Documentation

Oona's data visualization library

1. Personal information

Name: Oona Sohlberg

2. General description

My software is a numerical data visualization library. I developed it as a hard version. With my software, users can implement three types of data visualizations: line diagrams, histograms, and pie diagrams. The line diagram accepts 2 to 50 data points, while the histogram and pie diagram accept 1 to 20 data points. Histogram and pie diagram accept the same kind of data. Users can input data either as a file or manually within the software. After creating a visualization of the data, users can observe it to, for example, find out the median (for line diagrams and histograms), mean (for line diagrams and histograms), and perform linear regression (for line diagrams).

I chose to allow users to input data manually by typing it during program execution. I believe this is a useful feature, as users often may not have data in a file format. I didn't have this feature in my plan. Since my software is designed to visualize small amounts of data, manual input is likely to be frequently used. Initially, I planned for users to specify the headline, x-axis, and y-axis before creating the visualization. However, I decided to allow this customization only after the visualization has been created, as it may be easier for users to do so after seeing the visualization. My program has only one main window, while I planned to make several of them.

3. Instructions for the user

After starting the program, the user will choose type of the data visualization. Then, the user will select a file or manually input the data. If there are any issues with the input, the program will display an error message. In the upper right corner, there is an instruction button that opens a window with guidance on input data, including for example size, format, and number range.

Once the correct data is inputted, the visualization will appear. Afterward, depending on the type of visualization, the user can add features such as a grid, data sorting, headline and axis labelling, data information checking, and linear model. The user can always return to the menu to change the visualization type or input data without exiting the program.

4. External libraries

PyQt6

5. Structure of the program

My program has several files grouped by class or classes in the file.

The group name, where the classes of the files belong, can be seen in the beginning of the name of the file. (I had trouble with import statements when I grouped the files in folders, so I decided to group them in this way).

CW = central widget

DC = data calculations

DS = data structure

GI = graphics item

IC = input check

ID = input data

MW = main window

OW = opening window

The files main.py and test.py don't have a group name.

Most of the files contain only one class. I have a file for main function main.py which is used for running the program. This uses QApplication, which makes executing the program possible. My tests are in test.py, which also uses QApplication. This file has class Test, which inherits unittest.TestCase. It's necessary when using python's built-in unittests. Test has method setup, which creates the unit test cases.

My program has only one main window, and that is in file MW_main_window.py class MainWindow. It inherits QMainWindow, which has functionalities for main window. It imports all functionalities from QtWidgets and imports several classes from my program: Menu, File, GoThroughFile, Graph, NameAxisWindow, LIneInput, HistPieInput, CheckInputLine and CheckInputHistPie. MainWindow sets central widgets, that I have three in my program, Menu, File and Graph. It has some important methods, related to checking the input file or data. MainWindow has functionalities for some central buttons in the program, so it imports many classes from my program.

As I previously mentioned, my program has three central widgets, and they all inherit QWidget. The central widget set by default is Menu located in CW_menu.py. It imports all the functionalities from QtWidgets and QtGUI, and QtCore. User selects the type of visualization in the menu.

After choosing the visualization type, File form CW_file.py is set as a central widget. It imports all the functionalities from QtWidgets and QtGUI, and QtCore and my program's instructions windows, which I will be introduced later. There are four buttons in File, "Write data", "Select File", "Instructions for input data", and "Back to menu".

When "Write data" is clicked, a new window opens. If line diagram is chosen, LineInput from file ID_line_input.py opens as a new window. In case histogram or pie diagram is chosen, HistPieInput from file ID_hist_pie_input.py opens. These files import QtCore and all functionalities from QtWidgets and QtGui. LineInput and HistPieInput inherit QWidget. Both LineInput and HistPieInput collect data from users by QLineEdit.

If 'Select file' in File widget is clicked, it opens user's PC: s files by QFileDialog. Then the user can select a file from their own files. If "Instructions for input data" is clicked, the program opens a new window which has instructions. The opening window is in file OW_instructions_window.py, which has classes InstructionsLine, InstructionsHist and InstructionsPie. It imports all functionalities from QtWidgets and QtGui. These classes open the instruction window. When "Back to menu" is clicked, the program will set Menu as a central widget.

After there is input selected, the program will check the input. If the input has been written manually in the program, the input will be checked in CheckInputLine or in CheckInputHIstPie located in IC_check_input_line.py and IC_check_input_hist_pie.py respectively. If the input is a file, it will be checked in IC_check_file.py. File MW_main_window.py imports these classes, so the input data will go through class MainWindow. If the input data is not in right format, the program will throw an error message on File widget, and user can input data again.

After the user selects the right kind of file or writes right kind of input, the program will set Graph from CW_graph.py as a central widget. It imports all the functionalities from QtWidgets and QtGUI, and QtCore and my programs graphics items, information windows and show data -window. These will be introduced later. Graph sets the buttons which depend on the kind of visualization user has chosen. Graph has a “Back” button which sets File as a central widget.

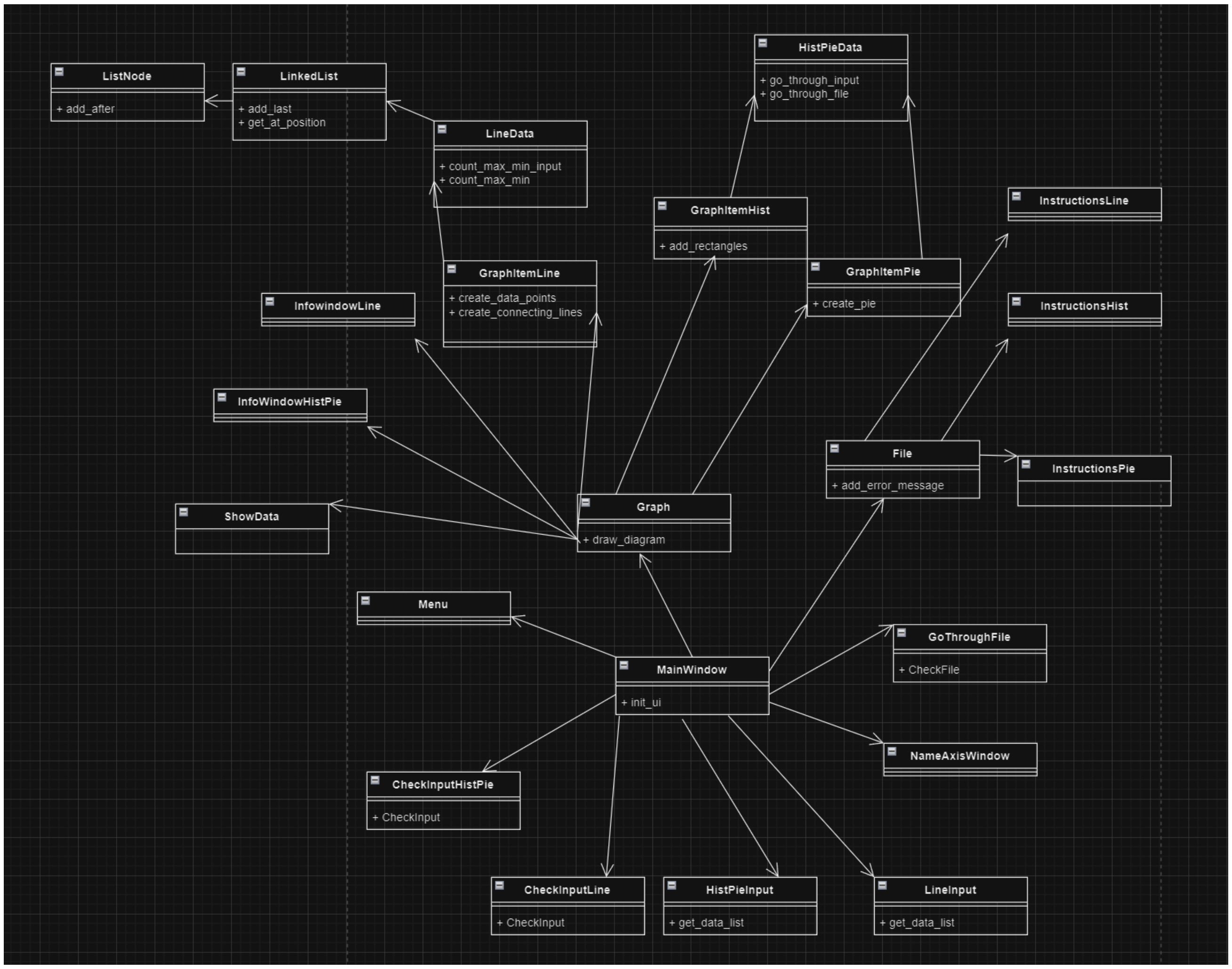
Graph has button “Name axis” (line diagram and histogram) or “Name headline” (pie diagram). When the user clicks that, a new window will open. This happens in class MainWindow. The opening window is class NameAxisWindow located in OW_name_axis.py. This file imports all functionalities from QtWidgets and QtGui, and QtCore. Class NameAxisWIndow inherits QWidget, and it uses QFileDialog to allow user to name headline and possibly axis.

There are buttons “Show data” and “Graph Information” in widget Graph. Both open a new window. Class ShowData is in file OW_show_data.py. It imports all functionalities from QtWidgets and QtGui. ShowData will open a window containing the input data, and it inherits QWidget. InfoWindowLine and InfoWindowHistPie are in OW_info_window_line.py and OW_info_window_hist_pie.py. These files import all functionalities from QtWidgets and QtGui. InfoWindowLine and InfoWindowHistPie will open a window containing some information about the data, and these inherit QWidget.

Graph will set QGraphicsView, where the visualization will be drawn. Depending on the type of visualization, the graphics item will be defined in class GraphItemLine, GraphItemHist or GraphItemPie, located in GI_graph_item_line.py, GI_graph_item_hist.py and GI_graph_item_pie.py respectively. These files import all functionalities from QtWidgets, QGui and QRectF from QtCore. These also import my program’s class LineData or HistPieData depending on the type of visualization. GraphItemLine, GraphItemHist and GraphItemPie inherit QGraphicsItem, as the visualization will be drawn in these classes.

The computation about the data happens in classes LineData and HistPieData. These are located in DC_line_data.py and DC_hist_pie_data.py respectively. DC_line_data.py imports class LinkedList. LinkedList is in DS_linked_list.py, and there is also class ListNode, which is used in LinkedList. LineData, HistPieData and LinkedList will be discussed more in sections Algorithms and Data Structures.

Down below is an imperfect UML-model of my program drawn by draw.io. There are all the classes except for class Test. I only added associations between classes and some central methods, as I explained the structure well here.



6. Algorithms

Algorithms are mainly located in DC_histpiedata.py and DC_linedata.py, that calculate necessary information about the data. Those have classes LineData and HistPieData respectively. I have used some python's built-in algorithms for simple calculations and coded by myself algorithms for more complex calculations.

In HistPieData, the methods 'go_through_input(self, data)' and 'go_through_file(self, data)' are present, while in LineData, the methods 'count_max_min_input(self)' and 'count_max_min(self)' determine the maximum and minimum y-values. Additionally, the methods in LineData also find the maximum and minimum x-values of the data. These methods go through the data line by line and store the maximum and minimum values in variables. Subsequently, they compare these variables to the next y-value (and x-value) and update them if the current value is the new maximum or minimum.

In the same way, the methods in HistPieData determine the maximum and minimum x-values and store them in a list. This is necessary because there might be several x-values having the same y-value. In both files, these methods calculate the sum of data points. In HistPieData, it's calculated using Python's len() function, while in LineData, it's calculated by incrementing one on each iteration of the for-loop. In LineData, method 'get_as_list(self)' returns the data as python's built-in list. It goes through the data as a linked list and stores it as a list.

Method 'get_scaled_values(self)' in LineData stores scaled values of the data points as a linked list. This is necessary, because visualizing data in GI_graph_item_line.py needs data as coordinates, so it can be drawn in visualization. It ensures that the data's minimum and maximum values are mapped to the edges of the diagram. This is why the maximum and minimum values in both the x and y dimensions must be different; otherwise, division by zero would occur in the algorithm. I found this the best way to visualize the data, because it's simple to make and shows all the data points clearly.

In LineData, the method 'get_as_list(self)' returns the data as a Python built-in list. It iterates through the data stored as a linked list and converts it into a list structure. The method 'get_scaled_values(self)' stores scaled values as a linked list. This is necessary because GI_graph_item_line.py requires data in the form of coordinates to be drawn in visualization.

Both classes use simple sorting methods. Since my program deals with a small amount of data (up to 20 or 50 points), I didn't prioritize finding the fastest sorting algorithm. Instead, I chose to implement basic sorting methods. I went with selection sort.

In LineData, you'll find 'sort_by_x_increasing(self)' and 'sort_by_y_increasing(self)' methods that sort data into linked list. In HistPieData, there's a 'sort_increasing(self)' method that sorts data into a dictionary. All these sorting methods loop through the data several times equal to the size of the dataset 'n'. Each time, they find the maximum or minimum value and put it into the sorted list. This approach has a time complexity of O(n^2), which is acceptable for small datasets.

In LineData the axis numbers on y- and x- axis are calculated in the same way. In both, the first and the last ones are maximum and minimum values of the data. The values in between are values between minimum and minimum values, with equal distance from one another. Except for the second and second-to-last numbers, since they are closer to the first and last number. That’s because there is a ten-unit margin from the axis to the first and last number. In HistPieData y-axis numbers are calculated for histogram. They simply start from 0 and end to the maximum y-value and have eight numbers between with equal distances.

In both classes, LineData and HistPieData there are methods for calculating the mean y-value of the data, and in LineData also for mean x-value. Mean for y-value is used in both line diagram and histogram. Mean for x-value is used in computation for standard deviation for x-value. Mean is computed simply by summing all y-values of the data together and dividing the sum by number of data points.

Median for y-value is calculated in LineData and HistPieData. It’s used in line diagram and histogram. It’s computed by taking the value of sorted data, which has an index half of the length of data, in case the length of data is odd. In the case of even data, median is the mean of two middle values of sorted data.

Standard deviation for both y- and x-values are calculated in LineData. These are used in computation of correlation and linear regression, and they are computed in the regular way of computing standard deviation.

Covariance of x- and y-values are computed in LineData. It’s used in computation of correlation and linear regression. It’s calculated in the regular way of calculating covariance.

LineData has a method for calculating linear correlation. It’s computed by dividing covariance between y- and x-values by the product of the standard deviation of y-values and standard deviation of x-values.

The slope of linear regression model is computed by dividing the covariance of x- and y-values by variance of x-values. If the variance is zero, the slope will be zero. This is the solution for slope in the method of least squares.

The intercept term of linear regression model is computed by reducing the product of the slope and mean of the x-values from the mean y-value. This is the solution for intercept in method of least squares.

In LineData class, there is a method for computing the starting and ending point of linear regression line. There should be two points which lie on the edge of the diagram, and the points should be on different edges, since the program requires that maximum and minimum values for both x and y should not be the same. The method goes through all the four edges of the graph. If the linear regression line crosses that edge, it will add the point of crossing to the points-list. The graphics item for line diagram will draw a linear regression between these points.

Both LineData and HistPieData have methods for rounding numbers. It will round numbers that have absolute value from thousand to 9999999 to four significant digits. Numbers from absolute value of 0.00001 to 999 will be rounded to three significant digits. In the case absolute value of number is under 0.00001, it will be rounded in seven decimal places.

Both LineData and HistPieData have methods for computing the length of a single number. It’s used for rounded axis numbers. It’s necessary, because the length of a number affects the place in the graph where the number is placed, so for example the number doesn’t overlap the axis.

HistPieData has a method for calculating the total sum of y-values. It’s used in pie diagram. The sum is rounded by round_num method.

7. Data structures

In my program, input data is stored differently depending on the type of visualization. In the case of line diagram, if data is written manually, it’s first stored as a two-dimensional array. In class LineData, it will be stored as a linked list. When the data is an input file, it is stored as a file until class LineData stores it as a linked list. I made a linked list, because it would be better with a bigger amount of data. Since my program ended up accepting only quite a small amount of data, it doesn’t have significant advantages. I made some of the methods in LineData to use two-dimensional array because it was easier in some cases than linked list. So, there is a method in LineData making a linked list as an array.
Class LinkedList uses class ListNode to store the data. ListNode is one data point in linked list. Adding points in linked list happens in method add_last in LinkedList. It places the new data point at the end of the list, using class ListNode’s method add_after. LinkedList has other methods, get_size, get_at_position and remove_position. I used a linked list which I made in TRAK Y- course and modified it by removing some unnecessary methods.
If the user has chosen a histogram or pie diagram, and if the data is written manually, it will be first stored as a two-dimensional array. In CheckInputHistPie it will be stored as a python’s built-in dictionary for the rest of the program. Dictionary is good for this kind of data, because the amount of data is small, and it doesn’t accept same keys. Data point’s value and frequency are different types, so dictionary is a natural choice. If the input data is a file, class HistPieData will store it from file to dictionary. Also, python’s build-in array is used in class HistPieData, if some method needs only the values or frequencies of the data.

8. Files

This program accepts text files, and the format of it is given in my program's instructions. Files aren't necessary, because the program allows the user to write the data while running the program.

9. Testing

I had planned that I would test the geometry of some items. That I didn't find necessary, because I could easily see those kinds of things from the code. My tests mostly focus on algorithms and data structures. I made unit tests for the classes that had lots of algorithms and data structures, like LineData and HistPieData, and tried to do some unit tests for other classes too. My tests don't test everything, but I think I implemented the most important tests.

10. The known shortcomings and flaws in the program

I think the biggest issue in my program is the structure. My methods aren't structured that well between classes. For example, my class MainWindow has some methods that don't certainly belong in it. In the beginning I didn't know where to place some methods, so I tested if it worked in some class, and then I just left them there. After finishing my program, I didn't want to change those anymore, because I thought it would get complicated, and it doesn't have effect on executing the program.

My program only throws a general error message when a wrong kind of data is given. I have quite a lot of rules for input data, so I didn't a find right way to show more specific error message(s). If I continued the program, I would like to print the wrong data for the user and highlight the data points including error. Now my program just throws an error message after seeing the first error in the input data. If I wanted show user specific error messages, I would like to show all the errors in data, not just the first line. Instead of showing the errors, I decided to make an instruction window, that should have clear instructions for input data.

11. 3 best and 3 worst areas

I think that the best three things in my program are that it runs in one window, usability and visual extensions. I think it's better for the user that the program is executed in one main window, as many more advanced software is created this way. Also, users can go back without crashing the program, which I think improves the user experience. My program has diverse usability, because it's possible to add data as a file or write the data. Input data has certain format, but it can vary a lot. There are a lot of visual extensions in my program, which I think are great. Depending on the type of visualization, user can for example see

the mean and median values in the visualization, add a linear regression line and sort the data.
The worst areas in my program are that the structure of methods and classes is a bit messy, my program only throws general error message, and my code is not that well commented. The first two things I discussed in “The known shortcomings and flaws in the program”, so I won’t add here anything to that. My code has at least one comment per method, but I think there should be more. I should’ve commented on the code more while programming.

12. Changes to the original plan

I made only small changes to the original plan, except for the extension that accepts input data written manually during execution. In my program user can’t choose the color of visualization, because I started to think it’s not necessary. My program will find out the maximum and minimum values of the data, but there isn’t a button for those. Those values are in the info window. I used selection sort algorithm instead of insertion sort, because it ended up being easier to make.

I planned that there would be several main windows, but I decided one is more suitable for this kind of program. In my plan I have written that I will only use python’s built-in data structures. Instead, I decided to use a linked list, because it’s more flexible data structure. Users can name the headline and possibly axis only after the visualization is drawn. I planned that users could do this at the same time writing the data. While programming, I thought that it’s better the naming happens after drawing the visualization.

I had made a schedule in my plan, but I ended up being faster than I expected. In my schedule I had planned that I would work on this approximately ten hours a week. Instead of that, on some weeks I worked on this project for twenty hours and on some weeks only for like five hours. That suited me well and I didn’t need to hurry at any point.

13. Realized order and scheduled

I coded first and coded a lot between 26.2 and 2.4. More exact dates are in GitLab. I started with the menu window. After that I created File window, but only ‘Select file’ button (I created ‘Write data’ button after I had made all the visualizations). Then I started working on Graph window and line diagram. First, I made a line diagram and the central calculations for that. After the line diagram started working, I started working on histogram and pie diagram. Finally, I made some non-mandatory extensions, like info window and linear regression line. After that I made only some small changes and cleaned the code. I started to make the tests 25.3. After finishing the code, I started writing this document. Lastly, I did the read me -file.

14. Assessment of the final result

I think my program is working well for its purpose. The visualizations are clear, and it has some useful extensions. It's easy for the user to use. Detailed error messages can be easily implemented, but it requires some time. Also, the structure can be fixed easily. In the case of wanting to add more data points to line diagram, some algorithms need to be redo. That's because the computations would be too time or space consuming. Linked list is a great data structure for larger data. There is no reason for histogram or pie diagram to accept too large data, because the visualization wouldn't be good anymore. So, the algorithms for those are ok.

Overall, my program is good for visualizing kind of small amounts of data. It's nice to use, because the program is not supposed to crash, and the user doesn't need any files to use the program. Users can see the main features of the data, which vary between visualizations. Maybe the ability to save the visualization would be good extension to add in the future.

15. References

A+ -materials:

CS-A1121 Ohjelmoinnin peruskurssi Y2: https://plus.cs.aalto.fi/y2/2024/toc/

CS-A1111 Ohjelmoinnin peruskurssi Y1: https://plus.cs.aalto.fi/y1/2022s/

CS-A1141 Tietorakenteet ja algoritmit Y: https://plus.cs.aalto.fi/a1141/2023/

How To Use QFileDialog To Select Files In PyQt6, publisher: @jiejenn, https://www.youtube.com/watch?v=V_TU0eCOVP8

Making File Dialogs In PyQt6 Using QFileDialog! PyQt6 Tutorial Part 7, publisher: @TurbineThree, https://www.youtube.com/watch?v=aiEz1G2-Mpg

https://whiteboard.office.com: Y2histgraph.png, Y2linegraph.png, Y2piegraph.png

Open AI Chat gpt: colors (graph_item_pie.py self.color_list, "Name 20 different kind of PyQT6 QColors")

MS-C1620 Statistical Inference, 2024, lecturer: Pekka Pere:

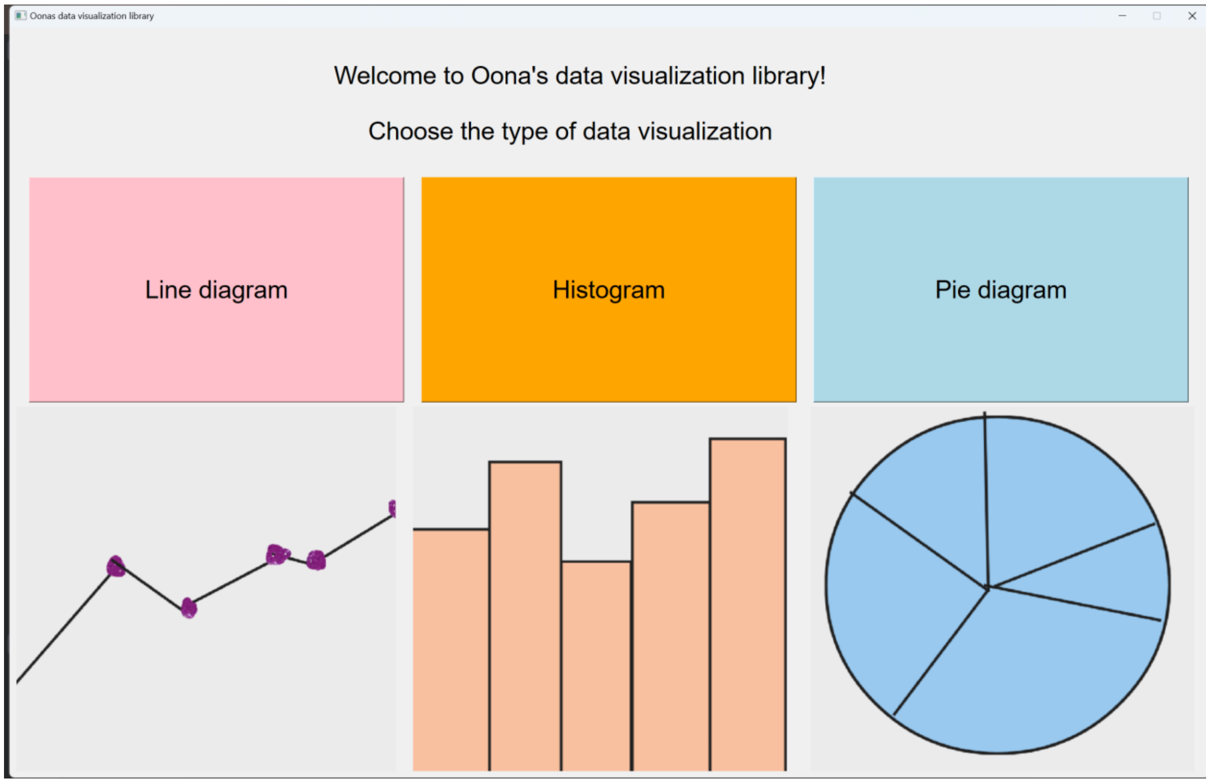
Lecture 6 Correlation and dependence:
https://mycourses.aalto.fi/pluginfile.php/1998237/course/section/240144/SI_lecture_6.pdf?time=1708015664905

Lecture 7 Linear Regression 1:

https://mycourses.aalto.fi/pluginfile.php/1998237/course/section/240144/SI_lecture_7.pdf?time=1709205874235

16. Attachments

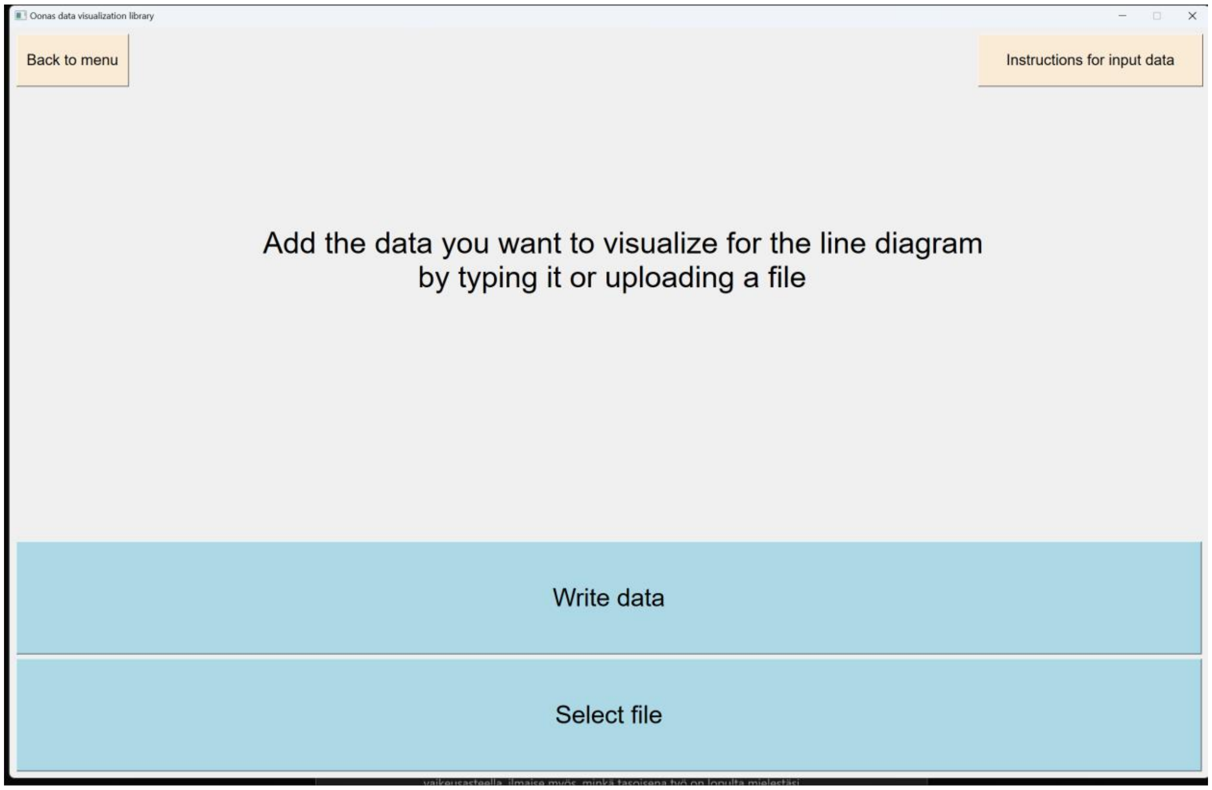
Menu:


Oonas data visualization library

Welcome to Oona's data visualization library!

Choose the type of data visualization

If ‘Line diagram’ is clicked:



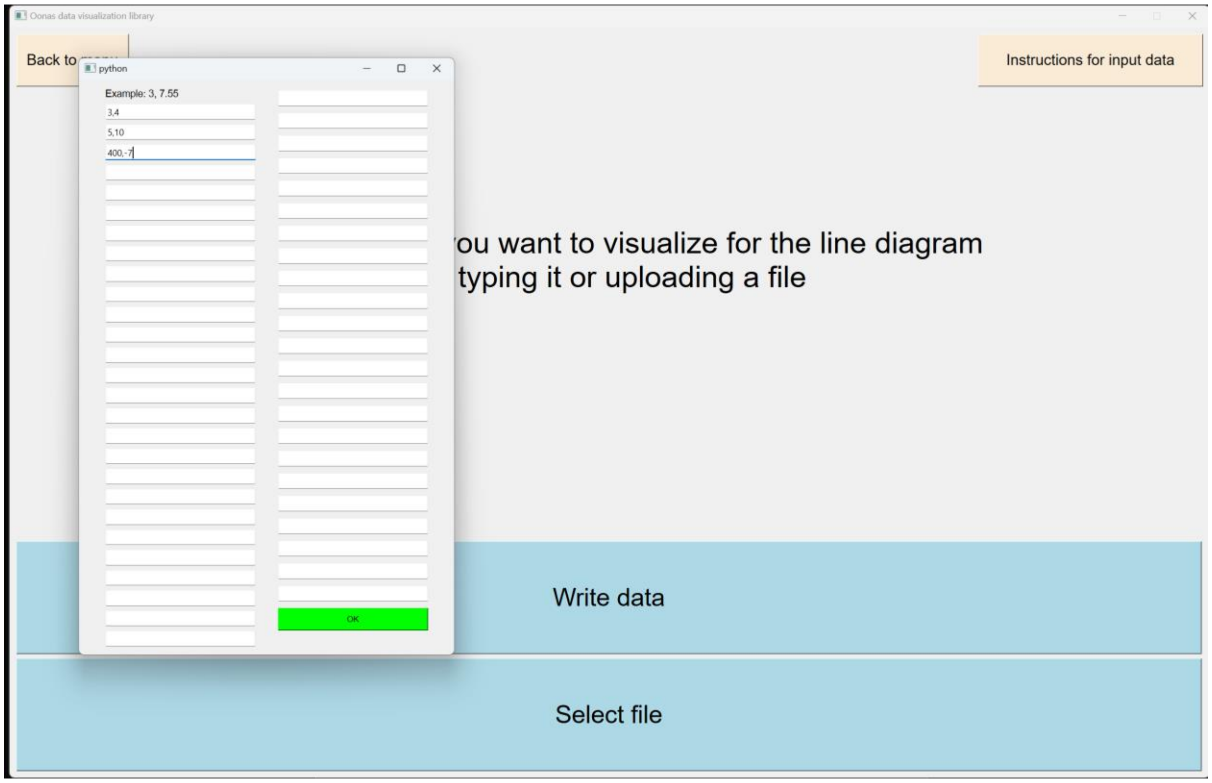
Oonas data visualization library

Back to menu

Instructions for input data

Add the data you want to visualize for the line diagram
by typing it or uploading a file

‘Write data’ is clicked:



Oonas data visualization library

python

Example: 3, 7.55

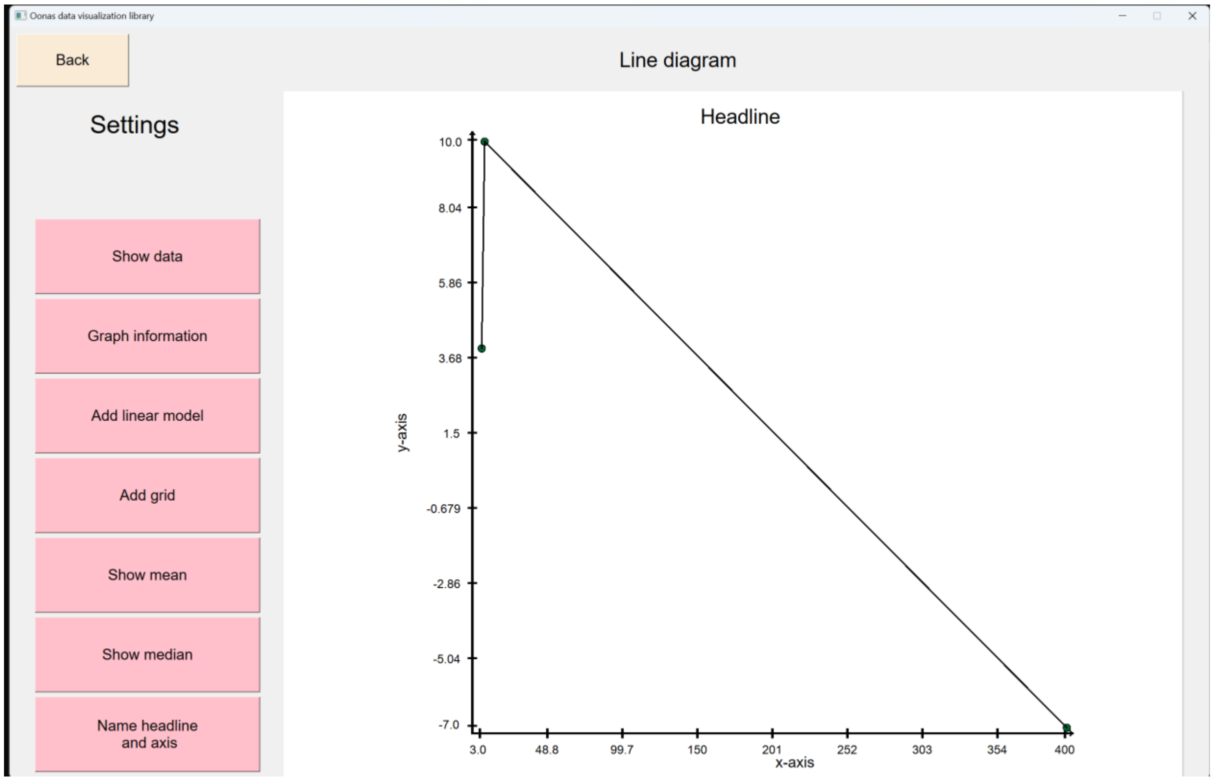
3,4

5,10

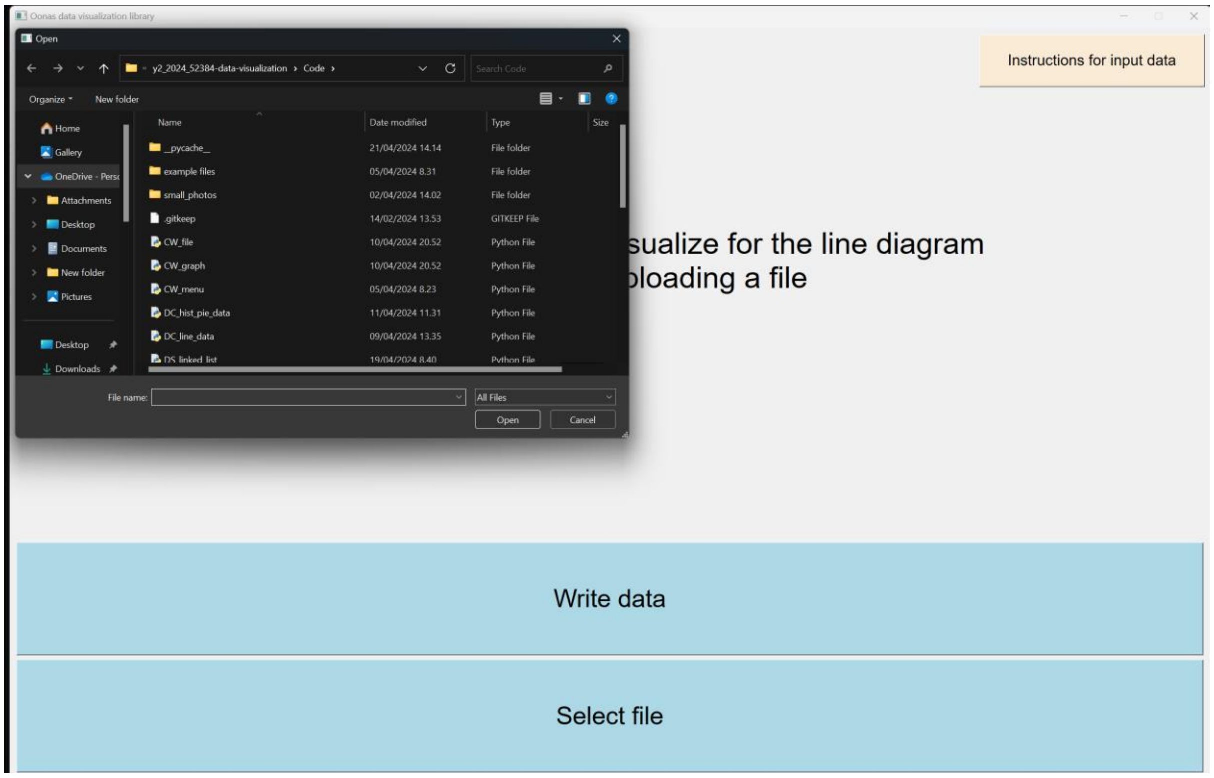
400,-7

Back to
Instructions for input data
OK
ou want to visualize for the line diagram
typing it or uploading a file
Write data
Select file

‘OK’ is clicked:

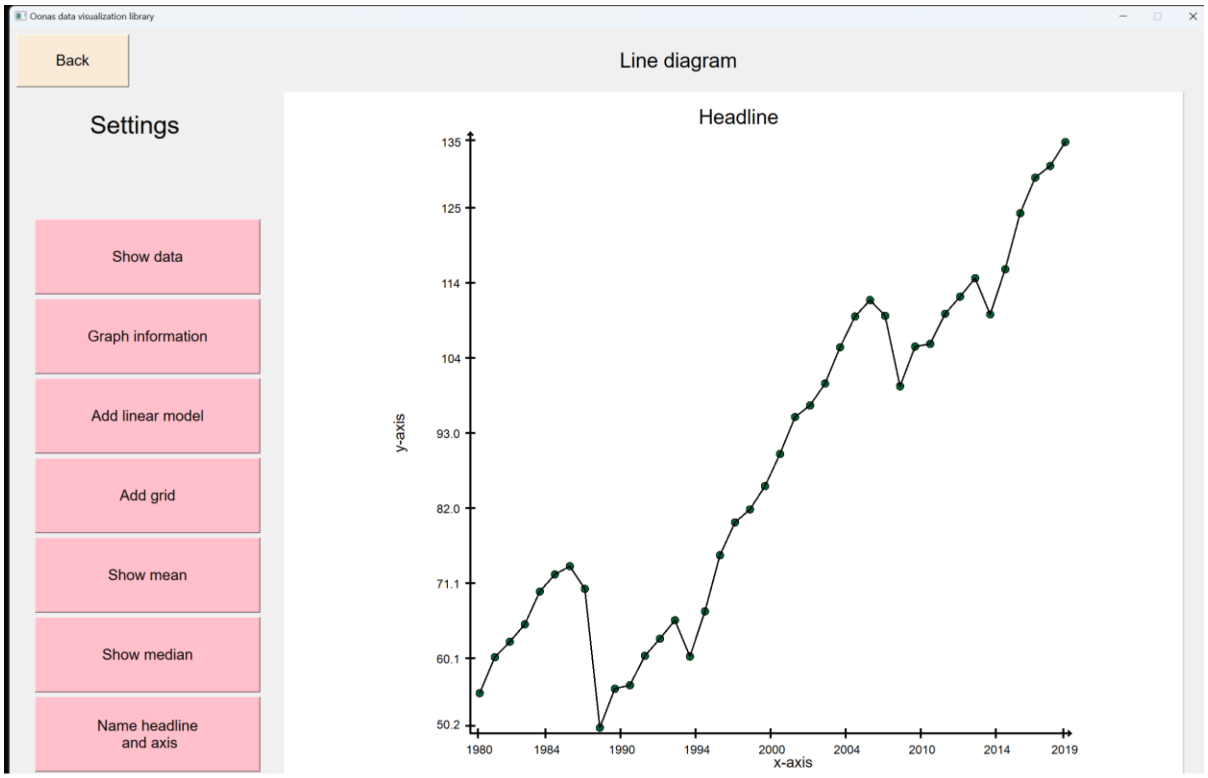


Then we click ‘Back’ and click ‘Select File’:

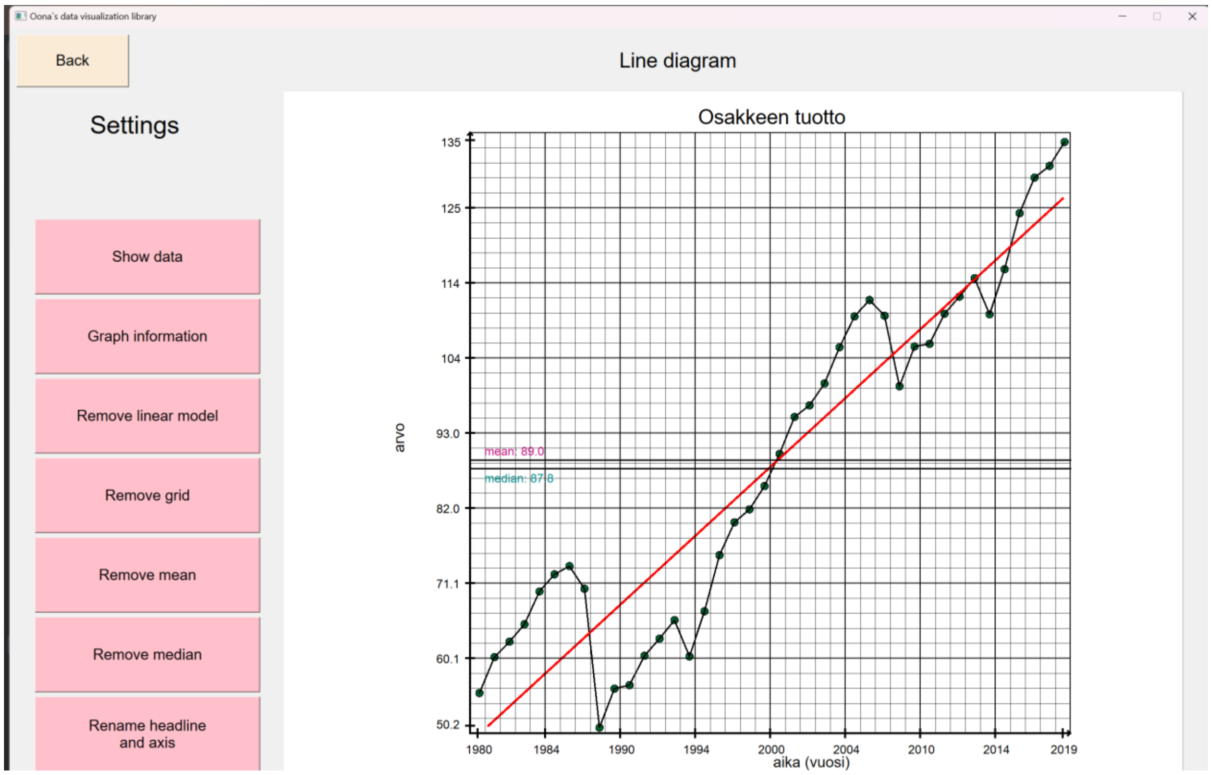


Name	Date modified	Type	Size
pycache	21/04/2024 14.14	File folder	
example files	05/04/2024 8.31	File folder	
small_photos	02/04/2024 14.02	File folder	
.gitkeep	14/02/2024 13.53	GITKEEP File	
CW_file	10/04/2024 20.52	Python File	
CW_graph	10/04/2024 20.52	Python File	
CW_menu	05/04/2024 8.23	Python File	
DC_hist_pie_data	11/04/2024 11.31	Python File	
DC_line_data	09/04/2024 13.35	Python File	
DS_linked_list	19/04/2024 8.40	Python File	

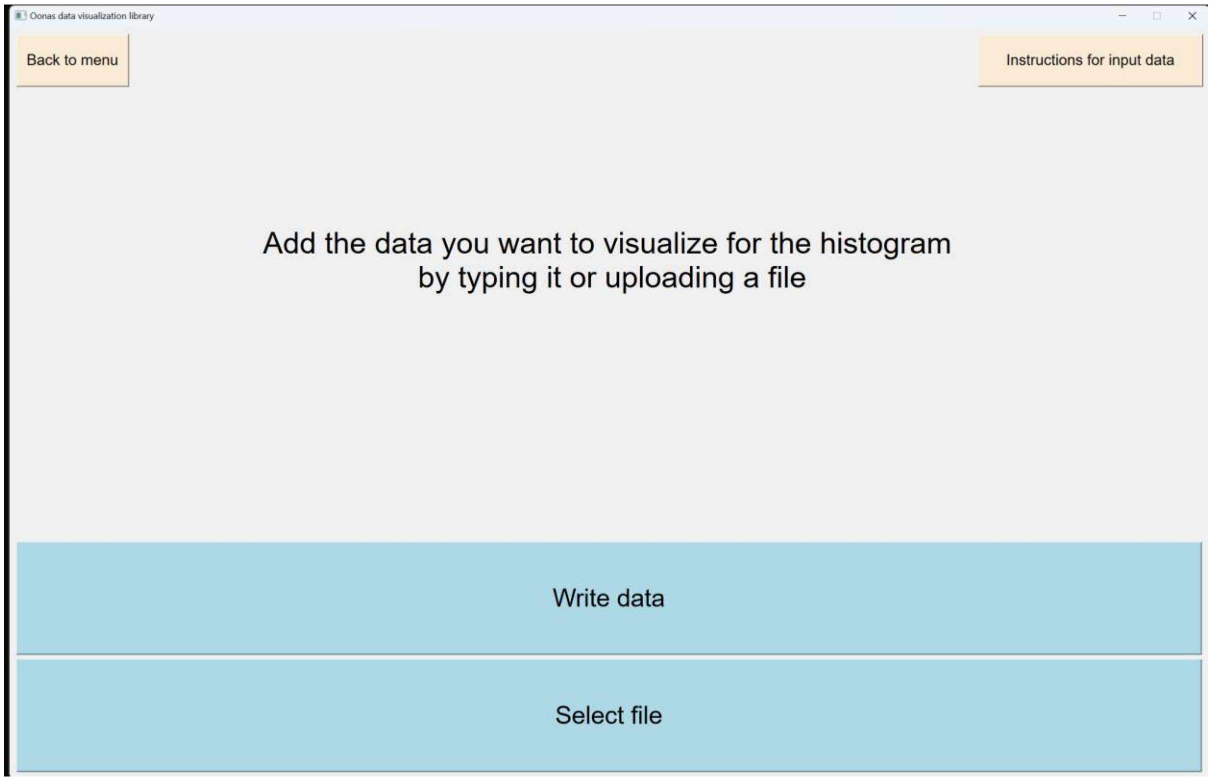
Then we select a file that suits for line diagram:



With visual extensions are clicked on, it will look like this:

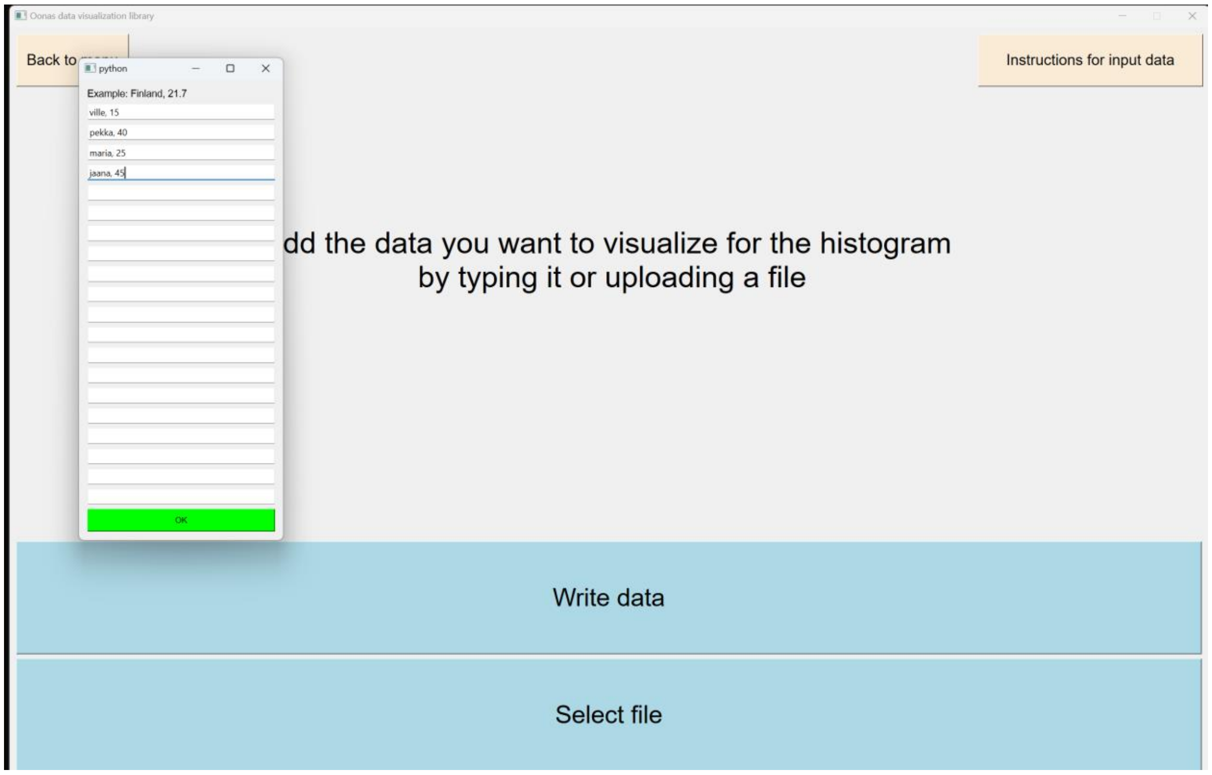


Then we click ‘Back’ and then we click ‘Back to menu’ and click ‘Histogram’:



Add the data you want to visualize for the histogram
by typing it or uploading a file

Then we click ‘Write data’ and write some data:

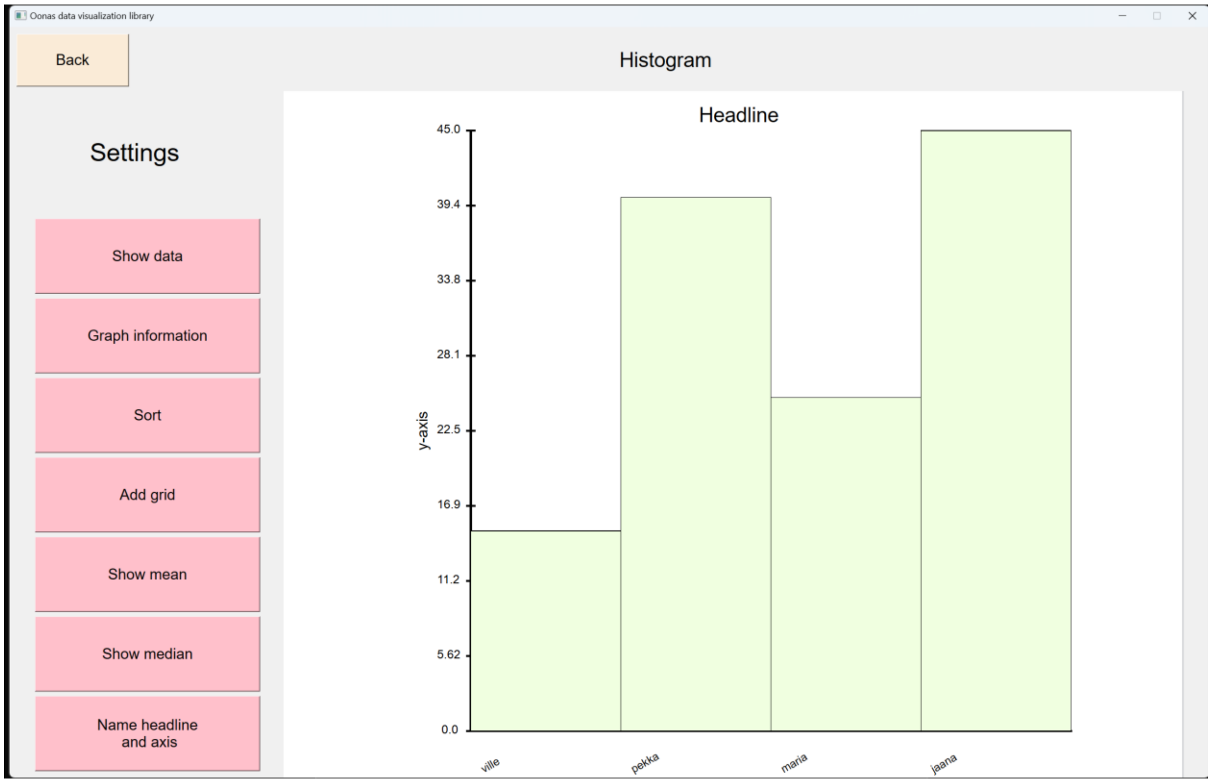


Example: Finland, 21.7

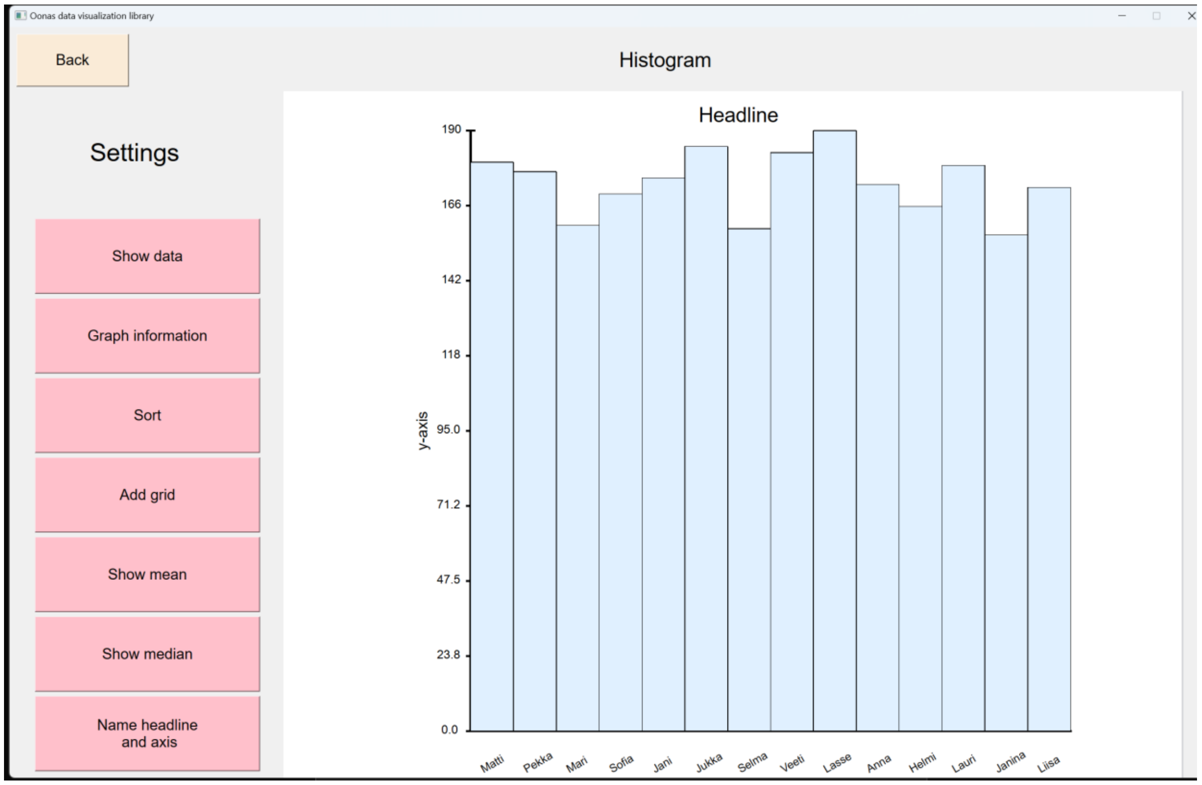
ville, 15
pekka, 40
maria, 25
jaana, 45

dd the data you want to visualize for the histogram
by typing it or uploading a file

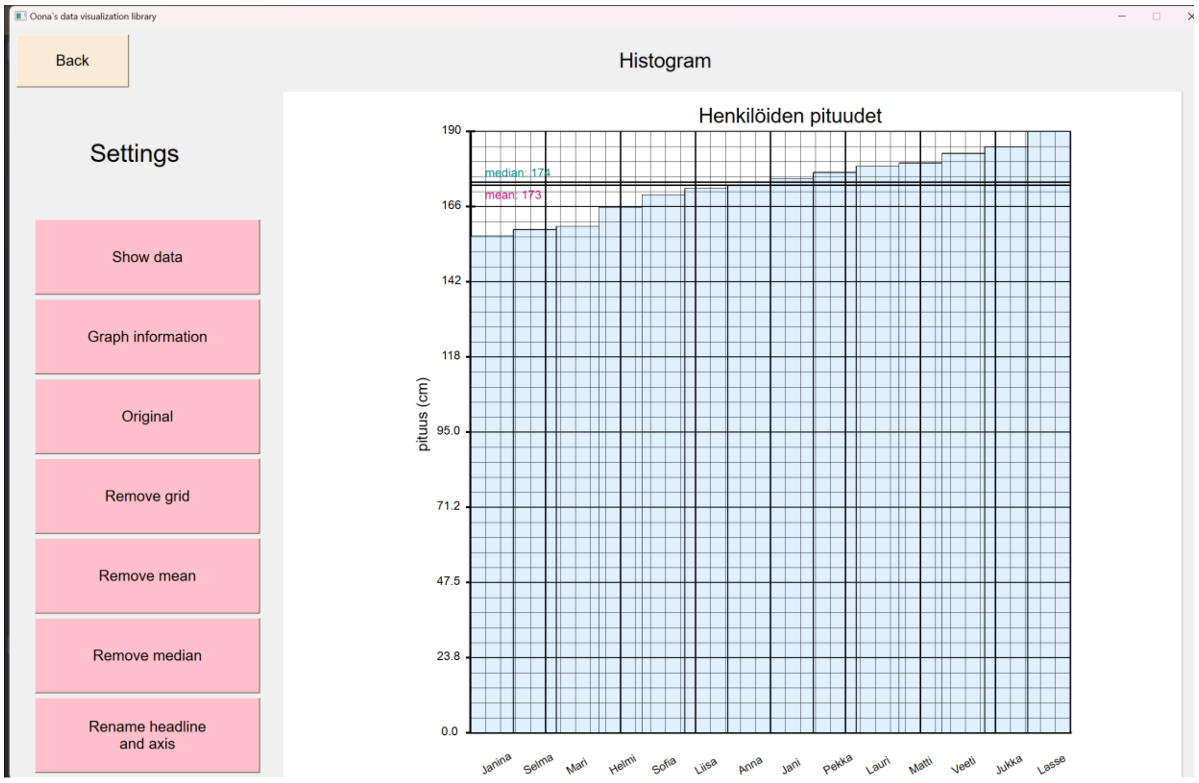
Then we click ‘OK’:



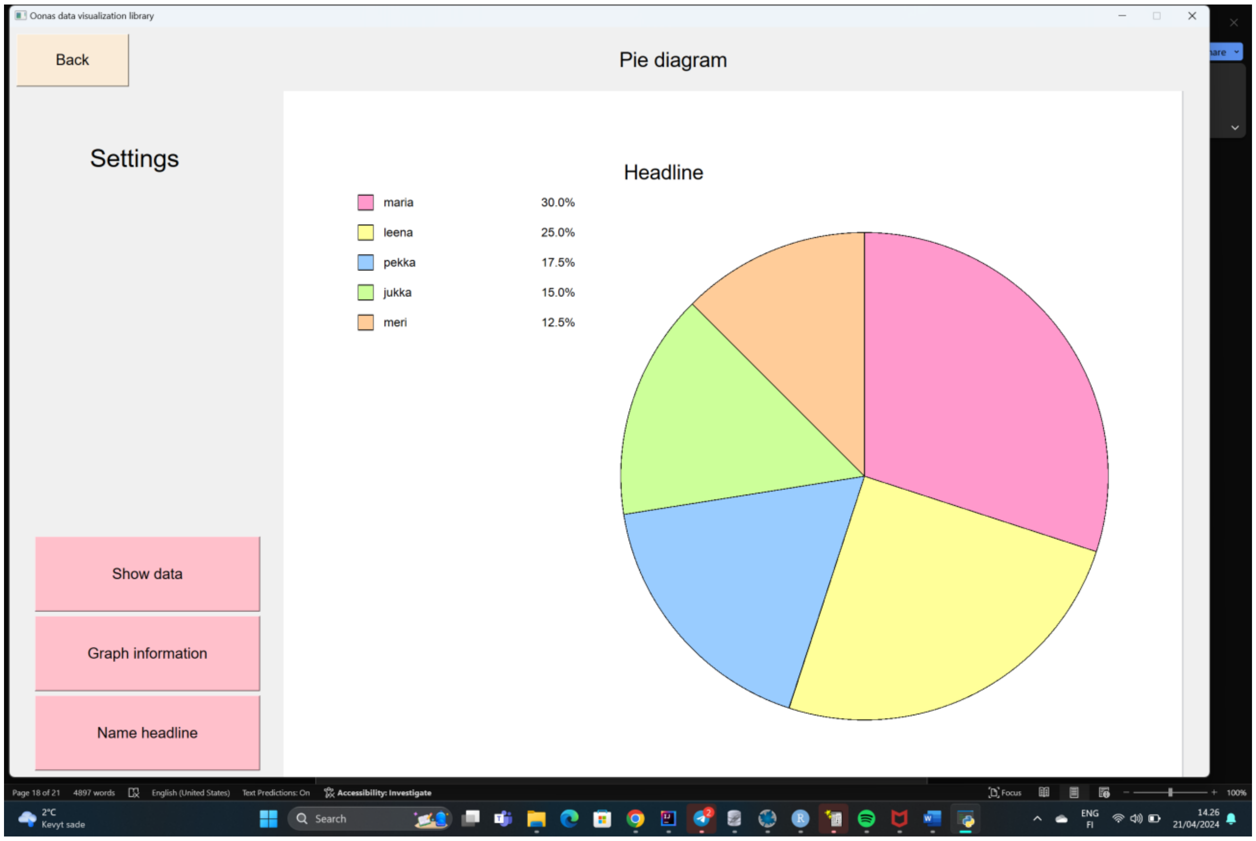
Then we click 'Back' and click there 'Select file' and select a file that suits for histogram:



And it will look like this when all the extensions are on:

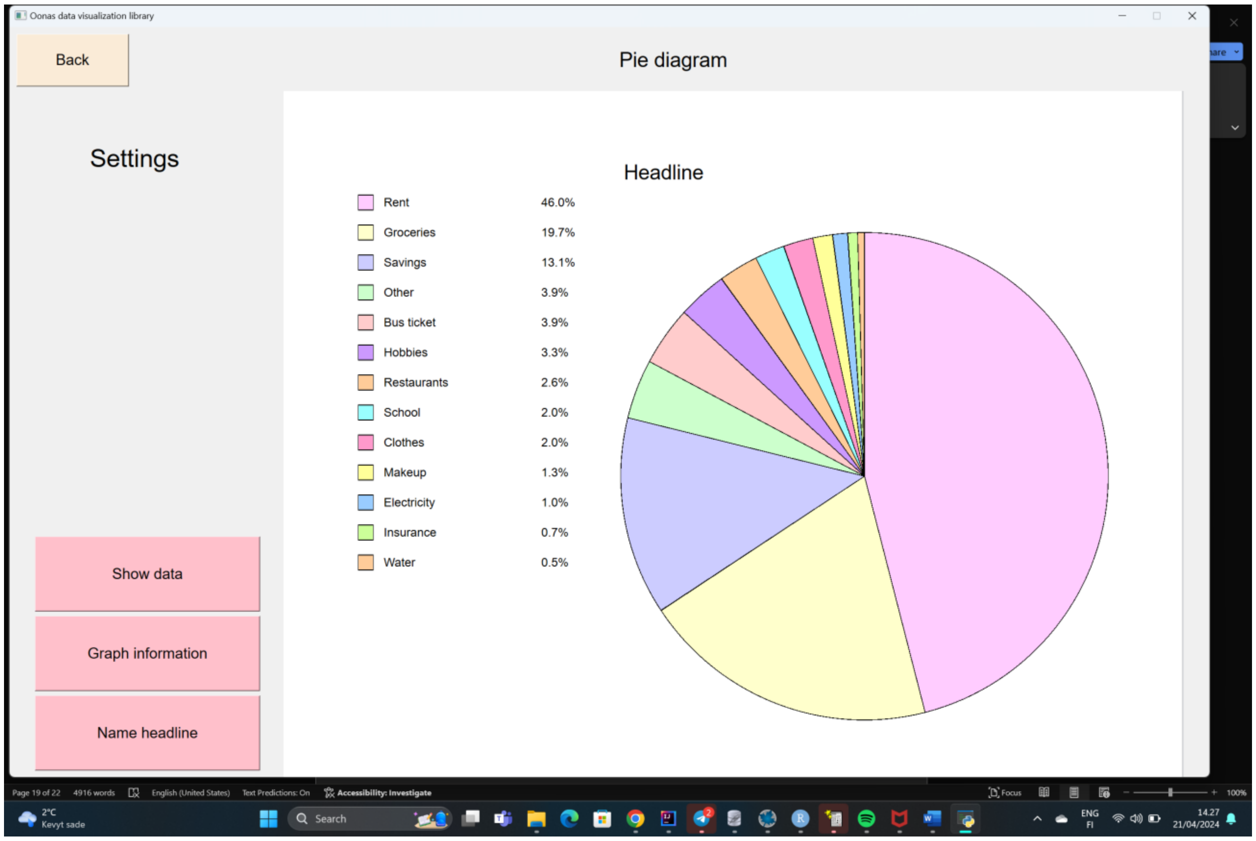


Then we click ‘Back’ and ‘Back to menu’ and from menu we select a pie diagram. Then we click ‘Write data’ and write some data that suits for pie diagram and click ‘OK’:



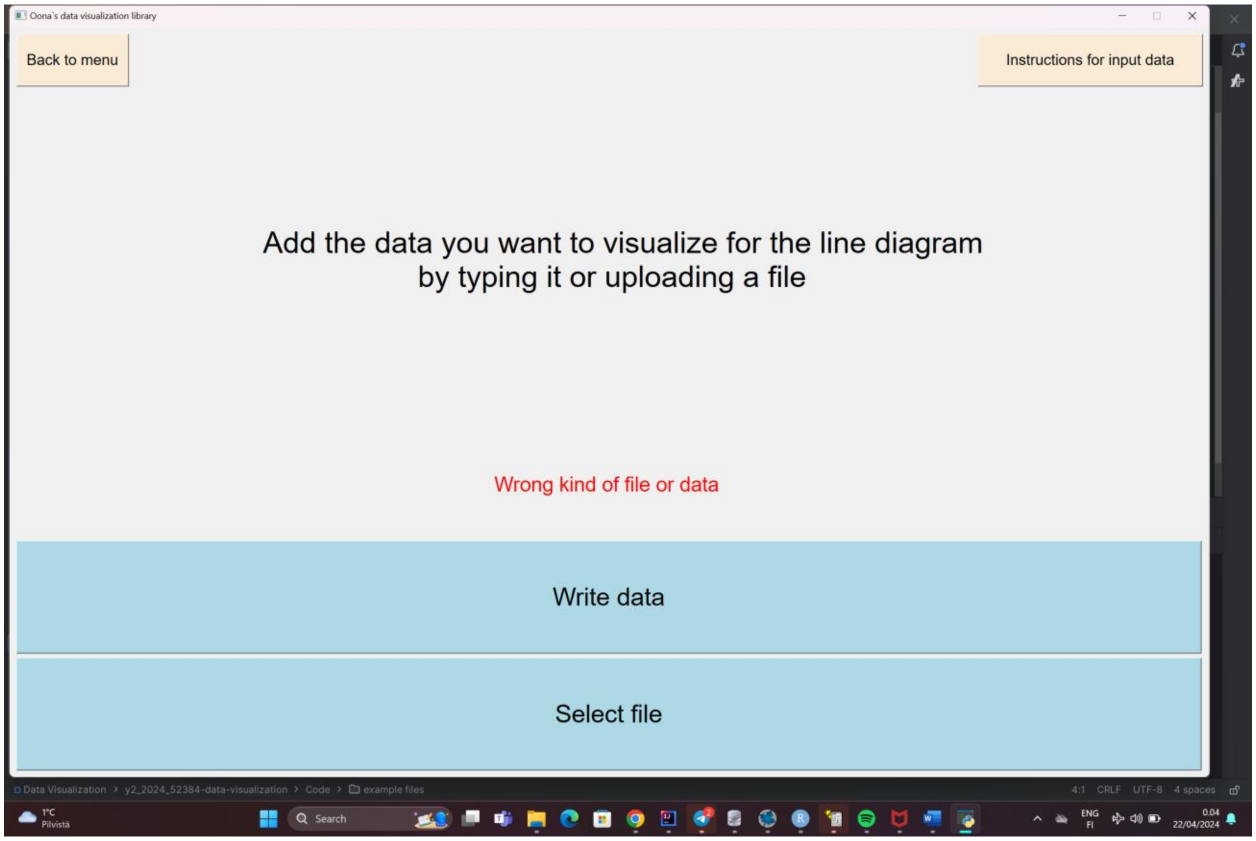
🟪	maria	30.0%
🟨	leena	25.0%
🟦	pekka	17.5%
🟩	jukka	15.0%
🟧	meri	12.5%

Then we do the pie diagram by clicking ‘Select file’ and selecting a file that suits for pie diagram:



Rent	46.0%
Groceries	19.7%
Savings	13.1%
Other	3.9%
Bus ticket	3.9%
Hobbies	3.3%
Restaurants	2.6%
School	2.0%
Clothes	2.0%
Makeup	1.3%
Electricity	1.0%
Insurance	0.7%
Water	0.5%

In case input is not in correct format:


Oona's data visualization library
Back to menu
Instructions for input data

Add the data you want to visualize for the line diagram
by typing it or uploading a file

Wrong kind of file or data