Project Manual Bachelor Year 1 Project 1.1 Computing - Tabulating - Recording and a bit of (Machine) Learning[*]

Pietro Bonizzi, Martijn Boussé, Philippe Dreesen,
Kurt Driessens, Evgueni Smirnov

Period 1.1, 1.2, and 1.3
Academic year 2023-2024
Data Science and Artificial Intelligence
Faculty of Science and Engineering
Maastricht University

Courses:

Introduction to Data Science and Artificial Intelligence (KEN1110)
Procedural Programming (KEN1120)
Discrete Mathematics (KEN1130)
Objects in Programming (KEN1220)
Calculus (KEN1440)
Logic (KEN1530)

[*]If you want to fully understand the title choice, look into the early days of IBM.

Welcome to project 1-1, you will love it and you will hate it[1].

Introduction

The goal of this project is for you to:

- Practise your Data Science and AI skills;
- Get experience with Scientific Research, Critical Thinking and Problem Solving;
- Learn to work in a project team.

These aren't trivial to learn, so we've come up with a plan: the plan is that you will learn all of these skills while you are analysing a database of student grades.

DACS has obtained what we believe to be the first gravitational wave-carried data communication ever discovered. It is believed that a very advanced alien society has used the collision of two massive black holes to send us their student grades. It is hypothesised that this data might contain hints on how to improve our education even further, so of course, we are very interested in a thorough analysis of these grades. However, given the current accuracy of the world's best gravitational wave detectors, there is also a hypothesis that this data represents random noise instead, so we need your help to find out the truth.

Overview

During this project, you will go through a number of data-analysis steps, ranging from learning to load data and compute statistics about the data, over visualisation of data, all the way to your first bit of machine learning.

Phase 1: You will learn to load data into memory and learn to handle and manipulate this data into separate sets and compute statistics on these sets. This will help you better understand concepts like data, information, and knowledge, and the differences among them, and start practising by looking at properties of a data set, and looking for patterns in data, as you have seen in *Introduction to Data Science and AI.* Also, this will make you practise with the most important concepts from *Procedural Programming*, handling multi-dimensional arrays in a number of nested loops. It will also reference topics such as machine learning and decision trees.

Phase 2: You will develop a user interface for simple data visualisation. Good data visualisations can be very useful and powerful tools, both during the data exploration phase and when presenting results to stakeholders concisely and effectively. It will also help you practise the new skills you will learn in the course *Objects in Programming.*

Phase 3: In the last phase, you will bring together parts you have built in the previous two phases, and you will develop a regression tree algorithm and use it to construct a state-of-the-art machine learning method called Random Forests that will allow you to produce fairly accurate predictions about unobserved data.

Phase 1: First Things First[2]

During the first phase of this project, you will learn

- how data can be loaded from a file,

[1] Not necessarily in equal amounts.

[2] "But not necessarily in that order." - Doctor Who

- how to clean data and learn its high-level properties, and
- how to partition data and find information on the different partitions.

We've attempted to provide you with a rather detailed outline of tasks that you can use to organise yourselves during this first week.[3] Each step described below roughly coincides with a day's work (for starting programmers).

Step 1: Loading and Studying Simple Data

While the most usual place for data to reside is inside some kind of database, and while there exist many ways to store data into computer files, we will start simple in this project. And one of the most simple yet convenient ways to share structured data across platforms is through a so-called *CSV-file*, in which *CSV* stands for *Comma-Separated-Values.*

For this project, when we talk about *structured data*, we will mean data that can be represented as a matrix of values where each row represents a single **observation** that consists of a number of measurements and each column represents such a measurement, namely a property or **attribute** that is measured for each observation. An example of a set of observations is given in Table 1, where each observation represents a state of the weather and the properties measured are Temperature, Humidity and Windspeed.

We refer to the different properties that are measured for each observation as attributes (sometimes also as **features**) and to the actual measurements as **values**. As a consequence, this type of data representation is known as an *attribute-value representation* of the data. As you might have already derived from the data, the attribute-values shown have different types, i.e., one is *real*, two are *ranked* (meaning ordered) and one is *integer*, for temperature, humidity and outlook, and windspeed respectively. There is a another common type of data, namely *nominal*, that represents measurements for which the values do not have an order. For example, color is a nominal attribute that can take different values such as blue, yellow, magenta or periwinkle, but these values can not be ordered.[4]

We will start simple, with a clean data set. In this case, "*clean*" refers to the facts that there is no missing data, no data that doesn't align with the expected type, etc.

- Use the provided code in the file `FileDisplayer.java` to read and display the file `GraduateGrades.csv`. The provided code will print out the data, which you will see is a collection of rows and columns. As is standard, each row represents a set of measurements taken together, each column represents an attribute that is part of the measurement. In this case, each row represents a graduate of the programme and every column represents a course. Each entry represents the grade of the corresponding graduate for the corresponding course.

[3]As you will notice along the way, gradually task descriptions will get less and less detailed, and you will need to develop the necessary self-directedness. This is a feature, not a bug, and one of the four foundations of Problem-Based Learning.

[4]Yes, we specifically added magenta and periwinkle for those of you who would argue for ordering them according to wave-length.

Table 1: Some example data

Temperature	Humidity	Outlook	Windspeed
23.2	High	Sunny	10
16.6	Low	Rainy	5
21.8	Medium	Rainy	12
5.3	Low	Overcast	25
12.0	Medium	Sunny	0

- Adapt the given code to read the data into a two-dimensional array and start writing code that can tell you things such as:
    1. Which courses are the most difficult/easy?
    2. Which students graduated cum-laude?
    3. Are there courses that seem similar or related? (We know this is a very vague question, but again, this is by design. You will have to define the kind or type of similarity and relatedness you are checking yourselves.)

Don't limit yourselves to just the questions above. Answering some of these questions might require you to compute some statistics about courses or students. One example is computing a course mean grade, but there are more interesting values to compute. Have a Google for some inspiration if required. You can also split yourselves into smaller groups and tackle a different question with each subgroup.

Figure out what you can learn about the courses, the graduates and the study programme from this data. To pass, you will need to answer the three questions above and come up with (at least one) additional good questions and write the code to answer them.

We continue looking at the grades data by having a look at the grades for current students. When you try to load the `CurrentGrades.csv` data with the same code as you used to load the `GraduateGrades.csv` file, you will probably notice that things don't go as expected. If you open the file in a different way, such as through the `FileDisplayer.java` code, you will notice why. Since this data is about current students, they do not yet have grades for all courses.

- Adapt your code so that it loads the data of the current students. Find a way to deal with the missing values in your code.
- Compute some of the statistics you calculated before on the graduates data again, and see if you notice any differences and whether you can explain these. This will of course require you to deal with the missing values correctly.

Step 2: Finding and Completing Hidden Information

Although one might consider the "`No Grade (NG)`" entries as annoying and as missing data, there are actually reasons why this data is missing, and these missing values can give us useful information about the study programme and how it is run and about the students and where they are in their education.

- Paying attention to the missing values, what can you discover about the study programme?
    1. Is there an order to courses in which they are offered to students for example?
    2. Which students are going to graduate soon?

Again, don't limit yourselves to the questions asked above, but see what you can discover about the student population and the rules and regulations of this programme. Answer the questions above and come up with (at least one) additional questions and write code to answer those questions. *(Hint: Think about the rules and regulations in your own programme and see if something similar can be observed in the data here.)*

We can now think about some predictions we want to make. For example, imagine the Student Affairs Office wants to start planning a graduation ceremony, but needs to know the number of students that is expected to graduate to book the right venue.

- Come up with your best guess for the number of students you expect to graduate this year. Maybe the questions below will help you come up with a strategy:
  1. How many students are eligible to graduate this year?
  2. Is it possible to predict passing percentages for the open courses?
  3. Is there useful information in the graduate grades for this?

Make sure you can motivate the way you approached this and explain how you arrived at the given estimate. For example, be explicit about the assumptions you are making.

Step 3: Tabulating and Finding the Best Tabulation

After trying to predict the total number of expected graduates for this year, we can dig a little deeper and try to see if there is anything else we can predict. For example, students might be wondering how they will perform in upcoming courses, so you could ask yourself, would it be possible to predict how they will do in the upcoming courses.

This question can be addressed using different levels of detail available in the data. While you could simply predict the average score for a course (that you obviously calculated on day one), this treats all students the same, and, as you should know by now, not all students are.[5]

Instead of just predicting a student to perform average, you can make use of additional information to make better predictions. The file `StudentInfo.csv` contains, to nobody's surprise, additional information on the students currently in the programme. While the feature names and their values might be hard to interpret, one can check for example the difference in average score for a course between students with different values for one (or more) of the properties listed.[6] For example, one could compute the difference in the average score for `ATE-007` comparing students with a high and a low `Hurni level`?

- Write a method that checks the difference in scores reached in a given course by students with a given property value. For this, you will need to be able to specify the course as an input to the method, but also a way to define how to separate the students into different groups, e.g., by specifying a property name and a *selection* or *boundary value* to apply. You can compare not only average scores, but also the difference in variation between the values.
- Besides the student information from the extra file, do the same using the scores obtained on previously passed courses.

Now that you can check the difference in scores between student subgroups according to a number of different properties, you can try finding the most informative property for guessing the score of a specific student for a specific course. This will result in a rule-like model, which predicts the expected grade of a student as: `if X then grade Y, else grade Z`, where `X` is a student-property and `Y` and `Z` are the mean scores of student with and without property `X`, respectively.

- Write code that for a given course, finds the best property to help guess the grade a student will get for that course. In this case, you can define best according to a measure called *variance reduction*.

[5]You are all individuals, just like everyone else. (Well ... except for you, the one reading this right now.)

[6]This splitting of data into different categories and counting how different categories differ in other properties is how IBM got started, building mechanical machines that would separate a stack of cards representing data into multiple stacks and reporting the number of cards in each stack. This was called "*tabulation*". This intermezzo is just here to show you how even these simple tasks can signal the start to a very big fortune.

- Then write code that uses the previous method to produced the best single guess for the future performance of a student.

This type of simple prediction model, where one splits the data into two subgroups to generate the best prediction is called a "*Decision Stump*". It's simple, but can be relatively effective.

Step 4: Combining Predictions: A Forest of Stumps

Building a model and asking it for a prediction on data is like calling up an expert and asking for their opinion. The great thing is, you can repeat this multiple times. So, instead of finding simply a single way to split the data and then guessing the after split mean as the prediction, one can take the, for example, 10 best ways to split the data and combine the predictions to get a better prediction. The idea behind this is that, when the prediction errors made are uncorrelated, they will correct each other when combined.

- Program a method that finds the best set of decision stumps. In this case, it is up to you to decide what makes up "the best" subset of splits. However, it is important that you are clear about your selected criterion and that you can motivate the choices made. (For a higher grade, you could come up with some alternatives and test them against each other.)

Now that you have a set of decision stumps, each predicting their own best guess for the student's grade, these predictions need to be combined into a single "*best*" prediction. Although the way to combine them might seem obvious, there might be alternatives that are less trivial but perform better.

- Implement (and again, for a higher grade, compare a number of ways) how to combine the selected stumps into a "*best*" prediction.

(Step 4 gives you some leeway in how to solve things. E.g., the text mentions "best" a number of times, and in different contexts. Comparing performance in a satisfactory way requires some form of performance measure definition. This assignment intentionally leaves this open for you to research and select. However, you need to be very clear about the performance measure you select and use.

Reporting on Findings

On the last day of the week, you will have to be ready to report on your findings about the data. Your grade will be determined by the variety, correctness and informativeness of your data analysis and predictions, and by how well you can motivate your conclusions. The code must be your own: do not use third-party software and do not cut and paste code found on the internet. We will use plagiarism tools to check this and we will ask you to provide an overview of who did what.

In the demonstration, you will be asked to present your insights into the data, together with the techniques that allowed you to reach these insights. Your demonstration should include a short PowerPoint-like presentation in which you summarise your findings. You will also be asked to show your predictions for both the number of expected graduates as well as the expected performance of a number of students on courses they didn't complete yet and you will be asked to explain how these predictions are made. Examiners might ask you to make a prediction for a specific student on a specific course, so make sure you have code ready to do this and although you should not show code in your presentation[7], they might ask you to show and explain a specific part of your code. You should also know

[7] You can include a flow chart or some simple pseudo-code when useful.

that we regard the ownership of the code as a group thing, so be prepared to answer also questions about the code written by your fellow team members.

To pass Phase 1 of the project, you need to at least complete steps 1 through 3. Groups that complete these three steps can expect a grade between 6 and 8, where the depth of your analysis, the quality of your investigation and the clarity of your motivations and decisions will determine where between those values your grade will lie. To get a higher grade, you will need to complete step 4, and also here, your precise grade will be determined by the quality of your work.

Phase 2: Visualising is Knowing

In Phase 2, you will build a visualisation tool that allows you to illustrate the findings you made in Phase 1. For this, you are allowed to use the `Swing`, `AWT` or `JavaFX` libraries only.

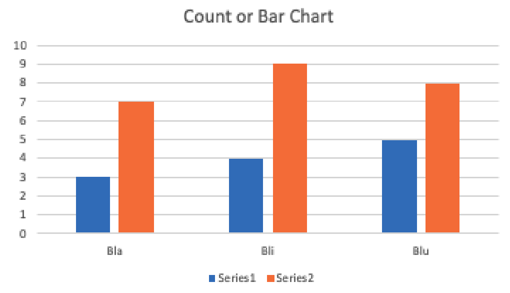
As we mentioned earlier, data visualisation is a very powerful way to explore your data and extract insights about the data structure and properties, and to present the results of complex analyses to stakeholders in a simple, effective, and impactful way. This phase also links to skills you will learn in (in particular) *Objects in Programming*. During Phase 2, you will need to structure and plan your group's work by yourselves more than in Phase 1, but of course, the project tutors are there to help you with project coordination and you can contact your course coordinators for questions related to their course content. Early in the second teaching block, you will be asked to submit a plan for Phase 2.

You will implement a number of data visualisations (of which a number of examples are given below), and you will make it so that the user of the system can select and compare the data for at least two separate subsets of data. Any visualisation should allow the user to select the variable they want to visualise (e.g., the grades for a certain course) together with a (set of) filter(s) that allows the user to select which subset of data to visualise (e.g., all students who scored higher than an 8 on a different course, or all students with a lower than medium Hurni Level. This should allow the user to, for example, confirm the optimal split for grade prediction suggested by your decision stump algorithm from Phase 1, but also to check the other properties or patterns that you discovered during Phase 1.

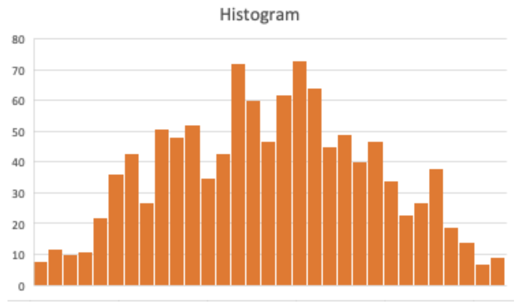
In fact, your goal during this Phase 2 is to find and implement visualisations for all of your findings from Phase 1, from average grades for courses and the associated variance, over correlations between course scores, to anything you might have discovered about the rules and regulations of this study programme.

Visualisations to include, but not limit yourself to.

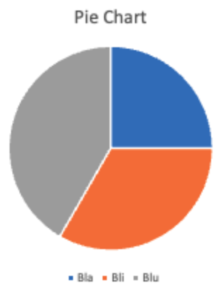
Bar Graphs One of the simplest data visualisations around. They usually represent a simple count.



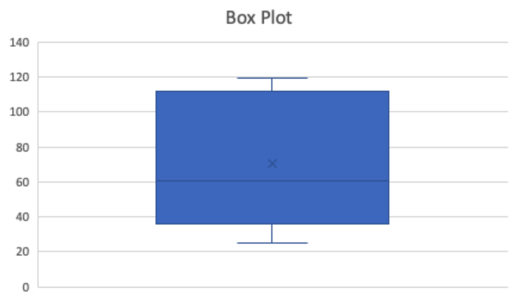
Histograms Very related, and in their most basic form actually equal, to Bar Graphs, they can be used to compare data distributions.



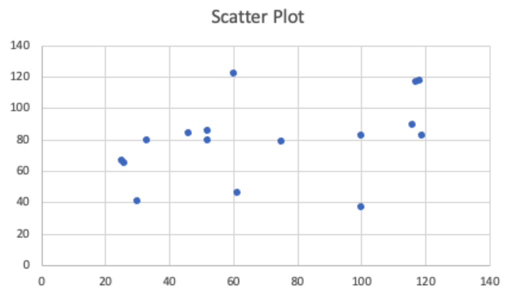
Pie Charts Everyone's favourite chart, because, if you select your data and settings correctly, you can make it look like PacMan. Draw multiple pies for comparisons.



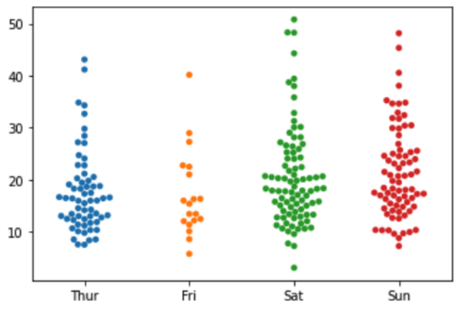
Box Plot Also meant to show data distributions; very compact, but contain a lot of information, so really useful for comparing different populations.



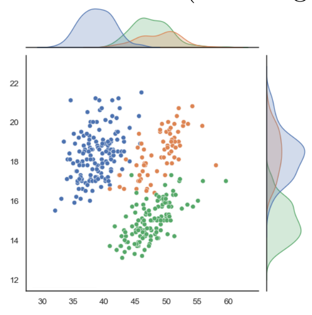
Scatter Plot These can, for example, be used to illustrate correlation between two variables.



Swarm Plot First of the fancy ones, a swarm plot is a more illustrative version of the box plot above.



Joint Plot And second of the fancy ones, a joint plot is a combination of a scatter plot and visualisations of the marginal distributions (or histograms).



But of course you can add others if you think they are useful and help illustrate one of your findings of Phase 1 better. By the way, there is something seriously wrong with all the plots shown above: None of the axis are labelled. Make sure you don't get caught making the same mistake.

To pass Phase 2, in your visualisation interface, it should be possible for the user to:

1. select the data they want to visualise (This will often be the course(s) for which to show grades.),
2. choose the filter(s) they want to apply to split the data, (This might be a student property, or grade boundary for a different course, etc.),
3. select the type of plot they want to see.

This type of visualisation tool is often part of something called a data dashboard in which a number of visualisations can be displayed.

Search around for some examples and be prepared to show off your dashboard during the demonstration at the end of this phase. Also come prepared to explain which visualisations you chose for what purpose, and how you approached generating the visualisation. Go back to the analysis you performed in Phase 1, and use your visualisations to support the claims you made then. You could even think about making some further claims based on the visualisations from your tool. Again, you will need to report who worked on what and examiners can and will ask you to demo the visualisation of certain data, to interpret the resulting image and to explain (parts of) the code that you built.

At the end of Phase 2, you are also expected to submit a draft of the report you will submit at the end of Phase 3. Of course, there is a lot of material still going to be developed in Phase 3, but you should aim at filling this draft report as best you can to save yourselves a lot of work in what is going to be a very busy Phase 3.

Phase 3: That bit of (Machine) Learning

In Phase 3 you will work full time on the project. The assignment for this phase, as a consequence, is a bit more extended and quite a bit more complicated than the assignments of Phase 1 and Phase 2. You will need to focus and organise yourselves well to deal with the complexities involved. Try not to dilly-dally but come up with a strategy to get you to the goals described below.

Recursion and Trees

During Phase 1, you built decision stumps, i.e., prediction models that split the data into two sets with a difference in population mean for the target variable. When you think about it, there is nothing holding one back from splitting the two sub-populations again, to get an even narrower target value distribution in each of the subsets. This recursive splitting of the population, according to the most suitable feature, leads to a very popular machine learning model called a *Decision Tree*. One can have classification trees that predict a nominal target value, or, as is the case in this project, you can have regression trees that predict a (quasi) continuous value.

Figure 1 shows an example of decision tree based on the data in Table 1. The blue oblong boxes are called "*nodes*" while the orange ovals are called "*leafs*". Each data point can be attributed to a leaf of the tree, by following the path as indicated by the tests in the nodes. In this case, the tree is a binary tree as each test results in either true or false. Another option can be to list all the possible values of a feature as branches in a node, but for this project we will only consider binary trees.[8] For this tree and the data used to construct it, you can see that there are three leaves, each with one or two data points in them. The tree can be used to make predictions about temperature. E.g., for situations where the `Outlook` $\neq$ `Overcast` and `Windspeed` < 10, the best guess for `Temperature` could be the average between 16.6 and 12.0, at least according to the data in Table 1 and under the assumption that a decision tree is the right way to model this temperature dependence.[9]

For this final phase, your first task is to:

[8]Students that want to go for non-binary trees should feel free to do so and, if successful, can expect this to impact their grade positively.

[9]This assumption about trusting the model being used is a necessary step in Machine Learning and referred to as the "*learning bias*". While being *biased* is often judged and regarded as a negative thing in daily life, there is no way of generalising, and consequentially of making predictions about unseen cases, without some form of bias that allows a model to treat a new, never before seen, case as similar to the data it was trained on.

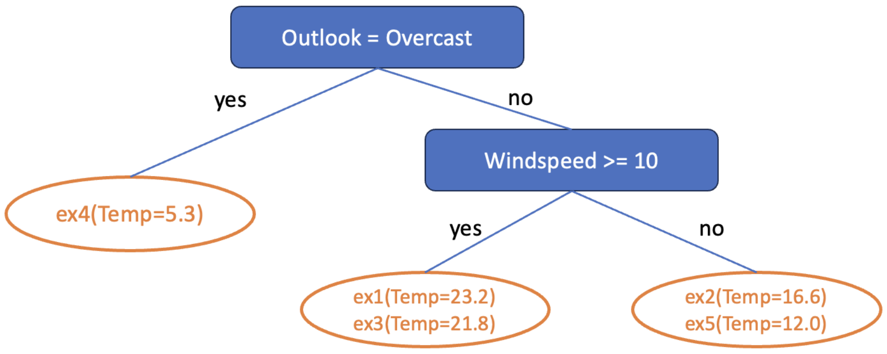


Figure 1: In the decision tree that makes predictions about temperature, every data point can be attributed to one of the leafs (orange ovals) by following the tests in the nodes (blue oblong boxes).

- Construct a regression tree algorithm by recursively calling a splitting method on the data subset that belongs to a leaf in the tree.

This will require your algorithm to keep track of which data points belong to which part of the tree and store the tree structure.

To be able to build this algorithm, you will need to store the structure of the tree somehow. While you haven't had the course on *Data Structures and Algorithms* yet, we are confident that you will be able to come up with something. While it might sound scary, a recursive data structure is probably the best for trees. You can imagine having a general data type *Tree* which can be either an internal node or a leaf, where internal nodes store a test and the references to two sub-Trees and leafs store (references to?) data points and a mean target value (and possibly a variance of that target value population).

Another important question that you will need to answer is when the recursive algorithm should stop splitting the data that belongs to a leaf of the tree? A shallow tree will over-generalise, predicting the requested value based on only a few features, and thus calculating the mean of a large set of possibly still very different data-points. A very deep tree with many levels of branches might base its decision on only very few, or even a single data point thereby losing its generalisation power and *overfitting* the data. One hint that we can give you is to search for terms such as: "*validation set*" and "*post-pruning*".

For debugging reasons, it might be good to construct some kind of visualisation tool for (relatively shallow) decision trees. This can most easily be done on the command line, in text format, but for a higher grade, you can also think about making a more fancy visualisation tool to become part of the dashboard of Phase 2.

Finally, a full grown forest ...

Just like in Phase 1, it is possible to build multiple models and combine their predictions in the hope that they will correct each other's mistakes. As it did back then, the success of this depends on the fact that the different models should make uncorrelated mistakes. For decision trees, this can be done by injecting a measure of randomness into the decision tree building algorithm. This randomness can come for example from constructing randomly selected subsets of data to use for constructing the tree, or from disallowing the tree to use all possible features to split on and instead only provide each tree in the forest with a random selection of features to use when splitting internal tree nodes.

Your final implementation task for this project is to:

- Construct a random forest algorithm that uses the self-correcting power of multiple, different, regression trees to produce a powerful model to predict student grades.

If you have updated your dashboard with a tree visualisation, you can also consider building in a way to visualise a random forest for a very high grade. This will probably require interaction with the user as drawing all trees of the forest might quickly become overwhelming.

Almost done ...

To wrap up the project, you are asked to write a report describing the functionalities of your software as well as a report on its performance. You are asked to provide not only predictions of grades, but also to estimate and compare the accuracy of your models' predictions for each of the systems that you developed. You can of course use the visualisations from Phase 2 to help illustrate. Your report should also include who worked on what parts of the project. When building your report, please follow the guidelines provided by the project coordinator.

A day before your final presentations, you will be sent a list of student IDs for which to make predictions about all missing course grades. We will compare your predictions with those from other groups to see who's model makes the most accurate predictions.

The final presentations will be held in a conference-style setup, where 4-5 groups are joined in a session. This means you can also expect fellow students to ask questions about your approach and results. Examiners will ask thorough questions during the Product & Report Examination.