```

While the difference in processing time is relatively small it is still noticeable. The cut() version of the code has a median that is around 15% smaller compared to the original code. For recording a variable in a bachelor thesis this alone would not be a convincing argument yet though since the data set is often not extremly big and the calcualtions do not take much time. Still, the verison with the cut() function is much more elegant compared to the first one.

## glm.predict and parallel computing

The predicts function from the glm.predict package has a built in option for parallel computing. The default is on FALSE however. When I used this package for my bachelor thesis I did not use this option since I did not really know it existed and I was also not dealing with a very large amount of data which would have made the search for a more efficient method necessary.

For this task I will first run some of the predicts() function I used in my BA-thesis and measure the time it takes for several iterations to run. In a second step I will use parallel computing and analyze the differences in time.

For this exercise I will directly load the fully recoded data set to save space.

```
load("data_BA.Rda")
library(texreg)
library(glm.predict)
```

## Simple model with few predictions

First I am running a very simple model which only contains one variable without any control variables yet. Since the independent variable has only two values it can take on it is very easy and quick to calculate predicted probabilities for these. I let both the non parallel version and the parallel version run for 10 times each and compared the results. As you can see the parallel version took around 400 times longer to run compared to the non parallel version. This is due to the fact that the model is very simple and few predicted probabilities have to be calculated. Further the number of simulations is low at 10'000.

The parallel version needs to detect the number of courses and set up the parallel computation for each of the 10 iterations which costs a lot of time compared to the little time saved in the computation of the very simple 10'000 predicted probability calculations.

In such a situation it is not only not necessary to use parallel code, that approach is even less efficient compared to a non parallel version.

```
#H1: Modell 1 - ohne Kontrollvariablen
h1.ml1 <- glm(voted.svp~ident.native.fact,
              data = migration.NWB, weights = weight_d_p,
              family = binomial(link = "logit"))
screenreg(h1.ml1)

#H1: Modell 2 - Mit Kontrollvariablen Alter, Geschlecht, Bildung, Einkommen
h1.ml2 <- glm(voted.svp~ident.native.fact + gender + age + edu.fact + income,
              data = migration.NWB, weights = weight_d_p,
              family = binomial(link = "logit"))
screenreg(h1.ml2)
```

```
microbenchmark(predicts(h1.ml1, "all", sim.count = 10000, type = "simulation",
                        set.seed = 1848),predicts(h1.ml1, "all",
                        sim.count = 10000, type = "simulation",
                        set.seed = 1848, doPar = TRUE), times = 10)
```

```
Unit: milliseconds

               predicts(h1.ml1, "all", sim.count = 10000, type = "simulation",      set.
 predicts(h1.ml1, "all", sim.count = 10000, type = "simulation",      set.seed = 1848, d
       min       lq       mean    median        uq       max neval
    8.0608    8.868    9.48881    9.2548    9.7313   11.3449    10
 3164.3358 3456.505 3851.71668 3822.4725 4207.0380 4582.2748    10
```

## more complex model with more predictions

The second model is an expansion of the first model which is nested in this second model. Additionally to the independent variable I now include various control variables that were of importance for my research question.

For the predicted probabilities I expanded the complexity a bit compared to my actual use case for my BA thesis so the calculations get more complex. I am now interested in the predicted probabilities for both values of the independent variable of whether someone identifies as native, for both genders and all ages for a person with the most common education level and a median income. Further I increased the number of simulations for this model to 100'000.

I am once again running both the non-parallel and the parallel version 10 times to compare the average time used for the calculations. One can see that the parallel version is slightly faster this time compared to the non-parallel version. This can be explained with the higher complexity of the model where the faster computing time is more significant relative to the time it takes to set the parallel computing up.

```r
microbenchmark(predicts(h1.ml2, "all;F;all;mode;median", sim.count = 100000,
                type = "simulation", set.seed = 1848),
                predicts(h1.ml2, "all;F;all;mode;median",
                sim.count = 100000, type = "simulation",
                set.seed = 1848, doPar = TRUE), times = 10)
```

```
Unit: seconds

               predicts(h1.ml2, "all;F;all;mode;median", sim.count = 1e+05,      type =
 predicts(h1.ml2, "all;F;all;mode;median", sim.count = 1e+05,      type = "simulation",
      min        lq      mean   median        uq       max neval
 13.16840 13.269367 13.492290 13.37518 13.584481 14.360890    10
  6.51909  6.668394  7.081963  7.07443  7.563851  7.632317    10
```

In the following Table 1 you can see the regression output of the two models used for reference.

```r
texreg(list(h1.ml1, h1.ml2), single.row = FALSE, stars = c(0.001,  0.01, 0.05),
        custom.model.names=c("Modell 1", "Modell 2"),
        custom.coef.names=c("Intercept", "Starke Identität als Einheimsiche*r",
        "Frau", "Alter", "Bildung (Sekundar II)", "Bildung (Tertiär)",
        "Einkommen"),
        digits=2,
        caption="Identität als Einheimische*r und die Wahl der SVP | Def. MH BfS",
        caption.above=T,
        custom.note=paste("%stars."))
```

Table 1: Models used for calcualting predicted probabilities

| | Modell 1 | Modell 2 |
|---|---|---|
| Intercept | $-2.20^{***}$ | $-2.22^{**}$ |
| | (0.35) | (0.75) |
| Starke Identität als Einheimsiche*r | $1.04^{**}$ | $1.05^{**}$ |
| | (0.37) | (0.38) |
| Frau | | $-0.02$ |
| | | (0.25) |
| Alter | | $0.02^{*}$ |
| | | (0.01) |
| Bildung (Sekundar II) | | $-0.22$ |
| | | (0.48) |
| Bildung (Tertiär) | | $-0.61$ |
| | | (0.50) |
| Einkommen | | $-0.08^{*}$ |
| | | (0.04) |
| AIC | 368.35 | 360.10 |
| BIC | 376.11 | 387.24 |
| Log Likelihood | $-182.18$ | $-173.05$ |
| Deviance | 427.31 | 406.87 |
| Num. obs. | 357 | 357 |

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$.