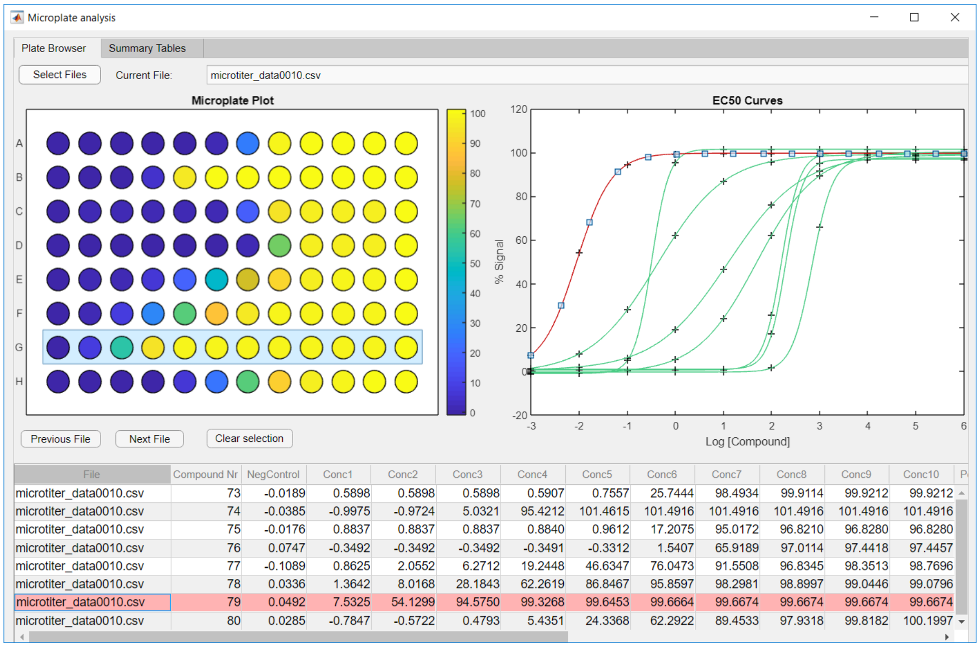
Fitting Dose Response Curve

Table of Contents

This is an example of how MATLAB can be used to facilitate a typical workflow for an experimentalist who needs to automate analysis across many (hundreds) of experiments with instrument data organized in flat files, TXT, CSV files, or spreadsheets. In this example, we want to find the top leads from hundreds experiments. The data are in CSV files from a Dose-Response Experiment in a 96-well plate format. Each row of data corresponds to a serial dilution (middle 10 wells) of a given compound with a negative control (first column) and positive control (last column. The dose response readings in each microwell are in arbitrary units (e.g. fluorescence, luminescence, absorbance) ranging from ~0 to ~100(%). We fit the experimental data to a four parameter model and add a quality control metric drawn from the scientific literature. In the A_Microtiter_Plate_One file we prototyped our workflow. In this script we are calling that workflow in a loop. The final result is a table of compounds sorted in decreasing potency and filtered by a quality control metric. This also includes a report, and a spreadsheet file.

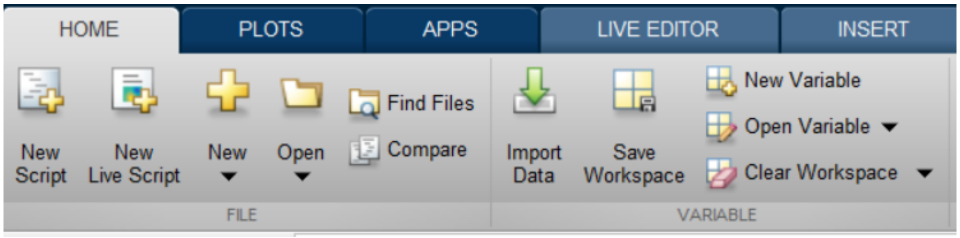


Current File: microtiter_data0010.csv

File	Compound Nr	NegControl	Conc1	Conc2	Conc3	Conc4	Conc5	Conc6	Conc7	Conc8	Conc9	Conc10
microtiter_data0010.csv	73	-0.0189	0.5898	0.5898	0.5898	0.5907	0.7557	25.7444	98.4934	99.9114	99.9212	99.9212
microtiter_data0010.csv	74	-0.0385	-0.9975	-0.9724	5.0321	95.4212	101.4615	101.4916	101.4916	101.4916	101.4916	101.4916
microtiter_data0010.csv	75	-0.0176	0.8837	0.8837	0.8837	0.8840	0.9612	17.2075	95.0172	96.8210	96.8280	96.8280
microtiter_data0010.csv	76	0.0747	-0.3492	-0.3492	-0.3492	-0.3491	-0.3312	1.5407	65.9189	97.0114	97.4418	97.4457
microtiter_data0010.csv	77	-0.1089	0.8625	2.0552	6.2712	19.2448	46.6347	76.0473	91.5508	96.8345	98.3513	98.7696
microtiter_data0010.csv	78	0.0336	1.3642	8.0168	28.1843	62.2619	86.8467	95.8597	98.2981	98.8997	99.0446	99.0796
microtiter_data0010.csv	79	0.0492	7.5325	54.1299	94.5750	99.3268	99.6453	99.6664	99.6674	99.6674	99.6674	99.6674
microtiter_data0010.csv	80	0.0285	-0.7847	-0.5722	0.4793	5.4351	24.3368	62.2922	89.4533	97.9318	99.8182	100.1997

```
clear; close all; clc;
```

Step 1: Access Data in MATLAB



Use the **Import Data** option from the toolstrip to bring in your data for analysis or use the **`readtable`** function

```
data = readtable('data.csv')
```

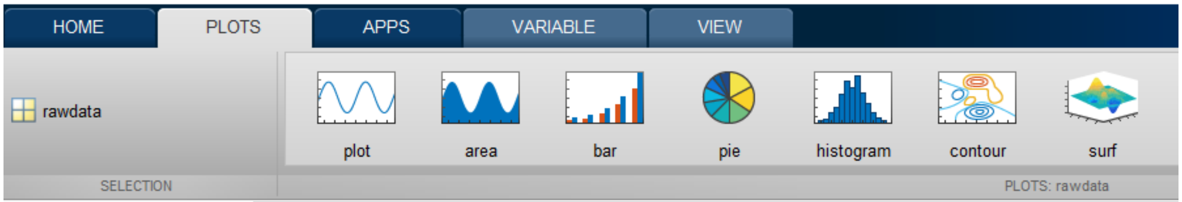
```
data = 8×12 table
```

...

	NegControl	Conc1	Conc2	Conc3	Conc4	Conc5	Conc6	Conc7
1	-0.0937	0.3389	9.9060	57.1390	92.1710	97.1720	97.6190	97.6570
2	-0.0336	-0.5264	-0.5264	-0.5264	-0.5264	-0.5264	-0.5263	-0.4762
3	-0.0137	-0.4983	3.2407	53.9460	99.3890	102.3700	102.4800	102.4800
4	0.0905	0.2136	0.2196	0.3525	3.2192	41.1440	94.8470	100.7800
5	-0.0775	-0.7785	1.7212	27.3410	85.9810	100.9600	102.2200	102.3100
6	0.0473	-0.0047	0.1516	1.9478	19.0260	73.4120	97.0630	99.8060
7	-0.0149	-0.3653	-0.3653	-0.3653	-0.3653	-0.3545	9.0751	100.0300
8	-0.0584	0.8869	0.8869	0.8870	0.9435	21.6330	101.4800	102.3300

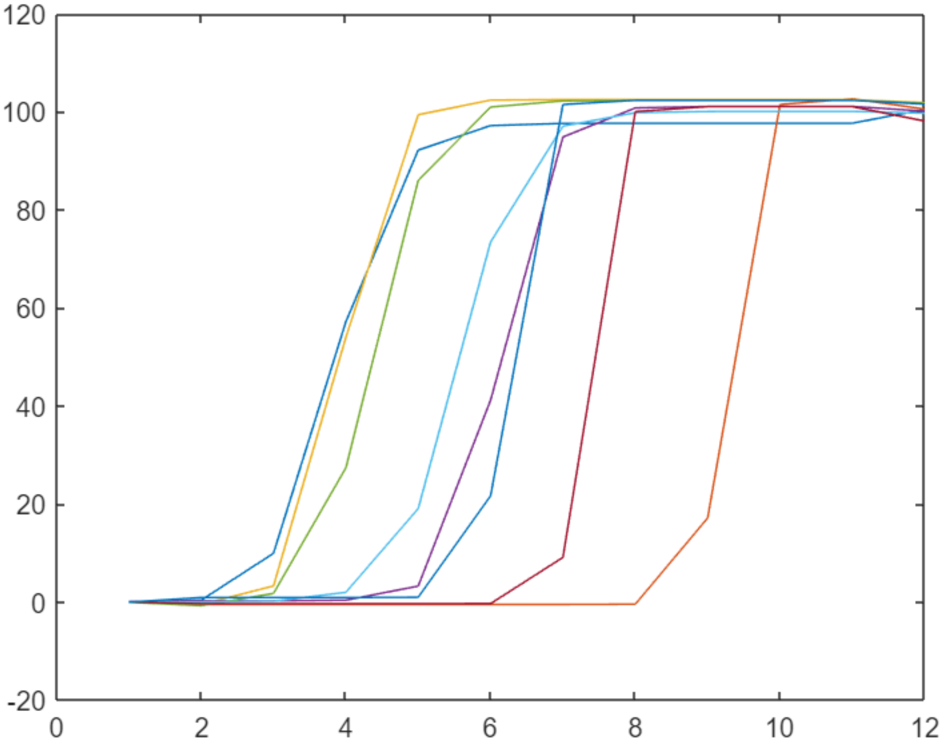
Step 2: Visualize the Data

When a variable in the workspace is selected, the **Plots Tab** automatically lights up with available visualizations that can accept that variable as input:



Let's try out a few of these visualizations and use the provided customization tools to adjust and automate custom data visualization

```
plot(data.Variables')
```



```
%bar3(data.Variables)
```

Step 3: Preprocess the Data

The data was imported into a **`table`**, one of several types of data containers. Tables are designed for columnar or tabular data, so accessing data can be done easily by name or by row and column numbers. The **`table`** also contains metadata accessible through properties.

```
data.Properties
```

```
ans = 
  TableProperties with properties:

             Description: ''
                UserData: []
          DimensionNames: {'Row'  'Variables'}
           VariableNames: {'NegControl'  'Conc1'  'Conc2'  'Conc3'  'Conc4'  'Conc5'  'Conc6'  'Conc7'  'Conc8'  'Co
    VariableDescriptions: {}
           VariableUnits: {}
      VariableContinuity: []
                RowNames: {}
        CustomProperties: No custom properties are set.
      Use addprop and rmprop to modify CustomProperties.
```

For example, let's assign row names to label each compound tested in the microtiter plate.

```
data.Properties.RowNames = cellstr("Compound"+(1:8))
```

```
data = 8×12 table
```

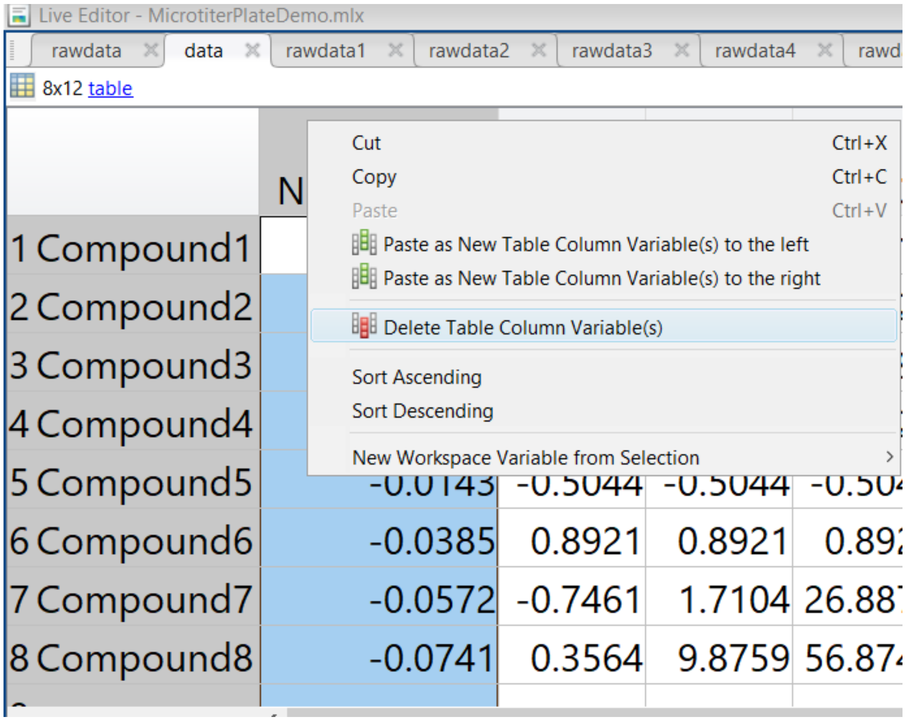
...

	NegControl	Conc1	Conc2	Conc3	Conc4	Conc5	Conc6	Conc7
1 Compound1	-0.0937	0.3389	9.9060	57.1390	92.1710	97.1720	97.6190	97.6570
2 Compound2	-0.0336	-0.5264	-0.5264	-0.5264	-0.5264	-0.5264	-0.5263	-0.4762
3 Compound3	-0.0137	-0.4983	3.2407	53.9460	99.3890	102.3700	102.4800	102.4800
4 Compound4	0.0905	0.2136	0.2196	0.3525	3.2192	41.1440	94.8470	100.7800
5 Compound5	-0.0775	-0.7785	1.7212	27.3410	85.9810	100.9600	102.2200	102.3100
6 Compound6	0.0473	-0.0047	0.1516	1.9478	19.0260	73.4120	97.0630	99.8060
7 Compound7	-0.0149	-0.3653	-0.3653	-0.3653	-0.3653	-0.3545	9.0751	100.0300
8 Compound8	-0.0584	0.8869	0.8869	0.8870	0.9435	21.6330	101.4800	102.3300

Before we can fit the data to our equation, we must first apply the positive and negative control measurements. **How do you do this in excel?**

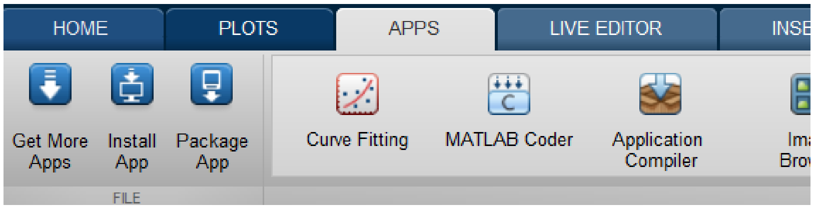
```
data.Variables = data.Variables - mean(data.NegControl);
data.Variables = data.Variables ./ data.PosControl*100;
```

Once this is done, we can remove the positive/negative control columns from our dataset. Try this from the **Variable Editor** and copy and paste the code below.



```
data = removevars(data, 'NegControl');
data = removevars(data, 'PosControl');
```

Step 4: Analyze the Data



MATLAB apps allow for easy and rapid prototyping while automatically generating the code for automation. Use the **Curve Fitting App** to fit each compound to the following equation:

$$R = \frac{\min - \max}{1 + (x/\mathrm{EC50})^{\mathrm{HillsSlope}}} + \max$$

where:

$$\begin{aligned} &1e-5 \le \mathrm{EC50} \le 1e7 \\ &0 \le \mathrm{HillsSlope} \le 10 \\ &0 \le \max \le 120 \\ &-20 \le \min \le 5 \end{aligned}$$

and input concentrations range from 1nM to 1M and we visualize them as a sigmoidal curve:

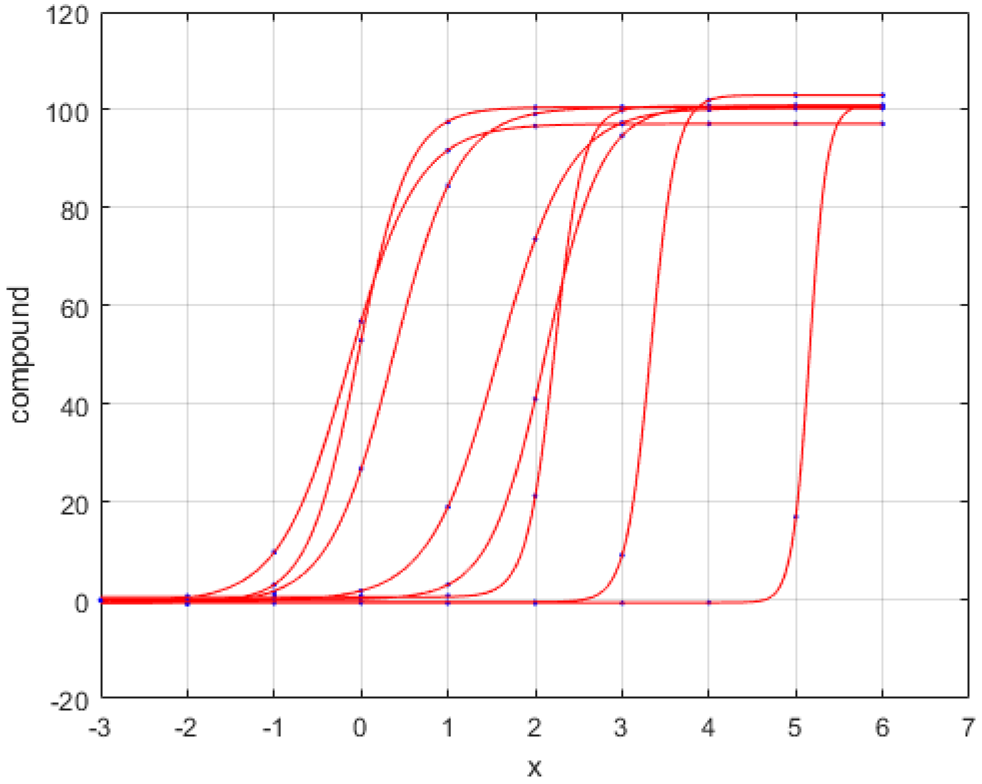
```
conc = [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000];
x = log10(conc);
```

Once we've gone through the fitting for one compound, we can ask the app to generate the MATLAB code in order to repeat the process for the rest of the data.

Step 5: Automate Data Analysis

In order to repeat the same analysis process for the entire dataset, use a **`for`** loop. A loop is a type of programming construct that allows control over the flow a process. Loop control statements permit a particular sequence of commands to be repeated any number of times. Below, the **`for`** loop repeats the fitting process for the full data **`table`**.

```
clf
h = height(data);
EC50resultsmtx = zeros(h,1);
for ii = 1:h
    compound = data{ii,1:10};
    [fitresult, gof] = createMicrotiterFit(x, compound);
    EC50resultsmtx(ii,:) = [fitresult.EC50];
end
```



```
EC50resultsmtx
```

```
EC50resultsmtx = 8×1
10^5  ×
    0.0000
    1.4373
    0.0000
    0.0013
    0.0000
    0.0004
    0.0215
    0.0017
```

Next Steps: How do I generalize to multiple data files?

Introducing **`datastore`** for handling multiple data files from multiple locations across multiple formats. **`datastore`** is a container datatype that points to where data lives and stores information about how it should be read in.

```
ds = datastore('Experimental_Data\*.csv');
ds.VariableNames =
{'NegControl','Conc1','Conc2','Conc3','Conc4','Conc5','Conc6','Conc7','Conc8','Conc9
','Conc10','PosControl'};
ds.ReadSize = 'file'
```

```
ds =
  TabularTextDatastore with properties:

                      Files: {
```

```
                               ' ...\MicrotiterPlateFittingDemo\Experimental_Data\microtiter_data0001.csv';
                               ' ...\MicrotiterPlateFittingDemo\Experimental_Data\microtiter_data0002.csv';
                               ' ...\MicrotiterPlateFittingDemo\Experimental_Data\microtiter_data0003.csv'
                                ... and 27 more
                               }
                      Folders: {
                               ' ...\MicrotiterPlateFittingDemo\Experimental_Data'
                               }
                 FileEncoding: 'UTF-8'
     AlternateFileSystemRoots: {}
           VariableNamingRule: 'modify'
            ReadVariableNames: false
                VariableNames: {'NegControl', 'Conc1', 'Conc2' ... and 9 more}
               DatetimeLocale: en_US

    Text Format Properties:
               NumHeaderLines: 0
                    Delimiter: ','
                 RowDelimiter: '\r\n'
               TreatAsMissing: ''
                 MissingValue: NaN

    Advanced Text Format Properties:
              TextscanFormats: {'%f', '%f', '%f' ... and 9 more}
                     TextType: 'char'
           ExponentCharacters: 'eEdD'
                 CommentStyle: ''
                   Whitespace: ' \b\t'
      MultipleDelimitersAsOne: false

    Properties that control the table returned by preview, read, readall:
        SelectedVariableNames: {'NegControl', 'Conc1', 'Conc2' ... and 9 more}
              SelectedFormats: {'%f', '%f', '%f' ... and 9 more}
                     ReadSize: 'file'
                   OutputType: 'table'
                     RowTimes: []

    Write-specific Properties:
       SupportedOutputFormats: ["txt"    "csv"    "xlsx"    "xls"    "parquet"    "parq"]
          DefaultOutputFormat: "txt"
```

Now we can perform the fitting on the preprocessed data. Here we initialize an empty array (**`results`**) to capture the EC50's and operate on the datastore until there is no more data left to read.

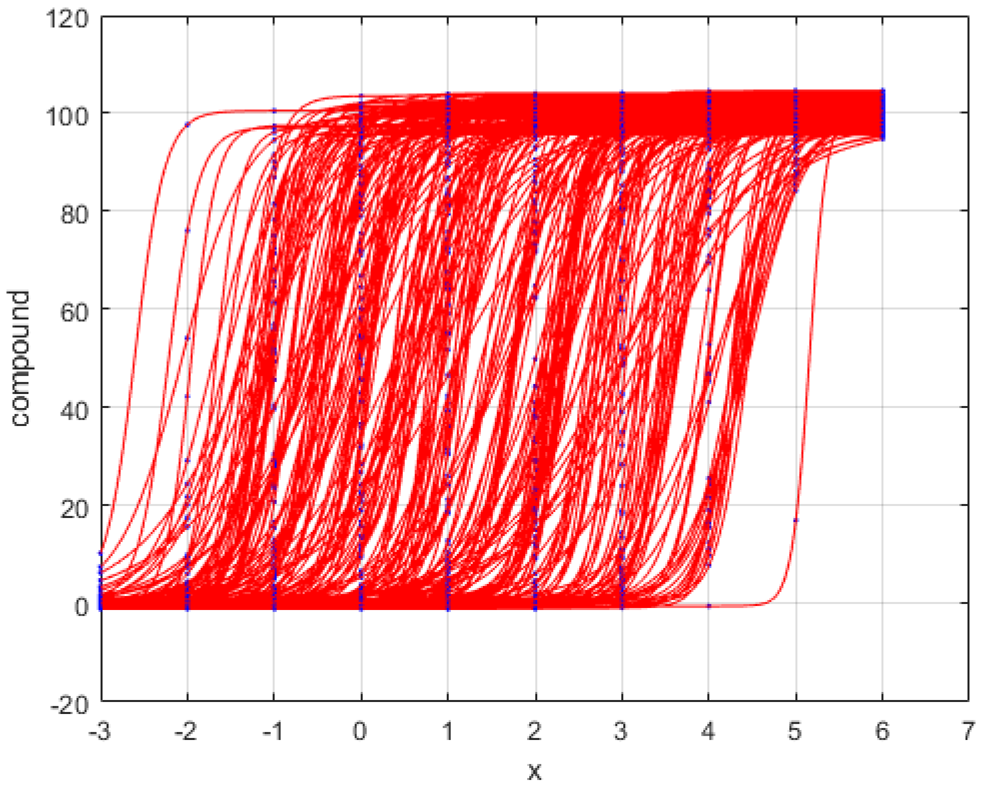
*NOTE: within the **`for`** loop, MATLAB Code Analyzer has identified that the variable, **`results`**, is increasing in size with every iteration. This is not a recommended practice and should be avoided when possible.

```
results = [];

while hasdata(ds)
    data = read(ds);

    data.Variables = data.Variables - mean(data.NegControl);
    data.Variables = data.Variables ./ data.PosControl*100;

    data = removevars(data, 'NegControl');
    data = removevars(data, 'PosControl');
    h = height(data);
    for ii = 1:h
        compound = data{ii,1:10};
        [fitresult, gof] = createMicrotiterFit(x, compound);
        results = [results;fitresult.EC50];
    end
end
```



```
writematrix(results,'myresults.csv')
reset(ds)
```

Extension: Alternatives for handling big data or scaling up

What if all the data needs to be processed at once? Is there any faster way of performing this analysis? There are several alternatives to explore...

```
do = false;

if do
    edit BigDataAndTall.mlx
end
```