Assignment 4

Software structure

We have three folders: resources, solutions, and tests. Resources contains the raw project .xlsx files. Solutions contain excel files of the input projects after performing calculations using minimum, expected, maximum, and random durations from the triangular distribution for both projects 'Warehouse' and 'Villa'. The Tests folder contains tests for the different classes.

The software is divided three files:

1. project.py contains two classes: **Project** and **Task**. A Project contains many tasks and implements the algorithms for calculating early and late dates for the project. Task is pretty much a data class but with some functionality for calculating durations of task depending on whether you use minimum-, expected-, maximum-, or triangular duration.
2. utils.py loads a project from file, saves a project to file and has functionally for creating .png files of the project in graph format for visualization (similar to a chess tree in assignment 2). Note that creating the .png graphs requires graphviz, but we have already created the graphs for the Warehouse and Villa project (with gates as well) so you don't have to.
3. projectsimulator.py contains a **ProjectSimulator** class that takes a risk factor and a project. It has functionality for simulating n number of projects using the triangular distribution, classifying the projects, calculating statistics and performing machine learning. Arguably, it is better to divide the machine learning functions to its own file, but we let it stay there since the software is so simple.

Task 1-3: Run utils.py to check that projects and tasks are implemented correctly and printed in a reasonable way. See excel files in the 'solutions' folder for the results from both projects with different durations as basis for calculating dates.

NB: To handle the case where task J.1 is created before J.2 and J.3 despite J.1 having J.2 and J.3 as predecessors, we check that all predecessors exist before adding them to the project, and save it for later if the predecessors aren't created yet.

Machine learning

The results of task 4, 5 and 6 can be verified by running the respective methods in project_simulator.py. We use scikit learn to perform all machine learning tasks. We have chosen the following classification models: logistic regression, decision tree classifier, and support vector classifier (support vector machine). We have chosen the following regression models: linear regressor, decision tree regressor, and random forest regressor.

Task 4:
This task has been solved in project_simulator.py under the static function named task_4. The function simulateNProjects simulates n projects and returns a list containing the durations of all the simulations. This list can then be given to our calculateStatistics function which returns all the statistics asked for in the assignment. We used chatGPT to help us create a function that neatly presents this information in a table in the terminal. We thought utilizing chatGPT for this is justified as it's only a cosmetic function.

Task 5:
This task has been solved in project_simulator.py under the static function named task_5. To solve this task we had to create a function in the Project class that adds a gate. When the gate is added we remember to delete the gate's predecessors' successors with the gate. The same goes for the gate's successors.

After we have added a gate we run 250 simulations for each of the risk factors. All these results are then added to a list. After this we split the data for training and test set and run the classification method. Mark that even though we call the train_test_split 3 times, we do it with the same seed each time, so effectively we train and test all the methods on the same data.

We use accuracy to measure the performance of the models. This is calculated using:

$$Accuracy = \frac{Correct\ predictions}{All\ predictions\ made}$$

As we can see in table 1 the decision tree and SVM perform better the later the gate is placed in the system. This makes sense as the later the gate is placed the less variance we have to predict the final result. The result produced by the logistic regression performs opposite to what is expected. We have concluded that there must be an error in the way we have implemented the method, but can seem to find out where the mistake is.

Gate placement	Logistic regression	Decision tree	SVM
After D1	66.5%	57%	66.5%
After H2 and H3	47.5%	70.5%	75%
After Q1	44.8%	98%	92.3%

Table 1: Shows how the different classification models performed based on where the gate was placed. The numbers represent accuracy.

Task 6:

This task has been solved in project_simulator.py under the static function named task_5. After we have added a gate we run 250 simulations for each of the risk factors. All these results are then added to a list. After this we split the data for training and test set and run the regression method.

We have used mean squared error (MSE) for measuring the accuracy of our predictions. This is a typical tool used in machine learning for measuring the performance of regression models. The smaller the number the more accurate the prediction is. The formula is:

$$MSE = \frac{1}{n} \Sigma(y - y_i)^2$$

From table 2 we get results that match with our intuition. We see that when the gate is placed late in the project's execution, the model performs better. Also we can see that the decision tree regressor always performs the worst, and linear regression always performs best.

Gate placement	Linear regression	Decision tree	Random forest
After D1	480	1051	557
After H2 and H3	171	392	205
After Q1	1.63	3.95	1.90

Table 2: Shows how the different regression models performed based on where the gate was placed. The numbers represent mean squared error.