Problem Statement:

How generally or narrowly do students learn? How quickly or slowly? Will the rate of improvement vary between students? What does it mean for one problem to be similar to another? It might depend on whether the knowledge required for one problem is the same as the knowledge required for another. But is it possible to infer the knowledge requirements of problems directly from student performance data, without human analysis of the tasks?

To predict student performance on mathematical problems from logs of student interaction with Intelligent Tutoring Systems. The available data takes the form of records of interactions between students and computer-aided-tutoring systems. The students solve problems in the tutor and each interaction between the student and computer is logged as a transaction.

There are four key terms important to build the data. These are **Problem, Step, Knowledge Component and Opportunity.**

- A problem is a task for a student to perform that typically involves multiple steps.
- A step is an observable part of the solution to a problem. This whole collection of steps comprises the solution. The last step can be considered the "answer", and the others are "intermediate" steps.
- A knowledge component is a piece of information that can be used to accomplish tasks, perhaps along with other knowledge components.
- An opportunity is a chance for a student to demonstrate whether he or she has learned a given knowledge component. A student's opportunity count for a given knowledge component increases by 1 each time the student encounters a step that requires this knowledge component.

One of the aspects of the dataset is not every student is given the same problem set. There were few cases where a problem was solved only by 2-3 students. Also students improve over the course as they master some skills while solving similar problems.

Let us get back to school days. Let us recall all those old batch-mates. Studying in the same class, spending same number of hours under the guidance of a common faculty, did all of us always score the same grades? The answer, we guess, is no. A student's proficiency does have a role to play.

If, however, we focus on just one of many, would that student be scoring the same grade after taking the same test repeatedly? Again, most likely the answer is no. A student's learning curve has its own impact. The outcome here is also a function of difficulty level of each step in the given problem statement.

The goal is to build a model which will predict the performance of a student which relates to the probability of getting the answers to a problem in the first attempt.

Data Format

Following are the raw features and their descriptions of the original dataset:

1. Row: This refers to the row number.
2. Anon Student Id: This is unique, anonymous identifier for a student.
3. Problem Hierarchy: This refers to the hierarchy of curriculum levels containing the problem.
4. Problem Name: This is unique identifier for a problem.
5. Problem View: This is total number of times the student encountered the problem so far.
6. Step Name: Each problem consists of one or more steps. The step name is unique within each problem, but there may be collisions between different problems, so the only unique identifier for a step is the pair of problem name and step name.
7. Step Start Time: This refers to the starting time of the step.
8. First Transaction Time: This refers to the time of the first transaction toward the step.
9. Correct Transaction Time: This refers to the time of the correct attempt toward the step, if there was one.
10. Step End Time: This refers to the time of the last transaction toward the step.
11. Step Duration: This refers to the elapsed time of the step in seconds, calculated by adding all of the durations for transactions those were attributed to the step.
12. Correct Step Duration: This refers to the step duration if the first attempt for the step was correct.
13. Error Step Duration: This refers to the step duration if the first attempt for the step was an error (incorrect attempt or hint request).
14. Correct First Attempt: This is the tutor's evaluation of the student's first attempt on the step. It takes value 1 if correct, 0 if an error. This is the target variable.
15. Incorrects: This refers to total number of incorrect attempts by the student on the step.
16. Hints: This refers to total number of hints requested by the student for the step.
17. Corrects: This refers to total number of correct attempts by the student for the step. This increases only if the step is encountered more than once.
18. KC (KC Model Name): KC refers to knowledge component. A KC model represents a set of identified skills that are used in a problem, where available. A step can have multiple KCs assigned to it. Multiple KCs for a step are separated by two tildes. Since opportunity describes practice by knowledge component, the corresponding opportunities are similarly separated by two tildes.
19. Opportunity (KC Model Name): This refers to a name of the skill a student has learnt in the past which can be used each time the student encounters a step with the listed knowledge component. Steps with multiple KCs have multiple opportunity numbers separated by two tildes.

Importing the dataset into the R Environment

On importing the dataset which is in a tab-separated variable (.tsv) format, we see transactions have been consolidated and displayed by student and step, producing a step record table.

A transaction is an interaction between the student and the tutoring system. Each hint request, incorrect attempt, or correct attempt is a transaction, and each recorded transaction is referred to as an attempt for a step.

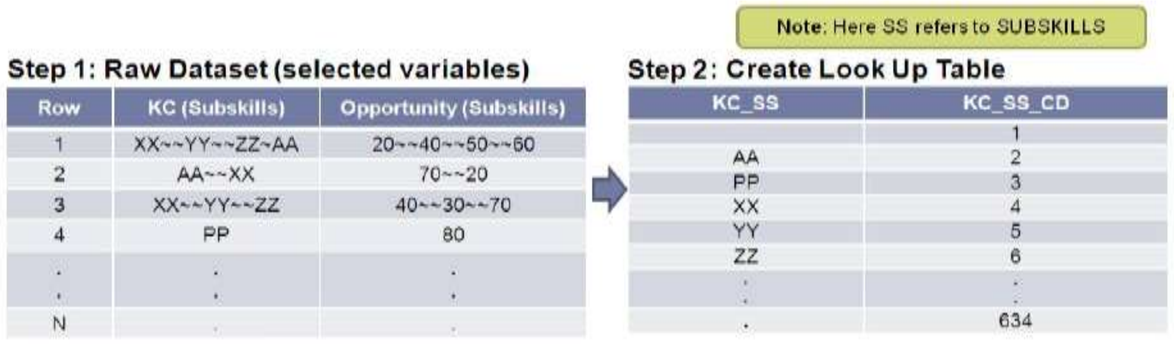
There are around 10 million transactions of 4796 unique students.

Feature Engineering

Our interest is to develop a model based on the problems a student works on instead of the steps he uses to solve the problem.

The first step is to keep track of the number (count) of skills that a step has to offer (KC Skill) and skill learnt by the student (Opportunity). Since a step can have multiple KCs assigned to it, multiple KCs for a step are separated by ~~ (two tildes). The opportunity describes practice by knowledge component, the corresponding opportunities are similarly separated by ~~.

In R, a function was written to count separate skills that each step will provide using a delimiter based on the number of '~~'.



Note: Here SS refers to SUBSKILLS

Step 1: Raw Dataset (selected variables)

Row	KC (Subskills)	Opportunity (Subskills)
1	XX~~YY~~ZZ~AA	20~~40~~50~~60
2	AA~~XX	70~~20
3	XX~~YY~~ZZ	40~~30~~70
4	PP	80
.	.	.
.	.	.
N	.	.

Step 2: Create Look Up Table

KC_SS	KC_SS_CD
	1
AA	2
PP	3
XX	4
YY	5
ZZ	6
.	.
.	.
.	634

Creation of KC variables look-up table

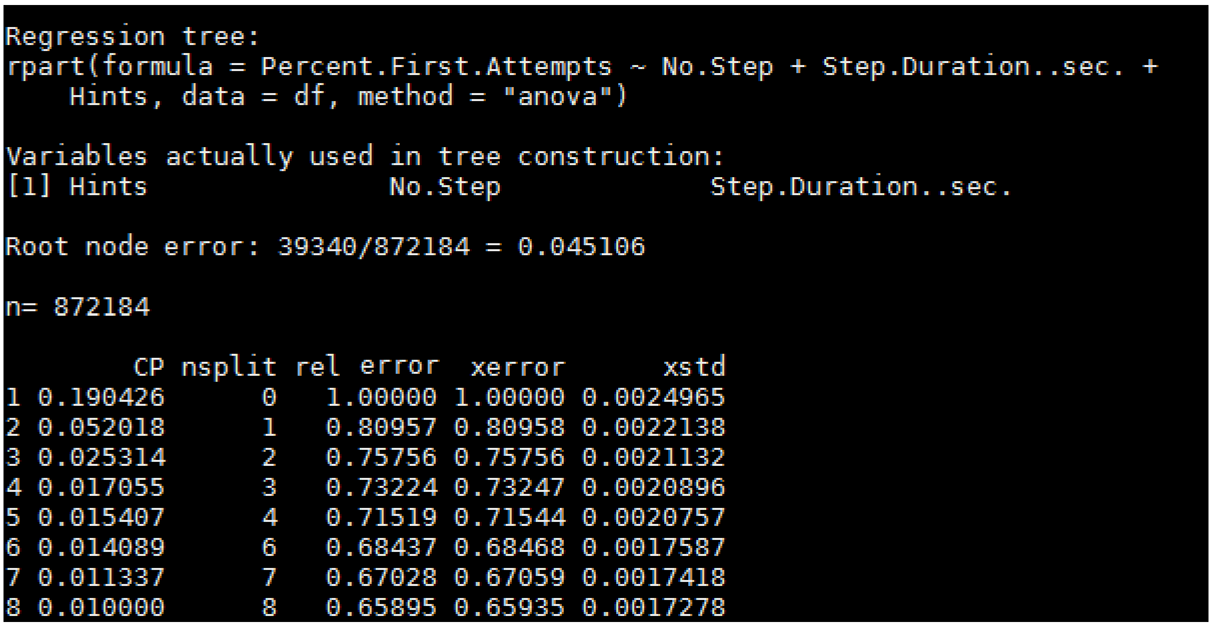
After we get the count of skills, we group the entire data based on the Problem Hierarchy and the Problem Name. For this, we remove all the unnecessary variables in the table and sum the rest of the data so that each entry is interaction of student when solving a particular problem. After the pre-processing, there are 872184 unique problems.

R Code

```r
library(stringr)
library(dplyr)

train_data <- read.table(file = "bridge_to_algebra_2008_2009_train.txt", header = TRUE, sep="\t")

train_data$Count.KC.SubSkills  <- str_count(train_data$KC.SubSkills., '~~') + 1
train_data$KC.KTracedSkills  <- str_count(train_data$KC.KTracedSkills., '~~') + 1

#Removing unnecessary variables
train_data <- train_data[,-c(1,6,7,8,9,10,12,13,18,19,20,21)]
train_data <- na.omit(train_data)

#New variable to calculate the number of steps in a problem
totalproblems <- data.frame(No.Step = rep(1, nrow(train_data)), train_data[,])

totalproblems <- totalproblems %>%
  group_by(Anon.Student.Id, Problem.Name, Problem.Hierarchy ) %>%
  summarise_all(funs(sum))
```

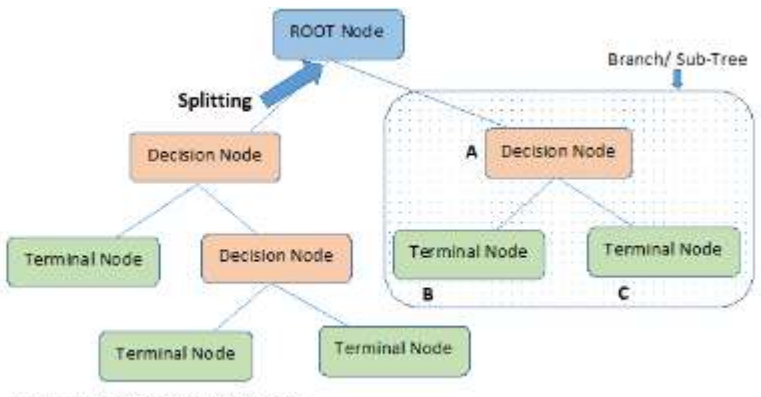
We then create new variable for checking the complexity by taking an aggregate of the number of steps in a problem and count of skills required to solve the problem. After doing the necessary pre-processing and feature engineering, we save the new data set as 'totalproblems.csv'.

Next we build a **decision tree algorithm** to predict the performance of the student which is taken as the Percentage of first attempt while solving a problem. The predictor variables were Number of steps, number of skills required to solve a problem and the total time taken to solve that problem.



```
Regression tree:
rpart(formula = Percent.First.Attempts ~ No.Step + Step.Duration..sec. +
    Hints, data = df, method = "anova")

Variables actually used in tree construction:
[1] Hints               No.Step             Step.Duration..sec.

Root node error: 39340/872184 = 0.045106

n= 872184

        CP nsplit rel error  xerror      xstd
1 0.190426      0   1.00000 1.00000 0.0024965
2 0.052018      1   0.80957 0.80958 0.0022138
3 0.025314      2   0.75756 0.75756 0.0021132
4 0.017055      3   0.73224 0.73247 0.0020896
5 0.015407      4   0.71519 0.71544 0.0020757
6 0.014089      6   0.68437 0.68468 0.0017587
7 0.011337      7   0.67028 0.67059 0.0017418
8 0.010000      8   0.65895 0.65935 0.0017278
```

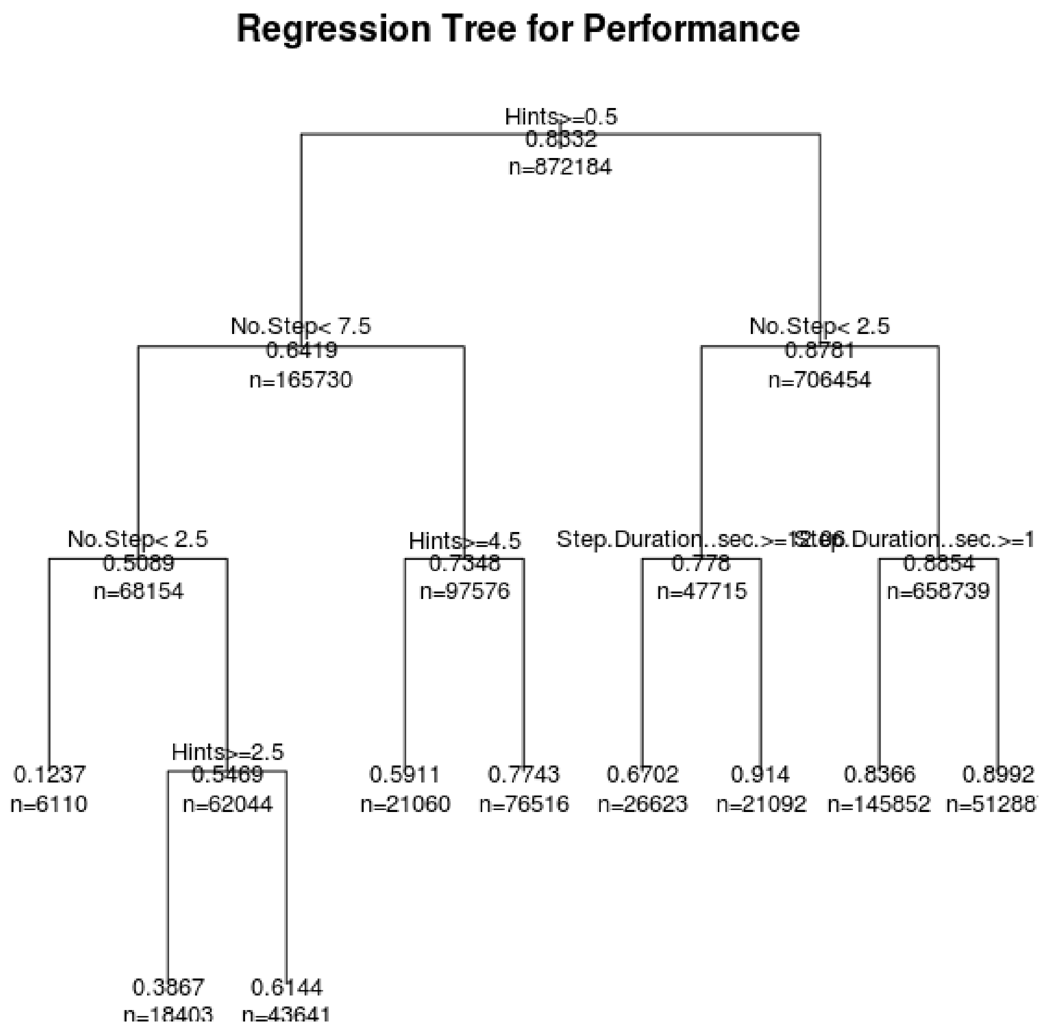
Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems.



Note:- A is parent node of B and C.

The decision of making strategic splits heavily affects a tree's accuracy. It helps us explore the structure of a set of data, while developing easy to visualize decision rules for predicting a continuous (regression tree) outcome.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.



Saving the dataset in MongoDB

MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas.

The matrix type data was converted into a nested JSON structure of the following schema

```
{
  "stu_000670b50e": {
    "Anon.Student.Id": "stu_000670b50e",
    "Problem.Name": ["GCF-04-12", "gcf-04-20", "GCF-06-08"],
    "Problem.Hierarchy": ["Unit GCF, Section gcf-1", "Unit GCF, Section gcf-1", "Unit GCF, Section gcf-1"],
    "No.Step": [25, 1, 21],
    "Problem.View": [25, 1, 30],
    "Step.Duration..sec.": [67.043, 26.354, 133.547],
    "Correct.First.Attempt": [25, 1, 20],
    "Incorrects": [0, 0, 1],
    "Hints": [0, 0, 0],
    "Corrects": [25, 2, 21],
    "count.subskills": [50, 2, 42],
    "count.ktracedskills": [50, 2, 42],
    "Complexity": [0.7071, 0.0283, 0.5939],
    "No.Complex.Step": [18, 0, 12],
    "Percent.First.Attempts": [1, 1, 0.952]
  }
}
```

The above is an example of sample of the data belonging to student with _id 000670b50e. All the data belonging to one student can be read by the unique index (i.e. the anonymous student id).

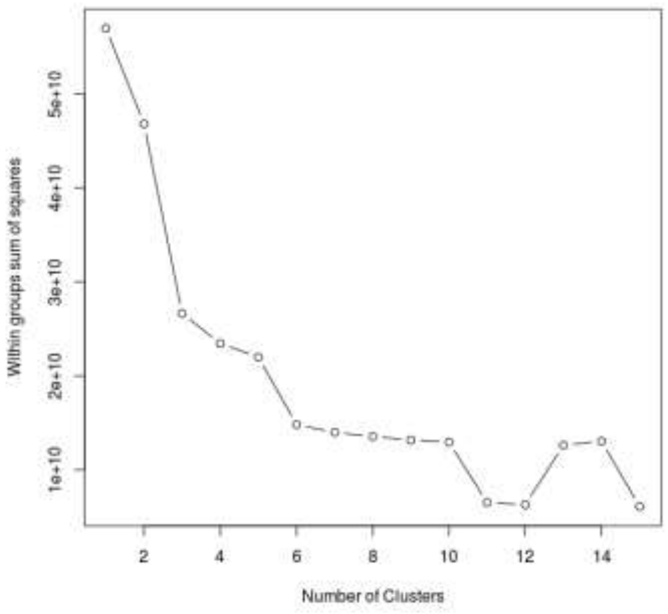
All the rows of the dataset are converted to JSON format and pushed to MongoDB.

R Code for converting the dataset to above JSON schema and pushing it to MongoDB

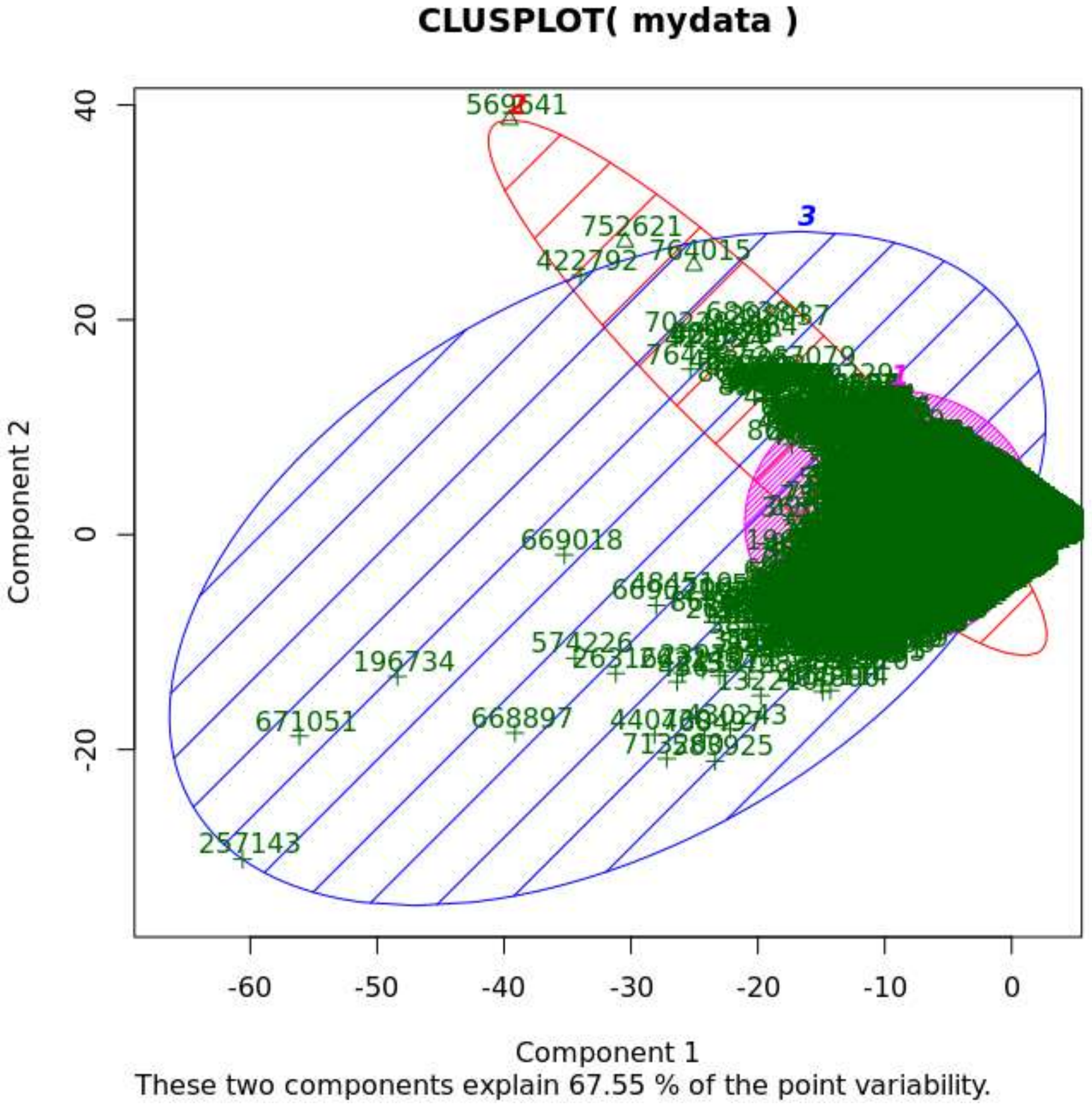
```
mongo <- mongo(collection = "test", db = "ashwin_db", url = "mongodb://localhost",verbose = TRUE)

convert_groupings <- function(key_df){
  key_df <- as.list(key_df)
  key_df$Anon.Student.Id <- unique(key_df$Anon.Student.Id)
  key_df
}

sample <- read.csv("totalproblems.csv", header = TRUE)
sample$X <- NULL

sample_list <- split(sample, sample$Anon.Student.Id)
sample_list <- lapply(sample_list, convert_groupings)


for(i in sample_list){
  sample_json <- jsonlite::toJSON(i, auto_unbox = T, pretty = T)
  mongo$insert(fromJSON(sample_json))
}
```

Clustering Algorithm to predict the toughness of a problem

We use a K-Means algorithm to cluster the data based on Number of steps, Total attempts to solve, duration to solve, skills and opportunities gained during a problem.



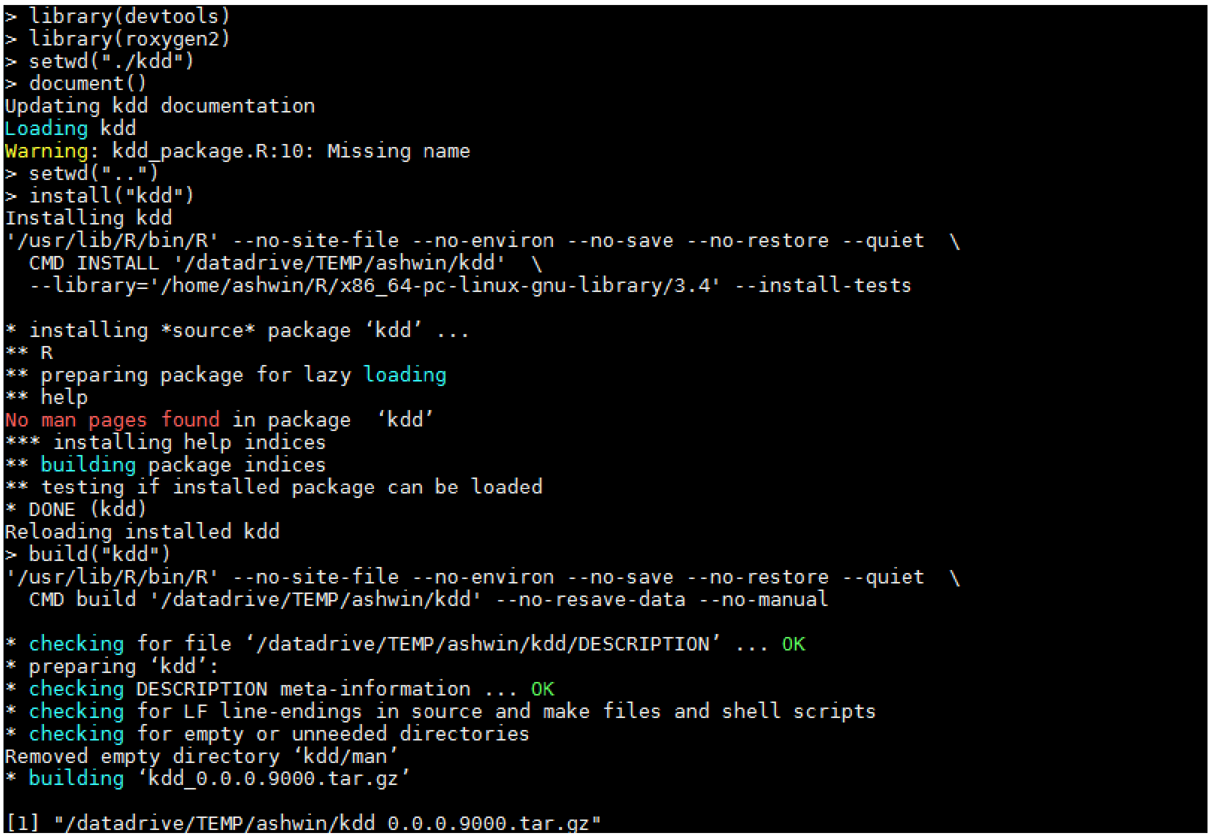
We cluster the data into 3 clusters namely **Easy, Medium, and Hard.**



The above models are incorporated as functions inside an R package ‘performance’.

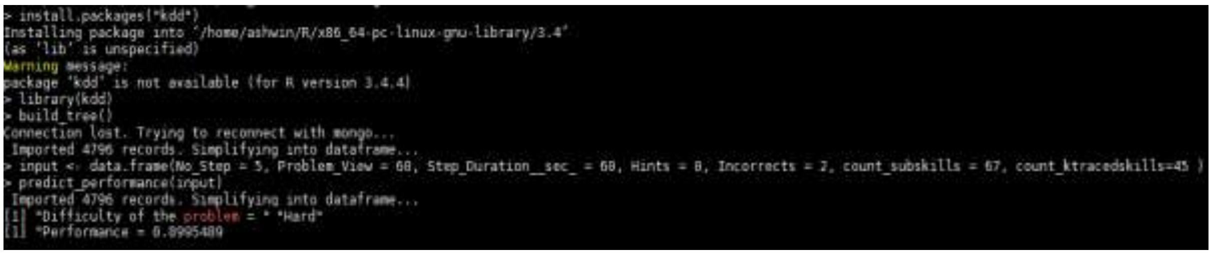
The various functions inside the package include **build_tree()** which is used to build a decision tree, **predict_performance()** to predict the performance of the student based on the above method, **predict_difficulty()** to predict the difficulty of the problem. It also has a **health_check()** function which is useful for asserting that the dependencies are up and running and your application can respond to HTTP requests.

Building the package



```
> library(devtools)
> library(roxygen2)
> setwd("./kdd")
> document()
Updating kdd documentation
Loading kdd
Warning: kdd_package.R:10: Missing name
> setwd("..")
> install("kdd")
Installing kdd
'/usr/lib/R/bin/R' --no-site-file --no-environ --no-save --no-restore --quiet  \
  CMD INSTALL '/datadrive/TEMP/ashwin/kdd'  \
  --library='/home/ashwin/R/x86_64-pc-linux-gnu-library/3.4' --install-tests

* installing *source* package ‘kdd’ ...
** R
** preparing package for lazy loading
** help
No man pages found in package  ‘kdd’
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (kdd)
Reloading installed kdd
> build("kdd")
'/usr/lib/R/bin/R' --no-site-file --no-environ --no-save --no-restore --quiet  \
  CMD build '/datadrive/TEMP/ashwin/kdd' --no-resave-data --no-manual

* checking for file ‘/datadrive/TEMP/ashwin/kdd/DESCRIPTION’ ... OK
* preparing ‘kdd’:
* checking DESCRIPTION meta-information ... OK
* checking for LF line-endings in source and make files and shell scripts
* checking for empty or unneeded directories
Removed empty directory ‘kdd/man’
* building ‘kdd_0.0.0.9000.tar.gz’

[1] "/datadrive/TEMP/ashwin/kdd_0.0.0.9000.tar.gz"
```

Working of the Package



```
> install.packages("kdd")
Installing package into '/home/ashwin/R/x86_64-pc-linux-gnu-library/3.4'
(as 'lib' is unspecified)
Warning message:
package 'kdd' is not available (for R version 3.4.4)
> library(kdd)
> build_tree()
Connection lost. Trying to reconnect with mongo...
 Imported 4796 records. Simplifying into dataframe...
> input <- data.frame(No_Step = 5, Problem_View = 68, Step_Duration__sec_ = 68, Hints = 8, Incorrects = 2, count_subskills = 67, count_ktracedskills=45 )
> predict_performance(input)
 Imported 4796 records. Simplifying into dataframe...
[1] "Difficulty of the problem = " "Hard"
[1] "Performance = 0.8995489
```

This package can be deployed via Docker using OpenCPU. **Docker** is a computer program that performs operating-system-level virtualization also known as containerization. **OpenCPU** is a system for embedded scientific computing and reproducible research. The OpenCPU server provides a reliable and interoperable HTTP API for data analysis based on R.