Turbo Data Analysis Machine- Instructions and the Process to Create it

Noah Urbanovsky

Thank you for choosing the Turbo Data Analysis Machine, ready to assist in all of your data analysis needs. This short guide will tell you how to get the most out of your time using it, in order to analyze data like a pro!

Upon opening the program through the file labelled main.m, a text-based menu will appear in the command window, giving eleven different options. You can decide which option you need by typing the corresponding number to the choice in the command window. The choices are as follows:

1. Set User Name: The program will ask you to enter your name; simply type it into the command window. The name given will be applied to any output files created to show it’s your file. If no name is given prior to the output file being created, “Name” will take its place, though this can be edited from the text file.
2. Load Data File: Upon selection, a separate window will open up that will allow you to pick any .txt file to load into the program. The file can only have one or two columns of data (only numbers), and if there are two columns, they must be the same length. Otherwise, many of the analyzing features would not work. Upon loading a file, some basic analysis of the data will be printed in the command window, including the mean, median, mode, variance, standard deviation, minimum and maximum values, and number of values for each column. This will also create a new .txt file as an output and add this analysis to it, along with any future calculations. As a warning, if this is run with the title of an existing file, it will replace that file, so be careful! Additionally, if there is already a loaded file, it will not let a new one load, so you must delete the old file first.
3. Clear Data from Memory: Clears all data, including the loaded file, output file’s name, username, and custom x and y axis titles for graphs. The output file created will still be present.
4. Set Output Filename: Upon prompt, input a title for the output file. This will change the output file of any future files to that name. This will also change the title of plots. You must also type the x and y labels, which will affect all plots.
5. Plot Histogram: Plots a histogram of the current data.
6. Plot Histogram Fit: Plots a histogram of the current data, along with a best fit curve.
7. Plot Probability Plot: Plots a probability plot.
8. Regression of Y on X: The user must input the degree of regression that they would like to use. The program then creates a plot containing each data point along with a best-fit line based on the regression. This feature will only work if the data given contains two columns.
9. Find Probability Given X or Z: You will be prompted to enter an x or z value and asked whether they entered an x or z value. Answer whether the data appears reasonable; the calculations will occur regardless, and this answer will be added to the output file. The probability and percentage will be calculated, presented in the command window, and added to the output file.
10. Find X or Z Given Probability: You will be prompted to give a probability and asked whether the data appears reasonable (like before, this doesn’t affect calculations). The x and z values will be calculated, presented in the command window, and added to the output file. The probability must be between 0 and 1; if it’s more than 1, it’s probably a percentage and should be divided by 100.
11. Exit: Ends the program.

That’s all you need to know in order to properly use this program and begin analyzing data like a pro! So get going- there’s tons of data just waiting to be analyzed!

In order to make this program possible, a lot of testing and debugging was done to make sure it runs smoothly and gives accurate results. One of the most challenging parts, oddly enough, was making sure the data could be properly loaded and that it would check to make sure the data was usable. In order to do this, I first had to make sure any files loaded were only .txt files. Thankfully, the command importdata(uigetfile('.txt')) provides this restriction without any hassle. Once that was solved, the program had to be able to figure out if a matrix is only made of numbers and that every column had the same length. Fortunately, the program can turn the file given into a matrix and this easily reveals empty spaces as NaN values, which are easy to spot using the isnan function. Since the data has to be all numbers, this made it clear whether a given file can even be considered. Finally, the file could only contain one or two columns of data, and this was very easy to check by checking the number of columns of the newly created matrix. (This matrix is what’s used for the rest of the program, in fact). For repeat loads, it first checks to see if the variable “data,” which holds the matrix form of the file, has anything in it already. If it does, then it cancels the loading process.

Once the data can be obtained as a matrix, the rest of the code can work successfully. Setting the username, output filename, and x and y labels is very simple, as each just take an input string (that all start with defaults). Clearing data is also very easy, as everything is set to its default value or string, and the data variable is reset to hold nothing (data = [];). The three functions that create plots simply use their respective commands (histogram, histfit, and probplot) and a few commands for adding labels and titles.

The regression of y on x is a little more complicated. It uses polyval to create the coefficients for the new equation using the x (the first column), y (the second column), and the degree given by user input. The program then creates a new, more comprehensive array of x values, called xfit, with linspace, then creates a corresponding array of y values, yfit, using polyval. Finally, the original x and y values are plotted as points and the xfit and yfit values as a line, creating the regression line. The only requirement is that there are an equal amount of x and y values, but because the function for loading the data checks to make sure the file is made up of two equally-sized columns, this requirement is already taken care of.

Calculating probability from x or z appears tricky until you realize that, using the normcdf value, corresponding x and z values give the same probability. Thus, it’s easier to just ask for a value and let the user specify if it’s an x or z value separately. The normcdf function gives the probability and multiplying that by 100 gives the percentage.

Going the opposite way, it made sense to just give the x and z values together instead of just one or the other, since both are useful together. The z value is found using the input probability in the norminv function, and the corresponding x value is given by the equation z = (x - μ)/ σ (rearranged to x = (z\* σ) – μ). Probability can only be between 0 and 1 in normal cases, so it checks for that and gives an error if the input doesn’t match that requirement.

For all inputs, it’s important that an accidental bad input, be it letter or just pressing enter, doesn’t crash the program. For that, I employed try-catch structures to stop letter-based inputs and the isempty function to avoid empty inputs. That way, the program can either just ask again, or add it’s own general values to avoid derailing the whole program.

In order to check the ability to load files properly, I personally created a couple small files of data (usually around 6 rows) that check for all of the warning scenarios. I created one with only one column, one with two equal columns, one with two unequal columns, and one with three columns. This allowed me to check all barriers for loading files, including only allowing files with one or two columns and only allowing files with columns of equal length. In order to make sure that the files weren’t filled with text or or that the program didn’t crash if the user cancelled out of file selection, I used a try-catch structure here as well so that it would simply go back to the main menu without ruining the whole program. Additionally, the TA’s were kind enough to provide us with files filled with data in different forms that would let us test all of the plots and calculations properly. (If you guys are reading this, then thank you very much!) Creating histograms was very simple so not much was needed to check those for accuracy. For the regressions, normal plotting was also very straightforward, so it could be used as a reference for the shape of the best-fit line created from its regression function. For the problems related to probability, x values, and z values, since the two functions possible were related (find probability and find z/x), it’s possible to check accuracy by finding each other with their respective values (ex. pick a z or x value to find probability, then find z and x using that probability and see if it’s the same as the x or z value you had before). Overall, I did many trials in order to make sure that this program is successful and that anyone can be a pro at analyzing data!