Part 2 - Laboratory Preparation

October 28, 2020

1 COMP60711 Part 2 Coursework Guide

For the coursework in Part 2 of COMP60711, we will be using Python and Jupyter notebooks. This guide serves to give you some guidance for setting things up, how to work with notebooks, how to submit your coursework, and some advice on plotting graphs in Python.

In general, some code snippets are provided for some questions to help get you started. You will, however, be required to write some Python code. It is an incredibly popular language, so if you are unfamiliar with it then this is a good opportunity to learn (of which there are endless resources available).

1.1 Table of Contents

1.2 Setting up your Anaconda environment

For this coursework, we will be using Python & Jupyter notebooks. In this section, we will help you get started with these.

The easiest way to use Python is via Anaconda. As this comes with many packages that you may not need, if hard-drive space is a concern then you can install Miniconda instead.

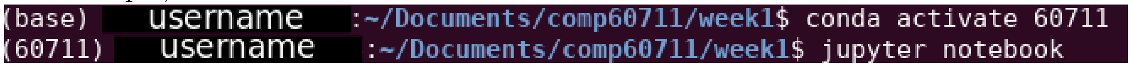
To ensure you have everything needed for this part of the coursework, we have created an environment for you. The file, `60711_environment.yml`, can be found on Blackboard. You can then create an environment for this course by running `conda env create -f 60711_environment.yml` in either the Anaconda Prompt (Windows) or the terminal (Mac OS/Linux). Environments are just groups of packages of particular versions, which can help ensure that everything runs as expected. Different versions of libraries have different features, and can have different function arguments!

You can then activate this environment via `conda activate 60711`. You'll need to do this before running any code, or starting any notebooks. Then, you can start your jupyter notebook, which we cover in the next section.

1.3 Using Jupyter notebooks

To start Jupyter notebooks, make sure you first activate the `60711` environment in either Anaconda Prompt (Windows) or the terminal (Mac OS/Linux). Then, you can run `jupyter notebook` here to open the Notebook Dashboard in your browser.

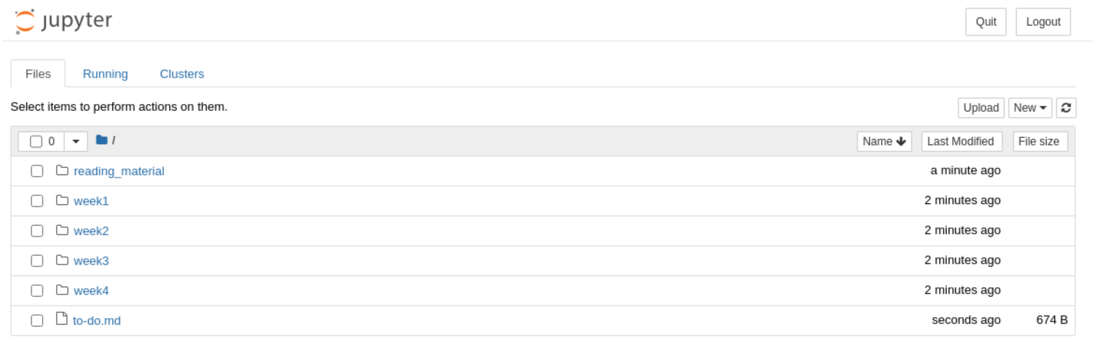
For example, we can run these two commands to start a notebook with our `60711` environment:



```
(base)    username    :~/Documents/comp60711/week1$ conda activate 60711
(60711)    username    :~/Documents/comp60711/week1$ jupyter notebook
```

Note, it is recommended that you start this notebook in the folder where you intend to work. Ideally, this is the same folder that contains the datasets you need for the coursework. For example, if we download the datasets and question notebooks to our folder `/home/name/Documents/comp60711/week1/`, then we should first navigate to this folder before starting the notebook. If not, then you will need to provide the full path (e.g. `"/home/name/Documents/comp60711/week1/dataset.csv"` rather than just `"dataset.csv"`).

Once you have started the notebooks, you can then open files. An example screenshot is below:



Select items to perform actions on them.

```
reading_material    a minute ago
week1               2 minutes ago
week2               2 minutes ago
week3               2 minutes ago
week4               2 minutes ago
to-do.md            seconds ago     674 B
```

Notebook dashboard. Here, we can create notebooks, and navigate through our folders.

We can create a notebook using the "New" button in the top-right. In the next section, we'll give some guidance for writing your answers using notebooks, though there is official documentation to help if you need it.

Caution! Notebooks are essentially an interactive session of Python, so that when you create variables they are stored in memory. This can be problematic when re-running cells out of order (i.e. running one cell before you run the one above it), and so your code may not behave as expected. This is particularly problematic for those who are less familiar with programming, but it can trip up even experienced users as it is unintuitive.

1.3.1 Writing Answers

There are two types of cell (that we are interested in) in a notebook: markdown, and code. The first is where you can write text, and how we expect you to write your answers. Each notebook links to a brief guide to Markdown, but in general.

We expect you to also submit, alongside the answer, the code you used to write it. You are not being marked on the quality of your code, just the output and your interpretation of it. Further in this document, we will provide a brief guide to making graphs, but you find that small snippets of code in your text will help to place graphs where you need them.

Adding Figures You may wish to add images that are not generated by code. For this, you can go to Edit –> Insert Image and select the file to insert. Please see the code in the above section (double click to reveal the text underneath) for the image there to see an easy way of adding a caption. If you are so inclined, you can also use HTML to add captions.

Citations There are some more complex ways to add citations to notebooks, but as long as it is clear what you cite and where it is fine.

The simplest approach is to use numbers inside square brackets so that you cite material where you need it, and provide a list of citations at the end of the cell or end of the document. Just remember to read the material that you cite[1], and that you use primary sources[2].

[1]: Simkin, M. V., & Roychowdhury, V. P. (2002). Read before you cite!. arXiv preprint cond-mat/0212043. [2]: Waters, N. L. (2007). Why you can't cite Wikipedia in my class. Communications of the ACM, 50(9), 15-17.

Note: Please ensure that the citations are separated onto new lines. You can add `<br>` to force this.

1.4 Submitting your coursework

Generally, it is advised that you freshly re-run the notebook before submitting, so that the first code block has `In [1]:` to the left of it. This ensures that all of your code is executed in order, and that we can follow what you have done.

In general, you should submit the coursework as .html file, by going to File –> Download as –> HTML (.html). Please then check this file so that it is the same as what you have seen.

In general, this is preferred over converting the notebook into a PDF, which can cause issues with graphs (in particular). If you do submit as a PDF, it is vital that you check the produced PDF before submission.

For the submission itself, there will be an area on Blackboard for the particular coursework to upload for the particular week.

1.5 Brief guide to plotting

Creating graphs is sometimes problematic in Python, and so this section will provide some examples using the most popular visualization package in Python.

1.5.1 Matplotlib

Matplotlib is an incredibly popular and powerful library for creating plots in Python. However, many users often find it unintuitive. Instead of listing all the different ways of creating plots, we will show a couple examples here which should help you in the creation of your own graphs.

First, let's create some dummy data:

[10]:
```python
import numpy as np
# Fix the random seed
np.random.seed(42)
# Generate two sets of random numbers and combine them
```

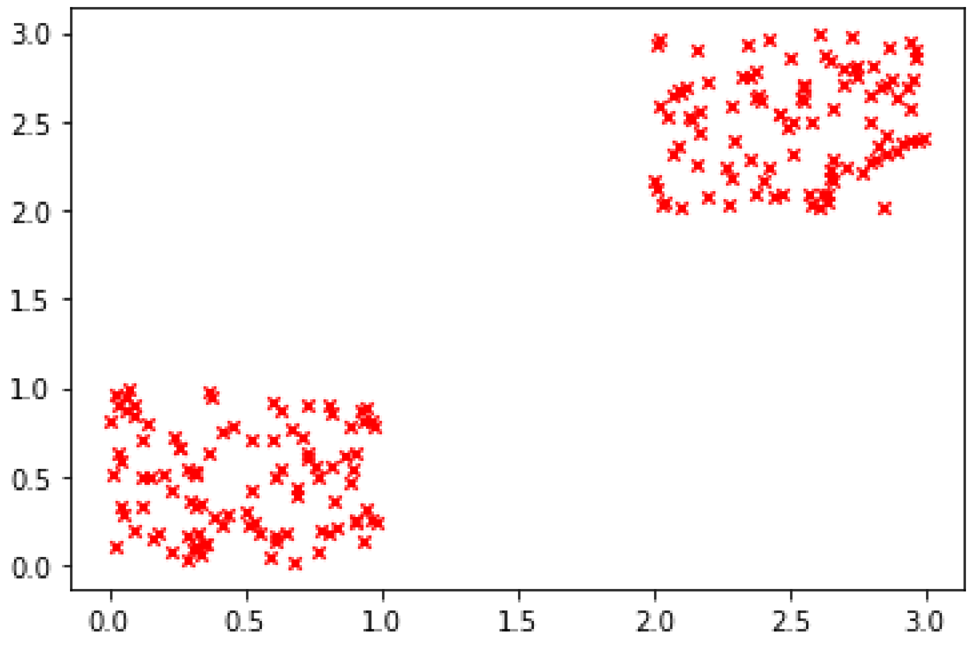
```
data = np.vstack([np.random.uniform(0,1,size=(100,2)),np.random.
 ↪uniform(2,3,size=(100,2))])
```

Then we can visualize this data:

```
[11]: # Import matplotlib
import matplotlib.pyplot as plt

# Create the figure and axes object
fig, ax = plt.subplots()
# Create a scatter plot with these axes using our data
# Note that, data[:,0] is our x-axis data, and data[:,1] is our y-axis data
ax.scatter(
    data[:,0], data[:,1], # Pass the data in
    c="red", # Colour
    marker="x", # The type of marker to use
    s=20, # The size of the marker
)
```

```
[11]: <matplotlib.collections.PathCollection at 0x7fea3046b748>
```

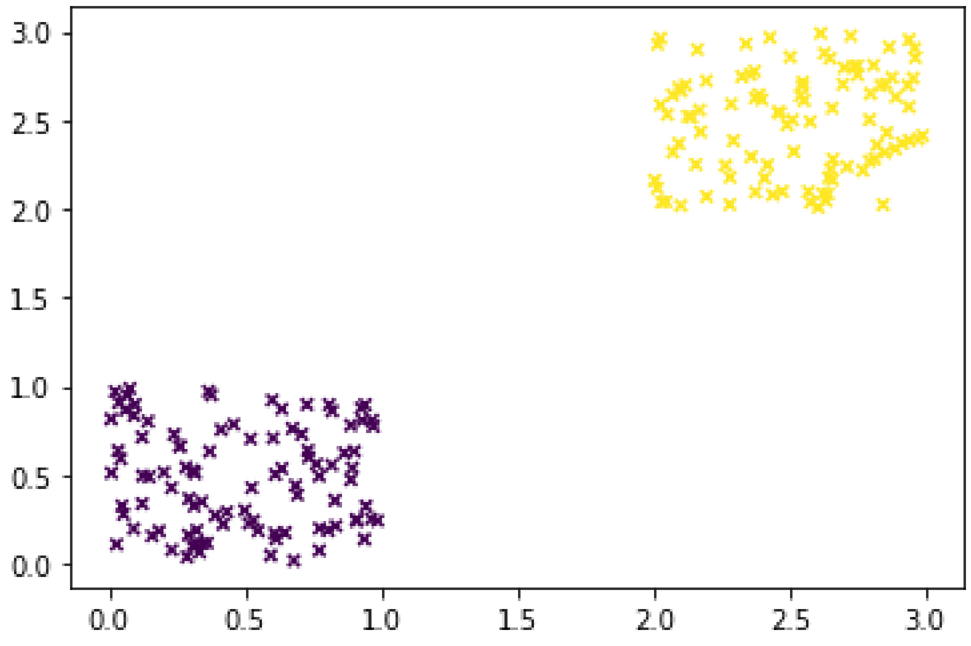


However, we may want to have multiple plots at once. One way to do this is to pass in a list of labels to use as the colour. As we generated this data, we know that the first 100 rows are for one square, and the second 100 rows are another colour.

[12]:
```python
# Create a vector of 0s the same length as our data
labels = np.zeros(data.shape[0])
# Set the second 100 rows to 1
labels[100:] = 1
```

[13]:
```python
fig, ax = plt.subplots()
# Create a scatter plot with these axes using our data and new labels
ax.scatter(
    data[:,0], data[:,1], # Pass the data in
    c=labels, # Colour
    marker="x", # The type of marker to use
    s=20, # The size of the marker
)
```

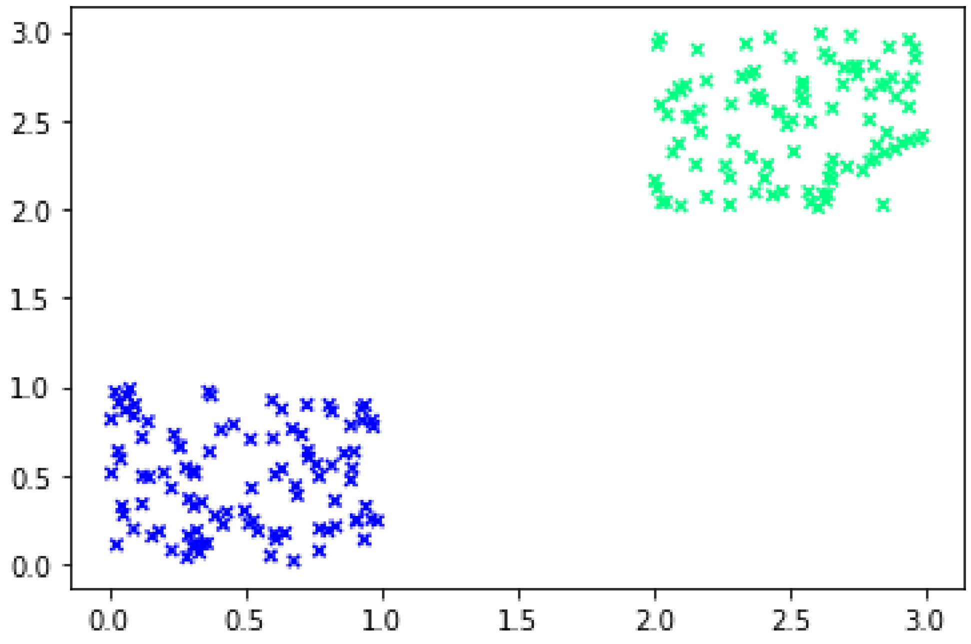
[13]:
```
<matplotlib.collections.PathCollection at 0x7fea30652da0>
```



A question here might be: why did it use purple and yellow? Underneath, matplotlib uses colormaps to determine what colours to use. You can select one and provide it using the `cmap` argument, as below:

[14]:
```python
fig, ax = plt.subplots()
# Create a scatter plot with these axes using our data and new labels
ax.scatter(
    data[:,0], data[:,1], # Pass the data in
    c=labels, # Colour
    marker="x", # The type of marker to use
    s=20, # The size of the marker
    cmap="winter" # Colourmap
)
```

```
[14]: <matplotlib.collections.PathCollection at 0x7fea31006ba8>
```



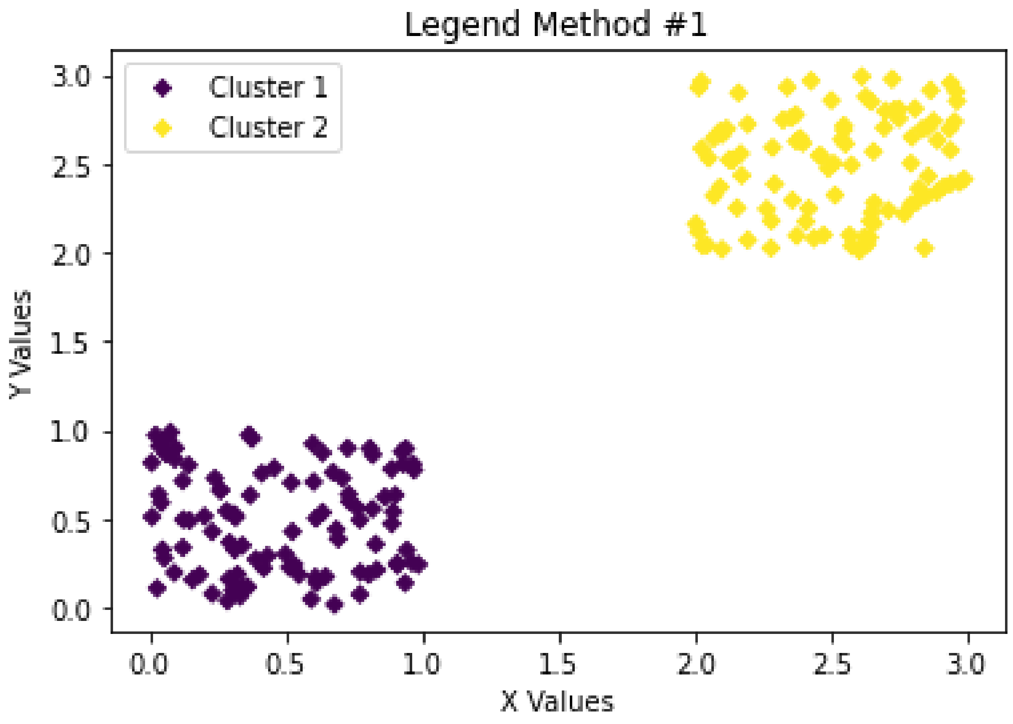
So far, we haven't labelled our graphs, or used a legend at all! This is a must. Let's plot the same data again, this time adding these. I'll show two ways of adding data which determine how you can create the legend.

The first method is plotting all of the data at once, as we have done before. Doing it this way is slightly more complicated, as we need to explicitly tell it what to put in the legend.

```
[15]: fig, ax = plt.subplots()
      # Create a scatter plot with these axes using our data and new labels
      scatter = ax.scatter(
          data[:,0], data[:,1], # Pass the data in
          c=labels, # Colour
          marker="D", # The type of marker to use
          s=20 # The size of the marker
      )
      # Add the labels
      ax.set_xlabel("X Values")
      ax.set_ylabel("Y Values")
      # Add a title
      ax.set_title("Legend Method #1")
```

```
# Create the legend manually
ax.legend(
    handles=scatter.legend_elements()[0],
    labels=["Cluster 1", "Cluster 2"]
)
```

[15]:
```
<matplotlib.legend.Legend at 0x7fea31126940>
```

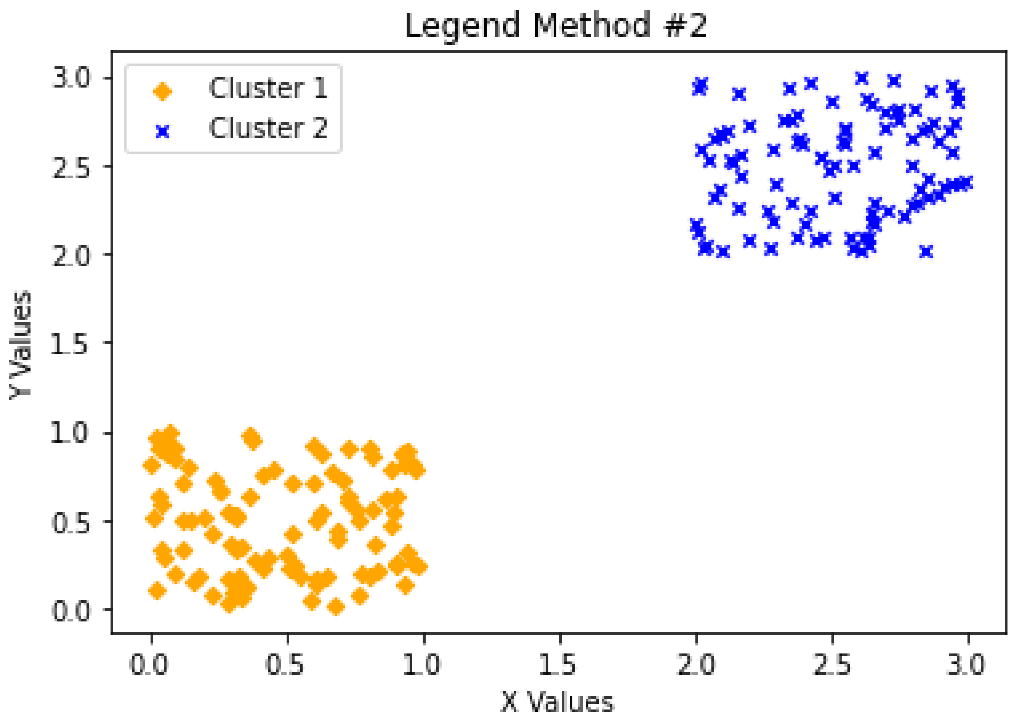


The easier way is to plot your data in groups that have a common label. In our case, that means splitting the data and calling `ax.scatter` on each split.

[16]:
```
fig, ax = plt.subplots()
# Create a scatter plot with these axes using our data and new labels
ax.scatter(
    data[:100,0], data[:100,1], # Pass the data in
    c="orange", # Colour
    marker="D", # The type of marker to use
    s=20, # The size of the marker
    label="Cluster 1"
)
# Add the labels
ax.set_xlabel("X Values")
ax.set_ylabel("Y Values")
# Add a title
```

```
ax.set_title("Legend Method #2")
# Plot the second set of data
ax.scatter(
    data[100:,0], data[100:,1], # Pass the data in
    c="blue", # Colour
    marker="x", # The type of marker to use
    s=20, # The size of the marker
    label="Cluster 2"
)
# Add the legend, which will automatically add the markers and labels
ax.legend()
```

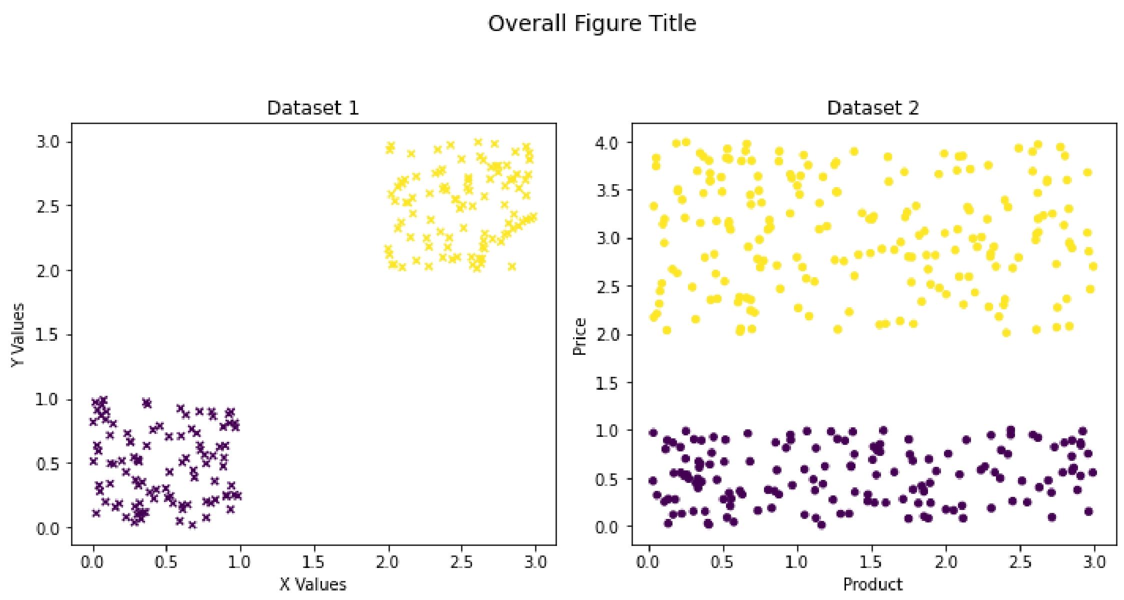
[16]:
```
<matplotlib.legend.Legend at 0x7fea31209128>
```



We may want to have multiple plots at once, however, if we want to show two datasets. Let's create another dataset and then do this! Note the use of `fig.tight_layout()` at the end - without this, our second y-axis label would overlap with the left-hand graph!

[17]:
```
# Create the second dataset
data2 = np.vstack([np.random.uniform([0,0],[3,1],size=(150,2)),np.random.
↪uniform([0,2],[3,4],size=(200,2))])
labels2 = np.zeros(data2.shape[0])
labels2[150:] = 1
```

```python
# Create a figure with 2 sets of axes in a row
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,5))
# Create a scatter plot with these axes using our data and new labels
ax1.scatter(
    data[:,0], data[:,1], # Pass the data in
    c=labels, # Colour
    marker="x", # The type of marker to use
    s=20, # The size of the marker
)
# Set labels
ax1.set_xlabel("X Values")
ax1.set_ylabel("Y Values")
# Set a title
ax1.set_title("Dataset 1")
# Do the same with the second set of axes
ax2.scatter(
    data2[:,0], data2[:,1], # Pass the data in
    c=labels2, # Colour
    marker="o", # The type of marker to use
    s=20, # The size of the marker
)
# Random axis labels
ax2.set_xlabel("Product")
ax2.set_ylabel("Price")
ax2.set_title("Dataset 2")
# Set an overall figure title
fig.suptitle(
    "Overall Figure Title",
    y=1.05, # Vertical position
    x=0.525, # Horizontal position
    fontsize=14 # Make it a larger font
)
# Ensyre everything is separate
fig.tight_layout()
```



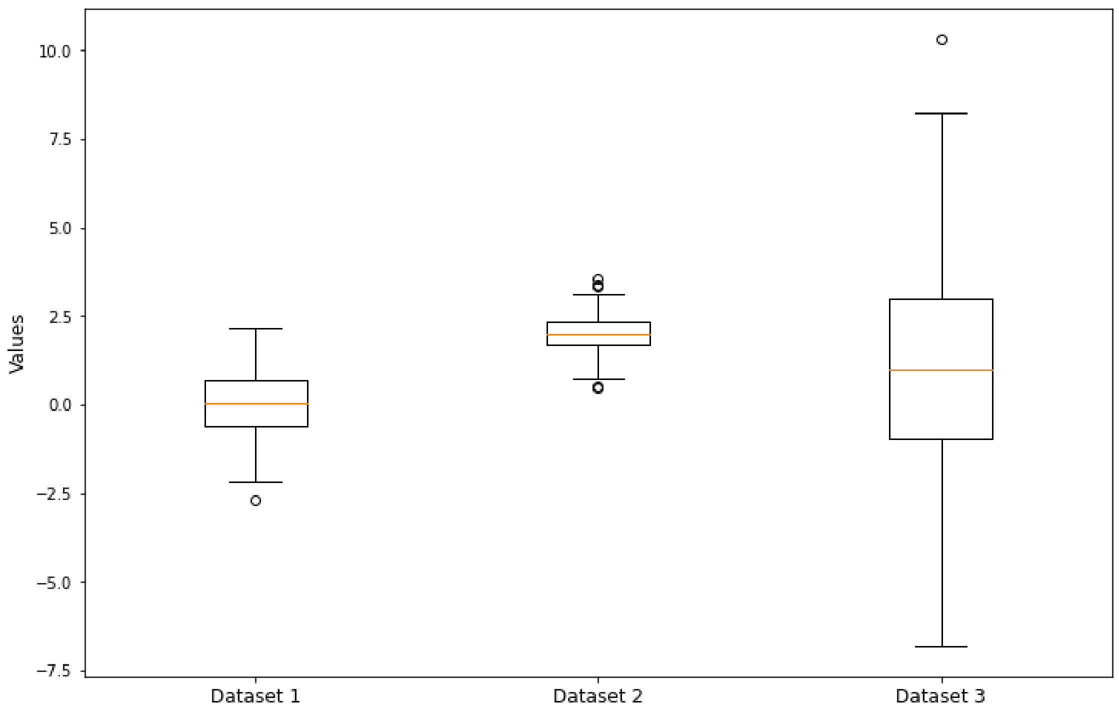
So far, we have only shown scatterplots. For the next example, we will show how to create boxplots instead.

[19]:
```
# Create 3 sets of random data

data1 = np.random.normal(0, 1, size=(200,))
data2 = np.random.normal(2, 0.5, size=(200,))
data3 = np.random.normal(1, 3, size=(400,))

# Change the figsize to make a bigger plot!
fig, ax = plt.subplots(figsize=(12,8))
# Set the y-axis label
ax.set_ylabel("Values", fontsize=12)
# Set the labels for our 3 boxplots
ax.set_xticklabels(["Dataset 1", "Dataset 2", "Dataset 3"], fontsize=12)
# Set equal to a variable to suppress the output
_ = ax.boxplot([data1, data2, data3])
```

```
/Users/jonathac/anaconda3/envs/60711_venv/lib/python3.6/site-
packages/ipykernel_launcher.py:12: UserWarning: FixedFormatter should only be
used together with FixedLocator
  if sys.path[0] == '':
```



This should give you some general templates for making graphs. Note that there are other types of plot you can make, which you can use in place of `.scatter` or `.boxplot`. Please refer to the documentation for this. The above examples are not the only way to create these figures, but it is one of the most robust methods that we recommend you use.