DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE DE COIMBRA

Research Methods

Milestone 1 - Exploratory Data Analysis and Linear Regression

Miguel de Oliveira Guerreiro
Luís Espírito Santo
André Carvalho dos Santos

November 10, 2021

1 INTRODUCTION

Experimental design is a systematic way of carrying out research to maximize precision and to draw specific conclusions from a set of pre-established hypotheses. In general, it can be thought of assessing the relationship between independent variables and some controllable dependent variables. The main purpose of the assignment explored in this document is to understand the formal steps of such methodology as well as assess some of the main challenges that one may encounter during this process.

With this in mind, the problem hereby addressed is a synthetic simplified problem with a clear problem statement, simple variables, a somehow prefixed and well defined experimental scenario and some out-of-the-box tools to accelerate this long methodological process, allowing a better learning experience.

This document is structured in 4 main sections. In this section, Section 1, a statement about the artificial problem addressed is provided (Subsection 1.1) as well as an explicit identification of the variables that play a role in this study (Subsection 1.3). The fixed experimental scenario and the tools provided for the experiments are also presented in Section 1. In Section 2 we explain our EDA and present the results we gathered from it. In section 3 we show our Linear Regression model for the problem and whether it holds or not. Finally, we present our preliminary conclusions in section 4.

1.1 PROBLEM STATEMENT

The problem addressed in this document can summarized by the following statement:

There are several exams to be scheduled in non-overlapping time slots such that:

1. **every student does not have to attend more than one exam simultaneously;**
2. **it uses the minimum number of time slots.**

How much time one needs to wait to obtain such schedule?

This seems a very simplified version of the known NP-hard Nurse Roastering Problem (NRP) which has several real applications and challenges and can be solved through different approaches [1]. This simplified version assumes non-overlapping time slots, non-divisible exams and only one kind of hard conditions - conditions like "exam A can't collide with exam B". More realistic versions of this problem would incorporate several types of soft conditions or overlapping time slots.

Given this simplified version, we can model this problem by using simple graphs (undirected graphs without self-loops) where the nodes represent exams and the edges represent the collision conditions we must satisfy. Given this model we may solve this problem by applying some graph partition algorithm aiming at minimizing the flow between partitions. We want to find a partition of the graph nodes with the least subsets such that there are no edges between nodes inside the same partition. Another way to solve this problem is to computer the complement graph and find the minimum clique node cover.

1.2 Experimental Setup & Tools

In order to study this problem we will study two different programs that were provided and implement two randomized backtracking algorithms to solve the exam scheduling problem. While code 1 will follow a top-down strategy, starting to search for acceptable schedules with the maximum number time slots and iteratively decrementing the number of time slots, code 2 will use the opposite bottom-up approach. These programs receive as an input: the data, a random seed (we'll refer to it as Seed 2) and a cutoff time. Another provided component of our experimental setup is an input data generator written in python that receives two input parameters: the number of exams and the probability that each pair of exams will have a student in common, besides a random seed which we named Seed 1.

To generate and manipulate the data we show in this report we developed some small tools in python and a notebook that helped us to visualize the data. These are fully available in a public github repository[1] to allow reproducibility. All the provided code and a document providing a more thorough explanation of the problem and provided resources are also accessible at the repo[1].

1.3 Variables

Independent variables, as the name suggests, aren't affected by the other variables we have in our problem. On the other hand, dependent variables' values change with respect to the independent variables. In a controlled experimental set up, independent variables correspond to the variables we can control and dependent variables correspond to the variables we want to study.

In this case, our main goal is to characterize the performance of two automatic scheduling programs in different scenarios. From the problem statement provided follows that the dependent variable is the total time the program takes to run, and the independent variables may be the number of exams, the number of time slots available, and the number of students that enrolled for each exam. However, on a second thought and according to the actual implementation of the software we are expected to use in our experiments, we cannot control the number of students or the available time slots directly. Rather, we control the probability that each pair of exams will have a student in common (p) and the total number of exams (n). Therefore, our actual independent variables - the ones we can control - are p and n. Besides, we also have two different implementations of the software which go by the name "Code 1" and "Code 2" respectively, so we actually have two different dependent variables - those that are assumed to be impacted by the independent variables - the runtime for "Code 1" (t_1) and for "Code 2" (t_2).

If we follow the strategy proposed at the end of Subsection 1.1 and represent the input data as a graph then we will see that there are n nodes and p is actually the probability of any two nodes sharing an edge. We can see then that actually we are generating problems using the Erdős–Rényi model [2]. Moreover, we can compute the number of vertices V and the expected value of the number of edges $E(E)$ using formulas 1.1 and 1.2, respectively. We can see that the mean number of edges E is a portion of all possible edges (n^2) in a graph with n vertices. Being a simple graph we need to make sure that there are no self-reference edges and that edges are undirected. From these formulas, we conclude that the connectivity of the graph increases with p, as we can visualize in Figure 1.1.

$$V = n \tag{1.1}$$

$$\mathbf{E}(E) = \binom{n}{2} \times p = \frac{n^2 - n}{2} \times p \tag{1.2}$$

[1] https://github.com/LESSSE/RM_UC/tree/main/assignment%201

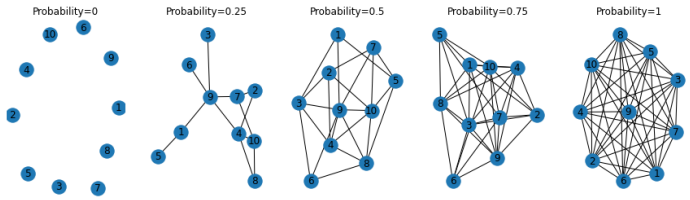

Figure 1.1: Graphical representations for the data generated by gen.py for $n = 10$ exams and various values for p.

2 Exploratory Data Analysis

Since we're starting studying this problem and still have so little knowledge on the data, in order to come up with hypotheses we need some initial observations. To characterize the performance and get aquainted with the data, we resort to Exploratory Data Analysis (EDA) to find relevant patterns, test initial guesses and detect anomalies and outliers. EDA usually consists of visualizing data employing several different graphical techniques, and extracting and extrapolating conclusions from them, with different graphic designs serving different purposes.

2.1 Data Generation

In order to gather enough data for our task of EDA, we ran a batch of experiments using different pairs of inputs for n and p. We ranged n from 5 up to 18 and p from 0 up to 1 with an increment of 0,01. We also ran each pair of inputs for 6 different values of seed 1 and 6 different values of seed 2, which resulted in a total of 36 samples for each pair of values. In total we produced 101808 samples which correspond to 36 samples for $14 * 101 = 1414$ different pairs of parameters for each one of the programs.

It is expected to observe that seed 1 changes the structure of the graph whereas seed 2 changes the order by which nodes are visited during the scheduling process. We decided that changing both seeds in a structured way would allow us enough variability and allow better reproducibility. If the random seeds were kept static, we should observe the same results, however, the execution time could still vary. We expect to see this variation of the runtime with static random seeds due to the CPU Workloads that are unpredictable. Under heavier CPU loads this could lead to several CPU context-switching operations, leading to a decrease of performance. As a future direction, it might be interesting to study this, to understand the impact of the CPU workloads on the runtime of our application. We ended up ranging our seed 1 from 1325 up to 1330 and seed 2 from 31235 up to 31240. This range should not be relevant for the problem because the pseudo-random number generator is using an hashing function behind-the-scenes.

It is also important to notice that we applied a 300 seconds threshold (cutoff time). Any execution that reached this runtime was forced to finish without ever reaching a solution. These results were discarded within our experiment.

2.2 Data Analysis

In Table 2.1 we can observe that the mean execution time for Code 1 is almost the same (slightly less) than the mean execution time of Code 2. The execution times within the 50% and 75% percentiles for both t_1 and t_2 are very low. However, the maximum value reaches 300 seconds, the mean execution times are way above this percentiles and the standard deviation is also considerable. From this information we can infer that that we are expecting a sudden large growth or maybe some clear outliers.

From the correlation heatmap in Figure 2.1 we can see that both seeds are not correlated with runtime as we assumed. The probability percentages and the number of exams are not correlated too, which indicates that we extensively explored different pairs of parameters in a balanced way. Both p and n are positively correlated with runtimes which demonstrates that increasing any of these parameters will result in an average increase of the runtime. The runtimes are perfectly correlated which indicates that we can strongly predict the the execution time of one code just by running the other.

Observing the solution histogram in Figure 2.2, we can understand that there are differences in the output from codes 1 and 2 when the solution is 1. Having knowledge that all the solution for all problems with $p = 0$ is 1 we may raise the hypothesis that Code 2 wrongly outputs 2 for those problems, which simultaneously explains the two most visible discrepancies between solutions in this diagram. We can also see that both distributions are skewed to the right and show a very long tail. One exception is the spike we see at higher runtime values (near

	Runtime Code 1 - t_1 [s]	**Runtime Code 2** - t_2 [s]
Mean	5.917431	5.994136
Std	39.394401	39.624103
Min	0.000004	0.000003
25%	0.000023	0.000008
50%	0.000077	0.000034
75%	0.000656	0.000670
Max	300.000004	300.000013

Table 2.1: Descriptive measures

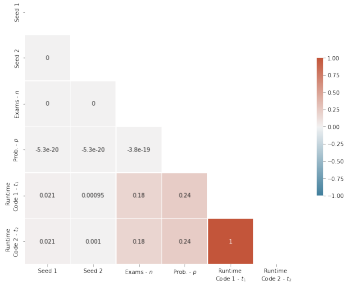

Figure 2.1: Correlation heatmap

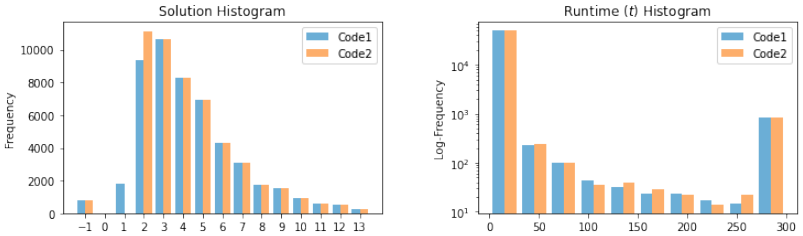

Figure 2.2: Solution and runtimes histograms

300 seconds) which we believe are caused by the cutoff condition we used for data acquisition. In sum, besides small differences, both distributions of solutions and runtime seem very similar for Code 1 and Code 2.

In Figure 2.3 we show the box plots for 25 different pairs of values for p and n. These show, for each pair of values, the median, both the 25^{th} and 75^{th} percentiles, min and max as well as the points that were considered outliers for the runtimes for both Code 1 and Code 2. The outlier classification took into consideration the interquartile range which is defined by the difference between the 75^{th} and the 25^{th} percentile. Points that were further than 1.5 times the interquartile range (IQR) from the median were classified as outliers. This outlier classification process was only used for this graph and the samples were not removed from our data. Thus any further graph still takes into consideration these points.

We can detect several patterns from these boxplots. For probabilities p close to 0 we can see that Code 2 usually performs better, whereas when p is really close to 1 is preferable to use code 1 because it achieves slightly smaller runtimes. We can also see that for non-extreme values of p, higher values of n mean more and wider outliers. These outliers disturbed our graphs and made harder to take conclusions on the behaviour of the IQR for growing n. In the last two graphs we see again that cutoff mostly impacts samples for large values of n and p completely masking any true variation of the data.

In Figure 2.4 we present a 3D representation of both mean value and standard deviation of runtimes for both codes, for every pair of values that was collected. It seems that both runtime means and dispersion grow in a log-scale both with n and p. The linear growth is particular perceptible for $p = 1$ and with increasing n. For large values of p and n we see a plateau that we once again assume to be caused by the imposed cutoff. We noticed a particularly interesting phenomenon. We observe what we consider to be a strange increase in deviation for small values of p in a shape of a wave, which seems to grow with n. Maybe this means that it might be worth to study the behaviour of variation for small values of p with increasing values of n.

In Figure 2.5, we show the variation curves of the median of runtimes when varying n and p. We also represent the interquartile range (portion where you can find 50% of our samples). From this graph we see that runtimes seem to grow linearly with n and p on the logarithmic scale, although in the latter case it is not as evident.

Finally, in Figure 2.6, we show a Quantile-Quantile plot that allow us to compare 2 distribtuions. We see that for a vast majority both distributions seem very similar. Although, for very small runtimes usually code 2 may perform better as indicate by the deviation we see in log-runtime. In the other hand, for large runtimes we get slightly better results with Code 1 and this difference seems to grow for even greater runtimes.

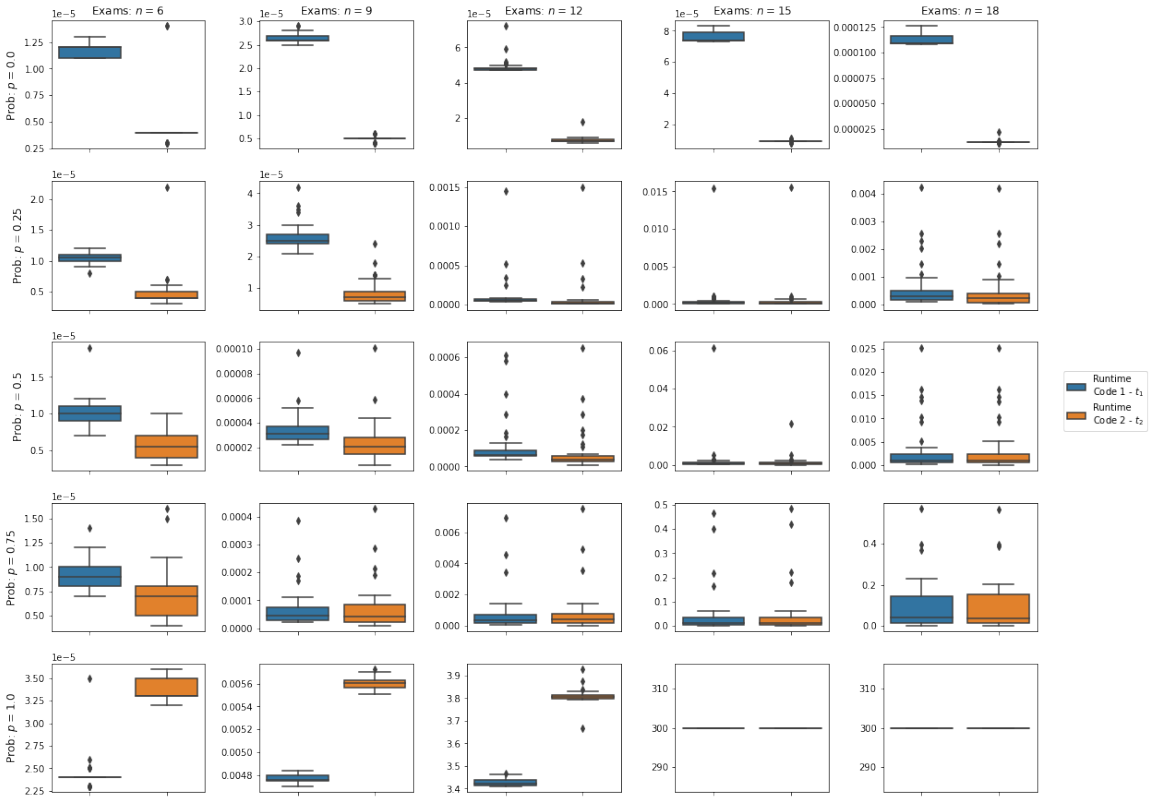

Figure 2.3: Graphical representations for distribution of runtimes for several pairs of values

3 Linear Regression

Another useful task in experimental design is to perform Linear Regression on the data since it allows us to predict previously untested behaviours. This assumes a linear model, which might not always be the case. Still, performing Linear Regression over a dataset provides us a simple and quick model that possibly describes how our system behaves for different input values.

From Section 2 follows that the mean runtime for either Code 1 or Code 2 to solve the problem grows exponentially with either the number of exams n or the probability p. Furthermore, from Figure 2.5, we can see that the runtime for either code approximately follows a linear tendency when we apply a logarithmic transformation to the data. This is true for both independent variables, even if it is not as apparent on the latter case.

To verify this, we tried to fit a linear model to each of the different variations of Figure 2.5, including the ones not shown, and calculated the Coefficient of Determination R^2:

$$R^2 = 1 - \frac{RSS}{TSS} \tag{3.1}$$

where RSS and TSS represent the sum of squares of residuals and the total sum of squares, respectively. R^2 gives us an idea of how good a model fits to the data, being that the closer it is to 1, the fitter the model is to the data is and vice versa.

In Figure 3.1 the different R^2 values for the different possible parameter variations are plotted. The two scatter lines indicate whether a logarithmic transformation was applied to the data or not. We can see that R^2 greatly improves for the transformed data, which suggests that the logarithmic transformed data is much closer to a linear model than the raw data, as expected. Not only that, but the R^2 values are very close to 1, especially when we increase either n or p. We should also point out that the linear model seems to fit better when we vary n, that is, when p is fixed for some value, R^2 reaches higher values than when n is fixed and we vary p, though the opposite is also true, even if to a lesser extent. Also, and especially when we fix a certain n but in general as well, the data seems to better fit a linear model as we increase its magnitude. Increasing n even further would prove

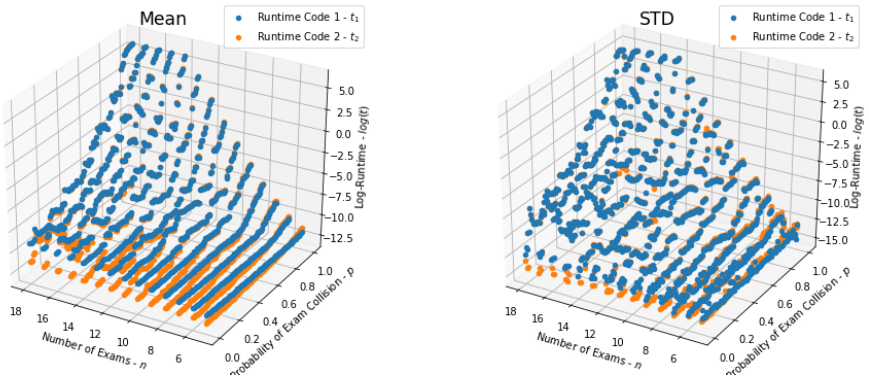

Figure 2.4: 3D representation of variation of mean of log-time of execution for different values for n and p.

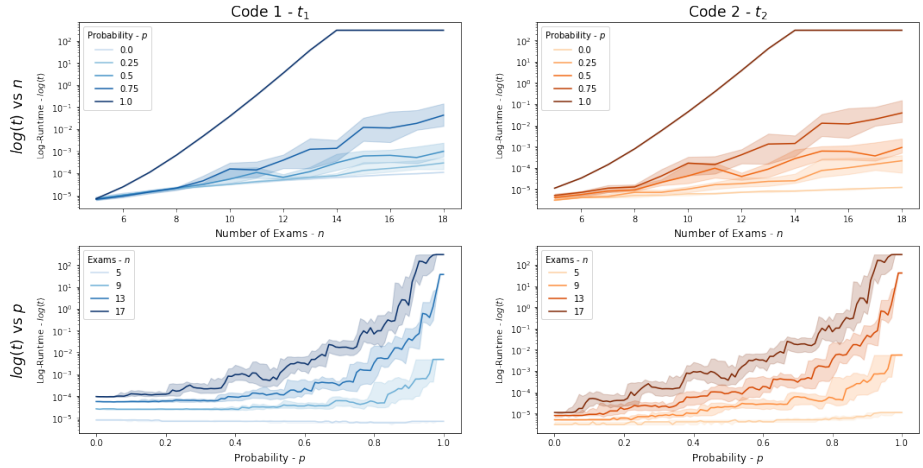

Figure 2.5: Median evolution while ranging the number of exams n (upper row) and probability of collision p (bottom row) and interquartile region.

useful for continued study on this. Lastly, it is interesting to notice that for Code 1 and for lower n, the linear model fails to fit the data well, which might indicate Code 2 performs better for lower n.

Finally, both Figures 3.2 and 3.3 show 6 different linear models fitted. We see again that we got greater values for R^2 when we range the number of exams n, and we fix p. We see again that extreme values of p are approximated better by using a linear model than middle ranged values of p. For these medium values of p we also can clearly see the non-homoscedastic nature of the data, i. e., we see greater dispersion for greater values of n. We also can question if linear models are the best way to model the impact that p has on t even after applying the log transformation. Maybe another kind of transformation or another kind of model such as a quadratic model could be testes in the future. Another possible future study could be to study the impact of the number of edges E in the graph instead of just the probability p. We can see also notice that for very small number of exams n, Code 1's runtime t_1 follows no visible linear pattern achieving poor results for R^2 and we predict that it results in a correlation coefficient with p closer to 0, while Code 2 grows with p for small n achieving greater R^2 and maintaining the positive values in its correlation coefficient.

4 Conclusion

After the experiments made, we can conclude that there isn't a clear efficiency increase by either using Code 1 or Code 2. Both have very similar execution times, however, Code 2 seems to perform better for lower value of p and Code 1 seems to perform better for higher values of p. As the number of exams and p increases, there is an exponential growth of the execution time for both code versions, ending up reaching the asymptotic value of the cutoff time. Since there is an exponential growth when it comes to the execution times, we applied a logarithmic

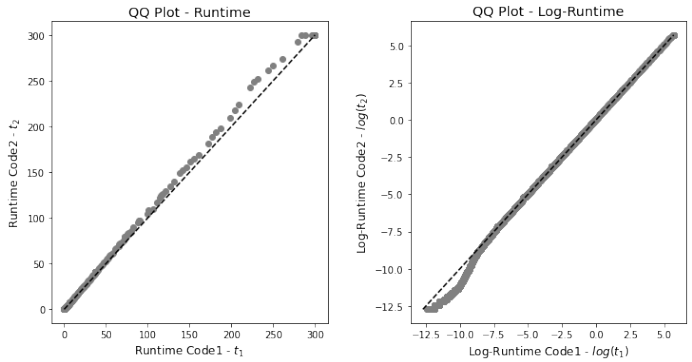

Figure 2.6: Quantile-Quantile plots between the two codes for both raw data (right) and log transformed data (left).

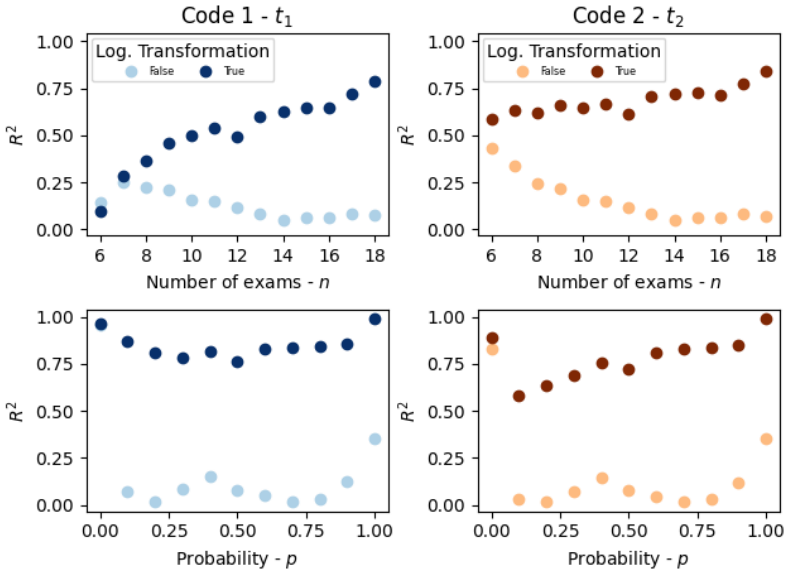

Figure 3.1: R^2 evolution for different n and p.

transformation in order linearize the data and improve the accuracy of the model. However, other models and possibly other transformations can also be tested to verify if model fitness improves. Nonetheless, for several combinations of p and n, the model fits well the data and has a high coefficient of determination R^2.

REFERENCES

[1] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. 7(6):441–499, November 2004.

[2] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

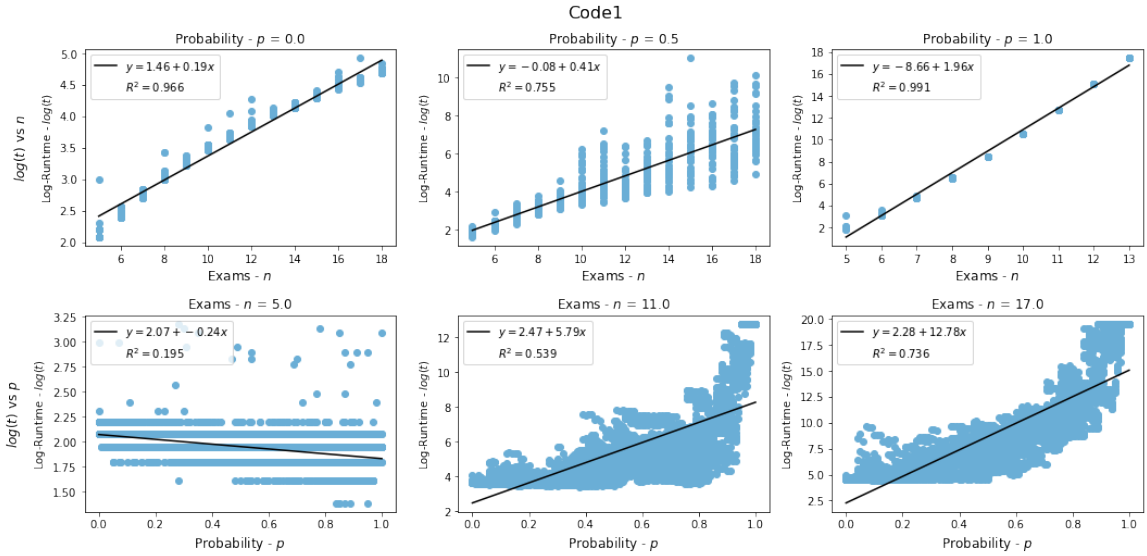

Figure 3.2: Linear model fitted to Runtime Code 1 t_1 while ranging the number of exams - n (upper row) and probability of collision - p (bottom row).

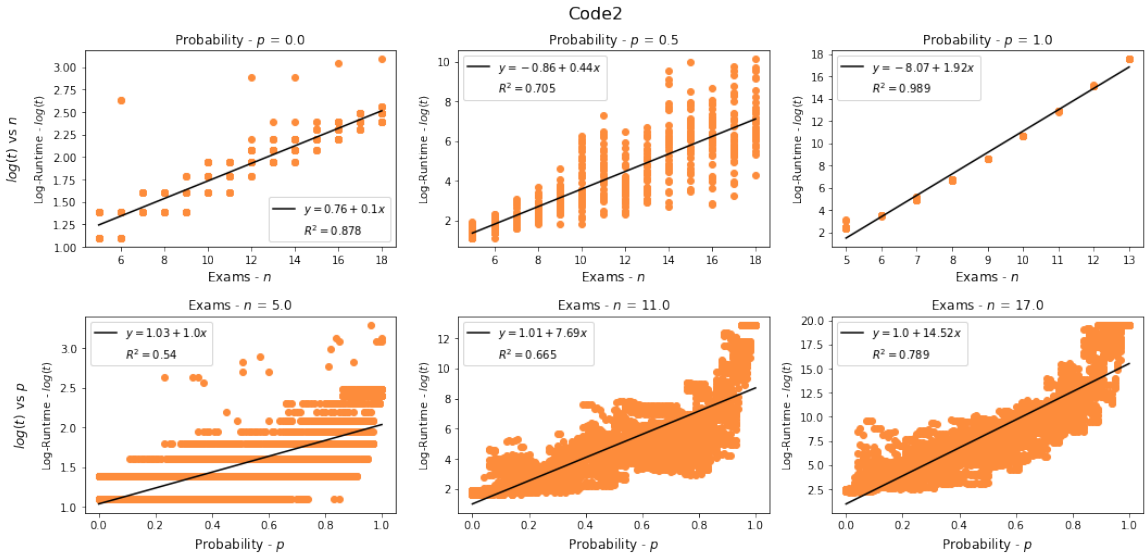

Figure 3.3: Linear model fitted to Runtime Code 2 t_2 while ranging the number of exams - n (upper row) and probability of collision - p (bottom row).