

Data Analysis – Making Sense of Data over Time

Electricity Demand Case Study

Whenever we switch on a television, boil a kettle or use a computer, we rely on electricity which is delivered to us across a transmission network. Unlike gas or water, electricity cannot be stored in large quantities. It is the job of the electricity grid operator to match generation capacity of power stations with demand.

This must be performed minute-by-minute, 24 hours a day, 365 days a year. Specialist forecasting teams use their knowledge and tools such as MATLAB to predict what electricity demand will be.

Rapid or unexpected changes in demand make this task even more challenging. Can you guess what drives the most common surges in electricity demand?

The biggest ever surge in demand was after England's world cup semi-final against West Germany in 1990, when demand soared by 2,800 megawatts - equivalent to more than a million kettles being switched on!

A key input to any forecasting model is historical data and this is what our example is going to focus on.

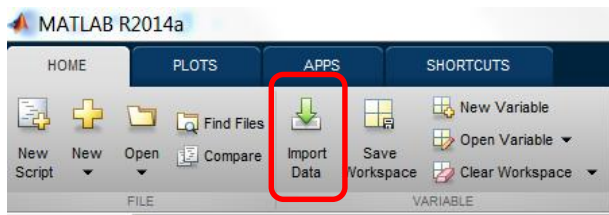
Our goal is will be to create a report which can be used by the electricity grid operator to forecast electricity demand based on historical data.

Importing Historical Electricity Demand Data

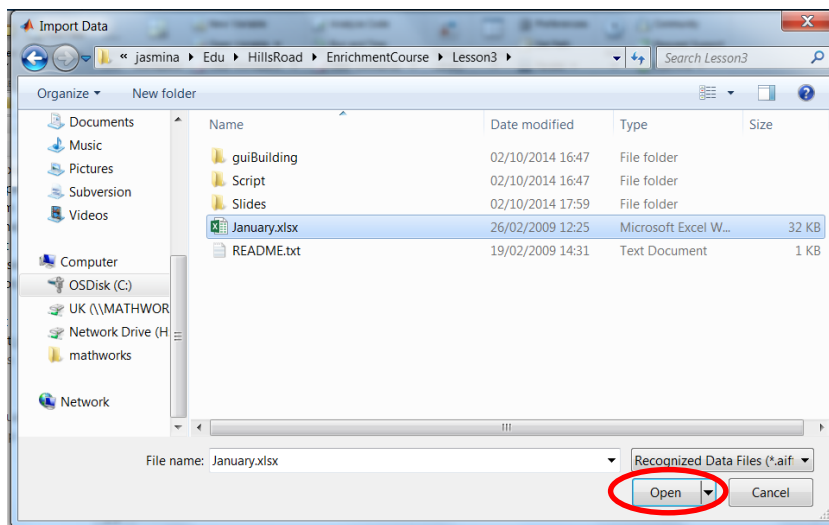
Our historical data for January is stored in the Excel file called 'January.xlsx'. Open the file in Excel to inspect the data. You will notice that there are four columns of data. The first column denotes the hour in the month (how many hours are there in the month of January?), the second column denotes the corresponding hour in a day, the third – the corresponding day in a week, and the fourth – the electricity consumption for the observed region within the current hour in megawatts.

Exercise 1: Importing Data into MATLAB Interactively

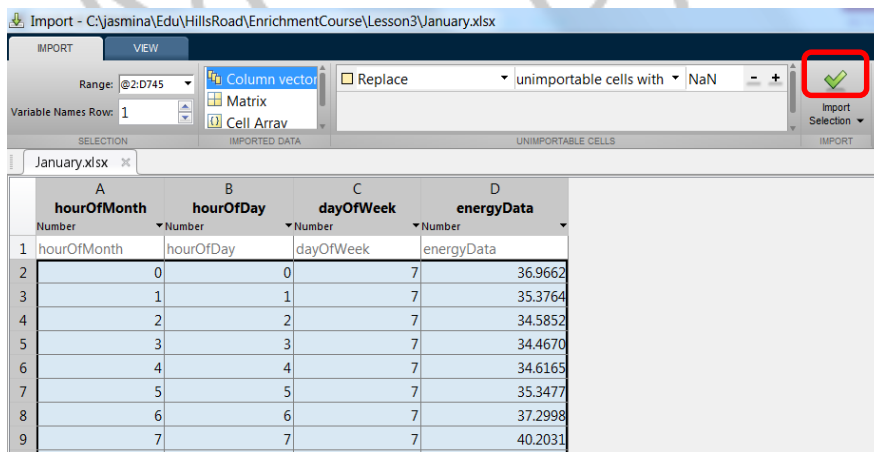
Go to MATLAB Home Tab and press the 'Import Data' button as shown below.



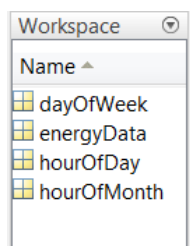
Navigate to the folder containing the 'January.xlsx' file, select that file, then click 'Open':



You will see MATLAB Import Tool starting up:



Click on the green check mark highlighted above. Has anything changed in your MATLAB workspace?



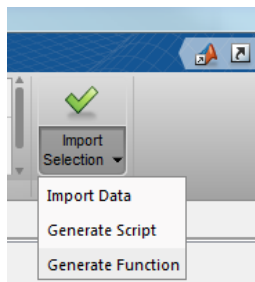
You can now double-click on any of the variables in MATLAB workspace to examine the data and confirm it is the same data we have seen in the Excel file.

Exercise 2: Importing Data into MATLAB Programmatically

So far, we have only imported data for one file, corresponding to a single month in a year. What if we needed to import data for all months since year 1950, with data for each month residing in a separate file? How many files would be there? How long do you think it would take you to import all data in this way?

As you can imagine, clicking around in the Import Tool is not the most time efficient way to bring your data into MATLAB. So what can we try next?

Let us go back to our Import Tool, but this time, instead of clicking directly on the green check mark, click on the small arrow right below it. This will expand a menu of different options for automatic MATLAB code generation to produce a MATLAB program for importing the data:

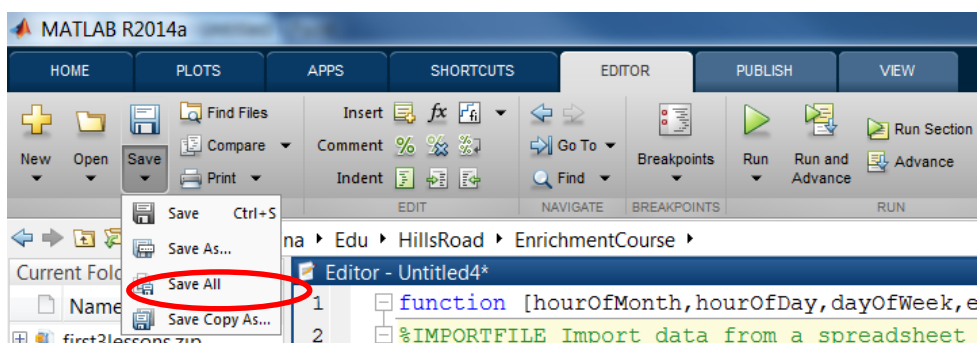


Let us choose the 'Generate Function' from the menu and see what happens. What can you see in your MATLAB editor?

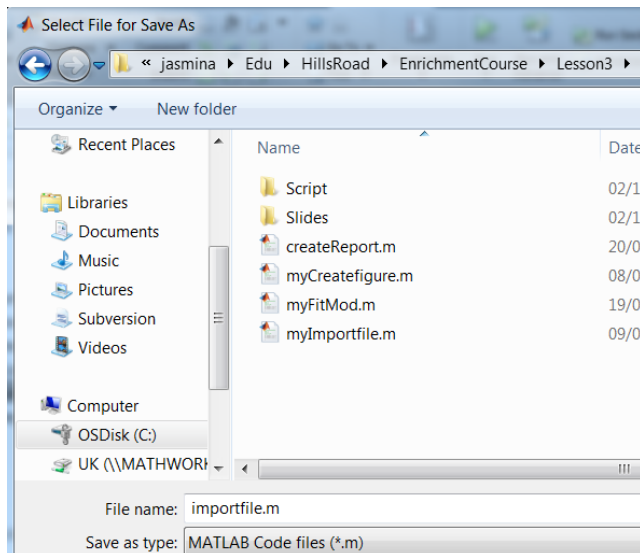
As you probably guessed, we have obtained a MATLAB function designed to import data from a given input file. What are the inputs to the function? And the outputs?

Say that, for a given Excel file, we just want to import all data from the first sheet, so we don't really need to worry about the sheet name or the rows we want to read. In that case, we can remove all input arguments following the 'workbookFile', including 'sheetName', 'startRow' and 'endRow'.

Now let's save this function as 'myimportfile.m'. Go to the Editor tab, then Save → Save As... :



Navigate to your Lesson3 folder and type in 'importfile.m' (without quotes):



To check if the function is working, clear everything from MATLAB workspace, by typing the following in MATLAB command line:

```
>> clear all
```

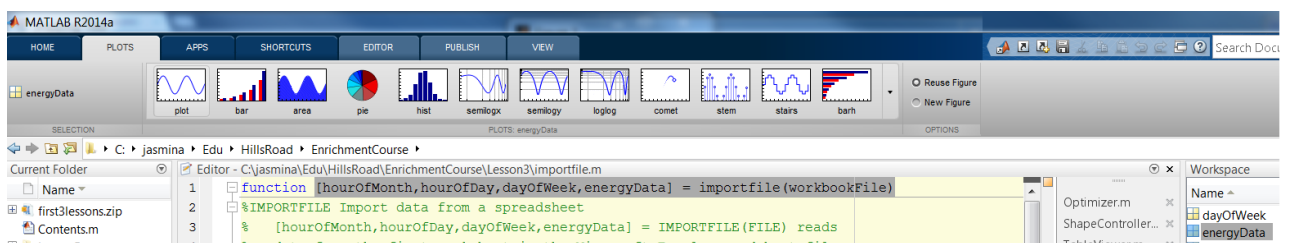
Then type (or copy and paste from the function file, changing `workbookFile` to `'January.xlsx'`):

```
>> [hourOfMonth, hourOfDay, dayOfWeek, energyData] = importfile('January.xlsx')
```

Data Exploration and Visualisation

Now that we have the data in, let us try and understand a little bit more how the data looks like and if there are any patterns and/or behaviours that we can identify.

Click (once) on the `energyData` variable in MATLAB Workspace to highlight it. Then go to the Plots tab and click on the first plot.



What do you get as a result? Can you say anything about the energy usage in January looking at the obtained plot?

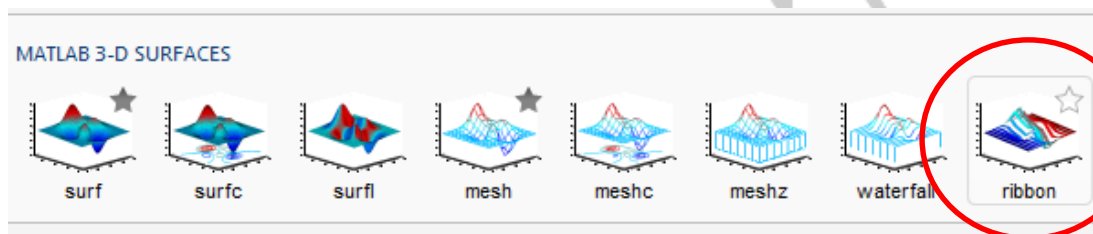
You will probably have guessed that there is some repetitive behaviour, but it is hard to observe anything more than that looking at the data arranged as it is. What can we do to make data easier to understand and interpret?

3D Plot Example

What if we rearranged data into columns of hourly data for each of the 31 days in January? How many elements would we have in each of the 31 columns?

```
>> energyDataGrid = reshape( energyData, 24, 31 );
```

Let us go on and visualise the newly rearranged data. Click on the energyDataGrid variable in MATLAB workspace and go to the Plots tab. However, rather than choosing the first plot from the list, scroll down and select the ribbon plot:



How would you interpret the resulting plot this time?

When do the spikes in the usage typically occur? What could be causing the spikes at these times? When would you expect to get the lowest energy usage throughout a 24h day? Is that consistent with what you are seeing in the plot?

Data Brushing and Linking

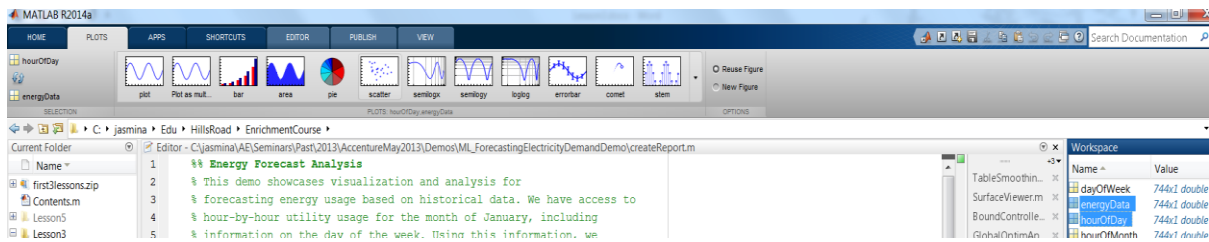
Let us plot energy data in two different ways, as a function of hour in a day and as a function of a day in a week.

Exercise 3: Plot energy data as a function of an hour in a day. Can you make a scatter plot?

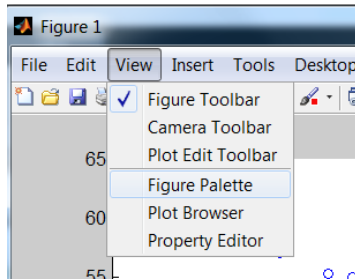
Exercise 4: Plot energy data as a function of a day in a week. Can you place this plot next to the previous plot on the same figure?

Solution to Exercises 3 and 4

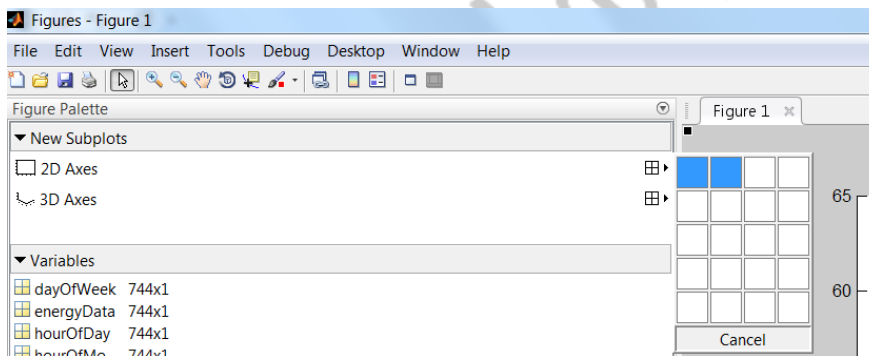
Plot energyData vs. hourOfDay by selecting those variables in the Workspace, then clicking on the 'scatter' plot in the 'Plots' tab.



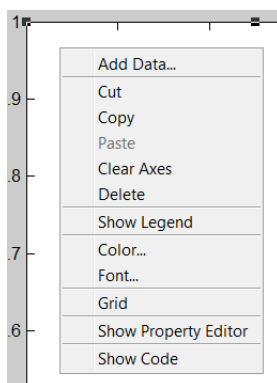
Once your figure with this plot shows up, go and select 'Figure Palette' in the 'View' menu:



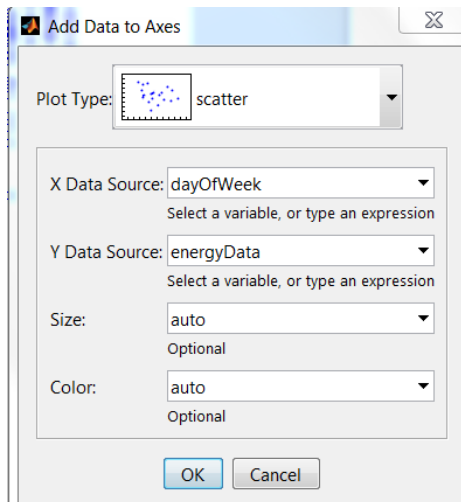
Add another plot in the same figure, on the right hand side of the current plot:



Right-click anywhere inside the axes of the second plot and select 'Add Data...':

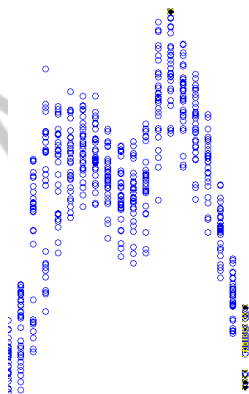
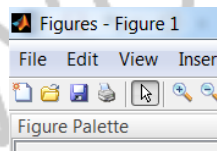


Then select your data and the type of plot you want (scatter):



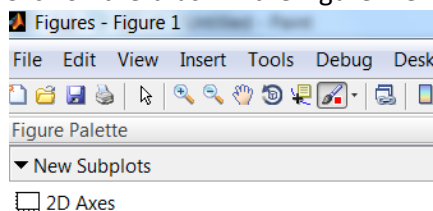
Now we can explore the relation between the data in the two plots by linking the data and then brushing to observe the relationships:

- Click on the arrow in the Figure menu:
- Click anywhere on the data in the first spot to ensure the data is selected:



(note the black selection points)

- Click on the brush In the Figure menu.



- Select some data points in the first plot and observe the corresponding selection in the second plot. Can you spot any clear relationships?

Before we proceed with our electricity demand example, let us first try something simple. Can you create a 10-element vector `x` of random numbers?

```
>> x = rand(1, 10)
```

Now type the following:

```
>> idx = x > 0.5
```

Do you understand the output of the last command? Can you guess what `x(idx)` will look like?

```
>> x(idx)
```

We could have obtained the last result without introducing variable `idx` explicitly:

```
>> x(x > 0.5)
```

Now create a vector `y` from vector `x`, which contains all elements of `x` less than 0.5. Also, can you remove all elements from `x` which are less than 0.5?

```
>> x(x < 0.5) = []
```

Challenge:

Create a vector `x` of 10 random numbers as above:

```
>> x = rand(1, 10)
```

Can you create vector `z` from vector `x`, containing only those elements which are less than their successors in `x`? E.g. if

```
x = [0.1576 0.9706 0.9572 0.4854 0.8003 0.1419 0.4218 0.9157 0.7922 0.9595],
```

the output needs to be:

```
z = [0.1576 0.4854 0.1419 0.4218 0.7922 0.9595]
```

Can you do it **in just one line of code**, without using loops? (Everything containing more than 80 characters is considered more than one line of code!)

Hint:

Can you figure out what function `diff` does? Try:

```
>> diff(x)
```

Curve Fitting

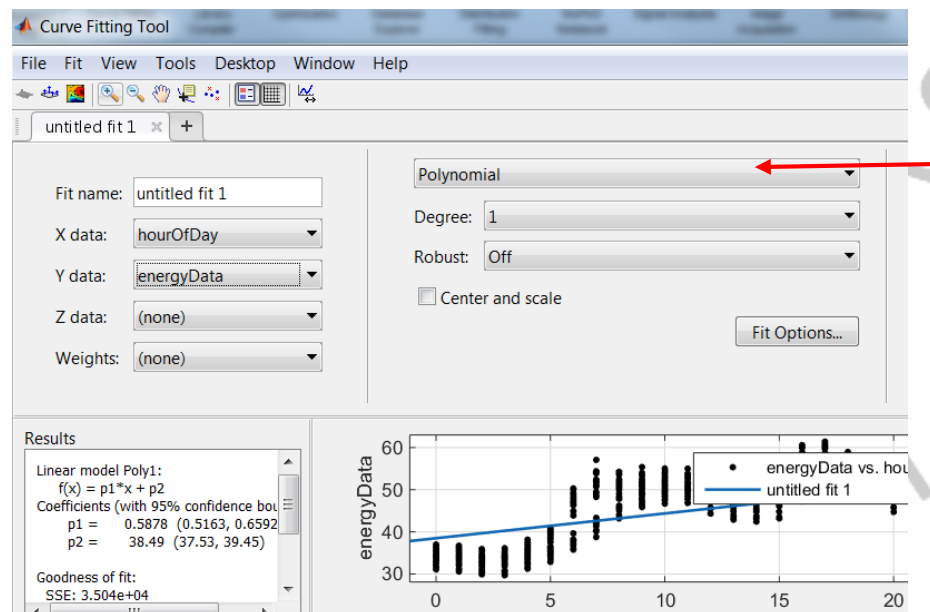
We have seen from a previous exercise that weekday energy usage data differs from weekend data. So let us separate out the data for working days. Now that you know about logical indexing, can you guess how we are going to do that?

```
>> weekdayIndex = dayOfWeek <= 5;  
>> weekdayEnergyData = energyData(weekdayIndex);  
>> weekdayHourOfDay = hourOfDay(weekdayIndex);
```


Let us try and figure out a model which represents how weekday energy usage depend on the time of day (represented by hour of day). Go to the 'Apps' tab and click on the 'Curve Fitting' icon.

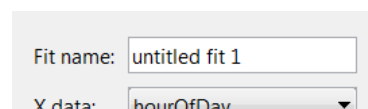


Then define your x-data as weekdayHourOfDay, and your y-data as weekdayEnergyData:

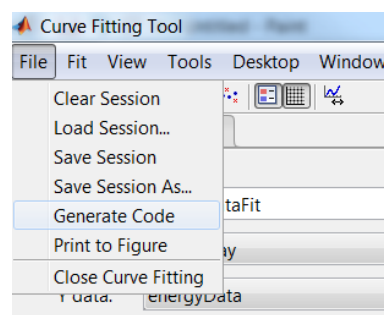


Can you play with different fitting functions (see arrow in the figure above)? Which one do you think fits the data best? Why?

Once you have chosen your best fit, enter your fit name, e.g. 'energyDataFit':



Then go to the 'File' menu and select 'Generate Code':



Save the automatically generated function as 'createFit.m' in your 'lesson3' folder. Now try out the following:

```
>> [fitresult, gof] = createFit(hourOfDay, energyData)
```

Exercise 5:

What is a typical energy usage at 10.20am on a working day?

Exercise 6:

Find a typical energy usage at 5min before midnight:

- a) Today
- b) Yesterday

(For the purposes of this exercise, assume the data for this October is the same as the data for the first 30 days of January, which we have already imported.)

Exercise 7:

At what time in a working day is electricity most used?

Exercise 8:

At what time during the working hours (between 9am and 5pm) in a working day is electricity least used?