CPSC 4300/6300: Applied Data Science

Week 1: Environment Setup

Clemson University
Instructor(s): Nina Hubig

In [1]:
```
""" RUN THIS CELL TO GET THE RIGHT FORMATTING """
import requests
from IPython.core.display import HTML
css_file = 'https://raw.githubusercontent.com/bsethwalker/clemson-cs4300/main/cs
styles = requests.get(css_file).text
HTML(styles)
```

Out[1]:

Getting and installing Python

You will be using Python throughout the course, including many popular 3rd party Python libraries for scientific computing. Anaconda is an easy-to-install bundle of Python and most of these libraries. We **strongly suggest** that you use Anaconda for this course.

For this course, we are using **Python 3**, not **Python 2**.

The **Jupyter** notebook runs in the browser. For me, it works best in Google Chrome or Safari. You probably want to use one of these for the course assignments.

Installing Anaconda

The Anaconda Python distribution is an easily-installable bundle of Python and many of the libraries used throughout this class. Unless you have a good reason not to, we recommend that you use Anaconda.

Mac/Linux users

1. Download Anaconda
2. Follow the instructions on that page to run the installer
3. Test out the Juypter notebook: open a Terminal window, and type

```
jupyter notebook .
```

Or use the Anaconda Launcher which might have been placed on your desktop.

A new browser window should pop up.

Click `New Notebook` to create a new notebook file. **Trick**: give this notebook a unique name by clicking on the word "Untitled" at the top.

You will have to occasionally work in the terminal on mac or linux. Mac users can access it (a terminal app is built into the mac) by typing "terminal" in spotlight or through Launchpad.

You might also want to choose a specific folder to work in, something like `cpsc4300` under `Documents`. In this case, open the terminal, and do:

`cd Documents/cpsc4300`

and then type:

`jupyter notebook`

Windows Users

1. Download the appropriate version of Anaconda
2. Follow the instructions on that page to run the installer. This will typically create a directory at `C:\Anaconda`
3. Test it out: start the Anaconda launcher, which you can find in `C:\Anaconda` or, in the Start menu. Start the Juypter notebook. A new browser window should open.
4. Click `New Notebook`, which should open a new page. **Trick**: give this notebook a unique name, like `my-little-rose`. Use Explorer (usually start menu on windows desktops) to search for this name. In this way, you will know which folder your notebook opens in by default.

Windows users should consider installing the program `git-bash`. It gives you a nice terminal to work with.

All Users

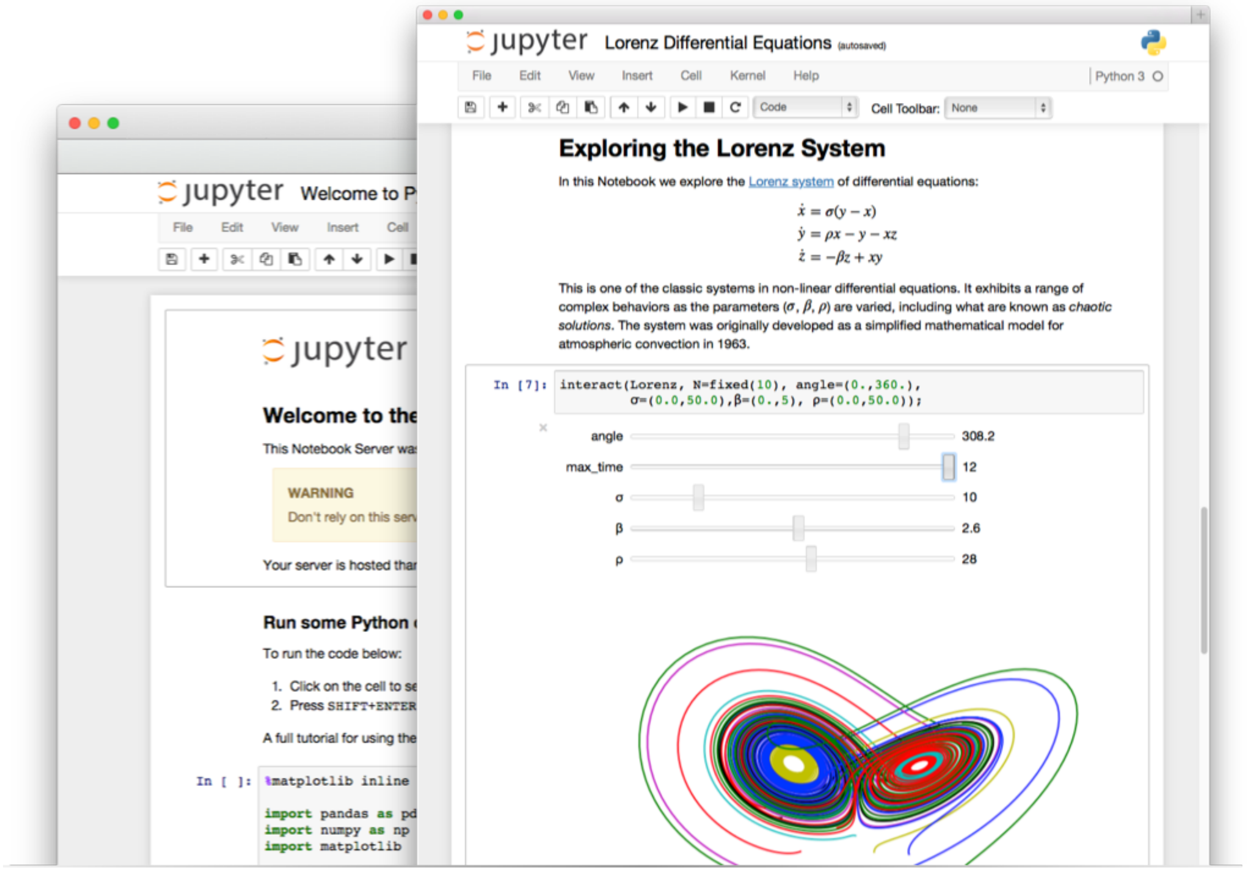
If you did not add Anaconda to your path, be sure to use the full path to the python and ipython executables, such as `/anaconda/bin/python`.

If you already have installed Anaconda at some point in the past, you can easily update to the latest Anaconda version by updating conda, then Anaconda as follows:

```
conda update conda
conda update anaconda
```

Hello, Jupyter

The Jupyter Notebook is a web application that allows you to create interactive document that contain live code, equations, visualizations and explanatory text


jupyter Welcome to P
File Edit View Insert Cell
jupyter Lorenz Differential Equations (autosaved)
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None

Welcome to the

This Notebook Server wa

WARNING

Don't rely on this serv

Your server is hosted tha

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

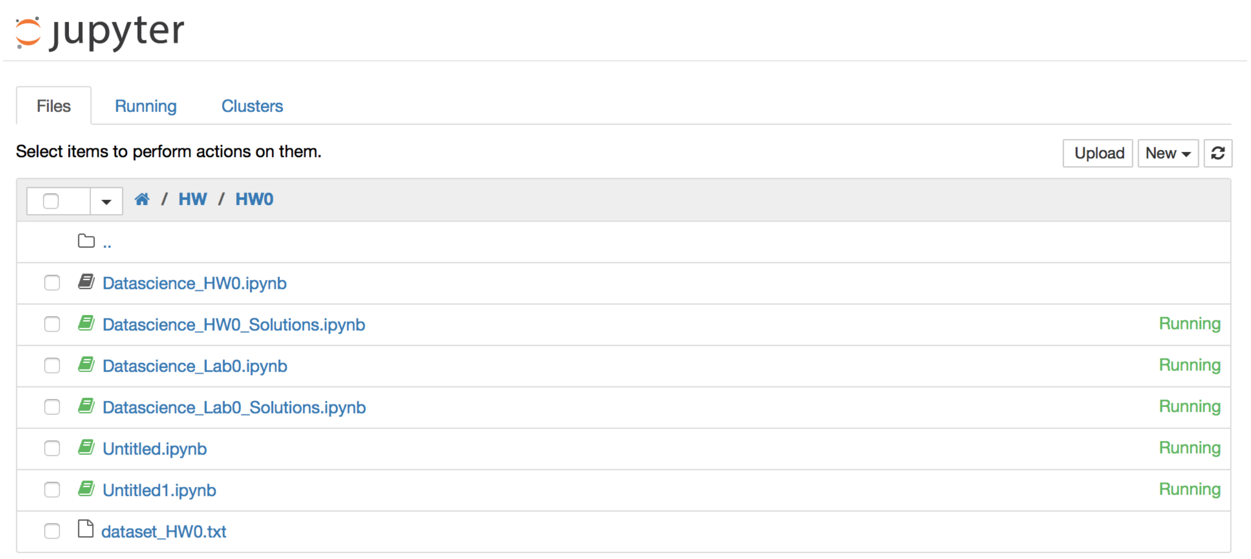
This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0));
```

angle	308.2
max_time	12
σ	10
β	2.6
ρ	28

</div>

When Jupyter app loads, you see a dashboard displaying files in the Jupyter home directory (you can reset this)



Select items to perform actions on them.

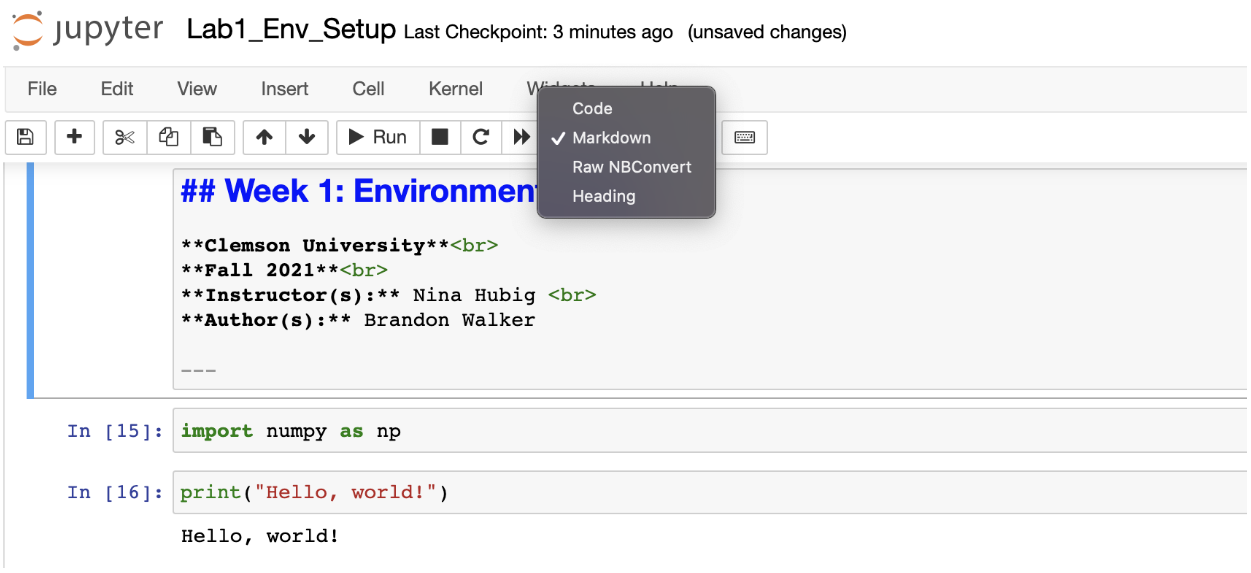
/ HW / HW0

..

Datascience_HW0.ipynb	
Datascience_HW0_Solutions.ipynb	Running
Datascience_Lab0.ipynb	Running
Datascience_Lab0_Solutions.ipynb	Running
Untitled.ipynb	Running
Untitled1.ipynb	Running
dataset_HW0.txt	

</div>

Each notebook consists of blocks of cells. Each cell can display rich text elements (Markdown) or code. Code is executed by a "computational engine" called the **kernel** . The output of the code is displayed directly below.

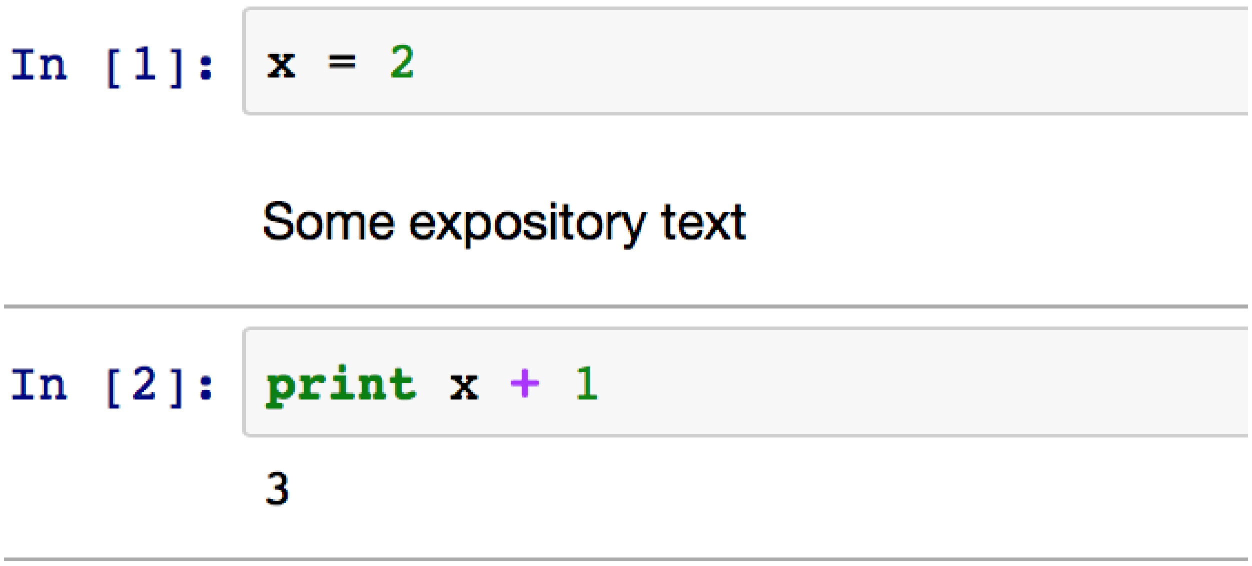


```
## Week 1: Environment

**Clemson University**<br>
**Fall 2021**<br>
**Instructor(s):** Nina Hubig <br>
**Author(s):** Brandon Walker

---
```

```
In [15]: import numpy as np

In [16]: print("Hello, world!")

         Hello, world!
```

</div>

Each cell can be executed independently, but once a block of code is executed, it lives in the memory of the kernel.



```
In [1]: x = 2
```

Some expository text

```
In [2]: print x + 1

        3
```

</div>

You'll be using them to complete labs and homework. Once you've set up Python, please download this page, and open it with Jupyter by typing

```
jupyter notebook <name_of_downloaded_file>
```

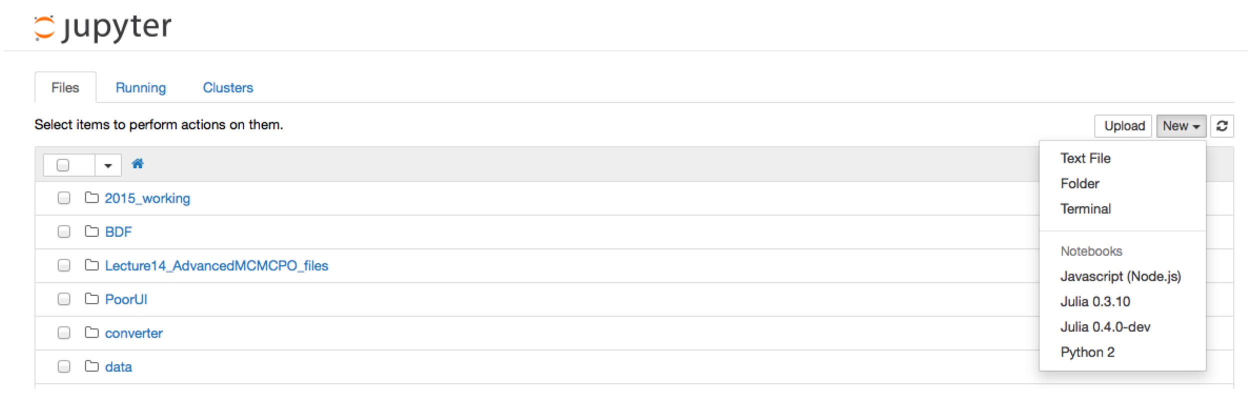
As mentioned earlier in the Mac section, you can also open the notebook in any folder by `cd` ing to the folder in the terminal, and typing

```
jupyter notebook .
```

in that folder.

The anaconda install also probably dropped a launcher on your desktop. You can use the launcher, and select "jupyter notebook" from there. In this case you will need to find out which folder you are running in.

It loolks like this for me:



Select items to perform actions on them.

2015_working
BDF
Lecture14_AdvancedMCMCPO_files
PoorUI
converter
data

</div>

Notice that you can use the user interface to create new folders and text files, and even open new terminals, all of which might come useful to you. To create a new notebook, you can use "Python 3" under notebooks. You may not have the other choices available (I have julia for example, which is another language that uses the same notebook interface).

For the rest of this setup test, use your local copy of this page, running on jupyter.

Notebooks are composed of many "cells", which can contain text (like this one), or code (like the one below). Double click on the cell below, and evaluate it by clicking the "play" button above, for by hitting shift + enter

In [2]:
```
x = [10, 20, 30, 40, 50]
for item in x:
    print("Item is {}".format(item))
```

```
Item is 10
Item is 20
Item is 30
Item is 40
Item is 50
```

You must be careful to make sure you are running the Anaconda version of python, since those operating systems come preinstalled with their own versions of python.

This is how you can see the version in the jupyter interface

In [3]:
```
import sys
print(sys.version)
```

```
3.8.2 (default, Apr  8 2021, 23:19:18)
[Clang 12.0.5 (clang-1205.0.22.9)]
```

You could also open a terminal and just type

`python` or

```
ipython
```

there. When the program starts up, you should see "Anaconda" printed out, similar to the above. If this is the case, your install went well, and you can quit the python "interpreter" by typing Ctrl-D.

If you've successfully completed the above install, skip to below the troubleshooting section. All of the statements there should run.

Troubleshooting

PROBLEM You are using a Mac or Linux computer. When you start python at the terminal or do `sys.version` in the notebook, you don't see a line like `3.5.3 |Anaconda custom (x86_64)| (default, Mar  6 2017, 12:15:08)`.

Reason You are most likely running a different version of Python, and need to modify your Path (the list of directories your computer looks through to find programs).

Solution Find a file like `.bash_profile`, `.bashrc`, or `.profile`. Open the file in a text editor, and add a line at this line at the end:

`export PATH="$HOME/anaconda/bin:$PATH"`.

Close the file, open a new terminal window, type `source ~/.profile` (or whatever file you just edited). Type

```
which python
```

-- you should see a path that points to the anaconda directory. If so, running `python` should load the proper version.

If this doesn't work (typing `which python` doesn't point to anaconda), you might be using a different shell.

Type `echo $SHELL`.

If this isn't `bash`, you need to edit a different startup file (for example, if `echo $SHELL` gives `$csh`, you need to edit your `.cshrc` file. The syntax for this file is slightly different:

```
set PATH = ($HOME/anaconda/bin $PATH)
```

PROBLEM You are running the right version of python (see above item), but are unable to import numpy.

Reason You are probably loading a different copy of numpy that is incompatible with Anaconda.

Solution See the above item to find your `.bash_profile`, `.profile`, or `.bashrc` file. Open it, and add the line `unset PYTHONPATH` at the end. Close the file, open a new terminal window, type `source ~/.profile` (or whatever file you just edited), and try again.

PROBLEM Under Windows, you receive an error message similar to the following: "'pip' is not recognized as an internal or external command, operable program or batch file."

Reason The correct Anaconda paths might not be present in your PATH variable, or Anaconda might not have installed correctly.

Solution Ensure the Anaconda directories to your path environment variable ("\Anaconda" and "\Anaconda\Scripts"). See this page for details.

If this does not correct the problem, re-install Anaconda.

IF YOU ARE STILL HAVING ISSUES ON THE INSTALL, REACH OUT TO THE COURSE STAFF FOR HELP!

Python Libraries

There are two main installing packages for Python, `conda` and `pip` . Pip is the Python Packaging Authority's recommended tool for installing packages from the **Python Package Index (PyPI)**. `Conda` is a cross platform package and environment manager that installs and manages conda packages from the **Anaconda repository** and **Anaconda Cloud**. Conda does not assume any specific configuration in your computer and will install the Python interpreter along with the other Python packages, whereas `pip` assumes that you have installed the Python interpreter in your computer. Given the fact that most operating systems do include Python this is not a problem.

If I could summarize their differences into a sentence it would be that conda has the ability to create **isolated environments** that can contain different versions of Python and/or the packages installed in them. This can be extremely useful when working with data science tools as different tools may contain conflicting requirements which could prevent them all being installed into a single environment. You can have environments with pip but would have to install a tool such as virtualenv or venv. You may use either, we recommend `conda` because in our experience it leads to fewer incompatibilities between packages and thus fewer broken environments.

Conclusion: Use Both. Most often in our data science environments we want to combining pip with conda when one or more packages are only available to install via pip. Although thousands of packages are available in the Anaconda repository, including the most popular data science, machine learning, and AI frameworks but a lot more are available on PyPI. Even if you have your environment installed via `conda` you can use `pip` to install individual packages

(source: anaconda site)

What are environments and do I need them?

Environments in Python are like sandboxes that have different versions of Python and/or packages installed in them. You can create, export, list, remove, and update environments.

Switching or moving between environments is called activating the environment. When you are done with an environments you may deactivate it.

For this class we want to have a bit more control on the packages that will be installed with the enviromnent so we will create an environment specifically for this course.

Creating an environment

You can create a new environment by running the following command in the terminal.

```
conda create -n cpsc4300 python=3.8
```

Activate the new environment:

```
source activate cpsc4300
```

You should see the name of the environment at the start of your command prompth in parenthesis.

Verify that the new environment was installed correctly:

```
conda list
```

This will give you a list of the packages installed in this environment.

References

Manage conda environments

Starting the Jupyter Notebook

Once all is installed, and your environment is active, go in the Terminal and type

```
jupyter notebook
```

to start the jupyter notebook server. This will spawn a process that will be running in the Terminal window until you are done working with the notebook. In that case press `control-C` to stop it.

Starting the notebook will bring up a browser window with your file structure.

For more on using the Notebook see: https://jupyter-notebook.readthedocs.io/en/latest/

Installing Modules

We will use specific Python Modules in this course. You can find installation instructions for most modules online, but installing a new module will typically follow this pattern:

```
conda install <module_name>
```

Before installing the module, make sure the virtual environment in which you want to install it is active. For example, to install Numpy, you would do the following:

```
# Activate environment
conda activate cpsc4300

# Install numpy
conda install numpy
```

Testing latest libraries

Run the cell below to print the version you have install for some key libraries we will use in this course. For reference, I included the version I have install on my computer. Packages are frequently updated, so you don't need to have the exact versions I have installed. However, the versions you're using should be close to, or newer than, mine.

In [1]:
```
#IPython is what you are using now to run the notebook
import IPython
print("IPython version:      %6.6s (mine is 7.24.1)" % IPython.__version__)

# Numpy is a library for working with Arrays
import numpy as np
print("Numpy version:        %6.6s (mine is 1.21.0)" % np.__version__)

# SciPy implements many different numerical algorithms
import scipy as sp
print("SciPy version:        %6.6s (mine is 1.7.0)" % sp.__version__)

# Pandas makes working with data tables easier
import pandas as pd
print("Pandas version:       %6.6s (mine is 1.2.5)" % pd.__version__)

# Module for plotting
import matplotlib
print("Matplotlib version:    %6.6s (mine is 3.4.2)" % matplotlib.__version__)

# SciKit Learn implements several Machine Learning algorithms
import sklearn
print("Scikit-Learn version: %6.6s (mine is 0.24.2)" % sklearn.__version__)

# Requests is a library for getting data from the Web
import requests
print("requests version:     %6.6s (mine is 2.25.1)" % requests.__version__)

import seaborn
print("Seaborn version: %6.6s (mine is 0.11.1)" % seaborn.__version__)
```

```
IPython version:      7.22.0 (mine is 7.24.1)
Numpy version:        1.19.5 (mine is 1.21.0)
SciPy version:         1.6.2 (mine is 1.7.0)
Pandas version:        1.2.1 (mine is 1.2.5)
Matplotlib version:     3.3.4 (mine is 3.4.2)
Scikit-Learn version: 0.24.2 (mine is 0.24.2)
requests version:     2.25.1 (mine is 2.25.1)
Seaborn version: 0.11.1 (mine is 0.11.1)
```

Kicking the tires

Lets try some things, starting from very simple, to more complex.

Hello World

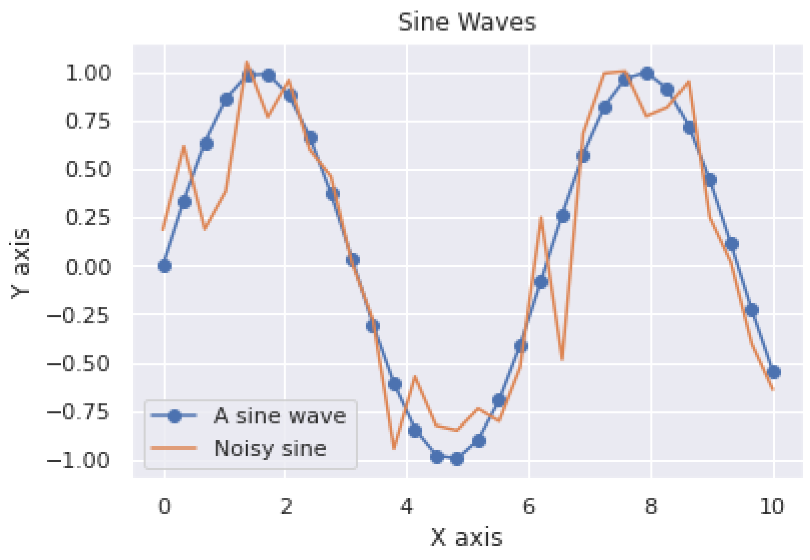
The following is the incantation we like to put at the beginning of every notebook. It loads most of the stuff we will regularly use.

In [2]:
```
# The %... is an iPython thing, and is not part of the Python language.
# In this case we're just telling the plotting library to draw things on
# the notebook, instead of on a separate window.
%matplotlib inline
#this line above prepares the jupyter notebook for working with matplotlib

# See all the "as ..." contructs? They're just aliasing the package names.
# That way we can call methods like plt.plot() instead of matplotlib.pyplot.plot
# notice we use short aliases here, and these are conventional in the python com

import numpy as np              # imports a fast numerical programming library
import scipy as sp              # imports stats functions, amongst other things
import matplotlib as mpl        # this actually imports matplotlib
import matplotlib.cm as cm      # allows us easy access to colormaps
import matplotlib.pyplot as plt # sets up plotting under plt
import pandas as pd             # lets us handle data as dataframes

import seaborn as sns # gives us more plotting options
sns.set()             # sets up styles
```

Hello matplotlib

The notebook integrates nicely with Matplotlib, the primary plotting package for python. This should embed a figure of a sine wave:

In [3]:
```
x = np.linspace(0, 10, 30)  # array of 30 points from 0 to 10
y = np.sin(x)
z = y + np.random.normal(size=30) * .2

plt.plot(x, y, 'o-', label='A sine wave')
plt.plot(x, z, '-', label='Noisy sine')
plt.legend(loc = 'best')
plt.xlabel("X axis")
plt.ylabel("Y axis")
plt.title("Sine Waves");
```



Hello Numpy

The Numpy array processing library is the basis of nearly all numerical computing in Python. Here's a 30 second crash course. For more details, consult the Numpy Documentation.

In [4]:

```
print("Make a 3 row x 4 column array of random numbers")
x = np.random.random((3, 4))
print(x,"\n")


print("Add 1 to every element")
x = x + 1
print(x,"\n")

print("Get the element at row 1, column 2")
print(x[1, 2])

# The colon syntax is called "slicing" the array.
print("Get the first row")
print(x[0, :])

print("\nLast 2 items in the first row")
print(x[0, -2:])

print("\nGet every 2nd item in the first row")
print(x[0, ::2])
```

```
Make a 3 row x 4 column array of random numbers
[[0.63882546 0.89362284 0.40407018 0.3923771 ]
 [0.76873493 0.70100644 0.72699223 0.25970835]
 [0.63645798 0.97014343 0.91514893 0.00897008]]

Add 1 to every element
[[1.63882546 1.89362284 1.40407018 1.3923771 ]
 [1.76873493 1.70100644 1.72699223 1.25970835]
 [1.63645798 1.97014343 1.91514893 1.00897008]]

Get the element at row 1, column 2
1.7269922322986964
Get the first row
[1.63882546 1.89362284 1.40407018 1.3923771 ]
```

```
Last 2 items in the first row
[1.40407018 1.3923771 ]

Get every 2nd item in the first row
[1.63882546 1.40407018]
```

Print the maximum, minimum, and mean of the array. This does **not** require writing a loop. In the code cell below, type `x.m<TAB>` , to find built-in operations for common array statistics like this

In [5]:
```
print("Max is  ", x.max())
print("Min is  ", x.min())
print("Mean is ", x.mean())
```

```
Max is   1.9701434337591233
Min is   1.008970081816894
Mean is  1.609671496156136
```

Call the `x.max` function again, but use the `axis` keyword to print the maximum of each row in x.

In [6]:
```
print(x.max(axis=1))
```

```
[1.89362284 1.76873493 1.97014343]
```

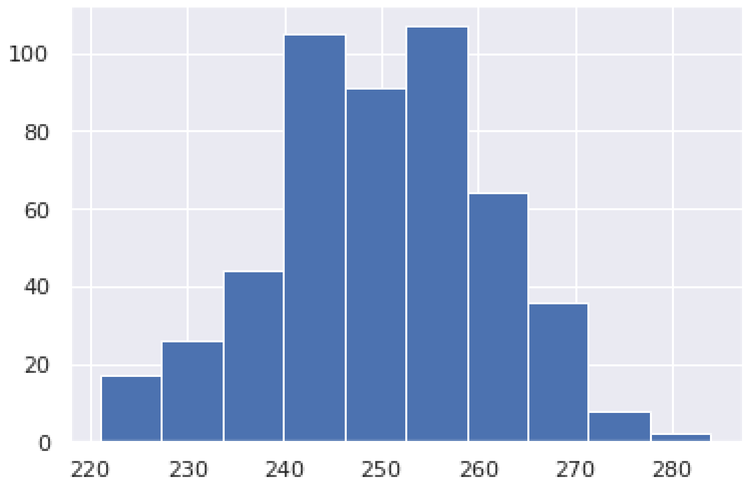
Here's a way to quickly simulate 500 coin "fair" coin tosses (where the probabily of getting Heads is 50%, or 0.5)

In [7]:
```
x = np.random.binomial(500, .5)
print("number of heads:", x)
```

```
number of heads: 253
```

Repeat this simulation 500 times, and use the plt.hist() function to plot a histogram of the number of Heads (1s) in each simulation

In [8]:
```
# 3 ways to run the simulations

# loop
heads = []
for i in range(500):
    heads.append(np.random.binomial(500, .5))

# "list comprehension"
heads = [np.random.binomial(500, .5) for i in range(500)]

# pure numpy, preferred
heads = np.random.binomial(500, .5, size=500)

plt.hist(heads, bins=10);
```



```
In [ ]:
```