DATA ANALYTICS
GLASGOW

Data Programming in Python

Project (Reassessment)

This assignment sheet is assessed and contributes 25% to your overall grade for Data Programming in Python. You can obtain a total of 20 marks for this project (see marking scheme at end).

Please upload your answers before Monday, May 24th, 10am (UK time).

To upload your answers, log on to Moodle and go to Data Programming in Python. *Under* Reassessment, *there is a link* Upload project. *Click on this link to upload the file containing your commented Python code. You can either upload a Python source file (file extension `.py`) or a Jupyter notebook (file extension `.ipynb`). Please do not upload your code in other formats.*

For this project you need two CSV files available in zip file `Project.zip`, which you can download from Moodle or from

`http://www.stats.gla.ac.uk/~levers/dpip/Project.zip`

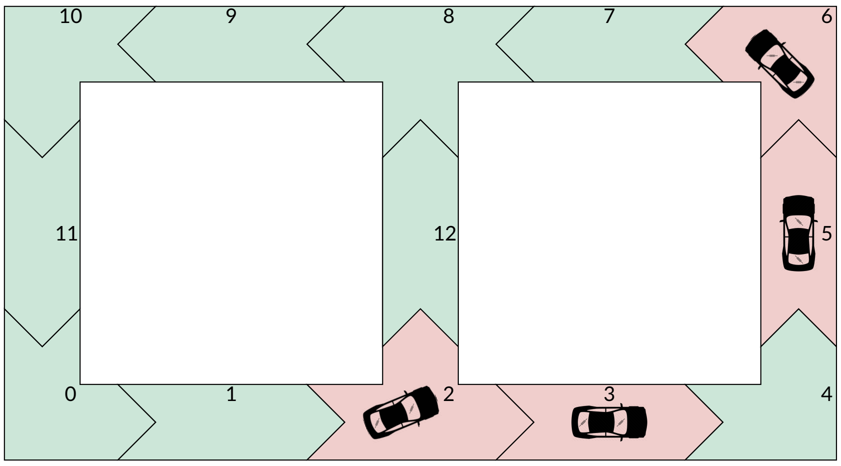
Background

In this project you will implement a simple traffic simulator and collect data from running it. More realistic and thus also more complex simulators, which work similar to the one you will implement, are being used, for example, to inform the choice of layouts for roads and junctions as well as to design signalling plans for traffic lights and schemes for variable speed limits.

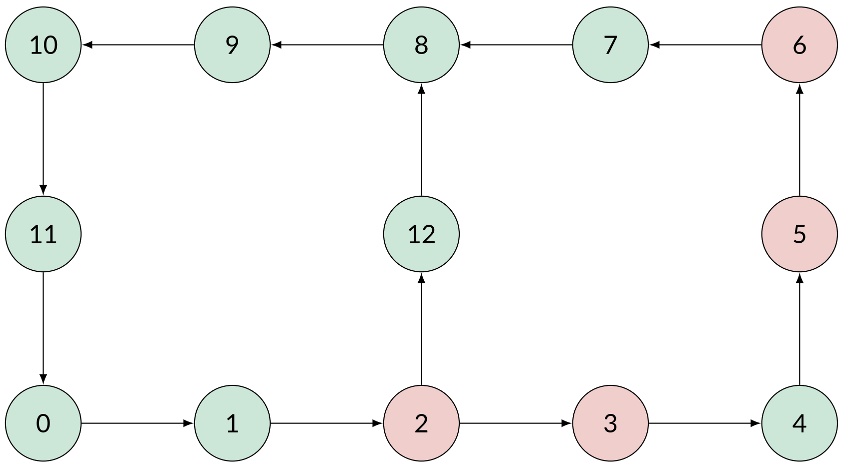
In our simplified simulator roads consist of connected segments. Akin to railway signalling, each block can either hold no car or exactly one car. We assume that all roads are one-way only. In other words the simulator will be based on a directed graph consisting of connected nodes representing the road network.

Example

The figures below show a simple example of such a network. It shows the road network with some cars on it.



In a slightly more abstract way we can represent this network as a directed graph. Each node in the graph represents a road segment. Red nodes in the graph represent nodes with a car and green nodes represent nodes without a car.



Details

Road network

In the ZIP file for the project there are two CSV files, which contain a more complex example layout of a traffic network.

- The file `nodes.csv` contains the coordinates of all nodes (road segments). Assuming that the origin $(0,0)$ is at the bottom-left of our toy example, the coordinates of the nodes of our toy example are.

```
##      x  y
## 0    0  0
## 1    1  0
## 2    2  0
## 3    3  0
## 4    4  0
## 5    4  1
## 6    4  2
## 7    3  2
## 8    2  2
## 9    1  2
## 10   0  2
## 11   0  1
## 12   2  1
```

  Tor example the third row indicates that segment 2 is at $(2,0)$.

- The file `edges.csv` contains the information which road segment connects to which other road segment. In the above toy example this would be

```
##      from  to
## 0       0   1
## 1       1   2
## 2       2   3
## 3       3   4
## 4       4   5
## 5       5   6
## 6       6   7
## 7       7   8
## 8       8   9
## 9       9  10
## 10     10  11
## 11     11   0
## 12      2  12
## 13     12   8
```

  For example, the third row indicates that segment 2 connects to segment 3, i.e. a car which is currently in segment 2 can move to segment 3. Segment 2 also connects to segment 12 as seen in the second-last row.

Simulation

The simulation is now carried out as follows.

For every (simulated) minute the simulator should run the following steps:

For every (simulated) second within the minute perform the following steps (60 times per simulated minute).

"Birth" of cars In every unoccupied road segment randomly set the state to "occupied" with probability p_{birth}. This represents cars joining the road network from properties, parking lots or parts of the road network that we do not simulate.

"Death" of cars In every occupied road segment randomly set the state to "unoccupied" with probability p_{death}. This represents cars exiting the road network to properties, parking lots or parts of the road network that we do not simulate.

Flowing traffic For every occupied road segment, move the car on to the next segment if it is free, i.e. set the state of the current segment to "unoccupied" and set the state of the selected segment to "occupied". If there is more than one segment the current segment connects to ("junction"), then randomly choose one of the segments it connects to. If this segment is occupied, do nothing. If this segment is unoccupied, move the car on to this segment.

For this step you should process the road segments in a random order. Otherwise merging of traffic does not happen in turn.

Let's illustrate this process using the example shown on the first page.

- Imagine we chose segment 6 to be updated. In this case we would move the car to segment 7 and segment 6 would become unoccupied.
- Imgaine we chose segment 5 instead. The only segment it leads into is segment 6, which is currently occupied, so we would not take any action.
- Imagine we chose segment 2 instead. We would then, with a probability of 50% move the car to segment 12, or, with a probability of 50%, leave the the car in segment 2 as segment 3 is currently occupied.

For every minute and every segment, your simulation should also collect the following summary statistics.

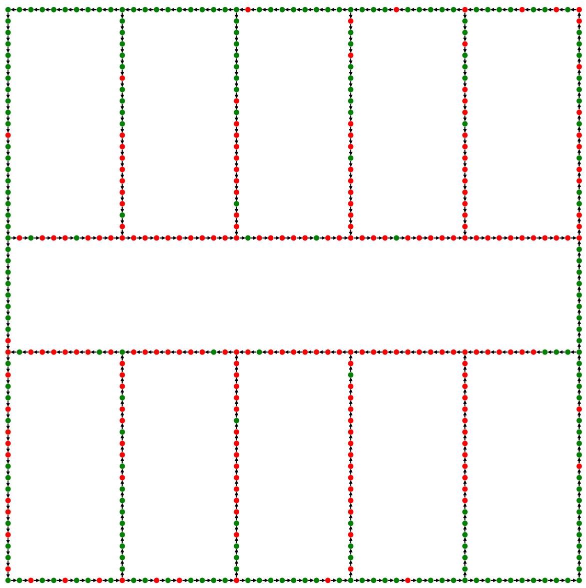
Occupancy Proportion of seconds of this minute during which this road segment was occupied.

Flow Number of times (during this minute) this road segment changed its state from "occupied" to "unoccupied" (depending on what is easier in your approach, you can either include the "death" of vehicles in this or not, if p_{death} is small this does not make a huge difference).

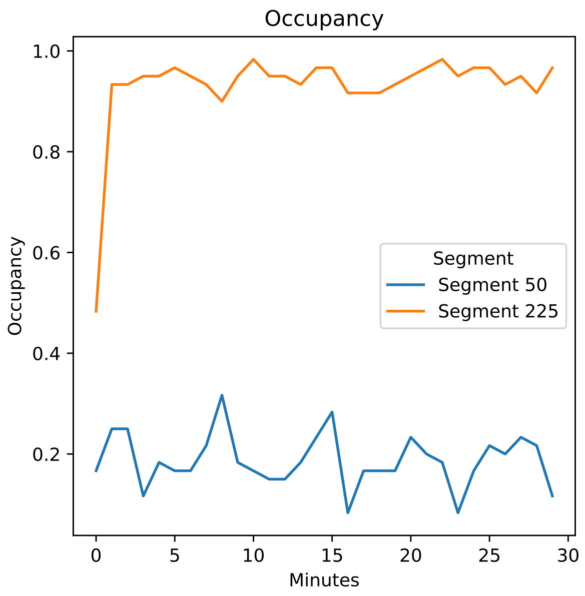
Implementation

Implement a class `Network` representing this traffic network.

- The class should be able to construct the network from the information in the files described above.
- There should be a method called `run` with takes the number of minutes, p_{birth} (default 0.001) and p_{death} (default 0.001) as arguments and which carries out the simulation as described above.
- There should be a method called `plot_network`, which creates a plot of the network in its most recent state, as shown below. Occupied segments should be shown in red and unoccupied segments should be shown in green.

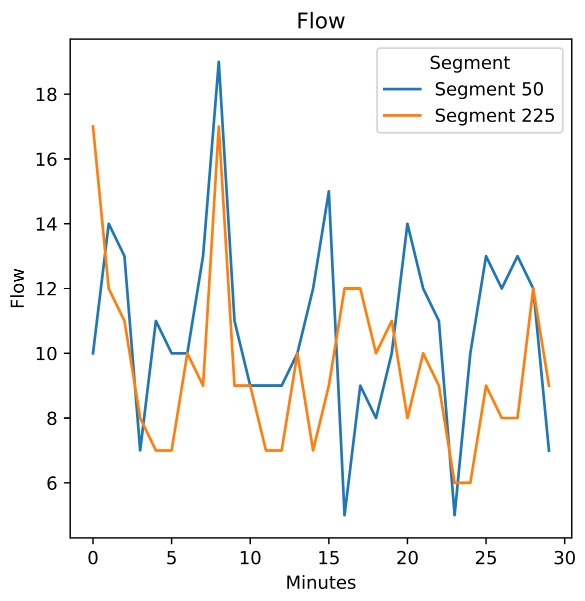


- There should be a method called `plot_occupancy`, which takes one ore more segment indices as argument(s) and which creates a line plot of the evolution of occupancy for each of these segments over time, as shown below. Each line should correspond to a segment.



If you store the occupancies in a data frame or array such that rows correspond to minutes and columns correspond to segments, then you need to create a line plot of the corresponding columns against time.

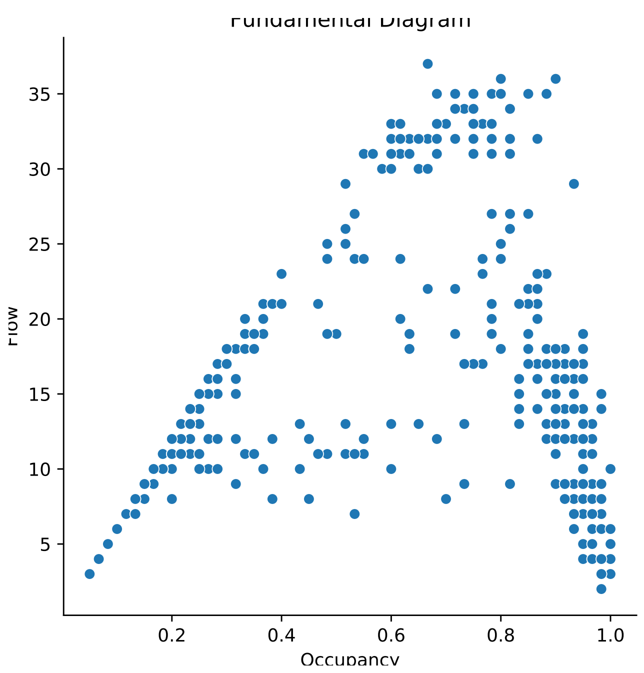
- There should be a method called `plot_flow`, which takes one ore more segment indices as argument(s) and which creates a line plot of the evolution of flow for each of these segments over time, as shown below. Each line should correspond to a segment.



If you store the flow in a data frame or array such that rows correspond to minutes and columns correspond to segments, then you need to create a line plot of the corresponding columns against time.

- There should be a method called `plot_fundamental_diagram`, which takes a minute as argument and which creates a scatter plot of the flow against occupancies observed at that minute, as shown below. Each point in this plot corresponds to a segment.

```
## /home/chaitanya/.local/lib/python3.6/site-packages/seaborn/_decorators.py:43: FutureWa
##   FutureWarning
```



If you store the occupancy in a data frame or array and the flow in a data frame or array such that rows correspond to minutes and columns correspond to segments, then you need to create a scatter plot of one row of the flow array against the corresponding row of the occupancy array.

All methods should check whether the arguments provided are valid and raise a suitable exception otherwise.

Sample code of how your class could be used is given below. (You are not obliged to exactly comply with this.)

```
n = Network("nodes.csv", "edges.csv")
n.run(minutes=30)
n.plot_network()
n.plot_occupancy(segment=[50,225])
n.plot_flow(segment=[50,225])
n.plot_fudamental_diagram(minute=10)
```

Your code should only make use of the following libraries or modules:`math`, `random`, `numpy`, `matplotlib`, `seaborn`, `pandas`.

Coding hints

- You might want to create a class to represent segments. It could hold the state (occupied or not) as well as the information about which other road segments this segment connects to (the "children" of the node in graph-speak). You can also use this class to store the data needed to compute the flow and occupancy for that segment for the current minute.
- To make a decision with probability α you can use

  ```
  import random
  if random.random()<alpha:
         ...
  ```

  To choose one number uniformly from $\{0, \dots, k\}$ at random you can use

  ```
  random.randint(0, k)
  ```

  To go through a list `l` in random order you can use

  ```
  for elt in random.sample(l, len(l)):
         ...
  ```

- To check and debug your simulation you can call the `plot_network` method every "second" inside the method performing the simulation.
- You do not need to draw arrows to indicate the direction of the one-way system, but you can use `plt.arrow` from `matplotlib` if you wish to do so.
- If you do not get one step to work feel free to simulate the required outpout at random or set it to some test values, so that you can implement and test subsequent parts.

Assessment

The following scheme will be used to assess your submission.

Criterion / "Unit test"	Marks if correct
Does the class read in the data and construct the network correctly?	4 marks
Can the network be plotted correctly?	2 marks
Is one step ("a second") of simulation carried out correctly?	3 marks
Is a longer run of the simulation ("several minutes") performed correctly?	3 marks
Is the data (occupancy and flow) being recorded correctly?	3 marks
Can the occupancy and flow data be visalised as described?	3 marks
Code clear and well organised?	2 marks