Machine Learning Course
Homework 3: Reinforcement learning

In this homework you will use Reinforcement Learning (RL) to solve a dynamic pricing problem. Dynamic pricing is a task of automatically adjusting the price for products or services in order to maximize income or other economic performance indicators. The decision about price change is made on the basis of market state, which could contain competitor's price, our previous price, time period and other exogenous factors.

In practise, to tackle this problem one requires sales data. For the sake of simplicity, in this homework you have to simulate sales data on your own.
The general framework for applying RL to dynamic pricing problem is the following:

1. Build forecasting model, which would predict sales based on market conditions - price, previous prices and other exogenous factors.
2. Build an environment simulator in which actions are prices; states are market conditions without current price; rewards - economic performance indicators, which are basically functions of sales.
3. Train an RL agent in a simulated environment.

Here is a decomposed sequence of stages, which you have to implement:

1. **Simulate sales**
   We'll model sales as a sum of a linear demand function dependent on price, a highly seasonal component dependent on time with a one-year period; a noise term. We'll use google trends data for a seasonal component. csv files with searches in Google are located in the *data* folder. Choose one file according to the remainder of the division of your student's number modulo 7.
   data_generation.ipynb - notebook for sales simulation.

$$sales(t, p) = k_o + k_1 p + s(t) + \epsilon$$

2. **Build forecasting model**
   In *sales_predictor.py* there is a defined class for predicting sales based on time and price. You should implement two methods of this class and use it to build a forecasting model.
   sales_prediction.ipynb - notebook for building forecasting model.
3. **Train agent on a simple environment**
   In *simple_market.py* script there is an implementation of the environment that uses our forecasting model. In this case a state is a number of a week (modulo 52 because of a one-year period) and an action is a price. The reward used in an environment is an income.

$$reward(t, p) = sales(t, p) \cdot (p - prime\_cost)$$

   In *simple_market.py* you have to implement tabular Q-learning agent and in *simple_agent_training.ipynb* you have to train an agent and compare agent's policy with greedy 1-step policy. The greedy policy is a policy that uses a price, that maximizes income in the current time step. It can be found analytically (note that time is a constant in this case).

4. **Train agent in a complex environment.**
   In *complex_market.py* there is an implementation of a more complex environment that uses previous price to calculate the sales. Thus it can better model the real market.

$$sales(p_t, p_{t-1}, t) = (s(t) + k_0 + k_1 p_t) \cdot tanh(k \cdot (s(t-1) + k_0 + k_1 p_{t-1}))$$

   Note that you shouldn't build another forecasting model. For simplicity, the updated environment just uses the previously trained model “twice”. Also, note that a state contains time and previous price, so that Markov property is satisfied.
   In *complex_market.py* you have to implement tabular Q-learning agent and in *complex_agent_training.ipynb* you have to train an agent and compare agent’s policy with greedy 1-step policy.