The present work was submitted to Lehr- und Forschungsgebiet Educational Technologies at DIPF

Learning the impact of Learning Analytics with an authentic dataset

Bachelor-Thesis

Presented by

Leute, Egon

6553108

First examiner:	Prof. Dr. Hendrik Drachsler
Second examiner:	Prof. Dr. Detlef Krömker
Supervisors:	Prof. Dr. Hendrik Drachsler

Frankfurt, August 12, 2019

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Frankfurt, August 12, 2019 Full Name

Acknowledgments

My deepest gratitude goes to my fiance for always supporting and motivating me in the past. Without her this thesis would have never been finished.

I would also like to thank Prof. Dr. Hendrik Drachsler and Prof. Dr. Detlef Krömker for supporting this thesis, without their help this journey would have been extremely hard.

And finally, a big shout-out to all of my friends and my colleagues at the Goethe University and Eva Kaufholz-Soldat from the Schreibzentrum am Riedberg.

Abstract

Nowadays, data sets of the interactions of users and their corresponding demographic data are becoming more and more valuable for companies and academic institutions like universities when optimizing their key performance indicators. Whether it is to develop a model to predict the optimal learning path for a student or to sell customers additional products, data sets to train these models are in high demand. Despite the importance and need of big data sets it still has not become apparent to every decision maker how crucial data sets like these are for the future success of their operations.

The objective of this thesis is to demonstrate the use of a data set, gathered from the virtual learning environment of a distance learning university, by answering a selection of questions in Learning Analytics. Therefore, a real world data set was analyzed and the selected questions were answered by using state-of-the-art machine learning algorithms.

Contents

1 Introduction

With an abundance of data, that can be gathered from Learning Environments, docents are facing challenges in benefiting from these kinds of data to improve their courses and to help their students in understanding complex issues with an improved pace. With better understanding of the available data, arising problems for students can be determined more in advance and acted to accordingly.

Previous studies [1] have shown that there are strong links in the interaction of students with Virtual Learning Environments (VLE) and academic achievements. But the question that remains is, how we can use these findings in actually improving education and to allocate valuable resources to those who would benefit most from it. To find these students it is essential to predict in an early stage of the semester which student should be targeted with additional measures. In the last research question, for the first time in learning analytics, a model will be calculated and evaluated that aims to serve this purpose.

With the rise of more capable processors and architectures that allow for processing large amounts of data in a reasonable time, scientists around the globe try to make use of that vast amount of information to answer the most urging questions in Learning Analytics [2]. Thanks to publicly available data sets of the interactions of students with VLEs we have enough data to develop and prove strategies that encounter these issues.

The goal of this bachelor thesis is to design use cases that show how to benefit from data that was gathered in a Virtual Learning Environment. For this task the “Open University Learning Analytics datasets” or short OULAD was chosen, because it provides information about the interactions of the students with the VLE, demographic data and data about the students’ assessments.

1.1 Research Questions

This thesis searches for answers to the following questions:

1. Is there a link between the time of enrollment in a course and the final exam results?
2. Is there a relationship between higher exercise points and higher final grades?
3. Are there parallels between the students interactions with course materials and their understandings of the subject matter?
4. How far in advance can we predict which students are likely to fail their final exam?

These questions will be answered in a format that can be used as seminar material to educate university staff about the importance of gathering data in a VLE.

1.2 Outline

This Thesis is structured as follows: Chapter 2 will provide an overview of the data set used in this thesis and shows the possibilities and limitations of the data set, Chapter 3 provides an overview of papers that either worked with the same data set or had similar goals like this thesis. Following, Chapter 4 presents describes the machine learning algorithms that were used to work the data set. Chapter 5 contains the Jupyter Notebook which aims to lecture university staff in the importance of learning analytics data sets. Finally, Chapter 6 provides a conclusion on the findings of this work on the OULAD and what can be done in the future.

2 The Open University Learning Analytics Dataset

Since the integration of virtual learning environments into the higher education sector, universities are able to collect huge amounts of data, showing how students interact with a platform and the designed course material. The data set can be downloaded from the website https://analyse.kmi.open.ac.uk/open_dataset. Making use of a data set like this is the aim of this thesis. But first a deeper understanding of the data set is needed to know how to connect the different tables and to see the limitations of a data set like the OULAD.

2.1 Data set schema

Figure 2.1 visualizes the connections between the different tables and on which keys they are connected with each other. Information that is collected in the VLE during the semester like click data is colored in orange, information about the modules, tests and content in green and last demographic information about the students in blue.

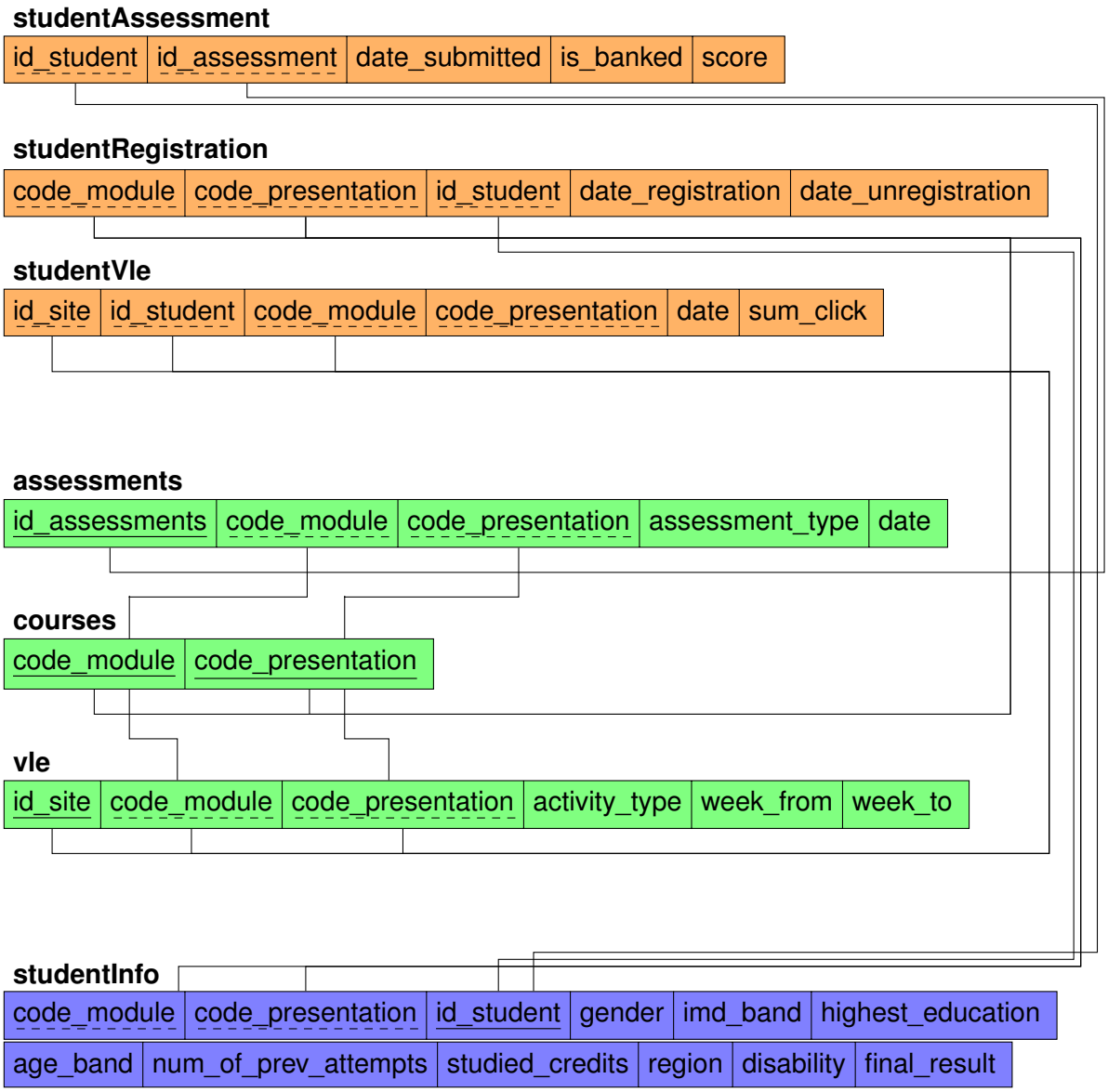

Figure 2.1: Data base schema

2.2 Table content

For privacy reasons private data of students like age was transformed to labeled data to make it impossible to trace back the date of birth for a specific student by using other public available information [3]
The tables shown in Figure 2.1 contain the following information:

- Table studentInfo:
  Contains demographic information about the students. Consists of around 32.000 rows
- Table courses:
  lists all module_code and module_presentation combinations present in the data set

- Table studentRegistration:
  When was the time of enrollment of a student into a course in relation to semester start
- Table assessments:
  This table contains information about when and which type of assessment was offered. There are three types of assessments available TMA, CMA and Exam.
- Table studentAssessment:
  Every row in this table represents a student's assessment, when it was submitted and what score was achieved by the student
- Table studentVle:
  Aggregated to a daily base and website this table holds the information on the interactions of each student with the virtual learning environment
- Table vle:
  Contains information about the web sites available in the vle and in which time range they were present in the vle

2.3 Limitations

From the description above the limitations of the OULAD become already more apparent. It will not be possible to conduct research like in the paper mentioned in section 2.1 by Bart Rienties et. al because the interactions are not on the same fine-grained scale like in the paper on an hour base rather on a daily base. Another limitation results of the anonymization and the relative short time span of the data set. It will not be possible to conduct research on how students developed over a span of multiple semesters.

2.4 Design for Goethe University Learning Analytics dataset

A data schema like in the OULAD, where clicks are aggregated per day, reduces storage space and makes a data base more performant. "However, the actual behavior of students occurs on much finer, hours per hour or even second by second level" (Rienties, 2018). Therefore, to not be limited in analytics, an optimal data storage method should hold the raw data gathered by the VLE to ensure that every activity can be replicated how it happened at the time it took place. It should also be possible to produce a time series for each student from the time they entered university to the time they graduated, to evaluate how they managed different obstacles and compare this to other students. By doing so research on the timing of interactions with the course material or the time of day the interaction took place can be conducted.

3 State of research

This chapter aims to inform about different studies that have been conducted on the OULAD, what their goals were and how they are connected to this thesis.

3.1 Linking Students' Timing of Engagement to Learning Design and Academic Performance [1]

With their research on students' timing of engagement and its relation to Learning Design, Bart Rienties, Michal Huptych and Quan Nguyen examine how the timing of engagement can predict if a student is likely to fail their final exam. They analysed how close students stick to the learning path that was designed by the instructor of the module and how close the students follow this path. They discovered that outperforming students tend to study more in advance while students that will fail their final exam spend more time on catching up with the course material. The data set used to conduct this study involves the behavior of 380 students in the two semesters in 2015 and 2016. In chapter 3 other factors that are correlated to a student's success and how to detect outperforming students by clustering their clicks in the VLE are demonstrated.

3.2 Student Success Prediction and the Trade-Off between Big Data and Data Minimization [4]

Hendrik Heuer and Andreas Breiter from the University of Bremen try to minimize the information needed to still make valuable predictions on fails and passes of students' on their final exams. The goal is to allocate data that doesn't need to be on a fine-grained scale, like the clicks on the VLE which they binarize, to still make accurate predictions and to make it easier to protect private and sensitive information of the students. According to the authors they can make very accurate predictions even without sensitive information. Following their work and results in this thesis, it will be tested at which point during a semester demographic information a therefore less sensitive information becomes less relevant in predicting the outcome of a semester.

3.3 OULAD MOOC Dropout and Result Prediction using Ensemble, Deep Learning and Regression Techniques [5]

In contrast to the previous paper, Nikhil Indrashekhar Jha, Ioana Ghergulescu, and Arghir-Nicolae Moldovan use the whole OULAD to improve predictions of dropout and result prediction performance. For Pass/Fail predictions, including all attributes into the training data like demographics, assessments and VLE interactions, they can achieve very high AUC scores of up to 0.93.

3.4 Relation to this thesis

Like in the paper “Linking Students’ Timing of Engagement to Learning Design and Academic Performance” in chapter 5.2 the time of enrollment of students will be examined and how it is related to the students’ performance at the final exam at the end of the semester.

In chapter 5.5. we will see like in the paper “Student Success Prediction and the Trade-Off between Big Data and Data Minimization” that only a limited amount of features is needed to make successful predictions about which students will pass their exams and who will fail. In addition to the findings on Data Minimization, this thesis will contribute on how data minimization can look like when predicting at an earlier stage of the semester opposed to at the end like in the paper of Hendrik Heuer and Andreas Breiter.

The paper “OULAD MOOC Dropout and Result Prediction using Ensemble, Deep Learning and Regression Techniques” shows benchmarks on what has already been achieved when forecasting students’ fail/pass rate on this data set. This thesis will expand the knowledge on how forecasting accuracies change when the predictions are not done at the end of the semester.

4 Machine Learning Algorithms

In this chapter I will describe the three machine learning algorithms that were used later in the Jupyter Notebook to examine the data sets. All machine learning algorithms were imported from the library scikit-learn [6].

4.1 K nearest neighbors - KNN

K nearest neighbors needs a numeric labeled data set as target variable, where all labels have to be integers and a numerical data set as input variables, where the input variables can be either integers or floats. Around 70 % of the data set should be used as training data and the remaining 30 % are used to evaluate the trained model. The KNN algorithm measures the distance of each data point in an m-dimensional space (where m is the number of variables in the data set) to a target variable. The most common label is then used to predict the most likely label.

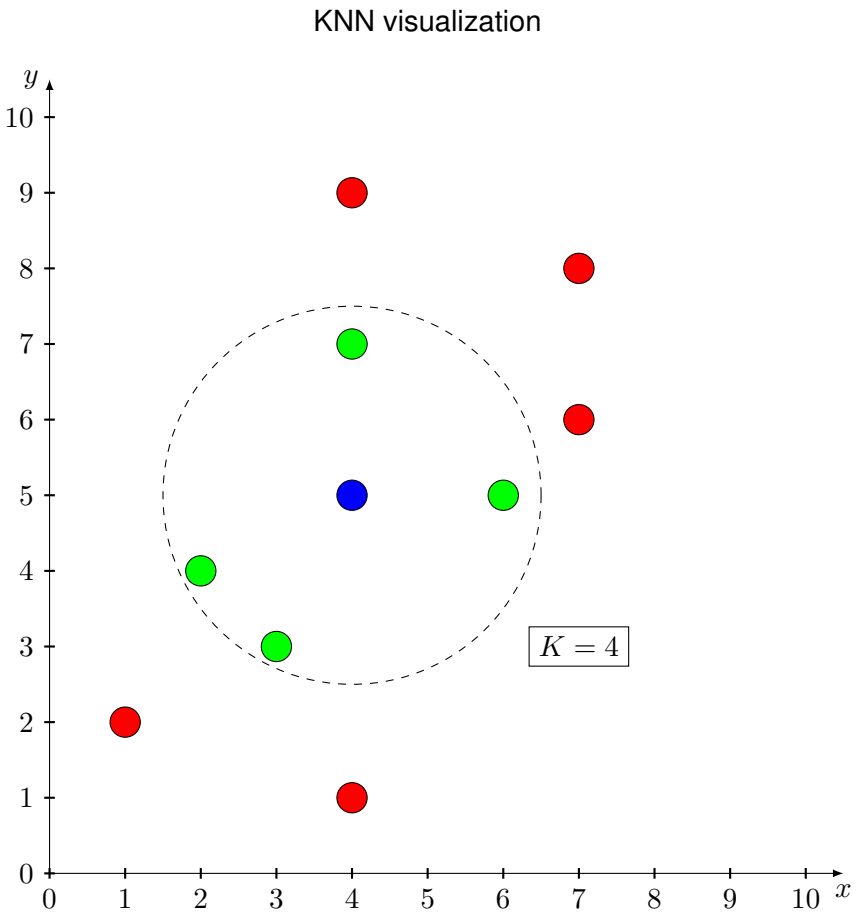

Figure 4.1: K nearest neighbors with K = 4 and 2 variables x and y

4.2 Decision Tree

The Decision Tree Algorithm needs as well as the KNN algorithm, labeled target variables and numeric input data. This algorithm forms a decision tree where in each node binary questions are asked on the variables that will divide the data set the most. With the algorithm provided by scikit-learn the generated decision tree can even be returned. Another useful factor when applying this algorithm is that the feature importance can be returned. Feature importance shows which variables were more decisive for dividing the data set.

4.3 K-means

In contrast to the KNN and decision tree algorithm the K-means algorithm does not provide a prediction for a data point, it tries to form K different clusters (where K is a natural number greater than 1). The error rate for each cluster is calculated by measuring the mean distance of each data point to it's center squared. To determine the optimal K the combined error rate of each K is plotted against K and take the K that is closest to the inflection point.

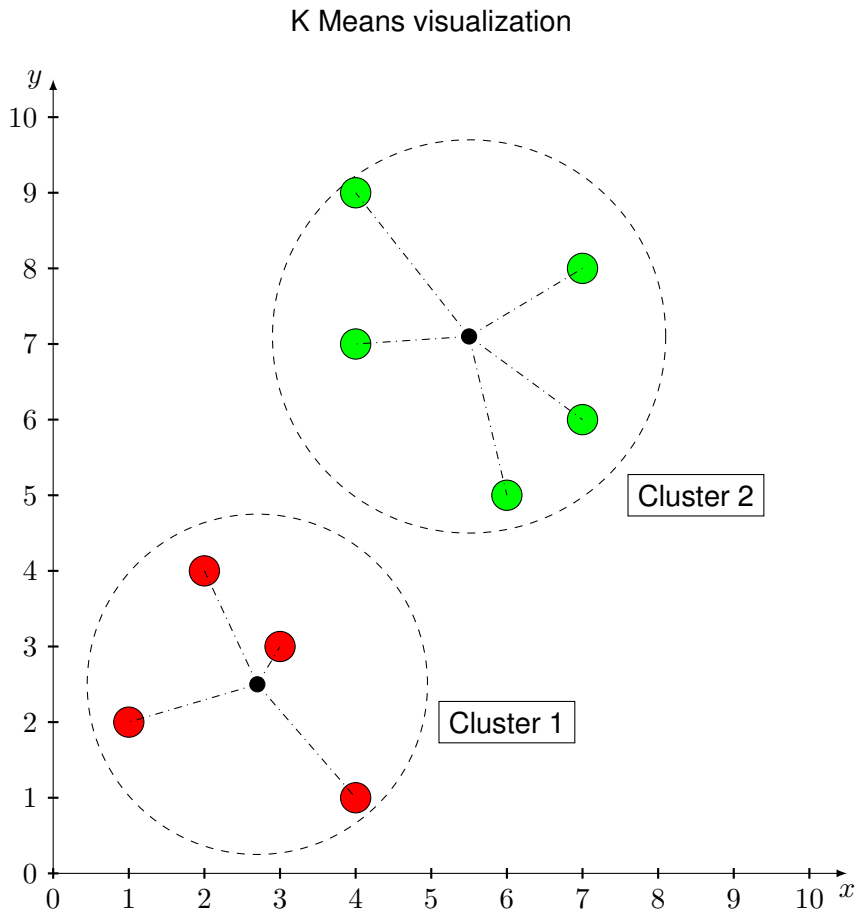

Figure 4.2: K-Means with K = 2 and two variables x and y

5 Jupyter Notebook

In this section the research questions, presented in chapter 1.1, will be answered using the programming language Python and the integrated development environment Jupyter Notebook. The following libraries are nedded to execute the code:

- Pandas [7]
- numpy [8]
- seaborn [9]
- Math
- sklearn [6]
- matplotlib [10]

5.1 Import Data

Download Data from https://analyse.kmi.open.ac.uk/open_dataset and save it to the same folder as this Jupyter Notebook. After this task was completed successfully we will import the Library *Pandas*, which helps to handle huge data sets more easily and performs vector operations with high speed.

```
In [1]: import pandas as pd

        studentRegistration = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//studentRegistration.csv', sep=',')
        assessments = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//assessments.csv', sep=',')
        courses = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//courses.csv', sep=',')
        studentAssessment = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//studentAssessment.csv', sep=',')
        studentRegistration = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//studentRegistration.csv', sep=',')
        studentVle = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//studentVle.csv', sep=',')
        vle = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//vle.csv', sep=',')
        studentInfo = \
        pd.read_csv(filepath_or_buffer='..//..//OULAD//studentInfo.csv', sep=',')
```

After importing all tables successfully into our Notebook, we can look into a data frame by typing DataFrame_name.head(n), where *n* is the number of lines shown.

```
In [2]: studentInfo.head(5)
```

Out[2]:

	code_module	code_presentation	id_student	gender	region	\
0	AAA	2013J	11391	M	East Anglian Region	
1	AAA	2013J	28400	F	Scotland	
2	AAA	2013J	30268	F	North Western Region	
3	AAA	2013J	31604	F	South East Region	
4	AAA	2013J	32885	F	West Midlands Region	

	highest_education	imd_band	age_band	num_of_prev_attempts	\
0	HE Qualification	90-100%	55<=	0	
1	HE Qualification	20-30%	35-55	0	
2	A Level or Equivalent	30-40%	35-55	0	
3	A Level or Equivalent	50-60%	35-55	0	
4	Lower Than A Level	50-60%	0-35	0	

	studied_credits	disability	final_result
0	240	N	Pass
1	60	N	Pass

```
2               60          Y    Withdrawn
3               60          N         Pass
4               60          N         Pass
```

```
In [3]: assessments.head(5)
```

Out[3]:

	code_module	code_presentation	id_assessment	assessment_type	date	weight
0	AAA	2013J	1752	TMA	19.0	10.0
1	AAA	2013J	1753	TMA	54.0	20.0
2	AAA	2013J	1754	TMA	117.0	20.0
3	AAA	2013J	1755	TMA	166.0	20.0
4	AAA	2013J	1756	TMA	215.0	30.0

```
In [4]: studentRegistration.head(5)
```

Out[4]:

	code_module	code_presentation	id_student	date_registration	\
0	AAA	2013J	11391	-159.0	
1	AAA	2013J	28400	-53.0	
2	AAA	2013J	30268	-92.0	
3	AAA	2013J	31604	-52.0	
4	AAA	2013J	32885	-176.0	

	date_unregistration
0	NaN
1	NaN
2	12.0
3	NaN
4	NaN

```
In [5]: courses.head(5)
```

Out[5]:

	code_module	code_presentation	module_presentation_length
0	AAA	2013J	268
1	AAA	2014J	269
2	BBB	2013J	268
3	BBB	2014J	262
4	BBB	2013B	240

```
In [6]: studentAssessment.head(5)
```

Out[6]:

	id_assessment	id_student	date_submitted	is_banked	score
0	1752	11391	18	0	78.0
1	1752	28400	22	0	70.0
2	1752	31604	17	0	72.0
3	1752	32885	26	0	69.0
4	1752	38053	19	0	79.0

```
In [7]: studentVle.head(5)
```

```
Out[7]:   code_module code_presentation  id_student  id_site  date  sum_click
        0         AAA             2013J       28400    546652   -10          4
        1         AAA             2013J       28400    546652   -10          1
        2         AAA             2013J       28400    546652   -10          1
        3         AAA             2013J       28400    546614   -10         11
        4         AAA             2013J       28400    546714   -10          1

In [8]: vle.head(5)

Out[8]:    id_site code_module code_presentation activity_type  week_from  week_to
        0   546943         AAA             2013J      resource        NaN      NaN
        1   546712         AAA             2013J     oucontent        NaN      NaN
        2   546998         AAA             2013J      resource        NaN      NaN
        3   546888         AAA             2013J           url        NaN      NaN
        4   547035         AAA             2013J      resource        NaN      NaN
```

After we have explored our dataset a bit, we can start to make the first assumptions and then try to prove them with the dataset.

5.2 Is there a link between the time of enrollment in a course and the final exam results?

To verify if there is a correlation between *time_of_enrollment* and *exam_score*, we will first need to prepare a data frame, where every row represents a student, module combination with the two variables *time_of_enrollment* in days and *final_exam_score*. Correlations can be very helpful in finding links between different variables, as they're easy to calculate, but should be treated with caution as they can be misleading.

First we will merge the two data frames *assessments* and *studentAssessment* on the key *id_assessment* to connect the assessments of the students to the modules and different presentations, by doing so we can split them into different groups and analyze them separately.

```
In [9]: exam_assessments = pd.merge(assessments, studentAssessment, on='id_assessment')
        exam_assessments = exam_assessments[exam_assessments['assessment_type'] == 'Exam']
        exam_assessments.head()
```

```
Out[9]:       code_module code_presentation  id_assessment assessment_type  date  \
        52923         CCC             2014B          24290            Exam   NaN
        52924         CCC             2014B          24290            Exam   NaN
        52925         CCC             2014B          24290            Exam   NaN
        52926         CCC             2014B          24290            Exam   NaN
        52927         CCC             2014B          24290            Exam   NaN

               weight  id_student  date_submitted  is_banked  score
        52923   100.0      558914             230          0   32.0
        52924   100.0      559706             234          0   78.0
        52925   100.0      559770             230          0   54.0
        52926   100.0      560114             230          0   64.0
        52927   100.0      560311             234          0  100.0
```

Now we merge our new data frame with the table *studentRegistration* to get the information when a student enrolled into the course

```
In [10]: exam_assessments_with_registration_date = \
                                  pd.merge(exam_assessments, studentRegistration, \
                                  on=['code_module', 'code_presentation', 'id_student'], \
                                  how='inner')
         exam_assessments_with_registration_date.head()
```

```
Out[10]:   code_module code_presentation  id_assessment assessment_type  date  weight  \
         0         CCC             2014B          24290            Exam   NaN   100.0
         1         CCC             2014B          24290            Exam   NaN   100.0
         2         CCC             2014B          24290            Exam   NaN   100.0
         3         CCC             2014B          24290            Exam   NaN   100.0
         4         CCC             2014B          24290            Exam   NaN   100.0

            id_student  date_submitted  is_banked  score  date_registration  \
         0      558914             230          0   32.0              -74.0
```

```
1       559706              234          0   78.0              -22.0
2       559770              230          0   54.0              -22.0
3       560114              230          0   64.0             -281.0
4       560311              234          0  100.0              -28.0

   date_unregistration
0                  NaN
1                  NaN
2                  NaN
3                  NaN
4                  NaN
```

After aggregating the cells we can investigate if there exists a correlation between *time_of_enrolement* and the final exam score. With *df.corr()* we can calculate the Pearson product-moment correlation coefficient (PPMCC) which returns a value between -1 and +1. -1 Meaning total negative correlation, +1 total positive correlation and 0 no linear correlation. The calculation of the PPMCC is shown below.

```
In [11]: from IPython.display import Math
         Math(r'\text{PPMCC} \\ \rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \cdot \sigma_Y}\
             \\ cov \text{ is the covariance} \\ \
             \sigma_Y \text{ is the standard deviation of } Y \
             \\ \sigma_X \text{ is the standard deviation of } X')
```

Out[11]:

PPMCC

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \cdot \sigma_Y}$$

cov is the covariance
σ_Y is the standard deviation of Y
σ_X is the standard deviation of X

```
In [12]: exam_assessments_with_registration_date[['score', 'date_registration']].corr()
```

Out[12]:

```
                      score  date_registration
score              1.000000           0.024529
date_registration  0.024529           1.000000
```

From the data frame above we can read that there is close to no correlation, meaning that it is very likely that a prediction for exam results based on the time of enrollment will not be successful. With seaborn, a library for visualizations, we can print a scattered plot with a regression line that fits the data points best.

```
In [13]: import seaborn as sns
         sns.lmplot(x='date_registration',y='score',\
                    data=exam_assessments_with_registration_date,\
                    scatter_kws={'alpha':0.07},\
                    fit_reg=True, line_kws={'color': 'red'}).set\
                     (xlabel='time of enrolement relative to course start', \
                      ylabel='final exam score');
```

In the plot above we can see that the regression line is almost vertical because of the low PPMCC. Also there does not seem to be a pattern which shows that there might be a correlation between *time_of_enrollment* and *final_exam_score*

Conclusion

As expected from the low PPMCC we do not get a clear picture for our prediction, because there seems to be no connection between the time of enrollment and exam results, since a coefficient of 0.024529 is considered random. However this is still helpful because we know now that we do not have consider time of enrollment any more when it comes to predictions about the final exam. In the next examinations we will try to combine other variables with the final exam points to find a pair with a higher coefficient.

5.3 Is there a relationship between higher exercise points and higher final grades?

Next we want to examine if the exercise points gathered by the students during the semester are a better method to predict the likelihood of a student succeeding the course. In the first step we will create another data frame with the same key combination (*id_student*, *code_module*), but now with the new variables *sore_exam* and *score_non_exam*. Where *score_non_exam* are assessments that get evaluated by a tutor or a computer and take place regularly during the semester.

```
In [14]: non_exam_assessments = pd.merge(assessments, studentAssessment, on='id_assessment')
         non_exam_assessments = \
         non_exam_assessments[non_exam_assessments.assessment_type != 'Exam']
         grouped_assessments = non_exam_assessments[['id_student', 'code_module', \
             'code_presentation', 'score']].groupby(['id_student', 'code_module', \
             'code_presentation']).sum().reset_index()

         exam_assessments = pd.merge(assessments, studentAssessment, on='id_assessment')
         exam_assessments = exam_assessments[exam_assessments.assessment_type == 'Exam']
         grouped_exam_assessments = exam_assessments[['id_student', 'code_module', \
             'code_presentation', 'score']].groupby(['id_student', 'code_module', \
             'code_presentation']).sum().reset_index()

         assessment_eval = \
         pd.merge(grouped_exam_assessments[['id_student', 'code_module', \
             'code_presentation', 'score']], \
                 grouped_assessments[['id_student', 'code_module',\
             'code_presentation', 'score']], \
                 on=['id_student', 'code_module', 'code_presentation'], \
             suffixes=('_exam', '_non_exam'))

         assessment_eval.head()
```

Out[14]:

	id_student	code_module	code_presentation	score_exam	score_non_exam
0	23698	CCC	2014J	80.0	590.0
1	24213	DDD	2014B	58.0	476.0
2	27116	CCC	2014J	96.0	744.0
3	28046	DDD	2013J	40.0	306.0
4	28787	CCC	2014J	44.0	224.0

```
In [15]: assessment_eval[['score_exam', 'score_non_exam']].corr()
```

Out[15]:

	score_exam	score_non_exam
score_exam	1.000000	0.270082
score_non_exam	0.270082	1.000000

In the cell above we can see that the PPMCC is significantly higher, which can be interpreted in a way that these two variables have a much higher significance than the others in our test before.

But 0.270082 is still not a remarkable strong correlation coefficient, but can be interpreted as a hint that we are on the right path. Now we want to plot the scatter plot to visualize the dependencies of the variables.

```
In [16]: sns.lmplot(x='score_exam',y='score_non_exam',data=assessment_eval,\
             fit_reg=True, scatter_kws={'alpha':0.07}, \
             line_kws={'color': 'red'}).set(xlabel='score final exam',
             ylabel='score exercises');
```

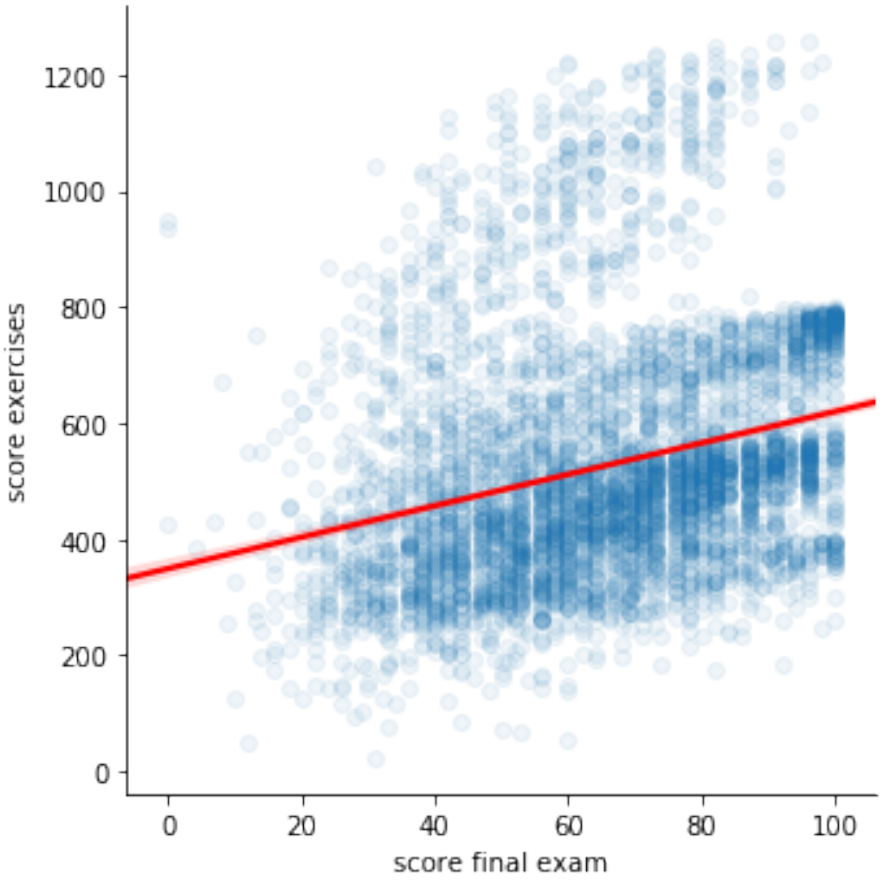

Figure 5.1: Regression between score exercise and score final exam

After plotting, we can confirm that there is also a visual correlation between exercise points and exam points, as we can see the points seem to get higher on the x axis the higher the y value is. Now we can look into the different modules and explore if there is any difference. With 'assessment_eval.code_module.unique()' we get a list of the distinct code_modules over which we will iterate to get results from each module. First, we will take a look at the correlation matrix and then print out the scatter plot.

```
In [17]: corr_module = pd.DataFrame()
         module_list = list()
```

```
for module in assessment_eval['code_module'].unique():
    module_list += [module, module]
    corr_module = pd.concat([assessment_eval[['score_exam', 'score_non_exam']]\
                    [assessment_eval.code_module == module].corr(), corr_module])
corr_module['module'] = module_list[::-1]
corr_module = corr_module.set_index(['module', corr_module.index])
corr_module
```

Out[17]:

module		score_exam	score_non_exam
DDD	score_exam	1.000000	0.134523
	score_non_exam	0.134523	1.000000
CCC	score_exam	1.000000	0.513080
	score_non_exam	0.513080	1.000000

In the output above, we can see that the PPMCC diverges notably in the two modules. This is a very important finding since it can have two meanings. Either in module CCC exercise are more connected to the final exam and therefore are a better indicator if a student passes or not than in module DDD, or we have to deal with an effect were correlations can disappear when a dataset is aggregated. After seeing this differences, we should definitely examine module DDD more closely. To accomplish this, we will print out a scatter plot of Module CCC and DDD to see if there are any visual differences.

```
In [18]: for code_module in assessment_eval.code_module.unique():
             scatter_plot = sns.lmplot(x='score_exam',y='score_non_exam', \
                 scatter_kws={'alpha':0.1}, \
                 data=assessment_eval[assessment_eval.code_module==code_module],\
                 fit_reg=True, \
                       line_kws={'color': 'red'}).set(xlabel='score final exam',\
                                                      ylabel='score exercises')
             fig = scatter_plot.fig
             fig.suptitle("Module: {}".format(code_module), fontsize=12)
```

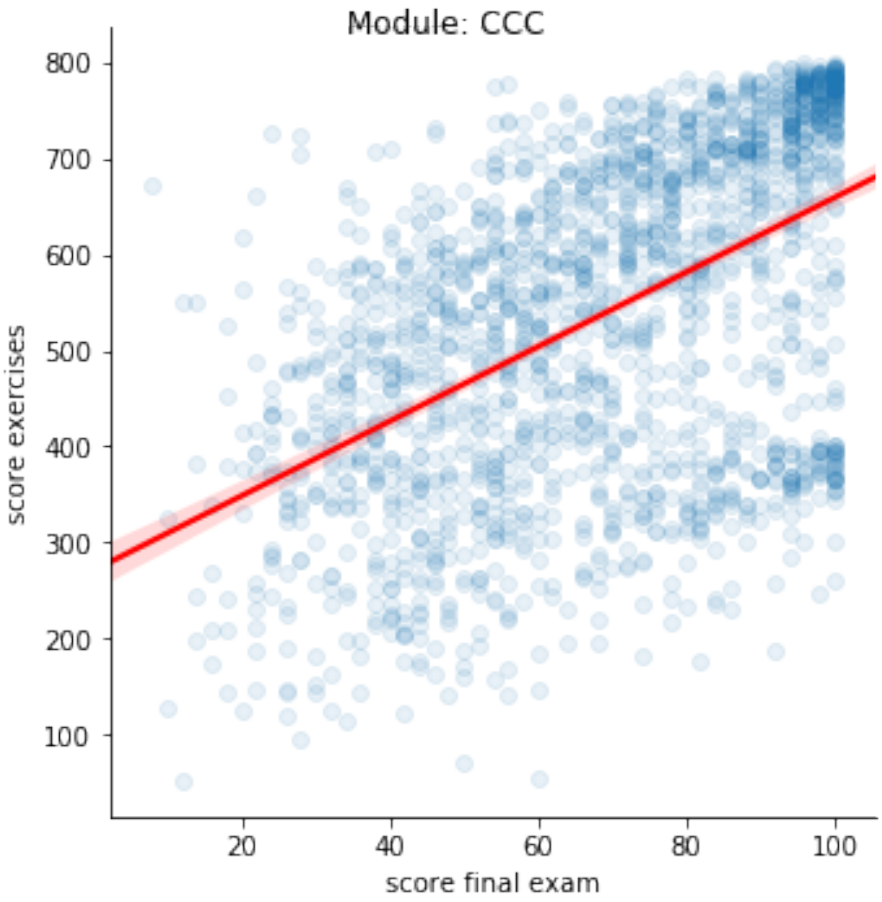

Figure 5.2: Regression between score exercise and score final exam for module CCC

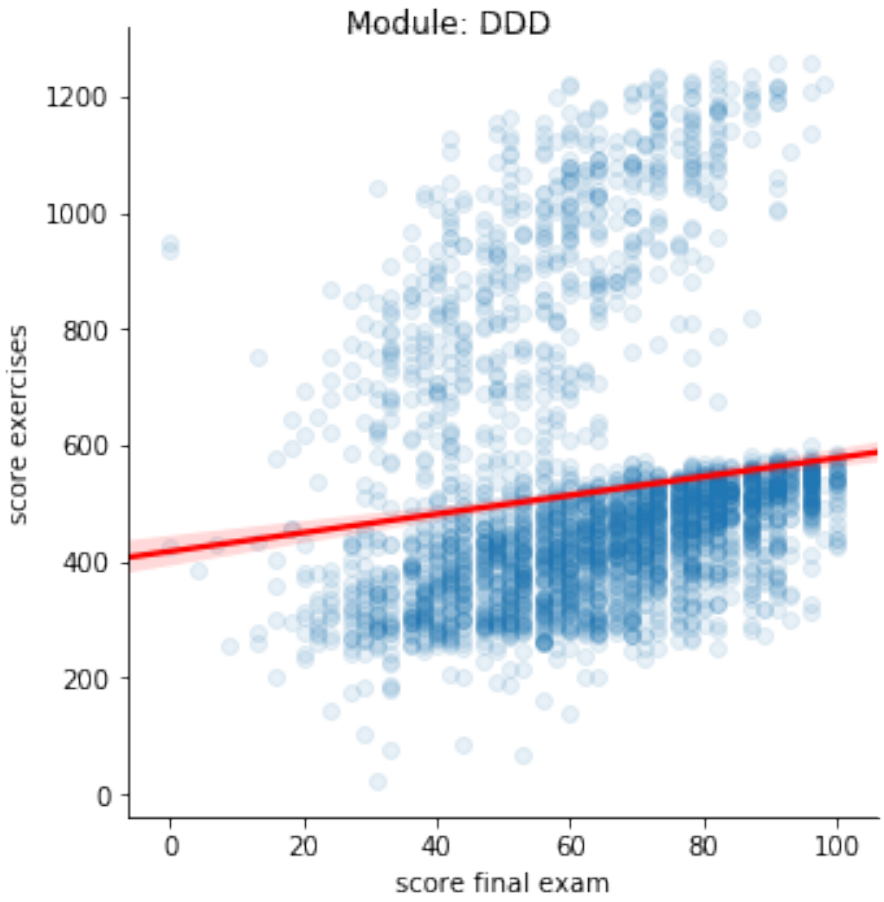

Figure 5.3: Regression between score exercise and score final exam for module DDD

As expected the visual representation is different in both modules. In the module *DDD* we can see 2 clusters, one bigger in the bottom part of the chart and a smaller one in the upper half. If data points are scattered like in the module *DDD* graph, we have to ask our self why there seem to be different clusters and how we can separate them to improve our correlation coefficient. Since we know from the data set description, that the data was accumulated over several semesters we can assume that these clusters formed because something changed in the course. To separate the different groups visually, we can plot the data again with the same code, but this time we will also specify the variable *hue* which assigns a different color to each semester.

```
In [19]: scatter_plot = \
         sns.lmplot(x='score_exam',y='score_non_exam', scatter_kws={'alpha':0.1},\
                    data=assessment_eval[(assessment_eval.code_module=='DDD')],\
                    hue='code_presentation').set(xlabel='score final exam', \
                                                 ylabel='score exercises')
         fig = scatter_plot.fig
         fig.suptitle("Module: {}".format(code_module), fontsize=12)

Out[19]: Text(0.5, 0.98, 'Module: DDD')
```

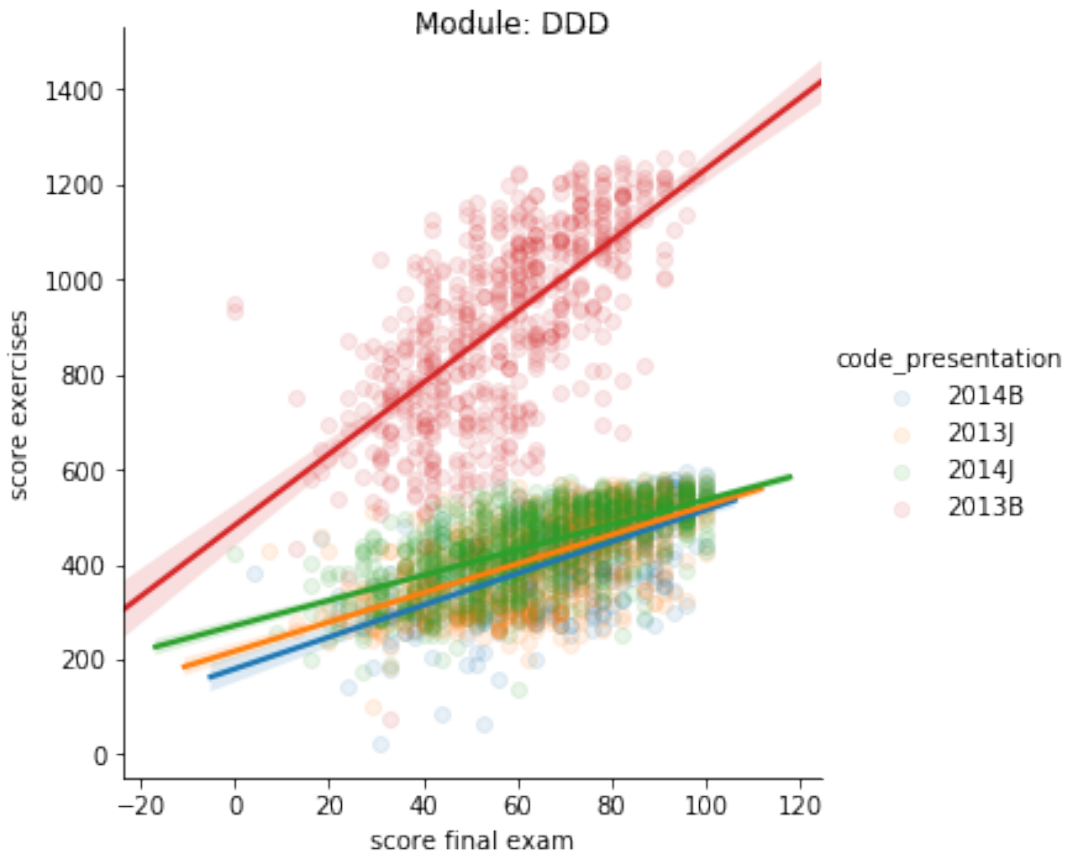

Figure 5.4: Regression between score exercise and score final exam for module DDD separated by semester

In the scatter plot above, we could identify the reason of the different clusters. In the Semester *2013B* it looks like the maximum amount of points that could be scored during the semester was higher. This tells us that we have to norm the exercise points (e.g. calculate percentage of points reached) of the students to get better predictions. Finally we want to calculate the PPMCC of the different semesters

```
In [20]: corr_DDD_presentation = pd.DataFrame()
         presentation_list = list()
         for presentation in assessment_eval[(assessment_eval.code_module == 'DDD')]\
                 ['code_presentation'].unique():
             presentation_list += [presentation, presentation]
             corr_DDD_presentation = \
                 pd.concat([assessment_eval[['score_exam', 'score_non_exam']]\
                        [(assessment_eval.code_module == 'DDD') & \
                         (assessment_eval.code_presentation == presentation)].corr(),
                            corr_DDD_presentation])
         corr_DDD_presentation['presentation'] = presentation_list[::-1]
         corr_DDD_presentation = corr_DDD_presentation.set_index(['presentation', \
```

```
                                    corr_DDD_presentation.index])
corr_DDD_presentation
```

Out[20]:

presentation		score_exam	score_non_exam
2013B	score_exam	1.000000	0.658489
	score_non_exam	0.658489	1.000000
2014J	score_exam	1.000000	0.637596
	score_non_exam	0.637596	1.000000
2013J	score_exam	1.000000	0.618482
	score_non_exam	0.618482	1.000000
2014B	score_exam	1.000000	0.635166
	score_non_exam	0.635166	1.000000

In the table above we the PPMCC was calculated for each semester. As expected we get a higher PPMCC when we separate the different semesters. With a PPMCC being around 0.63 we have found a statistical significant correlation.

Conclusion

When we examined the two modules that use rated exercises, *CCC* and *DDD*, we found that both modules have a PPMCC over 0.5. This means that exercise points are a statistical relevant estimator to predict the *final exam points* of a student. The exercise points of a student could also be used to find students who need additional material or courses to pass a module.

5.4 Are there parallels between the students interactions with course materials and their understandings of the subject matter?

From the previous analysis we could conclude that there is a correlation between exercise points and exam points. However why do certain students seem to have a better understanding of the exercises and are therefore better in the final exams? To investigate this circumstance we want to explore the click data of the students in the virtual learning environment (*vle*) and try to find groups that have a similar behavior. Ideally these groups will also have similarities in their mean exercise points or pass rate like the cluster they were assigned to. For building groups and classification tasks, K-means is a very common algorithm that helps to find clusters in a big data set. K-Means is an algorithm that can be best imagined in a 3-dimensional space, where it places K spheres around the data points. K is the number of clusters that fit the dataset, where K is an integer greater than zero. The algorithm places the center and radius of the spheres to reduce the combined distance of each data point to it's sphere center to a minimum. Clustering over so many variables will result in a large number of detailed clusters, which will be more exact than they need to be since we're more interested in the bigger picture. Why reducing the number of variable results in better clusters can be imagined best when we think about a weekend where we expect students to study and take time off to relax. We do not want to put a student who studied more on Saturday than on Sunday in a different cluster than a student that did it the other way around. This means we want to put students that study more in a similar day range (like on a weekend) in the same cluster. To reduce the selectivity of the K-means algorithm we will need to prepare the data set accordingly. A common practice to reduce the dimensionality and selectivity of a data set is the principal component analysis (pca). We can specify the numbers of components onto whom we want to reduce our data set to. The number of components should be greater than zero and smaller than the number of features we have (in our example the features are the days). In short the pca will take all features, find correlations among them and compute them to new features that have a greater entropy. Since we can determine the number of components (new features) we can control the dimensionality of our new data set. Our first goal will be to aggregate the data to a table where we have a column for each day of the semester and one row per student and course.

```
In [21]: vle_interactions = pd.merge(studentVle, vle[['id_site', 'code_module', \
                                'code_presentation', 'activity_type']], \
                                on=['code_module', 'code_presentation', 'id_site'])
```

First we merge the two tables vle and studentVle, to connect the information of the vle with the click events from the students

```
In [22]: vle_interactions.head()
```

Out[22]:

	code_module	code_presentation	id_student	id_site	date	sum_click	activity_type
0	AAA	2013J	28400	546652	-10	4	forumng
1	AAA	2013J	28400	546652	-10	1	
2	AAA	2013J	28400	546652	-10	1	
3	AAA	2013J	28400	546652	-10	8	
4	AAA	2013J	30268	546652	-10	3	

```
1       forumng
2       forumng
3       forumng
4       forumng
```

```
In [23]: interactions_agg_student_module_day = vle_interactions[['code_module', \
             'code_presentation', 'id_student', 'date', 'sum_click']].\
             groupby(['code_module', 'code_presentation', 'id_student', 'date']).sum().\
             reset_index()
```

Below we have a table with the total number of clicks per student per day per module per semester

```
In [24]: interactions_agg_student_module_day.head()
```

```
Out[24]:   code_module code_presentation  id_student  date  sum_click
         0         AAA              2013J       11391    -5         98
         1         AAA              2013J       11391     0         49
         2         AAA              2013J       11391     1        127
         3         AAA              2013J       11391     2          4
         4         AAA              2013J       11391     6          3
```

Now we need to fill in the days where the students were not active. If we would iterate with a for loop over the data frame it would take quite some time to compute this simple operation, reason being the large table we have created. Thats why it's more efficient to us the operation merge. We will create a dummy data frame with all possible days on whom we will merge our *interactions_agg_student_module_day* fill all empty values with zero.

```
In [25]: df_days = pd.DataFrame({'date': [day for day in range(-100,401)], 'dummy': 1})
         df_unique_student_module_presentation = \
         interactions_agg_student_module_day[['code_module', 'code_presentation', \
                                              'id_student']].drop_duplicates()
         df_unique_student_module_presentation['dummy'] = 1
         df_unique_student_module_presentation = \
         pd.merge(df_days, df_unique_student_module_presentation, on='dummy', how='left')
         del df_unique_student_module_presentation['dummy']
         df_unique_student_module_presentation.ffill()
         df_unique_student_module_presentation = \
         pd.merge(df_unique_student_module_presentation, interactions_agg_student_module_day,\
                  on=['code_module', 'code_presentation', 'id_student', 'date'], how='left')
         df_unique_student_module_presentation['sum_click'] = \
         df_unique_student_module_presentation['sum_click'].fillna(0)
```

Since not all modules have the same presentation length, we need to cut off the dummy days that are not between start and end of the module

```
In [26]: for module in df_unique_student_module_presentation.code_module.unique():
             for presentation in df_unique_student_module_presentation\
             [df_unique_student_module_presentation.code_module == module].\
```

```
            code_presentation.unique():
                min_date = \
                interactions_agg_student_module_day\
                [(interactions_agg_student_module_day.code_presentation == presentation) & \
                 (interactions_agg_student_module_day.code_module == module)]['date'].min()
                max_date = \
                interactions_agg_student_module_day\
                [(interactions_agg_student_module_day.code_presentation == presentation) & \
                 (interactions_agg_student_module_day.code_module == module)]['date'].max()
                df_unique_student_module_presentation = \
                df_unique_student_module_presentation\
                [~((df_unique_student_module_presentation.code_module == module) & \
                (df_unique_student_module_presentation.code_presentation == presentation) &\
                ((df_unique_student_module_presentation.date < min_date) | \
                (df_unique_student_module_presentation.date > max_date)))]
```

```
In [27]: df_unique_student_module_presentation.head()
```

Out[27]:

	date	code_module	code_presentation	id_student	sum_click
2207538	-25	DDD	2014J	8462	0.0
2207539	-25	DDD	2014J	25572	0.0
2207540	-25	DDD	2014J	27417	0.0
2207541	-25	DDD	2014J	33681	29.0
2207542	-25	DDD	2014J	33796	0.0

Above we can see a data frame were we have assembled all the data we need to preform our analysis. In the cell below we can choose the module and presentation (when the module started) we want to examine.

```
In [28]: module = 'CCC'
         presentation = '2014J'
```

After we have prepared our dataset and selected module and presentation, we can pivot it to form it to it's final form, for the principal component analysis (PCA) and K-means.

```
In [29]: studentvle_pivot = \
             df_unique_student_module_presentation[(df_unique_student_module_presentation.\
                 code_module == module) & (df_unique_student_module_presentation.\
                 code_presentation == presentation)]
         studentvle_pivot = studentvle_pivot[['date', 'id_student', 'sum_click']].\
             pivot(index='id_student', columns='date', values='sum_click')
         studentvle_pivot.head()
```

Out[29]:

date	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	...	\
id_student											...	
23698	4.0	0.0	0.0	0.0	1.0	6.0	0.0	2.0	0.0	77.0	...	
25261	30.0	10.0	8.0	4.0	0.0	0.0	1.0	10.0	0.0	0.0	...	
27116	0.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	0.0	0.0	...	
28787	0.0	5.0	0.0	0.0	13.0	0.0	0.0	0.0	0.0	0.0	...	

```
28952        0.0   0.0   0.0   0.0   0.0   0.0   0.0  28.0   0.0   0.0  ...

date         260   261   262   263   264   265   266   267   268   269
id_student
23698        0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
25261        0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
27116        0.0   0.0   4.0   0.0   0.0   4.0   0.0   4.0   0.0   3.0
28787        0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  25.0   0.0
28952        0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

[5 rows x 288 columns]
```

After pivoting, we will preform the principal component analysis to reduce the data frame.

```
In [30]: from sklearn.decomposition import PCA

         n_components = 5
         pca = PCA(n_components)
         pca.fit(studentvle_pivot)

         studentvle_pca = \
         pd.DataFrame(pca.transform(studentvle_pivot), \
                              columns=['PCA%i' % i for i in range(n_components)], \
                              index=studentvle_pivot.index)
```

```
In [31]: studentvle_pca.head()
```

```
Out[31]:                  PCA0      PCA1       PCA2       PCA3       PCA4
         id_student
         23698      -26.487352 -3.107317   4.325646 -13.751894 -15.049954
         25261      -26.048528  1.657808  18.557800 -48.723217 -33.935295
         27116       30.375342 -8.535808  11.407198 -18.994751 -27.446637
         28787      -43.647153 -4.473007  -1.291867 -11.675472  20.581356
         28952      -79.481406  1.407112   0.503242  -5.852509  22.683256
```

On this reduced data frame we can now apply the K-Means algorithm to find the optimal number of clusters and to divide the students into them, based on their behavior in the vle.

```
In [32]: from sklearn.cluster import KMeans
         from matplotlib import pyplot as plt

         X = studentvle_pca
         distorsions = []
         for k in range(2, 50):
             kmeans = KMeans(n_clusters=k)
             kmeans.fit(X)
             distorsions.append(kmeans.inertia_)
```

After calculating the squared errors for each number of clusters between 2 and 50 we can plot the elbow curve. As we can see below, with every cluster we add the error gets smaller, but the

decline of the curve gets flatter with every added cluster. In the plot we can read the inflection point, which lies somewhere around 8. This means that 8 clusters are best to divide our dataset in cluster with the optimal balance between squared error and number of clusters.

```
In [33]: fig = plt.figure(figsize=(15, 5))
         plt.plot(range(2, 50), distorsions)
         plt.grid(True)
         plt.title('Elbow curve');
```

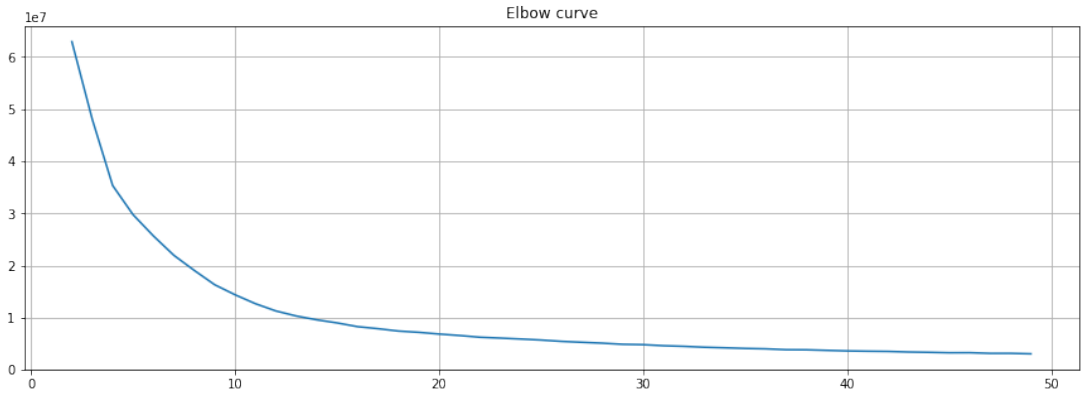

Figure 5.5: Elbow curve for cluster sizes between 2 and 50

```
In [34]: import numpy as np
         n_cluster = 8
         km = KMeans(n_cluster, random_state=100).fit(studentvle_pca)

         cluster_map = pd.DataFrame()
         cluster_map['data_index'] = studentvle_pca.index.values
         cluster_map['cluster'] = km.labels_

         cluster_map = \
         pd.merge(cluster_map, \
                assessment_eval[(assessment_eval.code_module == module) &\
                                (assessment_eval.code_presentation == presentation)], \
                              left_on= 'data_index', right_on='id_student')
         cluster_map['passed'] = np.where(cluster_map.score_exam >= 40, 1, 0)
```

After grouping the data into the cluster we can start to look into the clusters and examine if clustering by the online activities resulted in clusters that also have other similarities.

```
In [35]: results = pd.DataFrame()

         for cluster in range(0,n_cluster):
             ev = list()
             data_tmp = cluster_map[cluster_map.cluster == cluster]
```

```
results.loc['size', cluster] = len(data_tmp)
results.loc['rel_size', cluster] = len(data_tmp)/len(cluster_map)
results.loc['mean_score_exam', cluster] = data_tmp.score_exam.mean()
results.loc['mean_score_exercise', cluster] = data_tmp.score_non_exam.mean()
results.loc['overall_mean_score_exam', cluster] = cluster_map.score_exam.mean()
results.loc['mean_score_non_exam', cluster] = data_tmp.score_non_exam.mean()
results.loc['overall_mean_score_non_exam', cluster] = \
cluster_map.score_non_exam.mean()
results.loc['pass_rate', cluster] = data_tmp.passed.mean()
results.loc['overall_pass_rate', cluster] = cluster_map.passed.mean()
results.loc['rel_deviation_from_mean_exam_score', cluster] = \
(data_tmp.score_exam.mean() - cluster_map.score_exam.mean()) * 100 \
/ data_tmp.score_exam.max()
results.loc['rel_deviation_from_mean_exercise_score', cluster] = \
(data_tmp.score_non_exam.mean() - cluster_map.score_non_exam.mean()) * 100 \
/ data_tmp.score_non_exam.max()
results.loc['rel_deviation_from_mean_pass_rate', cluster] = \
(data_tmp.passed.mean() - cluster_map.passed.mean()) *100
```

The cluster sizes are quite unequal distributed, ranging from one up to 629 students. Also big differences in mean exam score and mean exercise score between the clusters show up, as well as differing pass rates. We can conclude from our findings that just with the number of actions per student per day we were able to find clusters of high performing students as well as students that might need further assistance to succeed in their academic career.

```
In [36]: results
```

Out[36]:

	0	1	2	\
size	629.000000	110.000000	1.000000	
rel_size	0.538527	0.094178	0.000856	
mean_score_exam	66.550079	75.145455	64.000000	
mean_score_exercise	529.430843	615.181818	489.000000	
overall_mean_score_exam	68.234589	68.234589	68.234589	
mean_score_non_exam	529.430843	615.181818	489.000000	
overall_mean_score_non_exam	547.501712	547.501712	547.501712	
pass_rate	0.850556	0.963636	1.000000	
overall_pass_rate	0.872432	0.872432	0.872432	
rel_deviation_from_mean_exam_score	-1.684510	6.910866	-6.616545	
rel_deviation_from_mean_exercise_score	-2.270210	8.556271	-11.963540	
rel_deviation_from_mean_pass_rate	-2.187507	9.120486	12.756849	

	3	4	5	\
size	13.000000	1.000000	217.000000	
rel_size	0.011130	0.000856	0.185788	
mean_score_exam	79.230769	40.000000	76.433180	
mean_score_exercise	646.230769	225.000000	615.617512	
overall_mean_score_exam	68.234589	68.234589	68.234589	
mean_score_non_exam	646.230769	225.000000	615.617512	

```
overall_mean_score_non_exam             547.501712  547.501712  547.501712
pass_rate                                 1.000000    1.000000    0.940092
overall_pass_rate                         0.872432    0.872432    0.872432
rel_deviation_from_mean_exam_score       10.996180  -70.586473    8.198591
rel_deviation_from_mean_exercise_score   12.450070 -143.334094    8.535814
rel_deviation_from_mean_pass_rate        12.756849   12.756849    6.766066

                                                6           7
size                                    196.000000    1.000000
rel_size                                  0.167808    0.000856
mean_score_exam                          60.244898   44.000000
mean_score_exercise                     488.428571  364.000000
overall_mean_score_exam                  68.234589   68.234589
mean_score_non_exam                     488.428571  364.000000
overall_mean_score_non_exam             547.501712  547.501712
pass_rate                                 0.806122    1.000000
overall_pass_rate                         0.872432    0.872432
rel_deviation_from_mean_exam_score       -7.989691  -55.078611
rel_deviation_from_mean_exercise_score   -7.477613  -50.412558
rel_deviation_from_mean_pass_rate        -6.630906   12.756849
```

```
In [37]: results = results.loc[:, results.loc["size"] > 10]
```

To further examine the cluster we drop the clusters having less than 10 students, as they are representing outliers, which leaves us with 5 clusters.

```
In [38]: cluster=results.columns
         assesment=['exam score','exercise points', 'pass rate']
         pos = np.arange(len(cluster))
         bar_width = 0.25
         deviation_from_mean_exercise_score=\
         list(results.loc['rel_deviation_from_mean_exercise_score'])
         deviation_from_mean_exam_score=\
         list(results.loc['rel_deviation_from_mean_exam_score'])
         pass_deviation=list(results.loc['rel_deviation_from_mean_pass_rate'])

         plt.bar(pos,deviation_from_mean_exercise_score,\
                 bar_width,color='blue',edgecolor='black')
         plt.bar(pos+bar_width,deviation_from_mean_exam_score,\
                 bar_width,color='green',edgecolor='black')
         plt.bar(pos+(bar_width*2),pass_deviation,\
                 bar_width,color='yellow',edgecolor='black')
         plt.xticks(pos+bar_width, cluster)
         plt.xlabel('cluster', fontsize=16)
         plt.ylabel('deviation in %', fontsize=16)
         plt.title(('Relative deviation of pass rate, exam and exercise points' + \
                    'from mean across clusters'),fontsize=18)
         plt.legend(assesment,loc=3)
```

```
plt.axhline(0, color='black', linestyle='--')
plt.show()
```

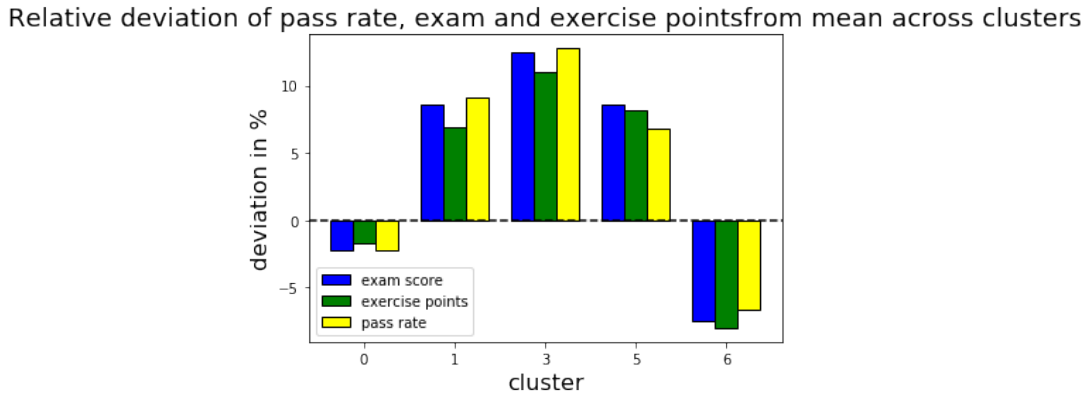

Figure 5.6: Deviation from mean for different clusters

Above we can see a bar chart that shows the relative derivation from the mean for exam points, exercise points and pass rate for the different clusters. The first thing to notice is that 3 clusters are well above average, one being close to average and one well below average. The two clusters below average, represent also the fast majority of students (around 70%). To get a better understanding of the clusters we examine the click rates that led to them.

```
In [39]: cluster_activity = pd.DataFrame()
         for cluster in results.columns:
             tmp_df = pd.merge(cluster_map[cluster_map.cluster == cluster],
                               vle_interactions,
                               on=['id_student', 'code_module', 'code_presentation'])
             activities = tmp_df['activity_type'].value_counts()

             for i in activities.index:
                 cluster_activity.loc[i, cluster] = \
                 activities[i]/results.loc['size'][cluster]
```

```
In [40]: cluster_activity
```

Out[40]:

	0	1	3	5	6
quiz	69.453100	215.527273	545.461538	198.935484	100.469388
homepage	66.740859	112.290909	195.076923	143.188940	78.102041
subpage	56.624801	105.118182	167.307692	117.599078	80.122449
resource	50.184420	78.072727	109.923077	99.336406	63.841837
forumng	34.815580	154.936364	621.538462	269.935484	60.765306
oucontent	25.370429	66.700000	92.846154	74.400922	42.852041
url	6.176471	12.445455	24.769231	15.023041	7.887755
oucollaborate	1.634340	3.509091	3.153846	5.672811	3.489796
page	1.279809	2.163636	3.076923	2.322581	1.576531

In the table above we have splitted the total clicks of each cluster into the 9 types of content that were available in this module. Now we will first create a bar chart that shows the mean clicks per student in a semester of the respected cluster. Following this we will take a look on how the different clicks were distributed on the different types of content.

```
In [41]: clicks_per_student_per_cluster = cluster_activity.sum()
         clicks_per_student_per_cluster

         plt.bar(pos,clicks_per_student_per_cluster,\
                 bar_width,color='blue',edgecolor='black')
         plt.xlabel('cluster', fontsize=16)
         plt.ylabel('absolute clicks', fontsize=16)
         plt.title('Clicks per student per cluster during semester',fontsize=18)
         plt.legend(['clicks per student'],loc=1)
         plt.show()
```

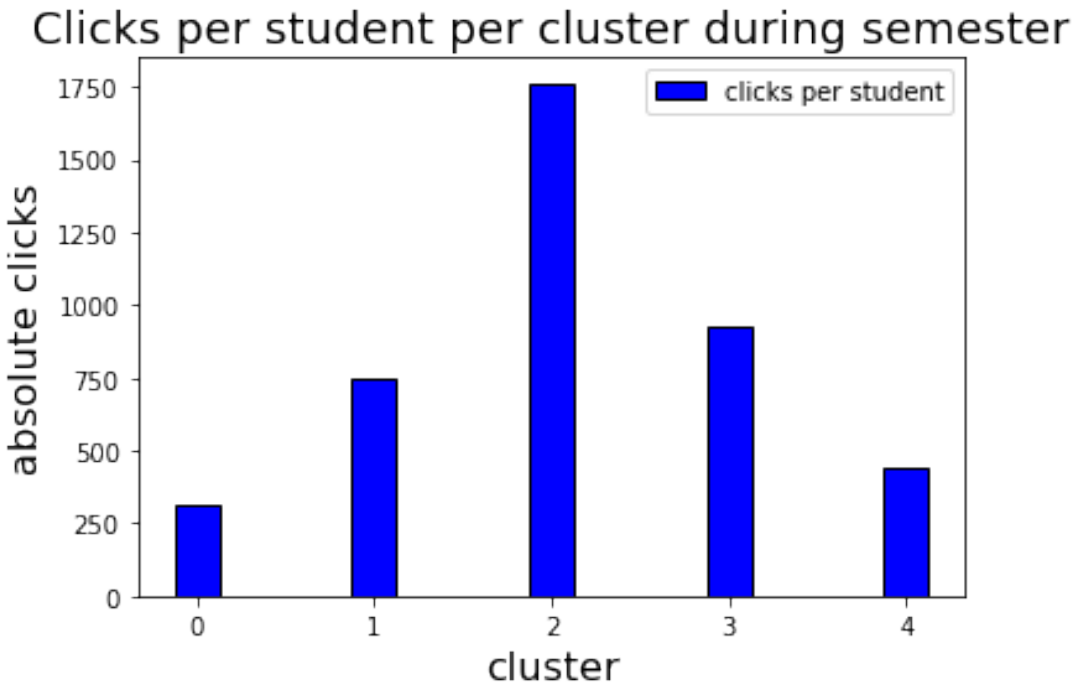

Figure 5.7: Mean click numbers per student per cluster

```
In [42]: ax = sns.heatmap(cluster_activity.div(cluster_activity.sum(axis=0), axis=1), \
                          linewidths=.5 ,robust=True ,annot_kws = {'size':14})
         ax.tick_params(labelsize=14)
         ax.figure.set_size_inches((12, 10))
```

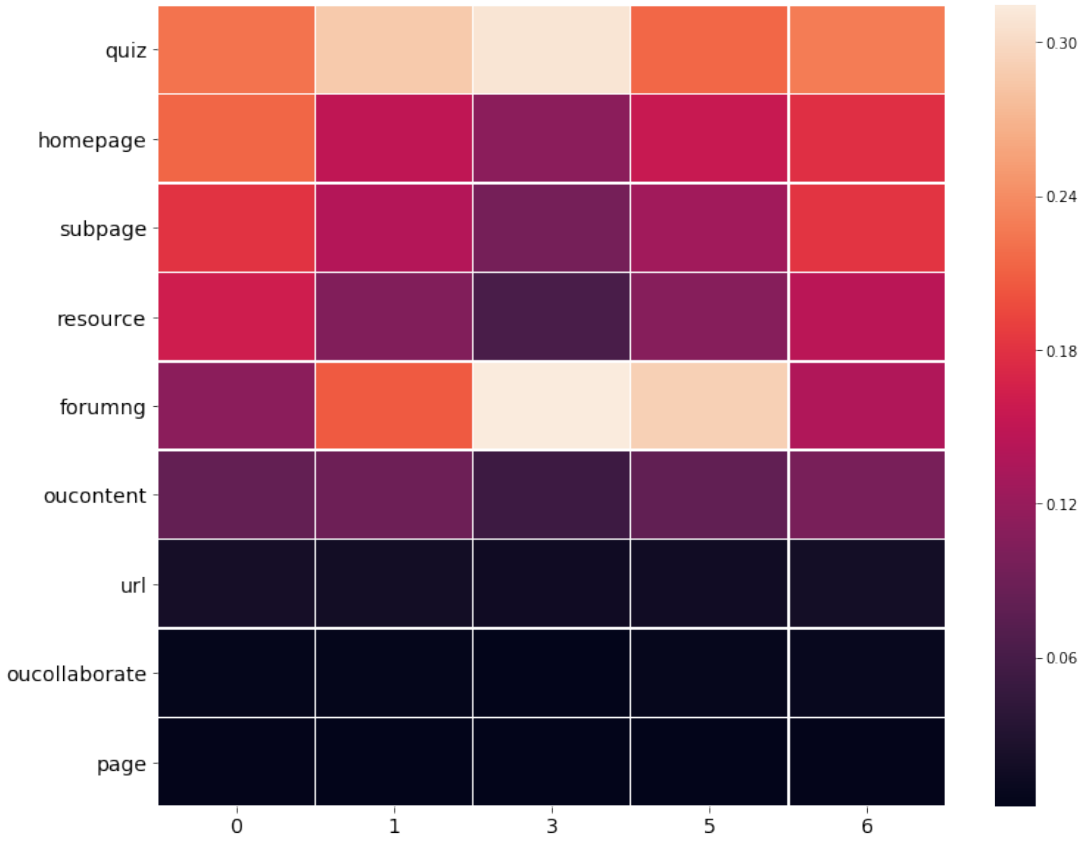

Figure 5.8: Click distribution on content per cluster

In bar chart we can see that cluster 1 and 5 accumulated more than 750 clicks during the semester and cluster 3 even over 1500 clicks. The clusters 0 and 6, which perform below average, only have around 400 clicks per semester. In the heat map we can look closer into the distribution of the clicks in the clusters. We can see that the high performing clusters spent the most clicks on *forumng* and *quiz* where the clusters 1 and 6 were not so active in the forum. This might indicate, that more social interaction of students also result in higher success rates.

Conclusion

With the principal component analysis in combination with K-means we could divide students into different clusters with different chances of succeeding a module by analyzing their behavior on the virtual learning environment. We could distinguish the clusters into 3 main types average, below and above average where the average group is made up of cluster 0 with 629 students, group above average consists of the clusters 1, 3 and 5 with a total of 340 students and the group below average gets covered by the cluster 6 with 196 students. From the data we have descried we can conclude that the distribution of clicks over the semester seems to be a good estimator if a student is doing well in their semester.

5.5 How far in advance can we predict which students are likely to fail their final exam?

To allocate students that need more help to pass the final exam as early as possible, we need to build a machine learning model that can predict the outcome of a students final exam with a limited amount of data. To simulate a limited amount of data we will aggregate the data how it would have looked like at x days (x being between -20 and 240) into the semester. When we take look at our data set we can see that we have data that is already available at the beginning of the semester. This data consist of the students demografic data like age, place of residence and sex as well as data that was gathered during previous semesters like the number of previous attempts the student took to pass a module or how much credit points he already has. Data that is not available at the start of the semester are for example the clicks the student will generate during the semester and his exercise points. We will start by creating a data frame that has one line for every unique *id_student*, *code_module*, *code_presentation*, *date* combination. In the previous chapter we have used already a data frame with the same porperties. So we just need to copy the data frame *df_unique_student_module_presentation*.

```
In [43]: df_timeseries = \
         df_unique_student_module_presentation\
         [df_unique_student_module_presentation.date <= 240]
         df_timeseries.head()
```

Out[43]:

	date	code_module	code_presentation	id_student	sum_click
2207538	-25	DDD	2014J	8462	0.0
2207539	-25	DDD	2014J	25572	0.0
2207540	-25	DDD	2014J	27417	0.0
2207541	-25	DDD	2014J	33681	29.0
2207542	-25	DDD	2014J	33796	0.0

Next we construct a data frame of all the non exam assessments in the data set to join it with our timeseries data frame.

```
In [44]: non_exam_assesments_to_date = \
         pd.merge(assessments[(assessments.assessment_type != 'Exam') & \
                              (assessments.date <= 240)], studentAssessment, \
                  on='id_assessment')
         non_exam_assesments_to_date.head()
```

Out[44]:

	code_module	code_presentation	id_assessment	assessment_type	date	weight	\
0	AAA	2013J	1752	TMA	19.0	10.0	
1	AAA	2013J	1752	TMA	19.0	10.0	
2	AAA	2013J	1752	TMA	19.0	10.0	
3	AAA	2013J	1752	TMA	19.0	10.0	
4	AAA	2013J	1752	TMA	19.0	10.0	

	id_student	date_submitted	is_banked	score
0	11391	18	0	78.0
1	28400	22	0	70.0

```
2       31604              17          0   72.0
3       32885              26          0   69.0
4       38053              19          0   79.0
```

```
In [45]: df_timeseries = \
         pd.merge(df_timeseries, \
                  non_exam_assesments_to_date[['date', 'id_student', 'code_module', \
                                               'code_presentation', 'score']], \
                  on=['id_student', 'code_module', 'code_presentation', 'date'], how='left')
         df_timeseries['score'] = df_timeseries['score'].fillna(0)
```

After we have successfully joined the two data sets we can calculate the cumulative sum of the exercise points and the clicks. This means that in the new rows *exercise_score_cumsum* and *sum_click_cumsum* are all the previous values from *score* and *sum_click* added to this day.

```
In [46]: df_timeseries['exercise_score_cumsum'] = \
         df_timeseries.groupby(['id_student', 'code_presentation', 'code_module'])\
         ['score'].transform(pd.Series.cumsum).fillna(0)
         df_timeseries['sum_click_cumsum'] = \
         df_timeseries.groupby(['id_student', 'code_presentation', 'code_module'])\
         ['sum_click'].transform(pd.Series.cumsum)
```

```
In [47]: df_timeseries.head()
```

```
Out[47]:    date code_module code_presentation  id_student  sum_click  score  \
         0   -25         DDD              2014J        8462        0.0    0.0
         1   -25         DDD              2014J       25572        0.0    0.0
         2   -25         DDD              2014J       27417        0.0    0.0
         3   -25         DDD              2014J       33681       29.0    0.0
         4   -25         DDD              2014J       33796        0.0    0.0

            exercise_score_cumsum  sum_click_cumsum
         0                    0.0               0.0
         1                    0.0               0.0
         2                    0.0               0.0
         3                    0.0              29.0
         4                    0.0               0.0
```

The data frame above now contains all the information, that can change on a daily basis. In the next step we will merge it with the data that stays consistent over the semester.

```
In [48]: df_timeseries = \
         pd.merge(df_timeseries, studentInfo[(studentInfo.final_result == 'Fail') | \
                                             (studentInfo.final_result == 'Pass')], \
                  on=['id_student', 'code_module', 'code_presentation'])
         tmp_module = df_timeseries['code_module'].copy()
         tmp_presentation = df_timeseries['code_presentation'].copy()
         df_timeseries = pd.get_dummies(df_timeseries, columns =\
                                        ['gender', 'region', 'highest_education', \
```

```
                                    'imd_band', 'age_band', 'disability', \
                                    'code_module', 'code_presentation'])
df_timeseries['code_module'] = tmp_module
df_timeseries['code_presentation'] = tmp_presentation
df_timeseries.head()
```

Out[48]:

	date	id_student	sum_click	score	exercise_score_cumsum	sum_click_cumsum	num_of_prev_attempts	studied_credits	gender_F	gender_M	...	code_module_DDD	code_module_EEE	code_module_FFF	code_module_GGG	code_presentation_2013B	code_presentation_2013J	code_presentation_2014B	code_presentation_2014J	code_module	code_presentation
0	-25	33681	29.0	0.0	0.0	29.0	3	60	0	1	...	1	0	0	0	0	0	0	1	DDD	2014J
1	-24	33681	0.0	0.0	0.0	29.0	3	60	0	1	...	1	0	0	0	0	0	0	1	DDD	2014J
2	-23	33681	0.0	0.0	0.0	29.0	3	60	0	1	...	1	0	0	0	0	0	0	1	DDD	2014J
3	-22	33681	0.0	0.0	0.0	29.0	3	60	0	1	...	1	0	0	0	0	0	0	1	DDD	2014J
4	-21	33681	0.0	0.0	0.0	29.0	3	60	0	1	...	1	0	0	0	0	0	0	1	DDD	2014J

```
[5 rows x 56 columns]
```

Now we have joined the consistent data to the timeseries data. To make labeled data like *gender* readable for a machine learning algorithm, we had to one-hot-encode them. This means that every label gets its own column with a binary value, where 1 stands for student has this label and 0 for not having the label. Next we have to join the target variable to the data frame and replace *Pass* with 1 and *Fail* with 0.

```
In [49]: df_tmp = \
         studentInfo[(studentInfo.final_result == 'Fail') | \
                     (studentInfo.final_result == 'Pass')]\
         [['id_student', 'code_module', 'code_presentation', 'final_result']]
         df_tmp['final_result'] = \
         np.where(df_tmp['final_result'] == 'Fail', int(0), int(1))
         df_timeseries = pd.merge(df_timeseries, df_tmp, \
                                  on=['id_student', 'code_module', 'code_presentation'], \
                                  how='left')
```

Finally we are done with manipulating the *OULAD* into a timeseries data stream. We have reconstructed on a day to day basis what a Docent would have known at the specified days into the semester to put us onto the same information base to guarantee that our findings can be replicated and applied to the real world. In the following steps we can now finally try to predict the final exam results for students. We will limit our data set onto different stages of the semester to simulate forecasting at the respective times during the semester.

```
In [50]: input_variables = ['exercise_score_cumsum',
                 'sum_click_cumsum', 'gender_F', 'gender_M',
                 'num_of_prev_attempts', 'studied_credits', 'region_East Anglian Region',
                 'region_East Midlands Region', 'region_Ireland', 'region_London Region',
                 'region_North Region', 'region_North Western Region', 'region_Scotland',
                 'region_South East Region', 'region_South Region',
                 'region_South West Region', 'region_Wales',
                 'region_West Midlands Region', 'region_Yorkshire Region',
                 'highest_education_A Level or Equivalent',
                 'highest_education_HE Qualification',
                 'highest_education_Lower Than A Level',
                 'highest_education_No Formal quals',
                 'highest_education_Post Graduate Qualification', 'imd_band_0-10%',
                 'imd_band_10-20', 'imd_band_20-30%', 'imd_band_30-40%',
                 'imd_band_40-50%', 'imd_band_50-60%', 'imd_band_60-70%',
                 'imd_band_70-80%', 'imd_band_80-90%', 'imd_band_90-100%',
                 'age_band_0-35', 'age_band_35-55', 'age_band_55<=', 'disability_N',
                 'disability_Y', 'code_module_AAA', 'code_module_BBB', 'code_module_CCC',
                 'code_module_DDD', 'code_module_EEE', 'code_module_FFF',
                 'code_module_GGG', 'code_presentation_2013B', 'code_presentation_2013J',
                 'code_presentation_2014B', 'code_presentation_2014J']
         target_variable = ['final_result']
```

After dividing the different variables into input and target variable we import two different classifiers, *KNeighborsClassifier* and *DecisionTreeClassifier*. The KNN algorithm is a quite simple algorithm, that places all data points in an m-dimensional space (m being the number of input variables). To make a prediction it looks up the most prevalent target variable of the K nearest neighbors of the data point to predict and forecasts this target variable. The decision tree tries to divide the data set by binary questions that have the highest information gain. The benefit of this approach is, that we can also have a look into the decision making and what variables are more decisive than others.

```
In [51]: from sklearn.preprocessing import Imputer
         from sklearn.model_selection import train_test_split
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.dummy import DummyClassifier
         from sklearn.preprocessing import StandardScaler
```

The first for loop will iterate over the days we want to predict. We start at 20 days before the modules have started and finish 240 days after the modules have started. In each iteration we divide the data set in a training set and a test set, where the training set consists of 70 % of the available data, and the test set of the remaining 30 %. With the KNN we will iterate also over the amount of neighbors, starting at 1 up to 50. For both algorithms we will save the best result in *best_score_knn* and *best_score_decision_tree*. When classifying the decision tree we will also save the feature importance to see which variables are more or less decisive and how these changes over time.

```
In [52]: eval_feature_importance = pd.DataFrame()
         eval_feature_importance['feature_names'] = input_variables
         best_score_knn = list()
         best_score_decision_tree = list()
         best_score_dummy = list()
         for day in range(-20,250,10):
             # prepare data for information base on day 'day'
             X = df_timeseries[df_timeseries.date==day][input_variables].values
             Y = df_timeseries[df_timeseries.date==day][target_variable].values
             Y = Y.ravel()
             X_train, X_test, y_train, y_test = \
             train_test_split(X, Y, test_size = 0.3, random_state = 100)
             tmp_best_score = list()
             # KNN Classifier
             for K in range(50):
                 K_value = K+1
                 neigh = \
                 KNeighborsClassifier(n_neighbors = K_value, weights='uniform', \
                                      algorithm='auto', n_jobs=-1)
                 neigh.fit(X_train, y_train)
                 y_pred = neigh.predict(X_test)
                 tmp_best_score.append(accuracy_score(y_test,y_pred))
```

```
best_score_knn.append(max(tmp_best_score))
# Decision Tree Classifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
y_pred = dtree.predict(X_test)
best_score_decision_tree.append(accuracy_score(y_test,y_pred))
eval_feature_importance[('data_till_day_' + str(day))] = \
dtree.feature_importances_
# Dummy Classifier
dummy = DummyClassifier()
dummy.fit(X_train, y_train)
y_pred = dummy.predict(X_test)
best_score_dummy.append(accuracy_score(y_test,y_pred))
```

After running the machine learning algorithms we plot the accuracy of both algorithms and see that the KNN always outperforms the Decision Tree. We can also read from the chart how both algorithms improve over time. This means that as the semester progresses, variables like *exercise_score_cumsum* become more important to predict if a student will pass this semester. With a pass rate around 55 % in the data set, measured over all modules, a random algorithm based on the distribution of fails and passes would produce an accuracy of also 55 %. From the plot below we can read, that with the KNN algorithm we achieve better results right from the beginning

```
In [53]: knn = plt.plot(best_score_knn, color='blue', label='KNN')
         decision_tree = plt.plot(best_score_decision_tree, \
                                  color='red', label='Decision Tree')
         dummy = plt.plot(best_score_dummy, color='green', label='Dummy Classifier')
         plt.legend()
         plt.xlabel('Days since modul start')
         plt.ylabel('Accuracy in percent')
         plt.title('Accuracy comparison of the Knn and Decision Tree Algorithm over time',\
                   fontsize=18)
         ax = plt.gca()
         ax.set_xticks(range(0,27,2))
         ax.set_xticklabels(range(-20,250,20));
```

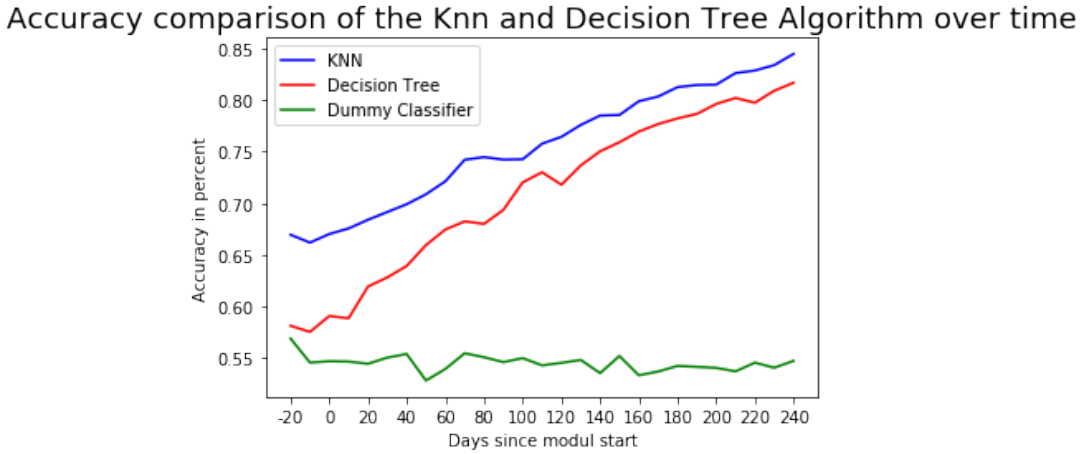

Figure 5.9: Accuracy history over time

Now we want to take a look into the feature importance. Since we saved all the data during the calculation of the classifiers into the data frame *eval_feature_importance* we can plot the data as a heat map easily. What stands out are the two most important variables *exercise_score_cumsum* and *sum_click_cumsum*. We can also see that in the beginning the clicks are more important than the exercise scores, but this changes around 2 months after semester start.

```
In [54]: plt.figure(figsize=(15, 15))
         sns.heatmap(eval_feature_importance.set_index('feature_names'));
```

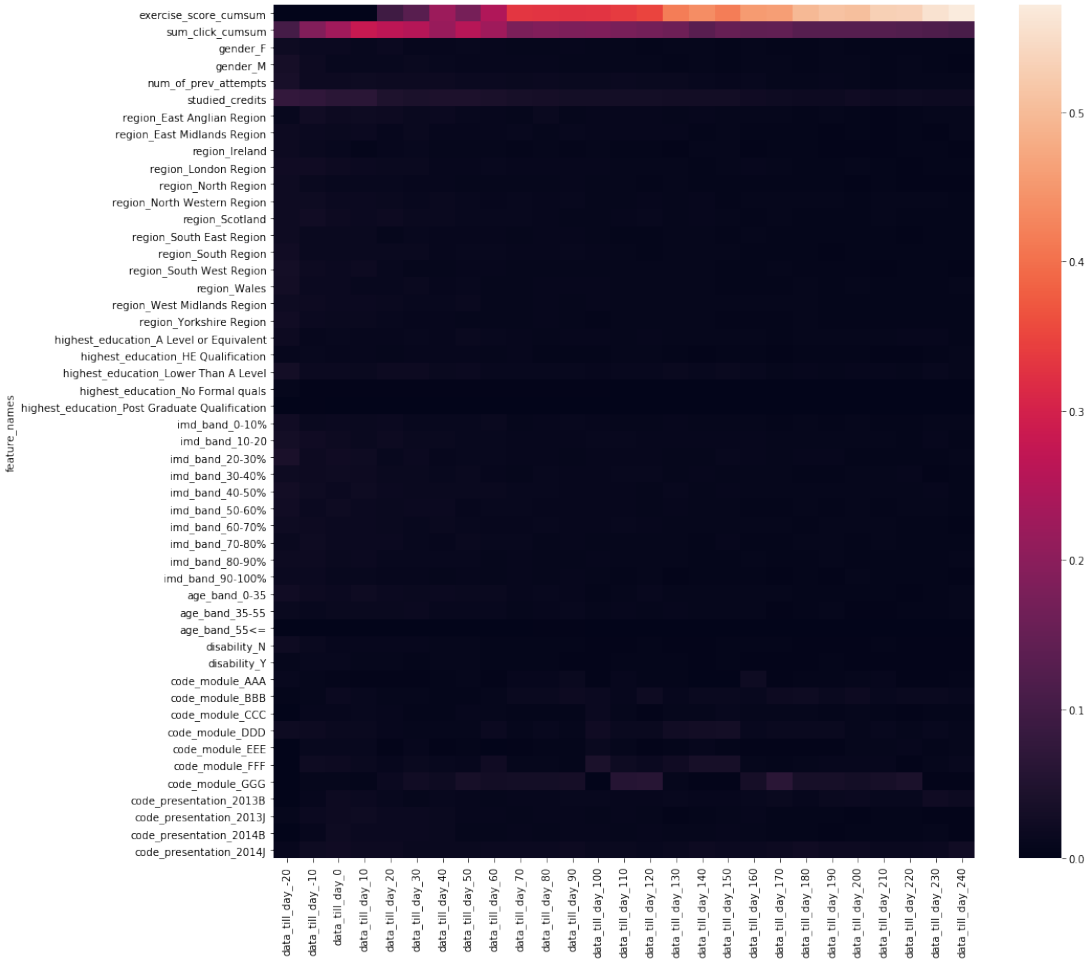

Figure 5.10: Feature importance over time

Conclusion

After 100 days into the semester we can already predict with over 70 % accuracy which students will fail and could therefore provide them with more intensive courses that could cater their needs more to help them to pass their final exam. We could also prove again like in *5.3 Is there a relationship between higher exercise points and higher final grades* that exercise points play a vital role in learning analytics and have a strong connection to a students success.

6 Conclusion

This chapter provides a summary by recapitulating the findings that have been made when answering the research questions

6.1 Summary

In chapter 2 the research object, the Open University Learning Analytics dataset, was presented. The chapter showed the structure of a Learning Analytics data set and the limitations that came with it because it needed to be anonymized to make it publicly available. Furthermore we outlined how a data set like the OULAD could look like for the Goethe University to make advanced analytic methods possible.

Next we took a look into other research that is centered around the data set and showed what approach others took with the OULAD and how this thesis contributes to their work.
After that the machine learning algorithms that were used in the Jupyter Notebook were introduced. We showed what kind of data these algorithms need, what kind of problems they can solve and how they are solving these questions.

In Chapter 5 the research questions were answered by examining the data set with the programming language Python in the integrated development environment Jupyter Notebook. The findings provided by the Jupyter Notebook can play an essential part in how students will study in the future. With these better thought-out learning paths for students can be designed and help can offered on a more fine grained scale

6.2 Result

Although there has been extensive research on the Open University Learning Analytics Dataset we could still discover new and relevant topics, like how the importance of variables change over the course of the semester when predicting fails and passes. The feature importance gathered from the Decision Tree algorithm in chapter 5.4 over the duration of the semester shows impressively the shift from click rates to exercise sum one to two months into the semester. Although the KNN in this chapter showed that already after 60 days into a semester an accuracy of 75% could e achieved.
Furthermore the importance of the distribution of clicks on the different types of content was proven in chapter 5.3 showing that outperforming students interact more with each other on the forum than low performing students.
A strong link between exercise points and the result of the students in their final exam could be

discovered as well, showing how great the impact of exercises on a student's understanding of a subject is. This can also be seen in the feature importance of the prediction model in chapter 5.4.

7 Outlook

The presented algorithms in chapter 5.4 can be further improved by engineering more features. For example could the clusters of chapter 5.3 be used as features. This would require to calculate the clusters for each new day into the semester new, which is very computation intensive. Also, a bigger data set is needed to provide a true train and test data set. The new train data set should consist entirely of predecessing semesters to the test data set to ensure that the data that is provided to the model could also be available in reality.

A big part of this thesis focused on finding students that are low performing and are having trouble to pass the final exam. The next step would be to try to distinguish the students that are genuinely trying to succeed by studying a lot from other students to better allocate valuable teaching resources.

8 Glossary

datframe	a pandas object that can store data
exercise_score_cumsum	cumulative sum of exercise points
final exam score	score of a student in his final exam
matplotlib	a open source library to plot data
pandas	an open source library that helps to handle and manipulate data
PCA	Principal component analysis; combines two variables that correlate with each other to one new component
PPMCC	Pearson product-moment correlation coefficient; a way to meassure the correlation between two variables
regression line	line that best fits the trend of a data set
score exercises	the sum of all exercise points per semester per student
score final exam	score of the final exam
score_exam	variable that holds the final exam score
score_non_exam	variable that holds the sum of all exercises per semester per student
sum_click_cumsum	cumulative sum of the number of clicks in the VLE
time of enrolment	when a student enrolled into a course relative to semester start

List of Figures

Bibliography

[1] Q. Nguyen, M.Huptych, and B. Rienties. 2018. Linking students' timing of engagement to learning design and academic performance. In: LAK'18: International Conference on Learning Analytics and Knowledge, March 7–9, 2018, Sydney, NSW, Australia. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3170358.3170398

[2] Greller, W., & Drachsler, H. (2012). Translating Learning into Numbers: A Generic Framework for Learning Analytics. Educational Technology & Society, 15 (3), 42–57.

[3] Kuzilek, J. et al. Open University Learning Analytics dataset. Sci. Data 4:170171 doi: 10.1038/sdata.2017.171 (2017)

[4] H. Heuer, and A. Breiter. 2018. Student Success Prediction and the Trade-Off between Big Data and Data Minimization. In: Die 16. E-Learning Fachtagung Informatik, Bonn 2018, 12 pages.

[5] Jha, Nikhil, Ghergulescu, Ioana and Moldovan, Arghir-Nicolae (2019) OULAD MOOC Dropout and Result Prediction using Ensemble, Deep Learning and Regression Techniques. In: Proceedings of the 11th International Conference on Computer Supported Education - Volume 2: CSEDU. SciTePress, pp. 154-164. ISBN 9789897583674

[6] Pedregosa et al. Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 201

[7] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

[8] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

[9] Michael Waskom, Olga Botvinnik, Paul Hobson, John B. Cole, Yaroslav Halchenko, Stephan Hoyer, Alistair Miles, et al. "Seaborn: V0.5.0 (november 2014)". Zenodo, November 14, 2014. doi:10.5281/zenodo.12710

[10] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007