Contents

Exercises for Week 12: Reusable Software

Grading Scheme:

Important: This Jupyter notebook needs to be completed and submitted via Blackboard before the due date to receive a non-zero grade.

- 5: Every question is completed and the solution is essentially correct. This means that the math formulation correctly models the given problem and the code outputs the correct results. However, it is okay if the code returns another optimal solution that is equally good.
- 4: Almost complete, but certain questions are blank, the code there does not run, or the output is clearly incorrect. For math formulations, this would be if the formulation is seriously flawed.
- 3: This score will not be assigned, as everyone should strive to get 4 or 5.
- 2: Not close to complete, but at least 50% complete.
- 1: At least 10% complete, but less than 50% complete
- 0: Less than 10% complete, or response is identical to someone else's, indicating plagiarism.

A perfect score is 5. Note that your code does not need to be absolutely perfect to receive a 5, but you need to complete every question and ensure that the outputs are correct on all of the sample runs included here. Furthermore, you should double check that your math formulations make sense, by plugging in concrete numbers and verifying that the constraints are satisfied.

These exercises are intended to be completed in 6-8 hours, including the individual-work time in class. You should budget at least this much time before the due date.

Name: XXX

Exercise 12.1: Optimizing Studying Time

Dominique is applying what he learned about optimization to improve his time usage while studying for final exams. He has created a table listing the various topics to study for, his current estimated level of mastery of each topic, the difficulty of the topic as measured by the number of hours studying needed for him to gain an additional level of mastery, and the importance of sufficiently mastering the topic for his future career. For each topic, a level of mastery of 5 is the threshold for "sufficient mastery," and although it is possible to achieve beyond level 5, it is not necessary for him at this point. The two desirable outcomes Dominique would like to achieve are as follows:

1. Attaining level 5 mastery for all topics as much as he can, especially for the most important topics.
2. Not spending too much time in total, so as to maintain a sane work-life balance and fulfill other responsibilities.

To achieve these goals, he has formulated the following optimization problem.

Data:

- T: set of topics.
- c_t: current level of mastery for topic $t \in T$ before studying for finals.
- d_t: difficulty of topic t as measured by the number of hours needed to gain each additional level of mastery.
- w_t: importance of achieving at least level 5 mastery for his future career.
- B: budget of time he would like to allocate in terms of the total hours of studying for these topics.

Decision Variables:

- x_t: number of hours to study for topic $t \in T$. (Continuous)
- g_t: auxiliary decision variable for each topic $t \in T$ representing the gap between his level of mastery after studying and the target mastery level of 5. (Continuous)

Objective and Constraints:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{t \in T} w_t g_t^2 \\
\text{s.t.} \quad & \\
& \sum_{t \in T} x_t \leq B \\
& g_t \geq 5 - c_t - \frac{x_t}{d_t} && \text{for each topic } t \in T. \\
& x_t, g_t \geq 0 && \text{for each topic } t \in T.
\end{aligned}$$

Note that this is a non-linear formulation because the objective involves the square of the decision variable g_t. To implement this, simply do $g_t * g_t$ in the Gurobi code.

Write a function called "optimizeTime" with two input parameters:

- **inputFile**: filename of a CSV file providing the data on the various topics. You may assume that the column headers of the input file are exactly as in the attached file "12-time-input.csv". The four columns correspond exactly to the first four data variables listed in the formulation.
- **B**: a positive number giving the value of the data variable B.

Your function should return a Pandas Series in which the index denotes the topic names and the value the optimal number of hours to study for that topic. Only topics with studying time at least 0.01 hours should be included. The topic names (for the index) are given as the first column of the inputFile, and the optimal number of hours of studying for topic t is given by x_t.

For the test run, you should download the input file attached to this exercise into the same directory as the Jupyter notebook. Below is what this file looks like.

```
[19]: import pandas as pd
      pd.read_csv('12-time-input.csv')
```

```
  Topic  Current Mastery  Difficulty  Importance
0    A1                2           5           4
1    A2                3          10           4
2    A3                6           4           4
3    B1                4           6           5
4    B2                2           3           5
5    B3                4           5           5
6    C1                4           1           2
7    C2                8           1           2
```

```
[20]: # Hint: you can create a Series as follows
      s=pd.Series(dtype=float)
      s['A1']=11.4
      s['A2']=5.79
      s
```

```
A1    11.40
A2     5.79
dtype: float64
```

```
[21]: # Write your code here
```

```
[22]: # Test code
      optimizeTime('12-time-input.csv',30)
```

```
A1    11.448087
A2     5.792350
B1     1.908197
B2     7.977049
B3     2.158470
C1     0.715847
dtype: float64
```

Exercise 12.2: Reusable Software for Project Selection

Recall the abstract formulation for the Project Selection problem introduced in Week 8 Session 16. This is also Example 3 from Week 11.

Data:

- P: set of projects
- C: set of conflicts. Each $(p_1, p_2) \in C$ is a pair of projects that conflicts with one another.
- R: set of prerequisite pairs. Each $(p_1, p_2) \in R$ is a pair such that project p_1 is a prerequisite to project p_2.

Decision Variables: x_p: whether to pursue project p. (Binary)

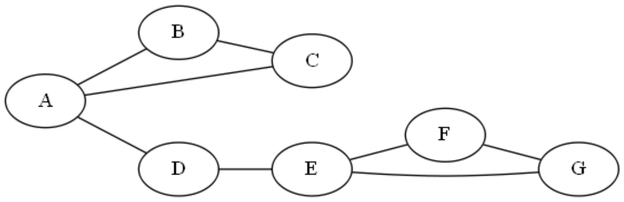
Objective and Constraints:

$$\begin{aligned} \text{Maximize} \quad & \sum_{p \in P} x_p \\ \text{s.t.} \quad & \\ x_{p_1} + x_{p_2} &\le 1 \qquad \text{For each conflicting pair } (p_1, p_2) \in C. \\ x_{p_1} &\ge x_{p_2} \qquad \text{For each pair } (p_1, p_2) \in R. \end{aligned}$$

Write a function called **projectSelection** with two input arguments:

- **inputFile**: an Excel file of the same format as the `12-projects-input.xlsx` file attached to this exercise.
- **outputFile**: the name of the output file that your function should generate, by obtaining the data from the given input file and solving the above linear optimization model. The format of your output file should match the `12-projects-sampleOutput.xlsx` file attached to this exercise.

Note that the `12-projects-input.xlsx` input file corresponds to the problem instance given in Week 8 Session 16. The set of conflicts corresponds to the edges in the following graph.



Moreover, project A is a prerequisite to project F, and project B is a prerequisite to project G. However, you should be reading all of these input data from the **inputFile** rather than hard-coding them in. Here's what the sample input `12-projects-input.xlsx` looks like.

```
[23]: import pandas as pd
      display(pd.read_excel('12-projects-input.xlsx',sheet_name='Conflicts'))
      display(pd.read_excel('12-projects-input.xlsx',sheet_name='Pre-reqs',index_col=0).
 ↪fillna(''))
```

```
  Project 1 Project 2
0         A         B
1         B         C
2         A         C
3         A         D
4         D         E
5         E         F
6         F         G
7         E         G
```

```
                   A    B C D E F G
Project\Pre-Req
A
B
C
D
E
F                1.0
G                     1.0
```

```
[24]: # Write your code here.
```

```
[25]: # Test code
      projectSelection('12-projects-input.xlsx','12-projects-yourOutput.xlsx')
      display(pd.read_excel('12-projects-yourOutput.xlsx',sheet_name='selection'))
      display(pd.read_excel('12-projects-yourOutput.xlsx',sheet_name='objective'))
```

```
  Projects to pursue
0                  B
1                  D
2                  G
```

```
   Maximum # of Projects
0                      3
```

Shift Scheduling

Problem Description

Imani is in charge of scheduling shifts for all nurses at the emergency department at Trojan hospital. **In order to retain quality staff, the hospital would like to create schedules that satisfy their preferences as much as possible, while respecting operational constraints and treating nurses fairly.** In the past, she created schedules by hand, which is very time consuming and she was never sure whether she has found the best schedule. Recently, after learning about linear optimization modeling, she would like to apply her knowledge to create an optimization tool that can be used to automatically generate the best schedule.

Based on her experience, she has created a template input file to store all of the relevant input data. See the `12-scheduling-input-1.xlsx` file for a smaller file for development purposes. The file contains 9 nurses and 1 week. See the `12-scheduling-input-2.xlsx` for a larger file containing a realistic instance, with 50 nurses and 9 weeks.

Ideally, a schedule should maximize the total preference scores of nurses. The preference scores are specified in the input data in the sheet called "Preferences". For each shift, a nurse may indicate a preference score of 0, 1, or 2. A score of 0 indicates that the nurse is unavailable to work that shift. A score of 1 indicates that the time is not preferred, but the nurse is willing to work if needed. A score of 2 indicates a preferred time slot.

Operational constraints:

- There are three nurse shifts in a day: morning, evening, and nights. Each shift should have exactly a certain number of nurses, which is specified in the input file in the sheet called "Requirements". You may assume that Shift 0 corresponds to a morning shift, shift 1 to an evening shift, shift 2 to a night shift, shift 3 to a morning shift, etc. Moreover you may assume that the total number of shifts is a multiple of 3.
- If a nurse works in a night shift, he/she must take the two prior shifts off as well as the two next shifts off. For example, if a nurse works on Shift 2, then she must not work in shifts 0, 1, 3, and 4. If a nurse works on Shift 5, then she must not work in shifts 3, 4, 6, and 7. This ensures that nurses get proper rest.
- No nurse may be scheduled to consecutive shifts. However, it is possible for example to work both shifts 1 and 3, as they are not consecutive and neither is a night shift.
- If a nurse indicates that he/she is unavailable to work in a given shift (i.e. preference score of 0 for a shift), then the nurse cannot be assigned to that shift.

Fairness constraint: The total number of shifts assigned to a nurse must be similar across nurses. Precisely speaking, there is a certain inequality k, and the total number of shifts worked by any given nurse must be within k of each other. For example, if $k = 2$, then the constraint would be violated if Alice is assigned 10 shifts and Bob is assigned 7 shifts. But it would not be violated if every nurse is assigned between 8 and 10 shifts (inclusive).

Write a function called `computeSchedule` with three input parameters:

- **inputFile**: an Excel file of the same format as the `12-scheduling-input-1.xlsx` and `12-scheduling-input-2.xlsx` files.
- **k**: the inequality parameter in the fairness constraint.
- **outputFile**: an output file generated by the function which contains an optimal schedule in one sheet. In the other sheet, the file contains the optimal objective value, the value of k, as well as the optimal objective value divided by the total number of shifts assigned, which can be interpreted as the average preference score for an assigned shift. For sample outputs, see the `12-scheduling-sampleOutput-1.xlsx` and `12-scheduling-sampleOutput-2.xlsx` files attached on Blackboard. Note that there may be multiple optimal schedules, so your code may output a different schedule for the same value of k, but the optimal objective value must be the same.

Exercise 12.3: Abstract Formulation for Shift Scheduling

Write an abstract formulation for the above problem. To help you get started, below are some data variables that may be helpful. You don't have to use these and you can add other ones as you see fit.

Data:

- I: the set of nurses.
- n: the number of days being scheduled.
- J: set of shifts, $J = \{0, 1, 2, \cdots, 3n-1\}$.
- J_{night}: set of night shifts, $J_{night} = \{2, 5, \cdots, 3n-1\}$.
- p_{ij}: the preference score of person i for shift j.
- q_j: the number of nurses needed for shift $j \in J$.
- k: the inequality parameter in the fairness constraint.

Decision Variables:

Objective:

Constraints:

Exercise 12.4: Reusable Software for Shift Scheduling

Create the function `computeSchedule` following the instructions in the problem description. Make sure you put all of your final code in one cell with the comment `# Final Code` such that if the Kernel is restarted and only that cell is run, the code will work. **At the end of your function, you should print a line to report that the function is finished, and print the optimal objective value, as in the sample outputs below.**

```
[26]: # Write your code here
```

```
[27]: # Test code 1 (it's okay if you have a different schedule with the same objective␣
 ↪value.)
      output_file='12-scheduling-myOutput-1.xlsx'
      import os
      if os.path.exists(output_file):
          os.remove(output_file)
      computeSchedule('12-scheduling-input-1.xlsx',3,output_file)
      import pandas as pd
      display(pd.read_excel(output_file,sheet_name='Summary'))
      display(pd.read_excel(output_file,sheet_name='Schedule'))
```

```
Finished Optimizing! Objective value: 83.0

   Objective  k  Average preference score
0         83  3                  1.693878

  name\shift_id  0  1  2  3  4  5  6  7  8  ...  11  12  13  14  15  16  17  \
0        Alexis  0  0  0  0  1  0  1  0  0  ...   0   0   1   0   1   0   0
1        Alyssa  0  1  0  0  1  0  0  1  0  ...   0   1   0   0   0   1   0
2       Anthony  0  0  1  0  0  0  1  0  0  ...   0   1   0   0   0   1   0
3       Brandon  1  0  0  1  0  0  0  1  0  ...   0   1   0   0   0   1   0
4       Brianna  0  1  0  1  0  0  0  0  0  ...   0   0   0   0   0   0   0
5         Caleb  0  1  0  0  1  0  0  1  0  ...   0   0   1   0   0   0   0
6       Cameron  1  0  0  0  0  0  0  0  0  ...   0   0   0   1   0   0   1
7         Chloe  1  0  0  0  0  1  0  0  1  ...   1   0   0   0   1   0   0
8   Christopher  0  0  0  1  0  0  1  0  0  ...   0   0   1   0   1   0   0

   18  19  20
```

```
0   0   1   0
1   0   0   0
2   1   0   0
3   0   1   0
4   0   0   0
5   0   1   0
6   0   0   1
7   1   0   0
8   1   0   0

[9 rows x 22 columns]
```

```
[28]: # Test code 2 (it's okay if you have a different schedule with the same objective␣
 ↪value.)
      output_file='12-scheduling-myOutput-2.xlsx'
      if os.path.exists(output_file):
          os.remove(output_file)
      computeSchedule('12-scheduling-input-2.xlsx',1,output_file)
      display(pd.read_excel(output_file,sheet_name='Summary'))
      display(pd.read_excel(output_file,sheet_name='Schedule'))
```

```
Finished Optimizing! Objective value: 4200.0

   Objective  k  Average preference score
0       4200  1                  1.904762

   name\shift_id  0  1  2  3  4  5  6  7  8  ...  179  180  181  182  183  \
0         Alexis  0  0  0  0  1  0  0  0  1  ...    0    0    0    1    0
1         Alyssa  1  0  0  0  1  0  1  0  0  ...    0    0    0    0    1
2        Anthony  0  0  1  0  0  0  1  0  0  ...    0    0    0    0    0
3        Brandon  0  0  0  1  0  0  0  1  0  ...    0    1    0    0    0
4        Brianna  0  1  0  0  0  1  0  0  0  ...    0    0    0    0    0
5          Caleb  0  1  0  0  1  0  0  1  0  ...    0    1    0    0    0
6        Cameron  1  0  0  0  0  0  0  0  0  ...    0    0    1    0    0
7          Chloe  1  0  0  0  0  0  1  0  0  ...    0    0    0    0    0
8    Christopher  0  0  0  0  0  0  1  0  0  ...    1    0    0    0    0
9         Daniel  0  0  1  0  0  1  0  0  0  ...    1    0    0    1    0
10         David  1  0  0  0  0  0  0  0  0  ...    0    0    0    0    0
11       Destiny  0  1  0  1  0  0  0  0  0  ...    1    0    0    1    0
12       Dorothy  0  0  0  0  0  0  0  1  0  ...    0    0    1    0    0
13         Ethan  0  1  0  0  1  0  0  1  0  ...    0    0    0    0    1
14       Gabriel  0  0  0  1  0  0  0  0  0  ...    0    0    1    0    0
15        Hailey  0  0  1  0  0  0  1  0  0  ...    0    0    1    0    1
16        Hannah  1  0  0  1  0  0  0  1  0  ...    0    1    0    0    0
17         Imani  0  1  0  1  0  0  1  0  0  ...    0    0    1    0    0
18        Isaiah  1  0  0  0  0  1  0  0  1  ...    0    0    1    0    0
19         Isaac  0  1  0  0  1  0  0  0  0  ...    0    1    0    0    0
20         James  0  0  0  0  0  0  1  0  0  ...    0    0    1    0    0
21       Jasmine  1  0  0  1  0  0  0  1  0  ...    0    0    1    0    0
22        Jayden  0  1  0  0  1  0  0  0  1  ...    0    1    0    0    1
23         Jayla  1  0  0  0  1  0  0  1  0  ...    0    0    0    0    0
...
[50 rows x 190 columns]
```

Multi-Product Supply Chain Planning

Problem Description

Nia is a data analyst at Trojan E-commerce, a medium sized online retailer with 17 fulfilment centers scattered across the US. She would like to apply optimization to minimize the weekly outbound shipping cost from fulfilment centers to customers. Trojan uses UPS 2-day delivery. Based on a regression analysis, she found that the cost of shipping each unit of item k from fulfillment center (FC) i to demand region j is $1.38w_k\delta_{ij}$, where w_k is the shipping weight of the item and δ_{ij} is the distance from FC i to region j.

While Trojan E-commerce sells hundreds of thousands of items, Nia conducted a clustering analysis to simplify the analysis and found 100 representative items, and she scaled up the demand for these so that they can serve as a proxy for all the items. Nia has also partitioned the US into 98 demand regions, and has estimated the weekly demand from each demand region for each of the representative items.

Trojan is committed to satisfying all customer demand for all of the items at all demand regions. The weekly demand for item k in region j is given as d_{jk}. However, Trojan can choose how much of this to provide from each FC. Using a closer FC would reduce the shipping cost, but capacity at each FC is limited. (For simplicity, assume that fractional amounts of items is allowed, so the amount shipped does not have to be an integer.) The company replenishes inventory every week. At any given FC, the amount of capacity required for processing each unit of item k is equal to s_k. The total capacity of FC i is given as q_i.

Write a function called `optimizeShipment` with two input parameters:

- `inputFile`: filename of the input file. The format is as in the `12-retail-toy-input.xlsx` and `12-retail-real-input.xlsx` files attached on Blackboard.
- `outputFile`: filename of the output file. The desired format is as in the `12-retail-toy-sampleOutput.xlsx` and `12-retail-real-sampleOutput.xlsx` files attached on Blackboard.

The function should be able to take in any input file of the same format, and create the corresponding output file.

Description of Data

- **12-retail-real-input.xlsx**: the data Nia prepared for her analysis. The excel workbook has five worksheets:
- Fulfilment Centers: the set of FCs, as well as capacity q_i for each FC i.
- Regions: the set of demand regions.
- Distances: the distance δ_{ij} from each FC i to each region j. Each row represents a region and each column a FC.
- Items: the set of items, as well as the shipping weight w_k and storage size s_k for each item k.
- Demand: the demand d_{jk} at each region j for each item k. Each row represents an item and each column a region.
- **12-retail-toy-input.xlsx**: a toy dataset of the same format as the above, for development purposes.
- **12-retail-toy-sampleOutput.xlsx**: the correct optimization output using the inputs from "12-retail-toy-input.xlsx."
- **12-retail-real-sampleOutput.xlsx**: the correct optimization output using the inputs from "12-retail-real-input.xlsx."

Exercise 12.5 Abstract Formulation for Multi-Product Supply Chain Planning

Write an abstract formulation for the above problem. To help you get started, the relevant data variables are already provided.

Data:

- I: the set of FCs.
- J: the set of regions.
- K: the set of items.
- q_i: the capacity of FC i.
- δ_{ij}: the distance from FC i to region j.
- w_k: the shipping weight of item k.
- s_k: the storage size of item k.
- d_{jk}: the demand for item k in region j.

Decision Variables:

Objective and Constraints:

Exercise 12.6: Reusable Software for Multi-Product Supply Chain Planning

Create the function `optimizeShipment` following the instructions in the problem description. Make sure you put all of your final code in one cell with the comment `# Final Code` such that if the Kernel is restarted and only that cell is run, the code will work. **At the end of your function, you should print a line to report that the function is finished, and print the optimal objective value, as in the sample outputs below.**

```
[30]: # One useful Syntax for DataFrames: transposing
      import pandas as pd
      df=pd.DataFrame([[1,2,3],[4,5,6]],index=['A','B'],columns=['i','ii','iii'])
      df
```

```
   i  ii  iii
A  1   2    3
B  4   5    6
```

```
[31]: df.T
```

```
     A  B
i    1  4
ii   2  5
iii  3  6
```

```
[32]: # Write your code here
```

```
[33]: # Test code
      output_file='12-retail-toy-myOutput.xlsx'
      if os.path.exists(output_file):
          os.remove(output_file)
      optimizeShipment('12-retail-toy-input.xlsx',output_file)
      display(pd.read_excel(output_file,sheet_name='Summary'))
      display(pd.read_excel(output_file,sheet_name='Solution'))
```

```
Finished optimizing! Objective value: 3400.769189999999
```

```
   Minimum Total Cost
0          3400.76919
```

```
  FC_name  region_ID  item_ID  Shipment
0       A          0        1       125
```

```
1       A          1        0       100
2       A          1        1       200
3       A          2        1       100
4       B          0        0       500
5       B          0        1        75
6       B          2        0       350
```

```
[34]: # Test code (might take a minute to run)
      output_file='12-retail-real-myOutput.xlsx'
      if os.path.exists(output_file):
          os.remove(output_file)
      optimizeShipment('12-retail-real-input.xlsx',output_file)
      display(pd.read_excel(output_file,sheet_name='Summary'))
      display(pd.read_excel(output_file,sheet_name='Solution'))
```

```
Finished optimizing! Objective value: 9841229.288170151
```

```
   Minimum Total Cost
0        9.841229e+06
```

```
     FC_name  region_ID  item_ID  Shipment
0       SAT1         10        0    2524.0
1       SAT1         10        1    2485.0
2       SAT1         10        3    2300.0
3       SAT1         10        4    3480.0
4       SAT1         10        5    6208.0
...      ...        ...      ...       ...
7870    MSP1         95       95      71.0
7871    MSP1         95       96      43.0
7872    MSP1         95       97      97.0
7873    MSP1         95       98      28.0
7874    MSP1         95       99      38.0

[7875 rows x 4 columns]
```

Exercise 12.7. Assigning of Final Grades

This question asks you to create software that can help a professor assign final grades in such a way so that the average GPA rounds to 3.5, while obtaining an assignment in which there are gaps in scores between consecutive grade levels, and no particular grade is assigned to disproportionally many students.

Data:

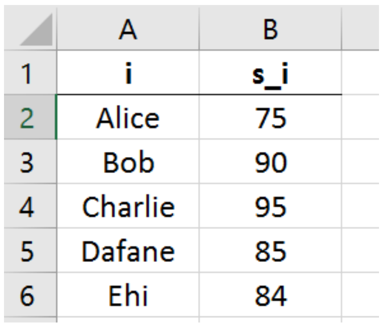
- I: the set of students.
- n: the number of students.
- $J = \{0, 1, \cdots\}$: numerical indices used to denote the various grade levels.
- s_i: the overall score of student $i \in I$ (between 0 and 100).
- g_j: the GPA corresponding to grade level $j \in J$.

Decision Variables:

- x_{ij}: whether to assign student i to grade level j. (Binary)
- t_j: the number of students assigned to grade level j. (Continuous)
- L_j: the score cutoff for grade level j. (Continuous)
- U_j: the maximum score in grade level j. (Continuous)

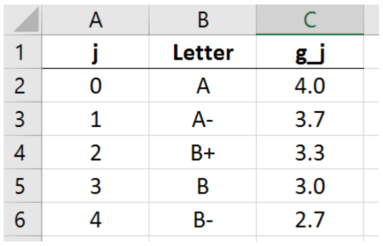
$$\begin{array}{rrl}
\text{Min} & \sum_{j\in J}(U_j - L_j) + 0.1\sum_{j\in J} t_j \times t_j & \\
\text{s.t.} & & \\
\text{(Average GPA)} & 3.495n \le \sum_{i\in I, j\in J} x_{ij} g_j \le 3.505n & \\
\text{(Assignment)} & \sum_{j\in J} x_{ij} = 1 & \text{for each } i \in I. \\
\text{(Max score)} & s_i x_{ij} \le U_j & \text{for each } i\in I,\ j\in J. \\
\text{(Min score)} & 100(1-x_{ij}) + s_i x_{ij} \ge L_j & \text{for each } i\in I,\ j\in J. \\
\text{(Correct totals)} & \sum_{i\in I} x_{ij} = t_j & \text{for each } j\in J. \\
\text{(Bounds)} & L_j \le U_j & \text{for each } j\in J. \\
\text{(Ordering)} & U_j \le L_{j-1} & \text{for each } j\in J \text{ with } j\ge 1. \\
 & t_j, L_j, U_j \ge 0 & \text{for each } j\in J.
\end{array}$$

The input data will be formatted as the attached file `12-grade-input.xlsx`. It is an Excel file with two sheets. The first sheet, named “Scores”, contains the score of each student, as in the following screenshot:



i	s_i
Alice	75
Bob	90
Charlie	95
Dafane	85
Ehi	84

The second sheet, named “Levels”, is as follows

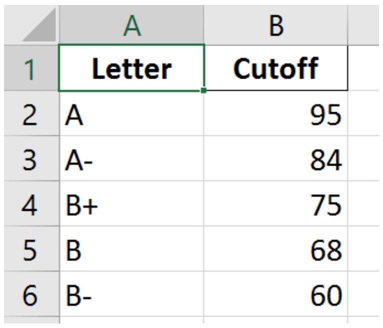


j	Letter	g_j
0	A	4.0
1	A-	3.7
2	B+	3.3
3	B	3.0
4	B-	2.7

Write a function called `assign_grades` with two arguments:

- inputFile: the name of the input file.
- outputFile: the name of the output file.

The function should create an Excel file with the name given by the parameter `outputFile`, which gives the cutoff (L_j) for each grade level j. It should look like the following screenshot:



	A	B
1	**Letter**	**Cutoff**
2	A	95
3	A-	84
4	B+	75
5	B	68
6	B-	60

```
[35]: # Write your code here
```

```
[36]: # Test code
      import os
      output_file='12-grade-output.xlsx'
      if os.path.exists(output_file):
          os.remove(output_file)
      assign_grades('12-grade-input.xlsx',output_file)
      pd.read_excel(output_file)
```

```
  Letter  Cutoff
0      A      95
1     A-      84
2     B+      75
3      B      68
4     B-      60
```