Data Mining - Homework 1

Valerio Trenta - matr. 1856471

October 27, 2019

1 Exercise 1

Before answering the three questions asked in this exercise, we need to clarify one thing: even if we know the two kids belonging to the family we are considering are both born in a random day of the week, the questions *do not take into account* this information.
This means, with respect to the third question, that we do not care about the fact that one of the two kids was born on Sunday - it just doesn't affect our problem and our final answer and solution to it. We are basically being questioned about the probability of the family having two daughters: the two events **A** = "the other kid is a female" and **B** = "the given daughter was born on Sunday" are not related, and in particular they are said to be **independent** of each other since the probability of **B** occurring in no way affects the probability of **A** occurring and viceversa. We will show this in points **1.3** and **1.2** in a while.

1.1 1.2

Let's start by answering the second question first. If we are not so sure how to answer this question - *"should I take into account the fact that one kid is a girl, or should I just go straight to the easiest answer and say the probability is 0.5?"* - the best way to do this is by taking into account the **sample space**, meaning all the possible events that can occur.
In this case, we obviously have **four** possible samples: boy and boy, boy and girl, girl and boy, girl and girl, considering the order as relevant.
Now, since we know one of the two children is a girl, the sample space in this case is reduced to **S = (boy and girl, girl and boy, girl and girl)**.
So, it is trivial to show that the probability of the other kid being a girl is **P("the other kid is a girl")** $= \frac{1}{3}$. This result may seem a little odd, but it is indeed correct: this probability is not equal to 0.5 simply because we do not know which of the two kids is a girl, if the one who was born first, or the other one - that's why we need to take into account such sample space and such probability.

1.2 1.3

The previous result is going to be the answer also to this question, but we need to - and we are going to - show why. A sample point in the sample space for this question is the following:

$$\{(\text{kid}_1, \text{kid}_2), (\text{day}_\text{i}, \text{day}_\text{j})\}$$

meaning we take into account two tuples: the first one, (kid_1, kid_2), is the one we have already analyzed in the previous point, so we have four possible cases for this that are (boy,boy), (boy,girl), (girl,boy), (girl, girl). Since we already know one of the kids is a girl, we can definitively exclude the tuple (boy,boy) from our possible cases, reducing them to three.
The second tuple, is (day_i, day_j) such that day_i is the day of birth of one kid, while day_j is the day of birth of the other kid, with i,j =1, ..., 7. So we have exactly 7*7*2 = 7^2*2 = **98** possible couples of birthdays over a week of seven days, that is (Monday, Monday), (Monday, Tuesday), ..., (Sunday, Sunday). So the number of samples in the sample space is |S| = 4*7^2*2 = 392 - for this multiplication by 2 and all the others below, see **1.1** below, towards the end of the paragraph.
Since we already know one of the kids is a girl, we are obviously looking for the case in which (kid_1, kid_2) = (girl, girl). Furthermore, we know that the given daughter was born on Sunday, which clearly restricts the couples of days we are looking for, and leaves us with a number of positive (desired) cases of (girl, girl) = 1*7*2 =14: (girl, girl, Sunday, Sunday), (girl, girl, Sunday, Monday), ..., (girl, girl, Sunday, Saturday), (girl, girl, Sunday, Sunday), (girl, girl, Monday, Sunday), ..., (girl, girl, Saturday, Sunday) - meaning, (girl girl, Sunday, x) such that x $\in$ seven days of the week, so seven possible combinations, and (girl, girl, x, Sunday), other seven possible combinations.
Since we know that one kid is a girl and is born on Sunday, our sample space can be reduced too (see once again **1.1** too), and we have |**S**| = 3*7*2 = 42.
So, the probability of event **X** = "Knowing one of the kids is a girl born in Sunday, also the other kid is a girl" is:

$$\mathbf{P(X)} = \frac{14}{42} = \frac{1}{3}$$

and this is the same result we would have obtained without taking into account the birthday of each kid, since as we have already stated, in the way the question is asked the two events "gender of the kid" and "birthday of the kid" are actually independent - indeed, the result is equal to the one we obtained in point **1.2**!

1.3 1.1

The sample space we have defined in the second question was enough to answer the third question, but we would have obtained the same result with a much less wide sample space that is $\mathbf{S_{reduced}}$ **= (boy and girl, girl and boy, girl and girl)**.
If we want to take into account also the days of the week in which the kids may be born, then we would have this sample space that was the one we exploited to answer the previous question, that is:

$$\mathbf{S = \{(kid_1, kid_2), (day_i, day_j)\}}$$

with (kid_1, kid_2) $\in$ {(boy, girl), (girl, boy), (girl, girl)} and (day_i, day_j) being all the possible combinations of pairs of days of the week, so 7^2 possible couples. Obviously, since we already know that one of the kids is a female, this leaves us with only 3 possible combinations that are (girl, boy), (boy, girl), (girl, girl). Furthermore, since the kid was born on Sunday, even if we don't know which of the two was, we only have 7*2 possible couples: we fix day_i = Sunday and let day_j be every day of the week, and then viceversa, so we end up with 7*2 = 14 possible couples of days.
Therefore, we have |S| = 3*7*2 = 42 sample points in the sample space.
Notice also that we could take into consideration another sample space, S_2, which would be enough to solve the problem in **1.3** and has less sample points: if we do not take into account the fact that we don't know which one of the two kids is born on Saturday, we end up with having only 7 possible couples of days fixing day_1 = Sunday and letting day_j be any day of the week, resulting in a probability of still $\frac{1}{3}$ (7 desired cases) and $|S_2|$ = 3*7 = 21. Indeed, we have multiplied by 2 both the possible (desired) cases and the number of sample points in point **1.3**, which is actually useless when computing the probabilities over the sample space, but since we are assuming we don't know which of the two kids is the one who was born on Sunday and is a girl, I think S_2 is theoretically not so fit for this exercise, so the best sample space to take into account should be **S**.

2 Exercise 2

See in the references: [How to Win a Guessing Game - Numberphile]

Just like the previous exercise, this one's solution goes way beyond human intuition, and that's why we need to start processing this by considering the hint that was given.
Indeed, we do not know the number **x** which we suppose to be between the two page numbers that have been chosen, and that's why we need to *randomly choose it*: we derive **x** by choosing it uniformly at random by exploiting a Gaussian distribution, or any other distribution which makes our task easier and grants

us in some way to have a strong gain from it.
Of course, the point in doing this is to have at the end of the day an **x** which is at least in the interval of the number of pages in the book and, possibly, even in a better interval.
Beware: the aforementioned step is actually crucial, since it is the one which will determine if our strategy will work or not: assuming the two challengers can pick the pages in an adversarial way, the distribution from which we pick our random number **x** must be as precise as possible, meaning that we want the number to be picked at least in the interval which contains the pages of the chosen textbook and, if possible, we want a distribution that can guarantee us to choose a good random number even if the numbers chosen by the challengers are chosen in a very small interval, like [1,2]. Assuming the two chosen numbers are **y** and **z** such that y < z and so x ∈ [y,z], we have three possible cases when our challengers reveal a number **n**:

- x ∈ [y,z] and x>n, so we choose the number which has not been revealed (not **n**=y) with probability $\frac{1}{2}$;
- x ∈ [y,z] and x<n, so we choose the number that has been revealed - that is, **n**=z - with probability $\frac{1}{2}$;
- x ∉ [y,z]: no information is added, so we can only guess randomly after we request one of the challengers to reveal their number, so we end up with a $\frac{1}{2}$ probability of winning the game.

In both the first and the second case, our random choose adds information to our problem, and that's where we are sure to guess correctly.
Since we pick **x** uniformly at random, and we have probability 1 to guess correctly in this case, we end up with these two events:

- A = "I choose x ∉ [y,z]";
- B = "I choose x ∈ [y,z]";

and we assume, since we pick **x** at random from a Gaussian distribution, that event B occurs with probability $\rho > 0$. So we get:

- P(win|A) = $\frac{1}{2}$ - "win" and event A are independent!
- P(win|B) = $\frac{1}{2}*\rho$ - in both cases we always win, but we pick the correct answer with probability $\frac{1}{2}$, and this of course is bounded to the probability that we choose **x** ∈ [y,z], that is ρ;

therefore:

$$\mathbf{P(win)} = \frac{1}{2} + \frac{1}{2}\rho = \frac{(1+\rho)}{2} > \frac{1}{2} \text{ since } \rho \in [0,1].$$

So the strategy is the following: choose randomly a number **x**, compare it with the number that will be revealed by one of the challengers and, if **x** is the smaller one between them, then choose the revealed number as the answer to the game, otherwise choose the one that has not been revealed yet.
Still, with this reasoning, the smaller is the gap between the two chosen pages, the harder it will be for me to win the game, meaning that in order to maximize the probability of winning this game it is important to choose carefully the distribution exploited to randomly pick **x** and the aforementioned gap should be as big as possible, but this would take the problem to a whole different level.

3 Exercise 3

3.1 3.1

We want to define a probability space Ω for the given random graph model $G_{n,p}$. Since we have n nodes, we define the set of nodes $\mathbf{N} \in G_{n,p}$ such that $\mathbf{N} = (n_1, n_2, ..., n_n)$, and the set of edges $\mathbf{E} \in G_{n,p}$ such that $e_{i,j} \in \mathbf{E}$ is the edge linking node n_i to node n_j, with i,j $\in$ (1, ..., n).
Since to each edge $e_{i,j}$ is assigned a probability $\mathbf{p} \in (0,1)$ which represents the probability of $e_{i,j}$ to actually *exist* - meaning, the probability of having an edge $e_{i,j} \in \mathbf{E}$ which connects the two nodes n_i, $n_j \in \mathbf{N}$ - defining a probability space over this model means to enumerate all the possible edges that are in **E** and to define all the possible graphs we can obtain with a different combination of edges which belong to **E** itself..
So in particular, we obtain that the number of the possible edges given a set of nodes **N**, with $|\mathbf{N}|=\mathbf{n}$, is equal to $\mathbf{E_p} = (E_1=\emptyset, E_2=(e_{1,2}), E_3=(e_{2,3}), ..., E_{\binom{n}{2}})$ which has a cardinality of $\binom{n}{2}$ which is, indeed, the number of possible pairs of nodes we can obtain from **n** nodes, and exactly the number of the edges in the *complete* graph given **n** nodes - and of course, we are assuming that the model does not contain any loop in assuming this.
Therefore, the probability space Ω coincides with all the possible graphs that can be obtained given the set of nodes **N** and all the possible combinations of edges linking them $\mathbf{E_p}$.
What we do when building a graph model like this, is select with probability **p** the $\binom{n}{2}$ possible pairs of nodes among which can be linked with the given set of nodes.
Basically, since given all the possible edges we can have in our model, a given pair of nodes that are linked through an edge can either exist or not exist, it is trivial to show that the size of this probability space is:

$$| \Omega | = 2^{\binom{n}{2}} = 2^{\frac{n(n-1)}{2}}$$

and that "2 power of" represents the aforementioned binary choice, since each graph in our model is obtained through a choice made over probability

p about which of the $\binom{n}{2}$ will exist in our graph: for each edge, if it exists it is included in our graph, otherwise it's not, so $2^{\binom{n}{2}}$ is the number of possible graphs we can obtain given **n** nodes.

3.2 3.2

Now, we have to define Ω as the set of the different sets of edges we can obtain, each being possibly equal, since **n** nodes by definition are fixed, to the final graph we will obtain. So, if we have $\mathbf{E_1} = \emptyset$, $\mathbf{E_2} = (e_{1,2})$ and so on, we have that $\Omega = (E_1, E_2, ..., E_{2^{\binom{n}{2}}})$ where $\mathbf{E_i}$ is the i-th enumeration of the combinations of nodes that are linked by edges that are present in the graph - that is, as we stated before, our final graph, since the nodes are fixed.
This means that, if we define probability **p** as the probability according to which we pick these edges - each of these edges - then this probability is applied to each edge, so out of all the combinations of edges that are possibly picked, each of them has the same probability of being chosen, so we have:

$$P(E_i) = \frac{1}{|\Omega|}$$

3.3 3.3

To answer this question - and the next one - we should modify a little the question itself: we want to check the probability of event **X**, where **X** = "three nodes form a triangle in the graphs, and the other nodes have no edges". This event **X** is given, actually, by these two events (assuming there are no multiple edges in the model, and assuming no loops): **A** = "3 nodes out of n have degree 2 in the graph" and **B** = "n-3 nodes have degree 0 in the graph", thus we have:

$$\mathbf{P(X)} = P(A \bigcap B)$$

Now, it is a well-known result, and it can be found in literature (check:[Erdos-Rényi Model Properties]), that the probability of a node in $n \in G_{n,p}$ of having degree k is equal to:

$$P(\text{degree of n is k}) = \binom{n-1}{k} * p^k * (1\text{-p})^{(n-1-k)}$$

Thus, since we want 3 nodes in our graph having degree 2, we have, since the probabilities are independent of each other for each node we pick:

$$\mathbf{P(A)} = P(\text{degree of } n_1 \text{ is 2}) * P(\text{degree of } n_2 \text{ is 2}) * P(\text{degree of } n_3 \text{ is 2}) = \left[\binom{n-1}{2} * p^2 * (1\text{-p})^{(n-3)}\right]^3$$

$$\mathbf{P(B)} = [(1\text{-p})^{n-1}]^{(n-3)}$$

and, since the two events **A** and **B** are independent since every edge is picked independently of each other edge, we have:

$$\mathbf{P(X)} = \mathrm{P(A} \bigcap \mathrm{B)} = \mathrm{P(A)*P(B)}$$

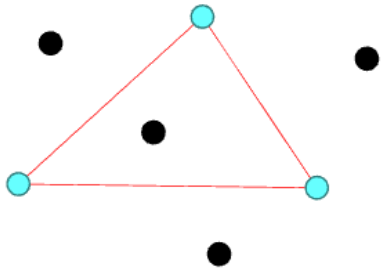

Figure 1: The desired result: three nodes having degree equal to two - connected by the red edges - and all the other nodes having degree equal to zero, thus forming a single triangle in the graph.

Notice that the results we accomplished in this point of the exercise, and the ones we will show later, are all reached under the given assumption that the edges in the model are all picked independently of each other.

3.4 3.4

The reasoning here is similar to the previous point, except we now consider these two events: **A** = "n-2 nodes have degree 2 in the graph" and **B** = "2 nodes out of n have degree 1 in the graph", so we have that **X** = "all the nodes are connected through a line in the graph" and $\mathbf{P(X)} = \mathrm{P(A} \bigcap \mathrm{B)}$.

Figure 2: The desired result: notice the degree of the intermediate nodes and the degree of the two nodes at the beginning and at the end of the line. The result is similar to a caterpillar data structure without leg nodes.

Given what we stated in the previous point, we have:

$$P(A) = [\binom{n-1}{2} * p^2 * (1\text{-}p)^{(n-3)}]^{(n-2)}$$
$$P(B) = [\binom{n-1}{1} * p * (1\text{-}p)^{(n-2)}]^2 = [(n\text{-}1) * p * (1\text{-}p)^{(n-2)}]^2$$

and again, we obtain:

$$\mathbf{P(X)} = P(A \bigcap B) = P(A) * P(B)$$

3.5 3.5

To compute the expected number of edges in our graph, we need to consider again the probability which we exploit to pick these edges, which is equal to **p**. We now need to consider all the possible pairs of nodes which can be picked given **n** nodes and can therefore be linked through an edge, which we have already proved to be equal to $\binom{n}{2}$.
If we pick the edges in the graph with probability **p**, this means that we expect to pick each of these pairs of nodes linked by an edge in our graph with probability **p**, so the expected number of edges in our graph must be equal to:

$$E[\text{number\_of\_ edges}] = \binom{n}{2} p = \frac{n(n-1)}{2} p$$

Again, this is a very well-known result in literature:
check [Erdos-Rényi Model Properties].

3.6 3.6

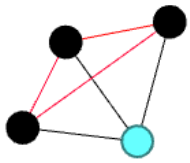

Figure 3: The desired result for a 3-star: in black, the edges we want our star to have, in red the ones we do not want it to have. The blue node is the center of the star. Notice other edges and nodes have not been taken into account, since they do not influence our final result.

To solve this, we should take into account all the possible nodes in the given graph which can possibly form a *3−star*, that are obviously all the possible quartets we can obtain with **n** given nodes, so $\binom{n}{4}$. Out of all these nodes, we have to pick with probability equal to **p** three edges - *(v_0, v_1), (v_0, v_2), (v_0, v_3)* - and we need *not* to pick the other edges linking the nodes that are not the

center of the 3-star, meaning other 3 edges.
A resulting 3−star can be observed in the previous page, in **Figure 3**.
Then, the expected number of 3−stars is equal to:

$$\text{E[number\_of\_3\_stars]} = \binom{n}{4}\text{p}^3*(1\text{-p})^3$$

Of course we do not take into account the other nodes that each node of the star could have, since they do not add any information to our result.

3.7 3.7

Same reasoning as the previous exercise, just generalizing the result, brings us to:

$$\text{E[number\_of\_k\_stars]} = \binom{n}{k+1}*\text{p}^k*(1\text{-p})^{\binom{k}{2}} = \binom{n}{k+1}*\text{p}^k*(1\text{-p})^{\frac{k(k-1)}{2}}$$

because for each *k+1−tuple* of nodes in the graph, we want it to have exactly **k** edges from v_0 (the center of the star) to all the other nodes v_1, ..., v_k, and we want it *not* to have exactly $\binom{k}{2}$ edges, that are all the edges which links the nodes from v_1 to v_k.

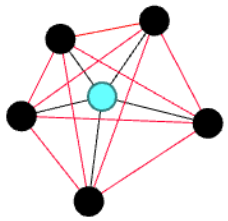

Figure 4: A possible k-star: in black, the edges we want our star to have, in red the ones we do not want it to have. The blue node is obviously the center of the star.

4 Exercise 4

4.1 4.1

In order to meet the requirements of this exercise, we need to write down a single, but efficient and pretty long UNIX command by exploiting some of the recommended tools. What we are trying to do with this command is: select, from the original files, only the columns we are interested in, which are **user_id** and

track_id and correspond with lines, in **awk**, $1 and $2, then sort the obtained file on **user_id** column.
At this point, we concatenate another instruction, where we exploit the **uniq -c** command, through which for each different value in a row - meaning, in this case, for each different user id, track id pair - we merge together those rows having same values and count them. Now, since we have counted basically for each user u_i the times u_i has listened to track t_j, through an **awk** command we filter out all the tracks that each user has listened less than 20 times and only keep the remaining user id, track id pair.
The last **awk** command is again a way to merge the rows with same **user_id** value, obtaining for each user a column containing all the ids of the tracks he has listened more than 20 times, in the format requested by the exercise. Here is the single-line command that was exploited:

awk '{FS=OFS="”"} {print $1,$5}' useridtimestampartidartnametraidtraname.tsv | sort −nk2 | uniq c | awk '{if ($1>19 && $2!="" && $3!="") print $2 "\t" $3}' | awk '{a[$1]=($1 in a ? a[$1] OFS:"")$2} END {for (i in a) print i "\t" a[i]}' | sort −nk2 > output_final_dataset.tsv

This is a much more readable version of the command:

```
awk '
    {FS=OFS="\t"}    /*setting the end of field character to TAB*/
    /*print only user_id, track_id from columns in original dataset*/
    {print $1, $5}' userid-timestamp-artid-artname-traid-traname.tsv
/*sort the obtained file and group by user_id each pair, and count them*/
    | sort -nk2 | uniq -c |
    awk '
/ *select only those pairs which occur >19 times and filter out empty fields*/
        {if ($1>19 && $2!="" && $3!="") print $2 "\t" $3}' | /
        awk '
         /*format the output in the desired way*/
            {a[$1]=($1 in a ? a[$1] OFS:"")$2}
            END
            /*print into the final output file after sorting again*/
                {for (i in a) print i "\t" a[i]}' |
                    sort -nk2 > output_final_dataset.tsv
```

The aforementioned command can be found under the same directory of this .pdf file, in the file called **exercise4.sh**.

4.2 4.2

Now we can exploit the obtained dataset, named as **output_final_dataset.tsv**, to process the data in Python.

I won't be including the **.py** code in this pdf file, but it can be found under its same directory with name **exercise4.py**, and here I will briefly show the output and explain the code, which is also commented in the **.py** file.
First, we exploit Python's **csv** to read the **output_final_dataset.tsv** file and turn it into a list which has the following format: [['user_id'] ['track_id1, track_id2, ...]]. Then, for each row of this list, we take the first element - **user_id** - and store it in a local variable, then for each user we take the *list of tracks* he/she has listened to more than twenty times and, since it is a string, we exploit the function **split()** to split it into different track_ids.
Now, each **track_id** is put as a key into the dictionary, that is the **inverted index** we want to build, and for each track_id we store a list of user_ids that are the users which have listened to the track.
After this loop, we obtain the desired **inverted_index** and we are now ready to design our algorithm for the final solution. I tried to build an algorithm with two loops at the beginning, but in the end I found out that the complexity of such an algorithm was quadratic and it took way too long for it to provide a satisfying solution, so I decided to count, for each key (track) in the index, the size of its value (a list of users), and then to sort the outcome of this operation through the **sorted()** function, in which in order to have a list which goes from the most listened track to the least one, I had to set the parameter *reverse=True*.
At this point, instead of looping on the entire index, I just took the first twenty elements of it - meaning, the twenty tracks that were listened by most of the users - which sounded clearly as a good approximation: if we want to take the top 10 tuples, there is no meaning in considering further tracks and, actually, even taking the first ten elements would be enough.
The loop was then made in order to build a dictionary which has tuples (track_1, track_2) as keys, and as values the sum of the users who have listened to track_1, plus the number of users who have listened to track_2, as shown in picture **Figure 5** at the next page: this way, the requested output is given.
The whole code can be found in the **exercise4.py** file.

5 Exercise 5

The solution for this exercise has been divided into two parts and two corresponding files written in Python: they can be found under this same .pdf's directory as **exercise5_createdataset.py** and **exercise5_compare.py**.
The first file creates the dataset which has been exploited to compare the different announcements, **dataset_kijiji.tsv**. The reasoning was the following: first, we need to identify the URL which corresponds to a **HTTP GET** request to the target website to retrieve all the announcements about apartments available for rent in Rome - that is,
"http://www.kijiji.it/case/affitto/roma−annunci−roma/?entryPoint=sb".
Once we have found it, with a for-loop we dynamically change the address to get **119** pages of such announcements (it's pretty self-explanatory in the code,

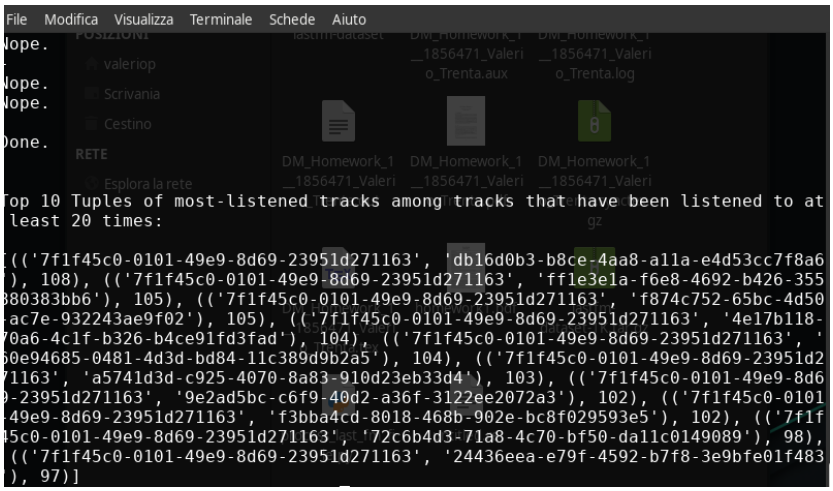

Figure 5: Requested output for exercise 4: Top 10 tuples of track ids.

we only have to modify a character representing the number of the page from the second page on) and store them in a dictionary to parse them by exploiting *TheBeautifulSoup* module with parameter *'lxml'*, which grants us the possibility of storing each page's *xml* description, while to get all the pages we exploit the **urllib3** module in the loop.
The **119** pages we chose to retrieve are actually all the pages stored in the server's website for announcements about apartments available for rent in Rome, at least at the time the code was written.
Actually, we could obtain even better results - in terms of time necessary to parse alle the pages - by exploiting the **lxml** module, but **TheBeautifulSoup** makes it easier to analyze the **xml** tree in the next steps and it is a well known module, so that's why it has been exploited.
We decide to parse all the pages of announcements, for a total of **119** pages and **2617** announcements (22 announcements per page, but the last one has only 21 *at the time the code was written*), setting a **time.sleep()** for the parsing loop of two seconds to avoid being rejected from the website, as suggested.
Beware: the number of pages and announcements in retrieved could change from day to day, so obviously when executing the code contained in the two .py files, one should also change some of their parameters!
Back to the code: we inspect the source code of the page itself with a browser - Mozilla Firefox was used in our case - to identify all the **xml tags** that are going to be useful for our goal, as shown in picture **Figure 6** in the next page.

It is easy to find among the **xml** description of the web pages, a line that is:

<ul id="search-results" ...>

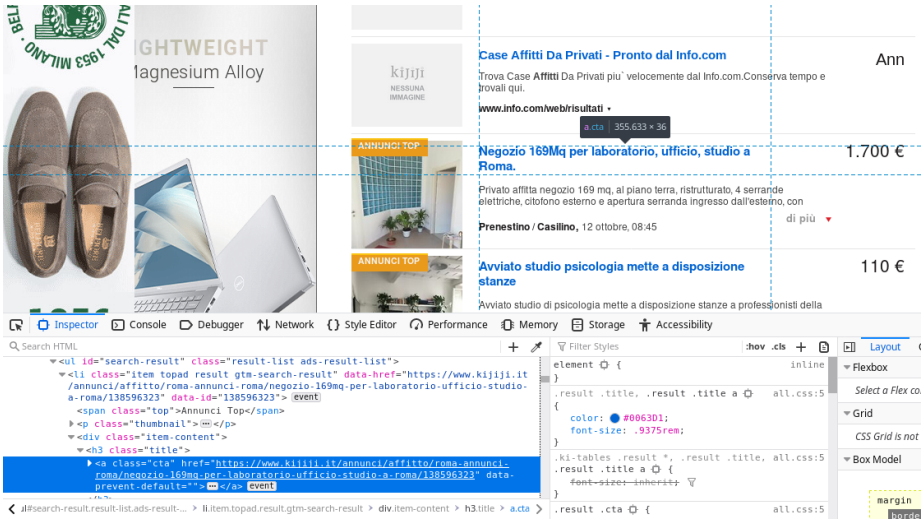

Figure 6: Inspecting the source code of the target pages.

this is where the list containing the search results for such pages begin, and where we want to start our processing phase; therefore, we store in a variable **start_tag** all the contents - meaning, the sub-nodes in the *xml* tree we obtained from parsing the web pages - of such list and, from there, again with a for-loop we look into the list every key value we need: the link of the announcement, its title, description, location and price, storing these values in a dictionary **(id, value)**, and of course we are sure that every value with a given **id** will correspond to the **id-th** announcement.
Then, we exploit once again the **csv** module in order to write our results into the **dataset_kijiji.tsv** file.

Now, for the second point of the exercise, we load the **dataset_kijiji.tsv** file again and read it with **pandas** module, which allows us to only select the two columns we are interested in and store them into two variables called **locations** and **prices**.
Now, some text pre-processing should be done, since some results are actually useless to our goal: for example, some of the announcements have no value specified for their **price** field, but instead the specify to directly contact the user who issued the announcement: these announcements have been omitted from our final result.
There are some announcements which are not about apartments, but shops or boxes available for rent. These announcements have been included in the result, even if in some cases they might have altered the final outcome for some locations by biasing the average price, but we decided to include them anyway since there was no boundary to them.
Also, the prices are reported in euros as strings, so we needed to cut the last two

characters from each of them (a space and the euro symbol, " €") and convert them through the **locale** module - again, by changing the environment where the code runs, some changes might be necessary in order to overcome this step, as explained in the comments of the code itself.
Next step was to iterate over the keys in the two dictionaries - which were of course the same and referring to the same apartments and, since we retrieved a dataset containing 2617 announcements, we had to iterate over 2617 keys - in order to count the number of announcements per location and sum each price for the same location, so that we obtained a dictionary **(location, (number of announcements, sum of prices))**.
Final step was to iterate again with a for-loop over the aforementioned dictionary to divide the sum of prices by the number of announcements in each location, thus obtaining the average price in the final output as a dictionary **(location, (number of announcements, average price))**, as shown in picture **Figure 7** below.

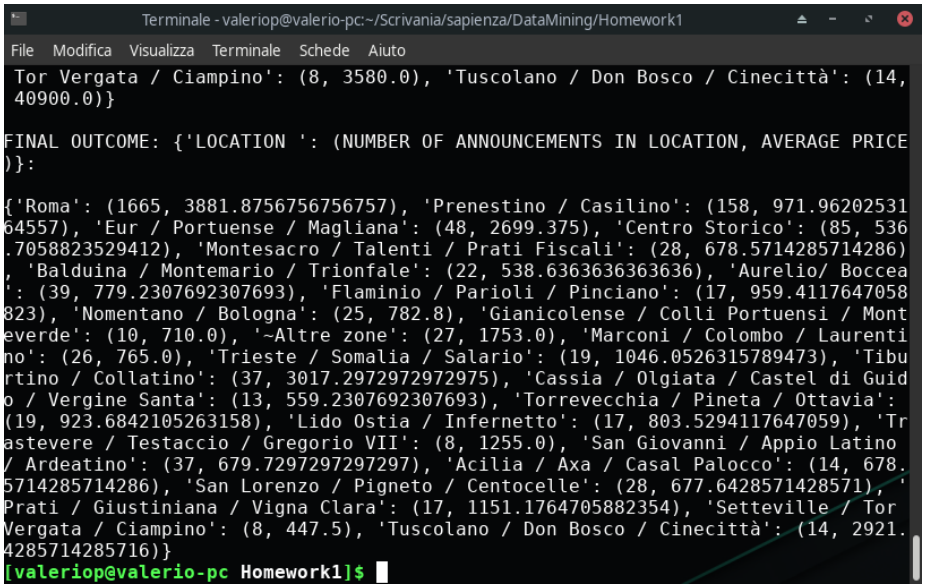

Figure 7: Final output.

Both **exercise5_createdataset.py** and **exercise5_compare.py** show the code that has been written in order to achieve these results - keep in mind that, of course, some of the *directory* variables contained in this code ought to be changed if codes are executed in different environments.

References

[How to Win a Guessing Game - Numberphile] The intuition for this answer came from the hint and from this interesting video I found on Youtube: https://www.youtube.com/watch?v=ud_frfkt1t0

[Erdos-Rényi Model Properties] https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model#Properties_of_G(n,_p)