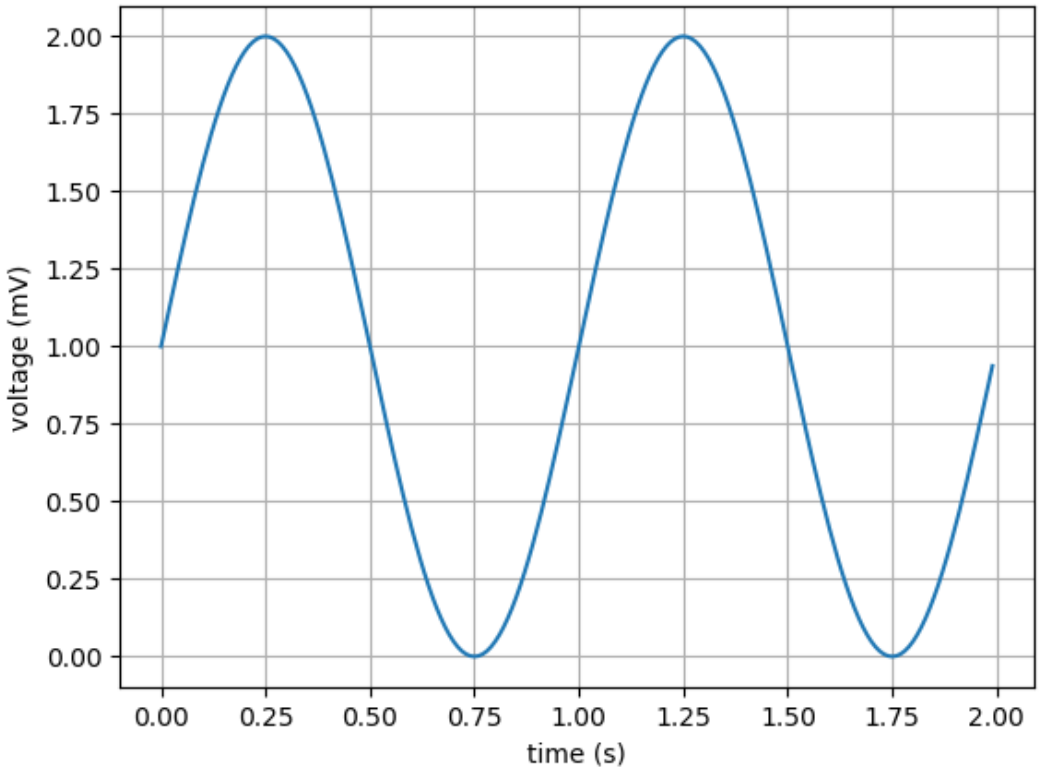
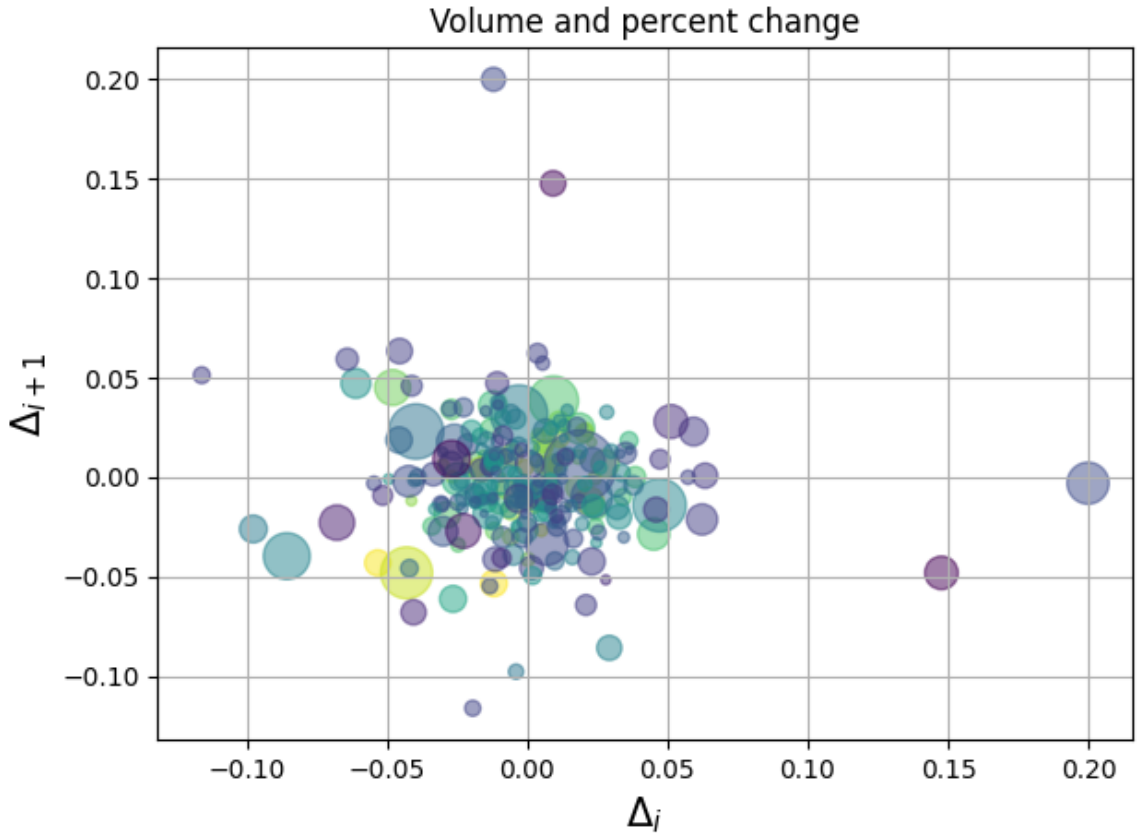
3 - Matplotlib

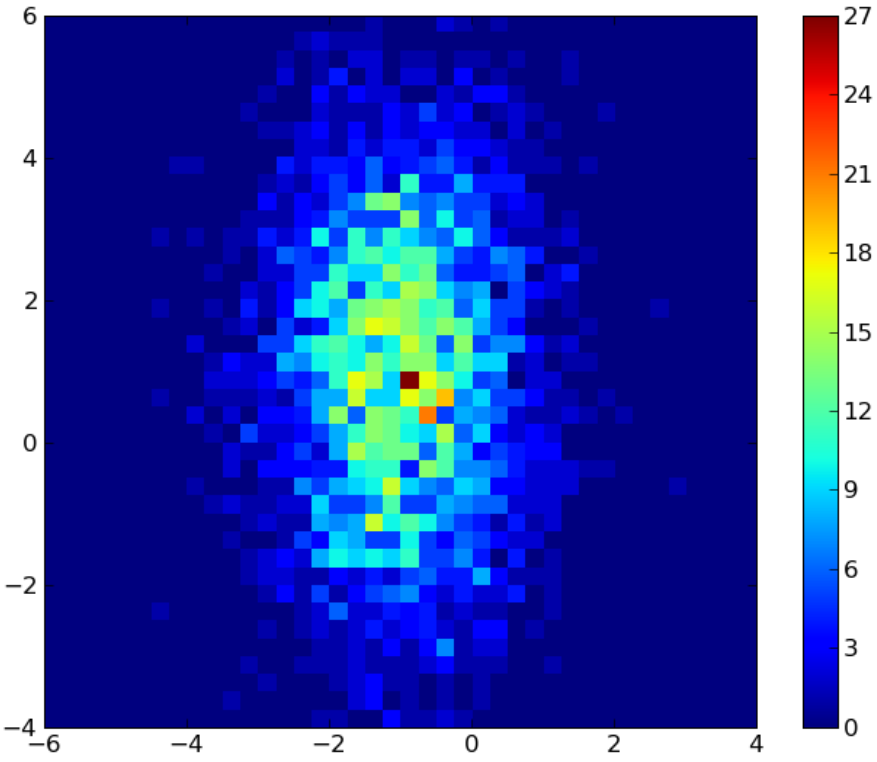
One the most important aspects of data science is *visualisation*. Ultimately, processing data and producing insight is only useful if you can *communicate your results*. One of the most effective ways of doing that is by making colourful, intuitive designs that explain clearly what can be challenging concepts. In Python one of the most popular tools for doing this is called **Matplotlib**.

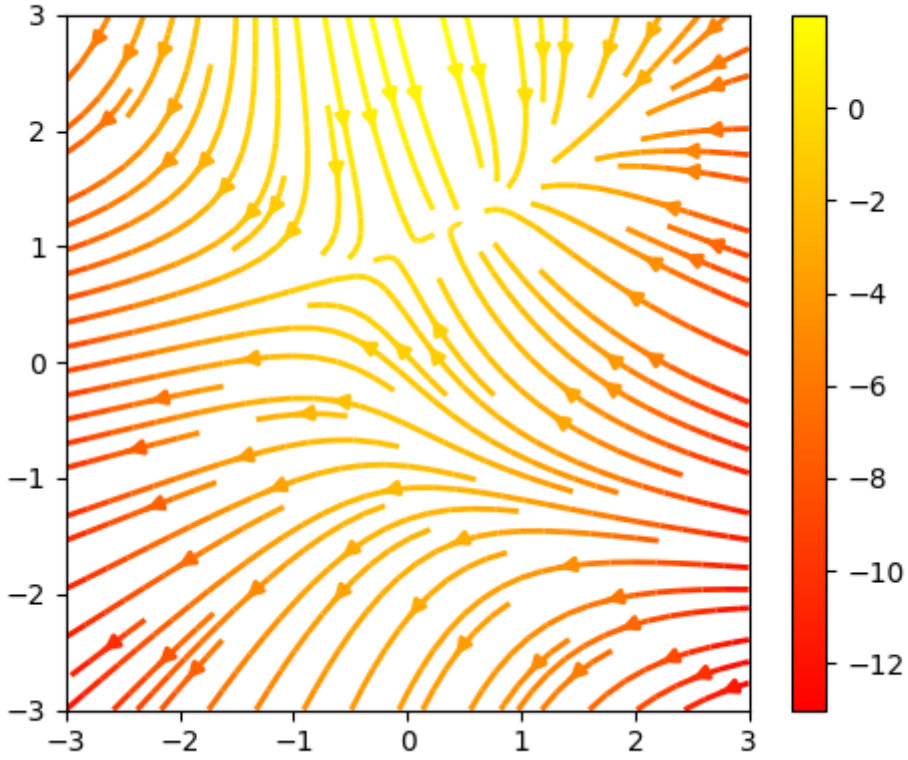
Matploltib is a an example of what's known as a *library*. A library is made up of pieces of code that other people have written which you can use directly in your own projects. A It's really useful to know some of the libraries that are out there - they can make your life a lot easier!

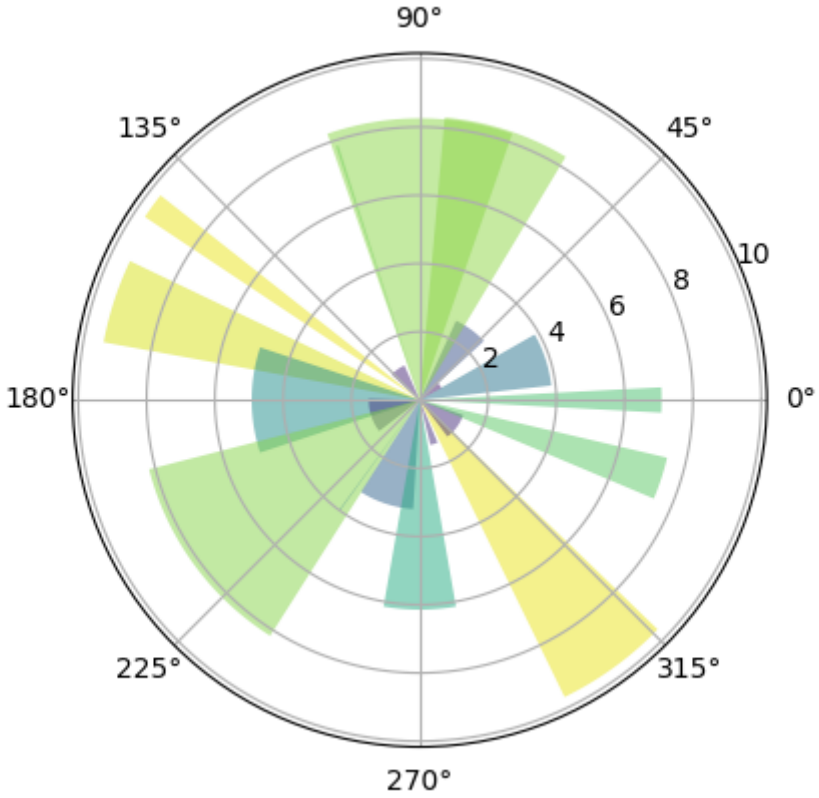
Here are some examples of the types of plot you can make with matplotlib

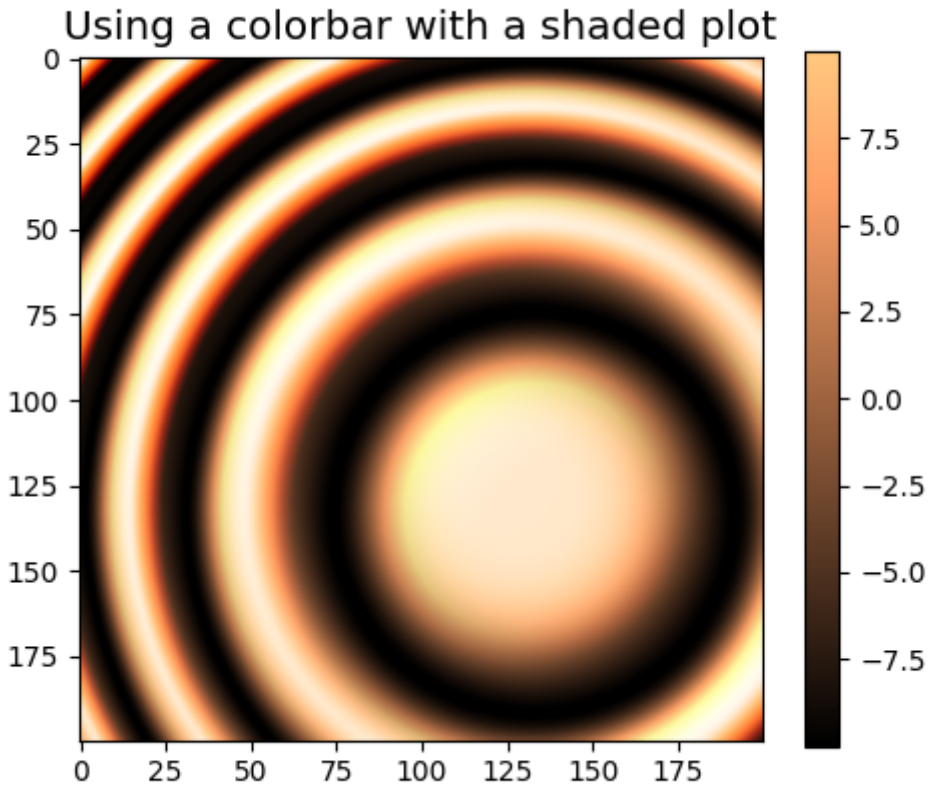


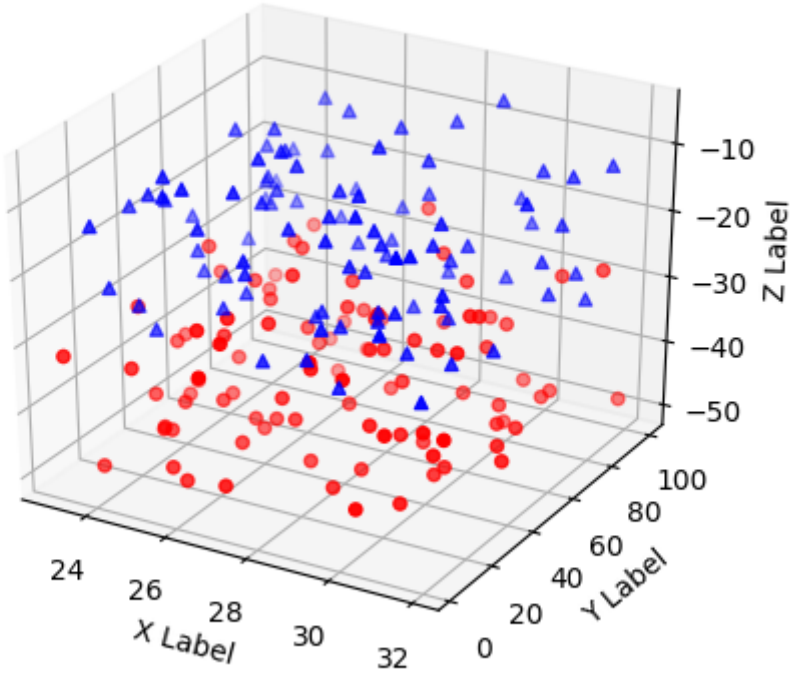

Volume and percent change
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
Δ_{i+1}
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
Δ_i

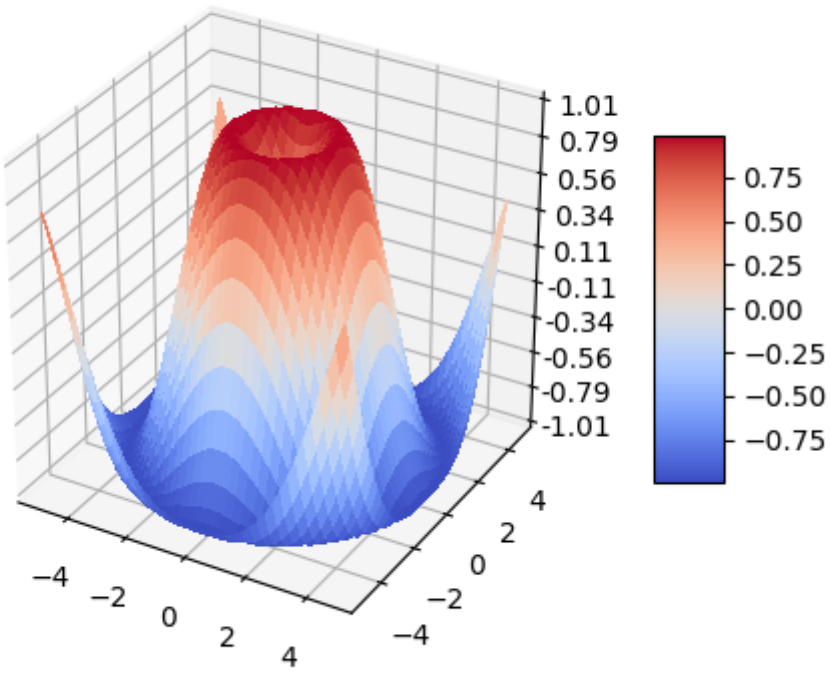

6
4
2
0
−2
−4
−6
−4
−2
0
2
4
27
24
21
18
15
12
9
6
3
0


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
0
−2
−4
−6
−8
−10
−12


90°
135°
45°
180°
0°
225°
315°
270°
2
4
6
8
10


Using a colorbar with a shaded plot
0
25
50
75
100
125
150
175
0
25
50
75
100
125
150
175
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5


−10
−20
−30
−40
−50
Z Label
100
80
60
40
20
0
Y Label
24
26
28
30
32
X Label



For many many more examples of what's possible with matplotlib, checkout this link where you can find the graphs along with the code that created them.

As you can see, there is a huge variety of options. Today we are going to focus on three simple types of plot: Scatter plots, line plots and histograms.

Installing Matplotlib

Unfortunately, matplotlib doesn't come automatically with Python, so before we can use it, we first have to install it. This will be a bit different how you have installed software before, but luckily it shouldn't be too difficult.

What you need to do is to open up the command line in exactly the same way as you did to start this notebook. But then, instead of typing

```
jupyter notebook
```

you need to type

```
pip install matplotlib
```

and hit enter. That's it! It shouldn't take more than a couple of minutes to install. If you get stuck, try using google to find help. If all is well you will get a message saying something like

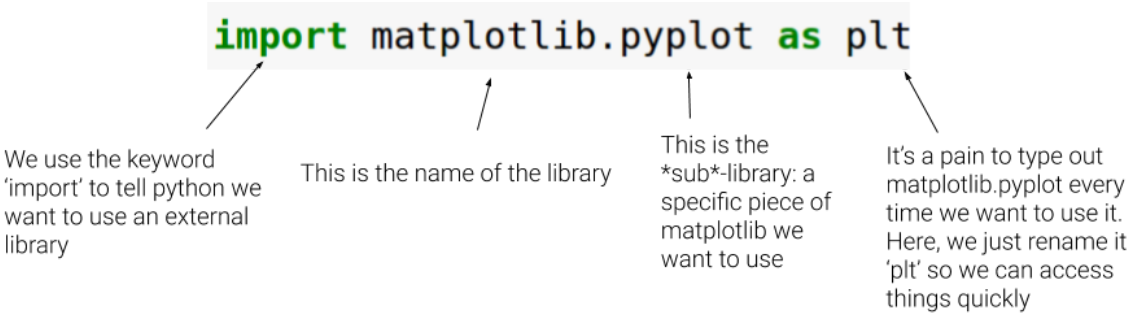
```
Successfully installed Matplotlib
```

Importing Matplotlib

If that all went smoothly you should be ready to import matplotlib in Jupyter. The way we do this in Python is as follows. Simply run the cell below as you would with any other cell

```
import matplotlib.pyplot as plt
```

Here's a quick explanation of what that means



We can now use all of matplotlib's functions simply by using the word `plt`! Another useful tip however is to run the line

```
%matplotlib notebook
```

twice. This is a bit confusing! It just allows us to get nice interactive plots - don't worry about it too much.

```
# RUN THIS CELL!
%matplotlib notebook
```

```
# AND THIS ONE!! :P
%matplotlib notebook
```

Now we're ready to rock!

1. Scatter Plots

Scatter plots are one of the most useful types of plot for data science. They allow us to show the *relationship* between two variables. Let's take an example

Example: your ice cream business!

Let's say we run an ice cream business and we want to understand how the temperature on a given day affects the number of ice creams we are likely to sell. Every day we go measure the temperature and then add up the total revenue from all our ice cream vans. This gives us a set of pairs of values on each day: temperature and revenue. We can visualise this information on a scatter plot by turning each observation into a point on a graph. The horizontal position will represent the temperature, and the height will represent the ice cream revenue. Doing this for every data point collected will allow us to see the relationship.

Some real data

Below you will see a list holding the temperature that was measured on each day.

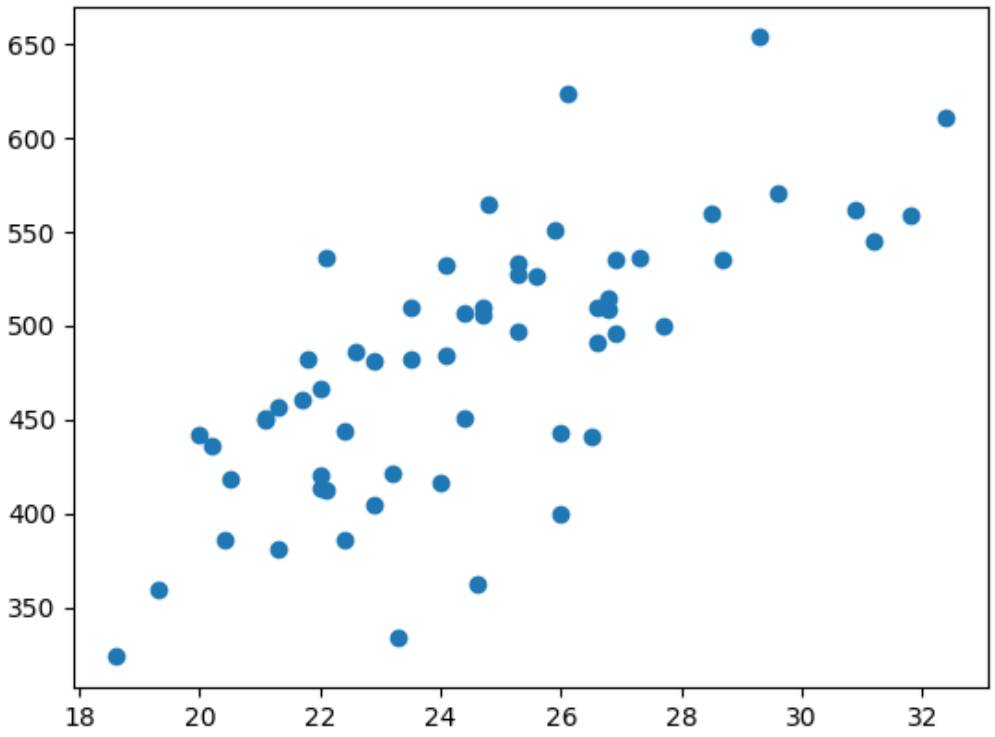
REMEMBER A list is an ordered sequence of things separated by commas. Here we have a list of numbers where each one represents the temperature on a given day.

```
temperature = [20.2, 22.0, 18.6, 25.3, 24.4, 20.0, 21.3, 23.5, 26.8, 22.9, 22.4,
               20.5, 24.0, 26.8, 25.3, 32.4, 19.3, 28.5, 24.7, 21.1, 23.2, 20.4,
               29.3, 25.9, 24.1, 25.6, 22.9, 26.5, 22.0, 28.7, 25.3, 24.7, 22.0,
               27.3, 26.1, 24.1, 31.2, 31.8, 23.5, 30.9, 26.9, 27.7, 26.0, 26.6,
               26.6, 26.0, 26.9, 22.6, 29.6, 23.3, 22.4, 24.6, 21.3, 21.1, 22.1,
               24.8, 24.4, 22.1, 21.8, 21.7]
```

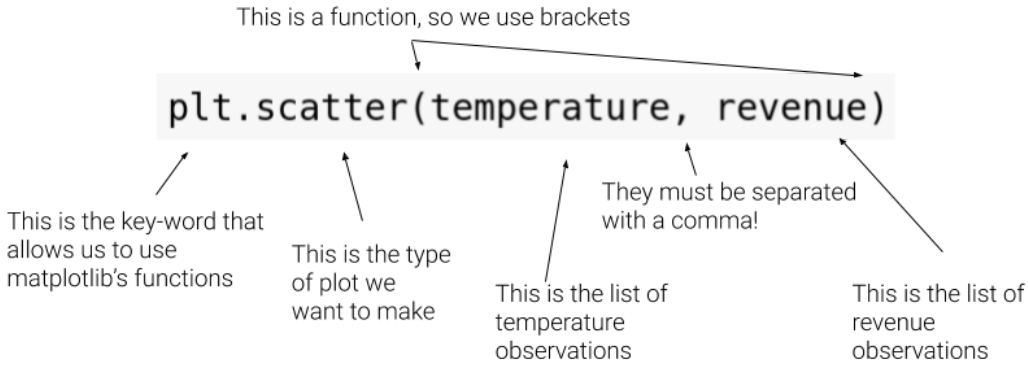
We also collect the total revenue in pounds on each day, which we have also put in a list

```
revenue = [435.83, 466.11, 323.65, 533.20, 506.79, 441.77, 456.31, 509.70,
           514.45, 404.82, 443.76, 418.71, 416.32, 508.53, 527.74, 611.48,
           359.78, 560.27, 510.14, 449.90, 421.14, 386.44, 653.93, 551.34,
           532.58, 526.22, 481.29, 441.01, 419.96, 535.47, 497.20, 505.45,
           413.64, 536.81, 624.03, 484.09, 545.41, 558.87, 482.68, 562.13,
           535.82, 499.59, 399.93, 509.68, 490.77, 443.27, 495.78, 486.18,
           570.36, 333.90, 385.74, 362.59, 380.74, 450.97, 412.60, 564.99,
           450.49, 536.18, 482.17, 460.85]
```

```
# RUN ME! :D
plt.figure()
plt.scatter(temperature, revenue)
```

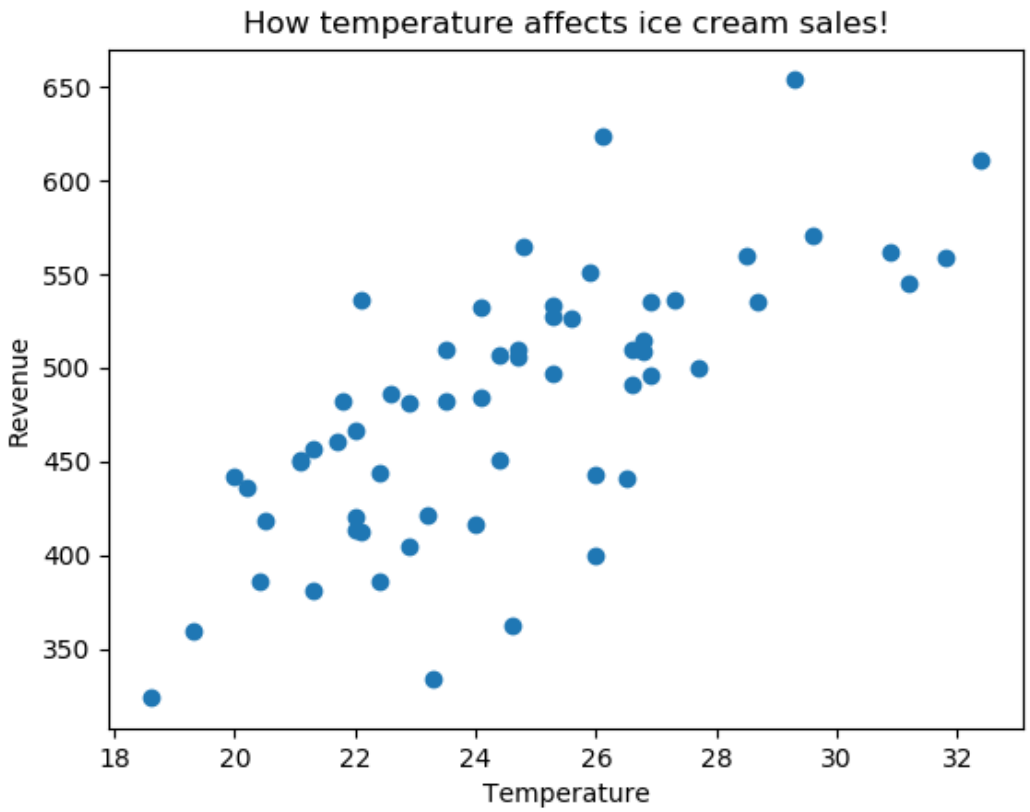


Hey-presto! We have a graph. The first line tells matplotlib we want to make a new figure. The second line is where the magic happens. This is where we plot our graph



When making a plot, it's always a good idea to have axis labels, preferably with a unit. Often it's nice to have a title too. We can do this using some more matplotlib commands

```
# RUN ME! :D
plt.figure()
plt.scatter(temperature, revenue)
plt.xlabel('Temperature')
plt.ylabel('Revenue')
plt.title('How temperature affects ice cream sales!')
```



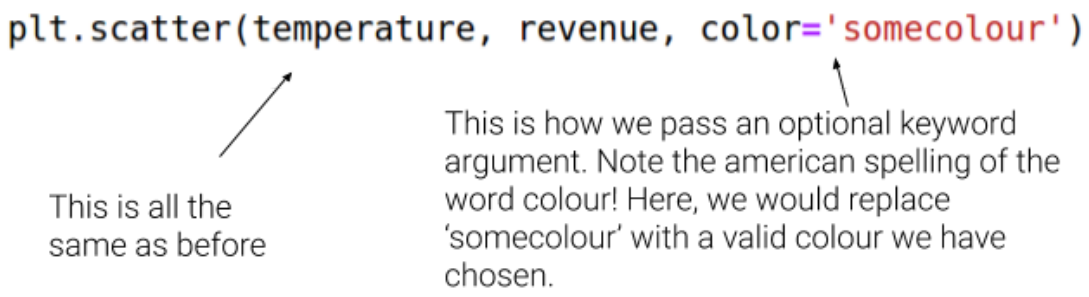
Notice that the text must be in quotes, because it is a string!

Exercise 1

Change the text in the code above so that it's clear what units temperature and revenue are being measured in. *Remember* units are always important.

Colors

That graph is a great start, but we can do better. One way we can make our plots more visually appealing is by making use of the wide range of colors that matplotlib provides for us. The way that we choose a color for our points is by passing in an extra keyword argument into the function `plt.scatter()`.



When we actually run this, we need to replace 'somecolour' with a valid colour. Here are a list of some of the possible valid colours in matplotlib.

Exercise 2

Copy the code from above. Now try adding the color keyword to `plt.scatter`. Choose one from the list above.

```
# YOUR CODE HERE :P
```

Multiple Scatters on one Graph

Often we want to compare the relationships of two or more distinct groups. In this example we are comparing the height and weight of a group of Olympic table-tennis players, and a group of Olympic weight lifters. Data taken from here!. Take a look at the lists below

```
# height of olympic table-tennis players in cm
tt_height = [170, 176, 178, 173, 183, 175, 186, 198, 172, 192, 168,
             175, 186, 182, 180, 166, 186, 178, 170, 175, 174, 180,
             178, 168, 185, 170, 168, 185, 184, 174, 178, 180, 177,
             173, 174, 179, 166, 170, 180, 183, 186, 181, 182, 173,
             182, 183, 196, 166, 176, 162, 186, 182, 187, 189, 174,
             170, 185, 193, 184, 175, 191, 182, 178, 180, 178]

# weight of olympic table-tennis players in kg
tt_weight = [72, 74, 76, 67, 54, 64, 67, 99, 63, 80, 60, 68, 71,
             74, 68, 51, 85, 77, 68, 67, 70, 74, 70, 61, 78, 65,
             69, 85, 75, 66, 85, 69, 77, 65, 80, 80, 60, 67, 75,
             73, 69, 74, 72, 73, 65, 82, 93, 73, 80, 51, 78, 77,
             85, 83, 68, 70, 76, 82, 77, 72, 87, 70, 72, 81, 65]

# height of olympic weight-lifters in cm
wl_height = [171, 180, 176, 185, 171, 182, 174, 183, 179, 165,
             180, 182, 176, 180, 183, 177, 193, 178, 173, 182, 183,
             185, 175, 180, 186, 181, 182, 186, 180, 183, 187, 172,
             172, 177, 170, 176, 180, 187, 175, 178, 185, 170]

# weight of olympic weight-lifters in kg
wl_weight = [100.5,  85, 105, 108,  96,  92, 105, 146,
             103, 101,  98.5, 108,  94, 107, 105, 131, 106,
             102, 105,  97.5, 109, 125, 118.5,  99, 107, 105,
             105, 100, 104, 108, 105, 105,  98.5, 106, 103,
             104.5, 104, 126, 107,  94, 101, 105.5]
```

Labels

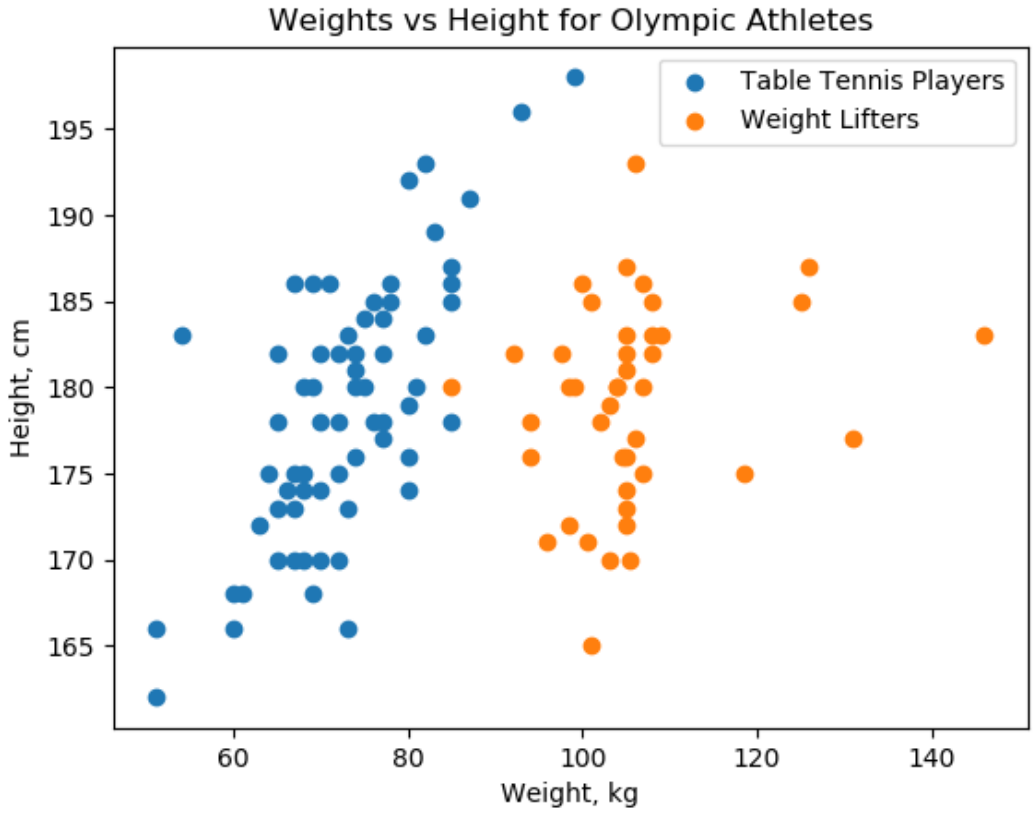
Once we have more than one thing plotted on the same graph, we will often want to have labels for the different things. This can be achieved by passing another keyword into scatter function called `label`. The other thing we need is to use the matplotlib function

```
plt.legend()
```

This tells matplotlib to add a key to the plot (legend is another word for key).

```
plt.figure()
plt.scatter(tt_weight, tt_height, label='Table Tennis Players')
plt.scatter(wl_weight, wl_height, label='Weight Lifters')
plt.legend()

plt.xlabel('Weight, kg')
plt.ylabel('Height, cm')
plt.title('Weights vs Height for Olympic Athletes')
```




```
plt.scatter(tt_weight, tt_height, label='My First Label!!')
```

Scatter function as before

The x-variable

The y-variable

Another keyword argument. Remember to wrap your text in quote marks - it is a string!

Exercise 3

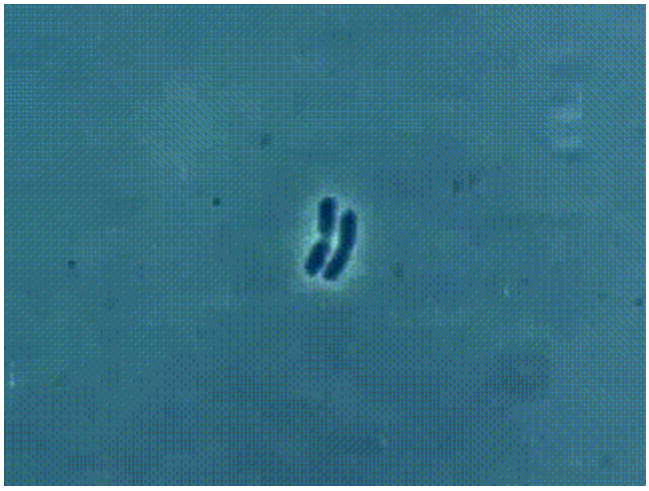
Copy the code from above. Try changing the labels. You can also individually change the colours of each scatter plot.

```
# YOUR CODE HERE :S
```

2. Line Plots

This is the second very important class of plots. Line plots are useful when we want to show some kind of line or curve. Here we don't want to necessarily show some noisy relationship between two variables - we want to show some sequence that has a specific order. That's all a bit abstract so let's take an example.

In biology people often study the way that cells multiple. A biologist is looking at the growth and development of some E.coli bacteria cells under a microscope. She starts with one cell, and then every half hour for 24 hours a computer takes a photo and counts the number of cells present.



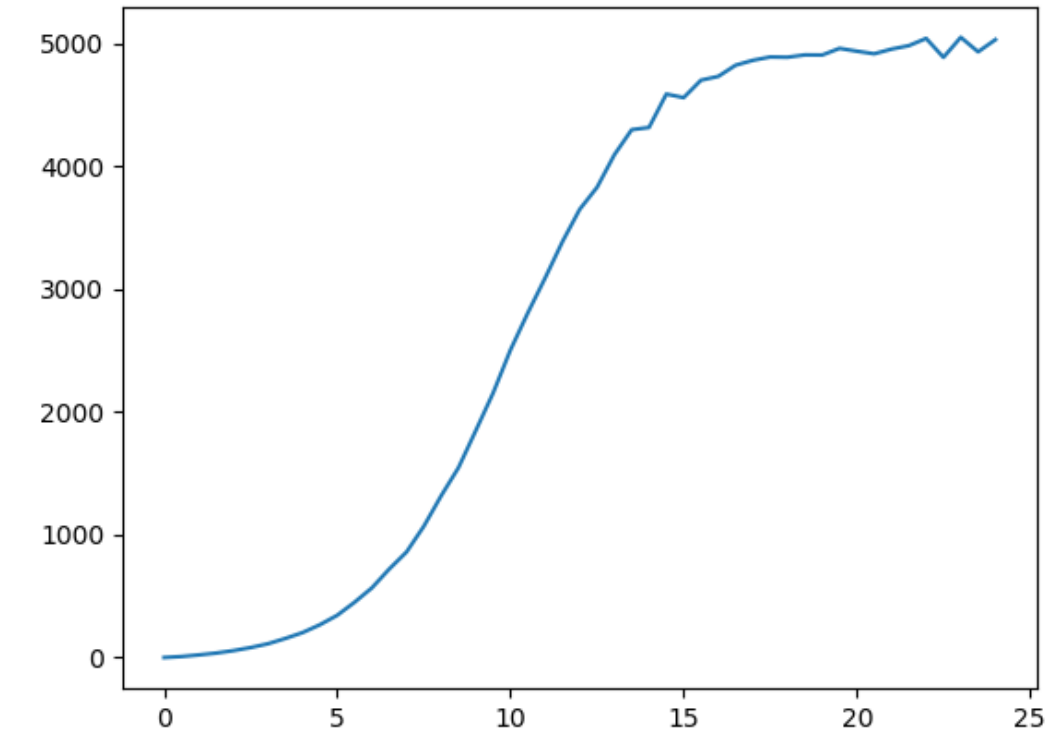
When she arrives in work the following morning, she observes the following count data

```
hour = [0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0,
7.5,
        8.0, 8.5, 9.0, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5,
14.0, 14.5,
        15.0, 15.5, 16.0, 16.5, 17.0, 17.5, 18.0, 18.5, 19.0, 19.5, 20.0, 20.5,
21.0,
        21.5, 22.0, 22.5, 23.0, 23.5, 24.0]

count = [1,    9,    22,   37,   57,   82,   113,  156,  204,  268,  345,
         451,  568,  721,  860,  1070, 1318, 1547, 1849, 2155, 2505, 2806,
         3090, 3386, 3650, 3829, 4094, 4298, 4316, 4588, 4559, 4701, 4732,
         4823, 4863, 4890, 4888, 4907, 4906, 4960, 4937, 4916, 4954, 4982,
         5041, 4887, 5050, 4932, 5031]
```

We can represent this data using a line plot

```
plt.figure()
plt.plot(hour, count)
```




```
plt.plot(hour, count)
```

This is a new matplotlib function called plot

These will be our x-points

These will be our y-points

This is actually very similar to how we make scatter plots: we pass a list of x-values and a list of y-values. However in this case, instead of getting a single point at each (x, y) pair, we get a solid line connecting them together in order.

We can still do all the same things with labels, titles, and colours.

Exercise 4

Try changing the colour and adding axis labels for the above plot.

```
# YOUR CODE HERE :J
```

Multiple lines, one plot

The biologist is testing an experimental gene-editing technique that slows the rate at which E.coli can multiply. She takes a new cell, which has had it genes edited, and performs the same experiment, counting the number of cells present every half hour for 24 hours. This time she observes the following count data

```
count_edited = [1,    10,   20,   31,   43,   56,   71,   89,   106,  126,  150,
                171,  197,  226,  256,  293,  322,  368,  410,  454,  508,  565,
                629,  672,  745,  827,  907,  983,  1063, 1137, 1243, 1337,
1471,
                1592, 1692, 1785, 1942, 2053, 2166, 2258, 2366, 2494, 2687,
2741,
                2886, 3019, 3129, 3178, 3374]
```

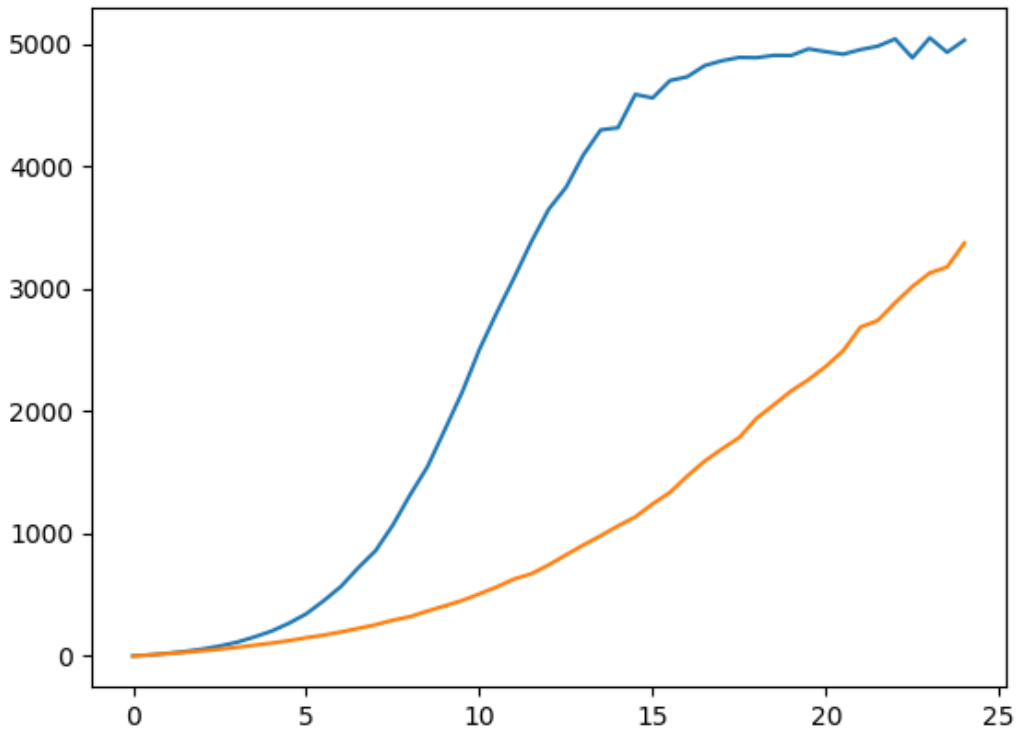
Exercise 5

Edit the code for the graph below to clearly show the results of the biologist. Your plot should include

- Axis labels
- A title
- A key
- Bright colors

From looking at this graph, what can we say about the biologist's new gene editing technique?

```
# EDIT THIS CODE :O
plt.figure()
plt.plot(hour, count)
plt.plot(hour, count_edited)
```



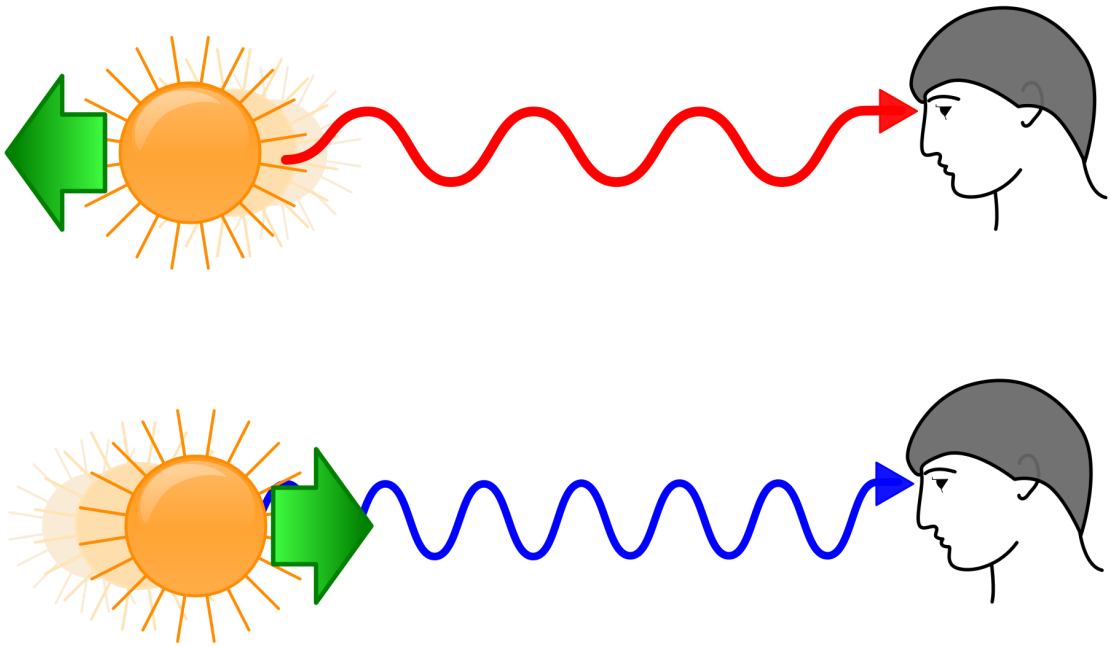
3. Histograms

The third and final type of plot we will look at is a histogram. Histograms are used when we want to show the distribution of a values in a certain dataset. They do this by indicating the number of data points that lie within a range of values.

Here's an example.

A team of physicists are working with image data taken from the Hubble telescope, which is mounted on a satellite orbiting earth. Every 10 minutes they receive a photo that looks something like this

Every single bright spot is a distant galaxy, made up of many billions of stars. The universe is expanding. That means galaxies that are further away move away from us faster. The scientists can estimate how far away a galaxy is by measuring it's *redshift*. Ever notice how an ambulance siren appears lower-pitched as it's moving away from you? The same thing happens with light. When something is moving away very fast, the light wave gets stretched out making it appear more red



This means, since further objects move away faster, is we know how fast something is moving away from us, we can estimate *how far it is*. The scientists want to know about the distribution of galaxy distances. Are there more galaxies closer? Or more further away? Using the technique of redshift, they compile a list of the approximate distances to all the galaxies they find in a single image, measured in billions of light years.

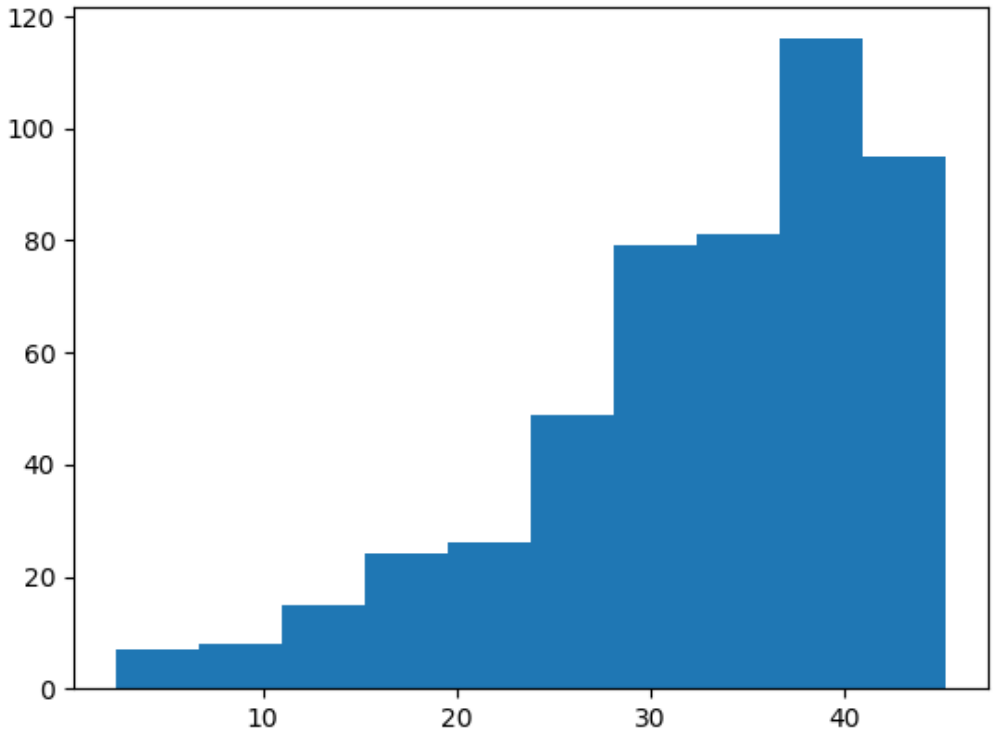
```
# galaxy distances in billions of light years

distances = [ 31.61, 32.63, 32.53, 42.21, 33.66, 41.33, 13.55, 25.9, 32.75,
34.33, 36.73, 41.22, 41.5, 28.24,
              33.47, 34.12, 32.45, 31.72, 36.56, 30.39, 39.04, 30.72, 14.39,
45.21, 42.03, 24.57, 38.68, 32.34,
              11.86, 40.58, 41.22, 35.32, 29.21, 37.83, 42.21, 37.79, 29.33,
36.91, 6.04, 37.46, 27.57, 13.58,
              28.62, 33.57, 16.79, 27.16, 27.2, 25.74, 23.74, 41.93, 40.69,
36.15, 25.48, 18.96, 37.4, 38.14,
              40.24, 31.98, 41.84, 11.31, 38.47, 36.29, 29.92, 32.71, 30.38,
25.3, 41.96, 32.38, 20.69, 16.53,
              37.2, 43.65, 22.38, 32.09, 23.06, 39.22, 31.61, 30.96, 26.07,
39.57, 36.12, 25.12, 37.08, 40.28,
              35.85, 23.83, 37.38, 2.38, 39.42, 37.01, 34.55, 29.23, 36.8, 33.31,
29.86, 42.86, 18.95, 43.5,
              27.06, 38.33, 38.51, 38.77, 29.91, 42.24, 34.2, 39.07, 30.89,
17.58, 15.16, 36.12, 33.75, 29.93,
              43.1, 19.1, 42.06, 40.49, 33.04, 22.56, 28.23, 30.26, 37.8, 4.31,
44.79, 29.2, 42.48, 38.37,
              18.18, 32.97, 41.44, 26.06, 16.55, 34.24, 41.93, 9.34, 43.42,
39.22,  6.04, 38.13, 35.98, 36.01,
              22.41, 38.93, 34.17, 36.79, 39.54, 40.97, 43.27, 30.31, 37.63,
43.19, 8.09, 33.52, 21.7, 42.23,
              39.56, 31.75, 42.39, 41.64, 44.43, 9.87, 39.97, 43.54, 25.57,
25.95, 20.92, 42.92, 32.08, 37.29,
              38.23, 37.96, 43.28, 35.67, 6.38, 23.03, 31.66, 25.39, 30.83,
28.65, 42.51, 36.47, 42.87, 40.62,
              38.98, 31.52, 30.65, 36.11, 42.28, 39.44, 42.24, 38.45, 25.17,
41.99, 18.73, 31.05, 41.52, 42.83,
```

```
            15.75, 29.76, 42.46, 7.15, 37.37, 40.1, 40.03, 39.44, 41.17, 37.78,
29.97, 31.99, 43.75, 37.89,
            10.99, 41.56, 9.26, 43.89, 41.06, 34.91, 41.18, 24.99, 29.89,
43.83, 25.73, 39.59, 36.63, 24.39,
            38.91, 34.67, 37.31, 37.01, 43.7, 44.99, 36.27, 32.01, 31.95,
29.21, 39.1, 27.25, 38.1, 35.87,
            41.73, 37.28, 23.41, 30.91, 28.08, 25.36, 33.47, 30.42, 27.71,
39.04, 38.46, 34.61, 24.56, 29.25,
            26.97, 37.98, 45.26, 36.93, 31.38, 42.24, 42.53, 14.34, 28.67,
43.27, 37.05, 15.44, 26.06, 43.67,
            38.1, 40.82, 28.92, 40.94, 32.48, 14.39, 18.95, 25.18, 37.92,
39.16, 32.03, 38.51, 42., 30.87,
            27.22, 42.13, 30.12, 13.29, 3.35, 35.6, 39.44, 29.28, 27.39, 30.42,
38.75, 34.51, 26.95, 31.07,
            37.57, 35.59, 44.44, 41.63, 38.78, 27.12, 33.15, 38.14, 19.33,
37.7, 39.92, 39.05, 41.1, 31.38,
            43.16, 33.11, 36.6, 20.34, 40.23, 21.26, 21.41, 39.79, 42.81,
38.69, 31.96, 15.44, 36.29, 39.97,
            41.68, 41.07, 27.12, 22.09, 32.25, 10.9, 24.32, 25.54, 17.21, 41.,
41.86, 41.91, 41.05, 35.87,
            43.69, 41.07, 40.08, 26.43, 15.71, 33.65, 29.52, 25.45,  5.39,
24.58, 21.81, 36.28, 37.78, 35.38,
            38.08, 17.35, 39.92, 42.45, 20.68, 35.07, 38.67, 40.97, 33.42,
15.41, 33., 36.03, 34.68, 15.19,
            29.33, 40.73, 23.59, 42.26, 18.5, 23.28, 41.65, 11.4, 26.77, 31.67,
39.58, 40.21, 41.23, 36.17,
            27.66, 40.61, 38.49, 28.4, 33.77, 39.51, 36.97, 24.83, 31.97,
26.16, 14.64, 25.4, 8.76, 43.95,
            40.34, 42.54, 21.84, 22.04, 12.83, 27.16, 38.79, 26.28, 14.52,
39.23, 37.11, 29.29, 25.6, 38.17,
            24.41, 39.44, 41.15, 42.59, 37.75, 41.19, 35.02, 42.76, 40.85,
42.55, 32.57, 44.33, 37.36, 37.01,
            41.98, 15.37, 44.31, 42.19, 40.36, 37.6, 36.66, 33.6, 20.15, 41.39,
18.06, 33.17, 38.21, 35.07,
            32.42, 31.84, 37.45, 20.32, 24.31, 34.74, 29.77, 32.92, 29.74,
37.01, 31.71, 36.02, 42.29, 42.53,
            34.5, 35.14, 33., 31.55, 41.66, 42.35, 31.83, 43.51, 34.7, 39.21,
31.51, 34.96, 23.33, 18.51, 29.37,
            28.85, 32.55, 32.1, 34.36, 29.38, 33.51, 31.52, 41.89, 44.2, 38.9,
44.32, 32.03, 31.89, 38.42,
            30.99, 32.25, 33.61, 33.64, 36.67, 39.22, 34.65, 25.58, 18.5,
29.28, 15.75, 35.62, 37.06, 21.46,
            39.05, 21.81, 10.9, 35.33, 19.91, 38.47, 30.32, 41.09, 41.02]
```

They then want to plot a histogram, to get a sense of this distribution. In matplotlib, we can create a histogram simply by typing

```
plt.figure()
plt.hist(distances)
```



Nice! What exactly is this plot showing?

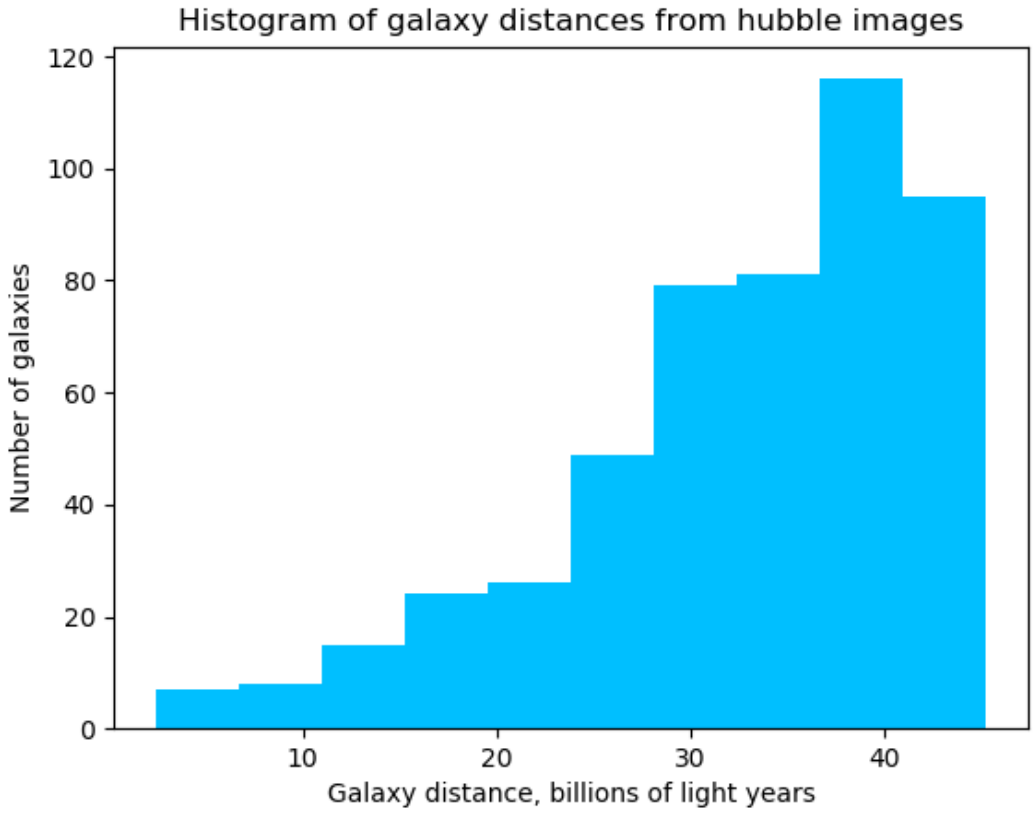
Along the x-axis we have the galaxy distance. Along the y-axis, we have the number of galaxies thhat fall within that range. From this graph, how many galxies were counted between roughly 28 and 32 billion light years away?

Something to think about

What does this histogram tell us about the distribution of galaxy distances from the hubble images?

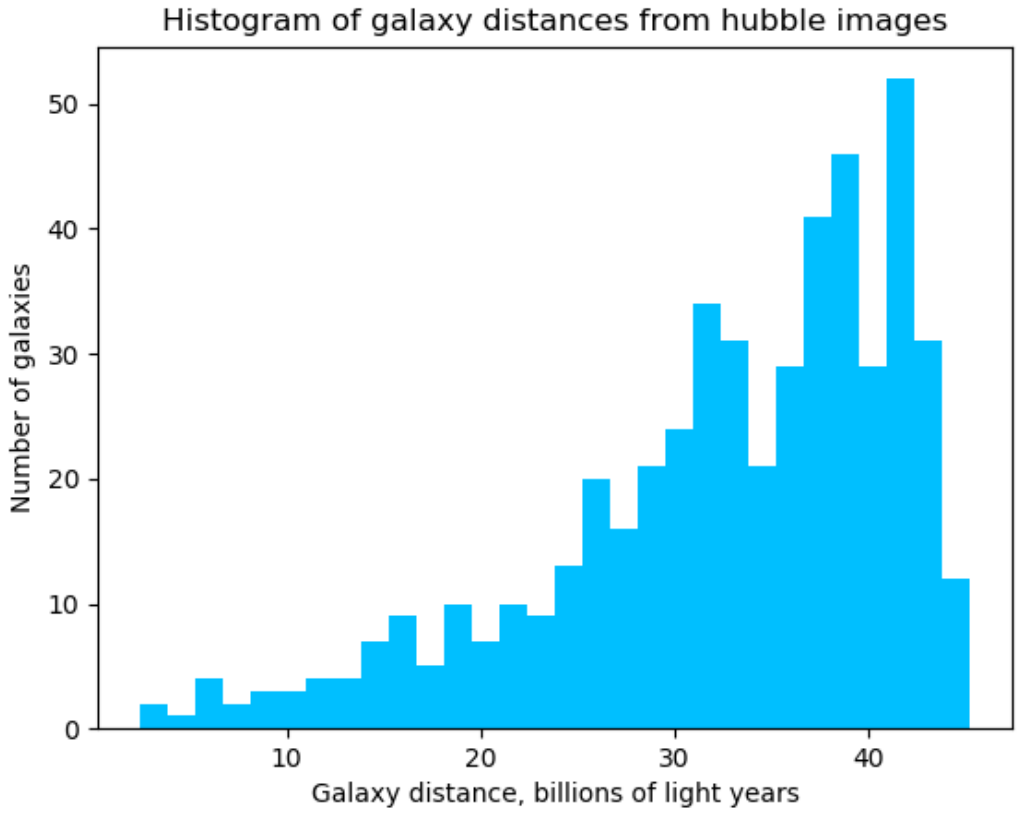
As before, we can adjust the colour and add axis labels.

```
plt.figure()
plt.hist(distances, color='deepskyblue')
plt.xlabel('Galaxy distance, billions of light years')
plt.ylabel('Number of galaxies')
plt.title('Histogram of galaxy distances from hubble images')
```



Another thing we might want to do is adjust the number of bars we have. This can be controlled by the *bins* parameter.

```
plt.figure()
plt.hist(distances, color='deepskyblue', bins=30)
plt.xlabel('Galaxy distance, billions of light years')
plt.ylabel('Number of galaxies')
plt.title('Histogram of galaxy distances from hubble images')
```



Exercise 6

Try adjusting the number of bins in the histogram. What do you think is appropriate for this data set?

That's all folks :P