Line Graphs in Matplotlib

Nouman M Durrani

1 Introduction to Matplotlib

- Line Graphs in Matplotlib — Interactive Lesson
- Line Graphs Concepts — Multiple Choice Quiz
- Sublime Limes' Line Graphs — Freeform Project
- Different Plot Types — Interactive Lesson
- Plot Types — Multiple Choice Quiz
- How to Select a Meaningful Visualization — Article
- Recreate Graphs using Matplotlib! — Interactive Lesson

1. Introduction
2. Basic Line Plot
3. Basic Line Plot II
4. Linestyles
5. Axis and Labels
6. Labeling the Axes
7. Subplots
8. Subplots Part II
9. Legends
10. Modify Ticks
11. Figures

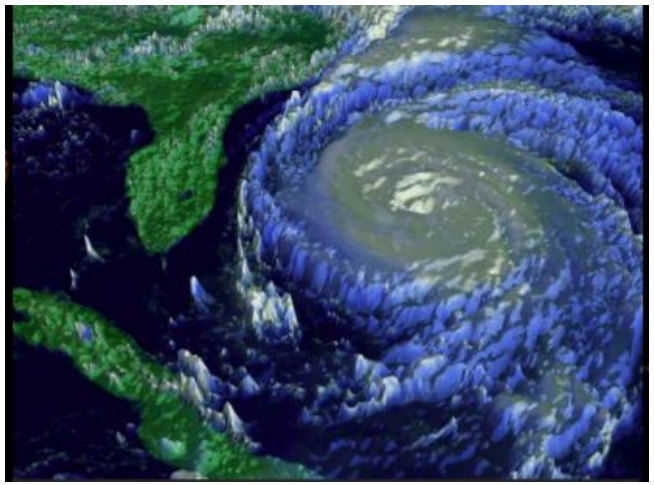
2 Introduction to Seaborn

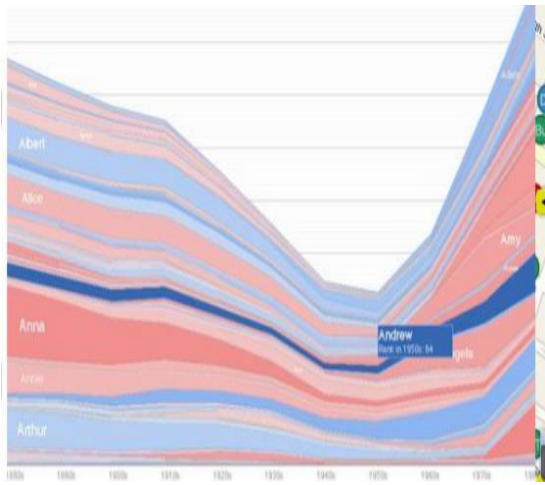
- Learn Seaborn Introduction — Interactive Lesson
- Learn Seaborn: Distributions — Interactive Lesson
- Introduction To Seaborn — Multiple Choice Quiz
- Seaborn Styling, Part 1: Figure Style and Scale — Article
- Seaborn Styling, Part 2: Color — Article
- Visualizing World Cup Data With Seaborn — Freeform Project

Data visualization

Sometimes data does not make sense until you can look at in a visual form, such as with charts and plots.

- Data visualization is all about understanding data by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed.
- Data visualization is visual representation of data for exploration, discovery, and insight of data.
- Interactive component provides more insight as compared to a static image.





Types of Data Visualization

- **Scientific visualization, information visualization, and visual analytics are the three main branches of visualization.**
- Scientific Visualization
  - Structural Data – Seismic, Medical, ..
- Information Visualization
  - No inherent structure – News, stock market, top grossing movies, Facebook connections
- Visual Analytics
  - Use visualization to understand and synthesize large amounts of multimodal data
  - audio, video, text, images, networks of people ..

Line Graphs in Matplotlib

- Matplotlib is a Python library used to create charts and graphs

- The concepts you will learn include:
  - Creating a line graph from data
  - Changing the appearance of the line
  - Zooming in on different parts of the axis
  - Putting labels on titles and axes
  - Creating a more complex figure layout
  - Adding legends to graphs
  - Changing tick labels and positions

- Before we start working with Matplotlib, we need to import it into our Python environment

```
from matplotlib import pyplot as plt
```

Line Graphs in Matplotlib

Basic Line Plot:

Line graphs are helpful for visualizing how a variable changes over time.

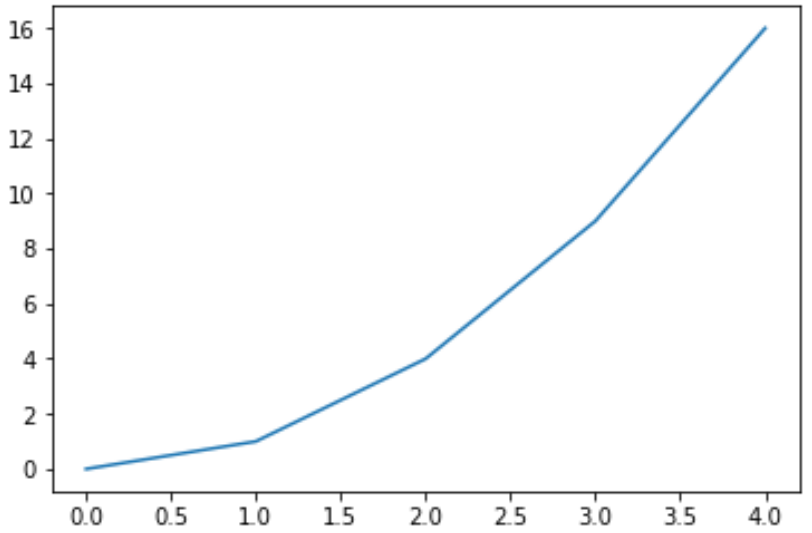
Some possible data that would be displayed with a line graph:

- average prices of gasoline over the past decade
- weight of an individual over the past couple of months
- average temperature along a line of longitude over different latitudes

Using Matplotlib methods, the following code will create a simple line graph using .plot() and display it using .show():

```
x_values = [0, 1, 2, 3, 4]
y_values = [0, 1, 4, 9, 16]
plt.plot(x_values, y_values)
plt.show()
```

plt.plot(x_values, y_values) will create the line graph



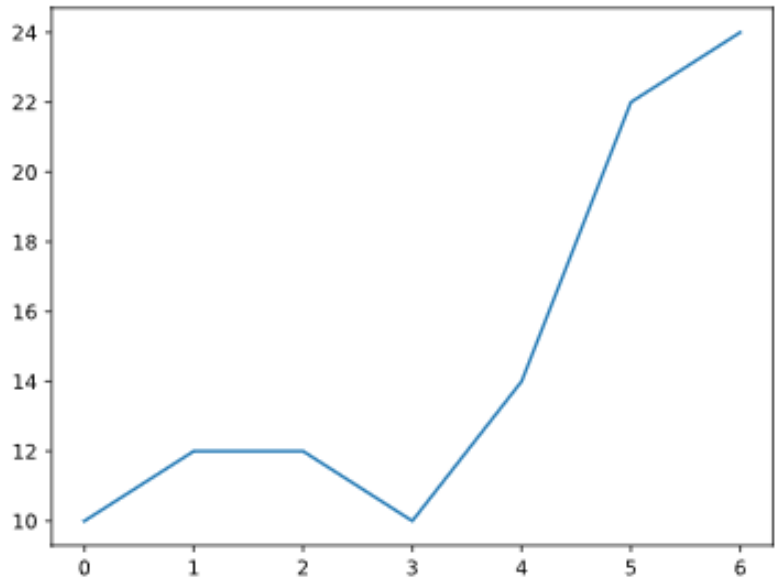
Line Graphs in Matplotlib

Example: We are going to make a simple graph representing someone's spending on lunch over the past week.

First, define two lists, days and money_spent, that contain the following integers:

Days	Money Spent
0	10
1	12
2	12
3	10
4	14
5	22
6	24

```
import codecademylib
from matplotlib import pyplot as plt

days = range(7)  # days = [0, 1, 2, 3, 4,5, 6]

money_spent = [10, 12, 12, 10, 14, 22, 24]

plt.plot(days, money_spent)

plt.show()
```



Plot days on the x-axis and money_spent on the y-axis using plt.plot().

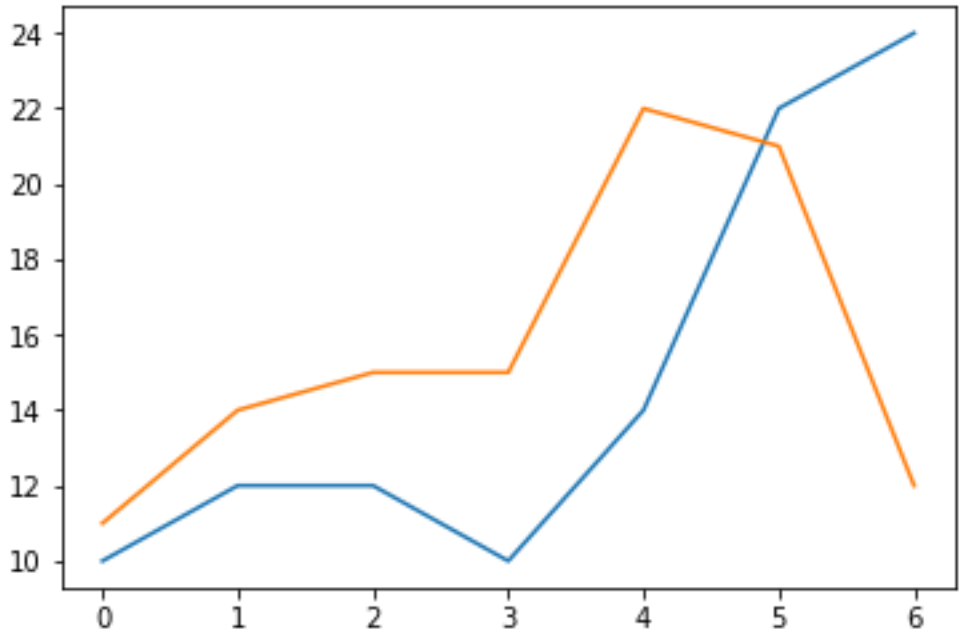
Multiple Line Graphs in Matplotlib

We can also have multiple line plots displayed on the same set of axes.

This can be very useful if we want to compare two datasets with the same scale and axis categories

Matplotlib will automatically place the two lines on the same axes and give them different colors if you call plt.plot() twice

```
# Days of the week:
days = [0, 1, 2, 3, 4, 5, 6]
# Your Money:
money_spent = [10, 12, 12, 10, 14, 22, 24]
# Your Friend's Money:
money_spent_2 = [11, 14, 15, 15, 22, 21, 12]
# Plot your money:
plt.plot(days, money_spent)
# Plot your friend's money:
plt.plot(days, money_spent_2)
# Display the result:
plt.show()
```



Multiple Line Graphs in Matplotlib

Lab Task 1:

We have defined lists called time, revenue, and costs.

```
time = [0, 1, 2, 3, 4]
revenue = [200, 400, 650, 800, 850]
costs = [150, 500, 550, 550, 560]
```

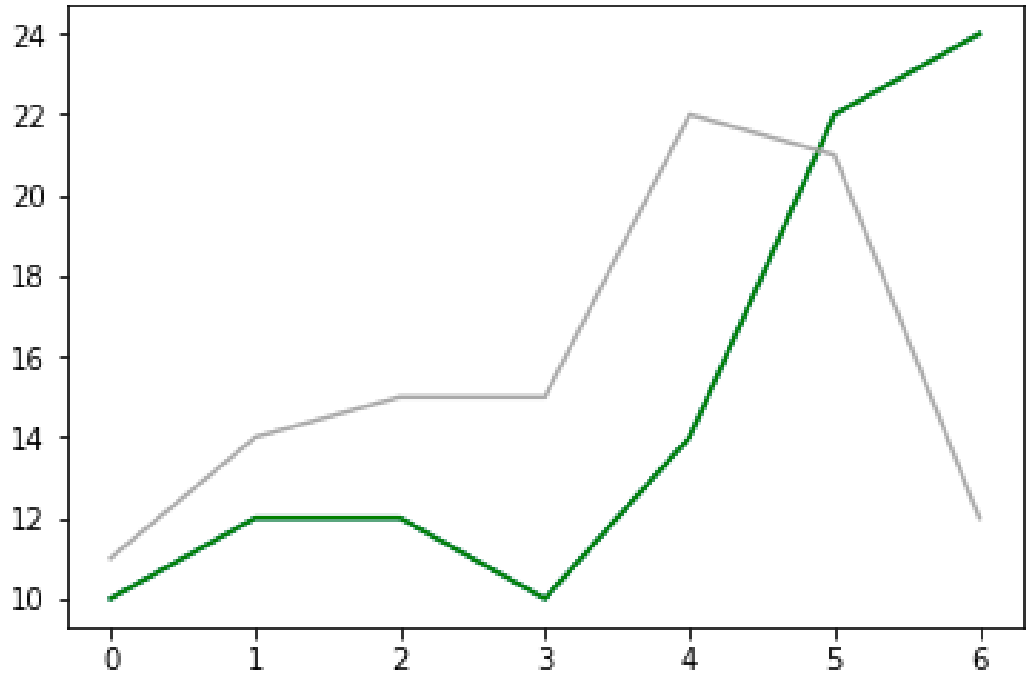
(a) Plot revenue vs time.

(b) Plot costs vs time on the same plot as the last line.

Line Graphs in Matplotlib: **Linestyles**

We can specify a different color for a line by using the keyword color with either an HTML color name or a HEX code

```
days = [0, 1, 2, 3, 4, 5, 6]

money_spent = [10, 12, 12, 10, 14, 22, 24]

money_spent_2 = [11, 14, 15, 15, 22, 21, 12]

plt.plot(days, money_spent, color='green')

plt.plot(days, money_spent_2, color='#AAAAAA')
```



Line Graphs in Matplotlib: Linestyles

We can also make a line dotted or dashed using the keyword linestyle.

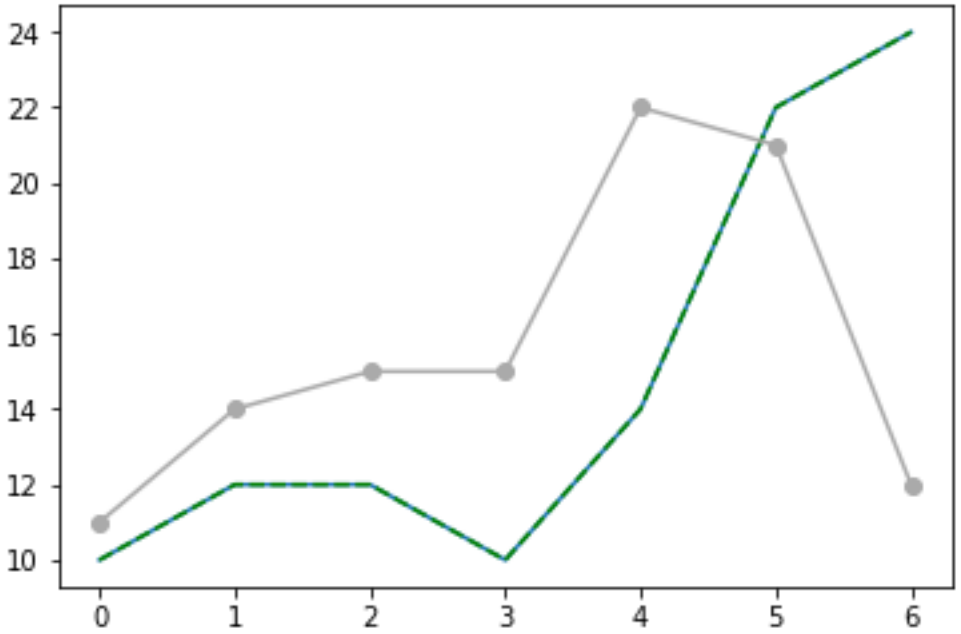
```
# Dashed:
plt.plot(x_values, y_values, linestyle='--')
# Dotted:
plt.plot(x_values, y_values, linestyle=':')
# No line:
plt.plot(x_values, y_values, linestyle='')
```

We can also add a marker using the keyword marker:

```
# A circle:
plt.plot(x_values, y_values, marker='o')
# A square:
plt.plot(x_values, y_values, marker='s')
# A star:
plt.plot(x_values, y_values, marker='*')
```

Line Graphs in Matplotlib: **Linestyles**

```
days = [0, 1, 2, 3, 4, 5, 6]
money_spent = [10, 12, 12, 10, 14, 22, 24]
money_spent_2 = [11, 14, 15, 15, 22, 21, 12]
plt.plot(days, money_spent, color='green', linestyle='--')
plt.plot(days, money_spent_2, color='#AAAAAA',  marker='o')
```



Line Graphs in Matplotlib: Linestyles

Lab Task 2:

a) Plot revenue vs. time as a purple ('purple'), dashed ('--') line.

b) Plot costs vs. time as a line with the HEX color #82edc9 and square ('s') markers.

Line Graphs in Matplotlib: Axis and Labels

Sometimes, it can be helpful to zoom in or out of the plot, especially if there is some detail we want to address. To zoom, we can use plt.axis().

We use plt.axis() by feeding it a list as input. This list should contain:

The minimum x-value displayed

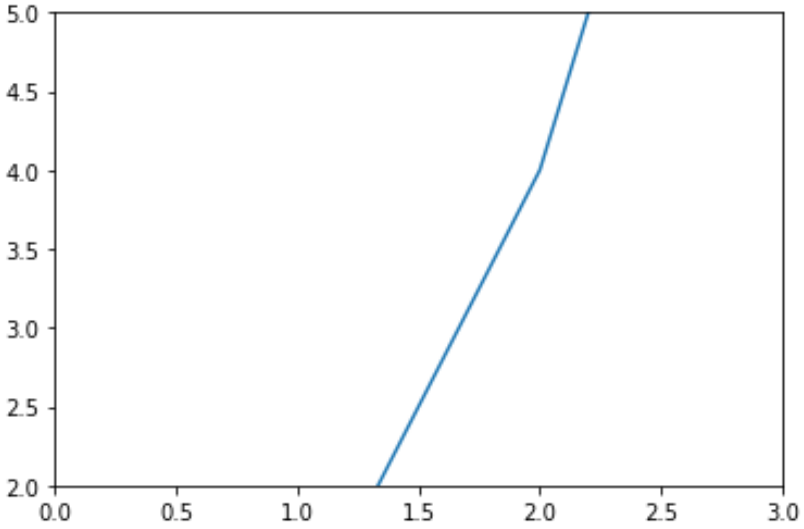
The maximum x-value displayed

The minimum y-value displayed

The maximum y-value displayed

For example, if we want to display a plot from x=0 to x=3 and from y=2 to y=5, we would call plt.axis([0, 3, 2, 5]).

```
x = [0, 1, 2, 3, 4]
y = [0, 1, 4, 9, 16]
plt.plot(x, y)
plt.axis([0, 3, 2, 5])
plt.show()
```



Line Graphs in Matplotlib

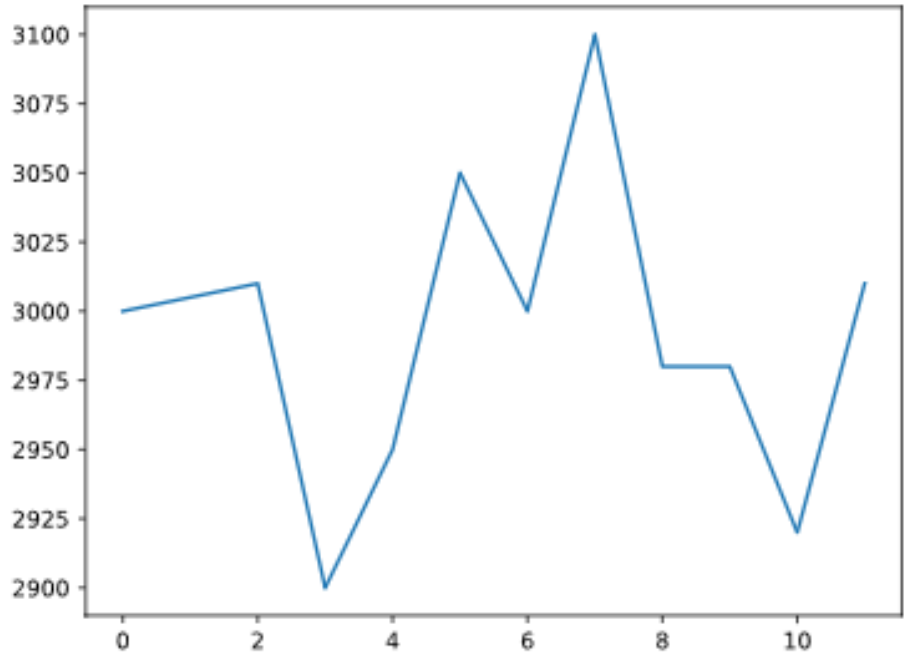
Lab Task 3:

We have plotted a line representing someone's spending on coffee over the past 12 years.

Let's modify the axes to zoom in a bit more on our line chart.

Use plt.axis() to modify the axes so that the x-axis goes from 0 to 12, and the y-axis goes from 2900 to 3100.

```
from matplotlib import pyplot as plt
x = range(12)
y = [3000, 3005, 3010, 2900, 2950, 3050, 3000, 3100, 2980,
2980, 2920, 3010]
plt.plot(x, y)
plt.show()
```



Line Graphs in Matplotlib: Labeling the Axes

Adding labels to the x-axis and y-axis, and giving the plot a title:

- We can label the x- and y- axes by using plt.xlabel() and plt.ylabel()
- The plot title can be set by using plt.title()

```
hours = [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
happiness = [9.8, 9.9, 9.2, 8.6, 8.3, 9.0, 8.7, 9.1, 7.0, 6.4, 6.9, 7.5]
plt.plot(hours, happiness)
plt.xlabel('Time of day')
plt.ylabel('Happiness Rating (out of 10)')
plt.title('My Self-Reported Happiness While Awake')
plt.show()
```

Line Graphs in Matplotlib: Labeling the Axes

Lab Task 4:

Consider the following code:

```
from matplotlib import pyplot as plt


x = range(12)
y = [3000, 3005, 3010, 2900, 2950, 3050, 3000, 3100, 2980, 2980, 2920, 3010]
```

a) Label the x-axis 'Time'.

b) Label the y-axis 'Dollars spent on coffee'.

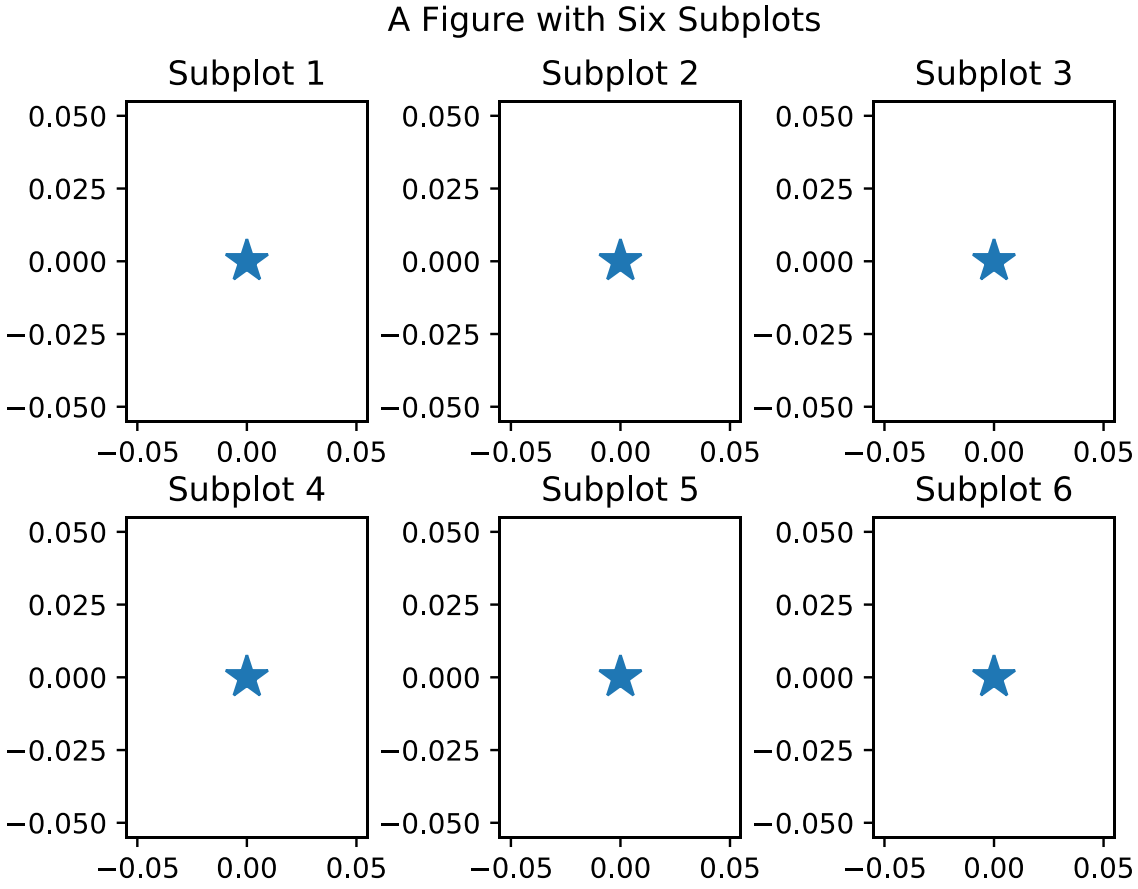
c) Add the title 'My Last Twelve Years of Coffee Drinking'.

Line Graphs in Matplotlib: Subplots

Sometimes, we want to display two lines side-by-side, rather than in the same set of x- and y-axes.

When we have multiple axes in the same picture, we call each set of axes a subplot.

The picture or object that contains all of the subplots is called a figure.



This figure has six subplots split into 2 rows and 3 columns

We can create subplots using **.subplot()**

Line Graphs in Matplotlib: Subplots

The command plt.subplot() needs three arguments to be passed into it:

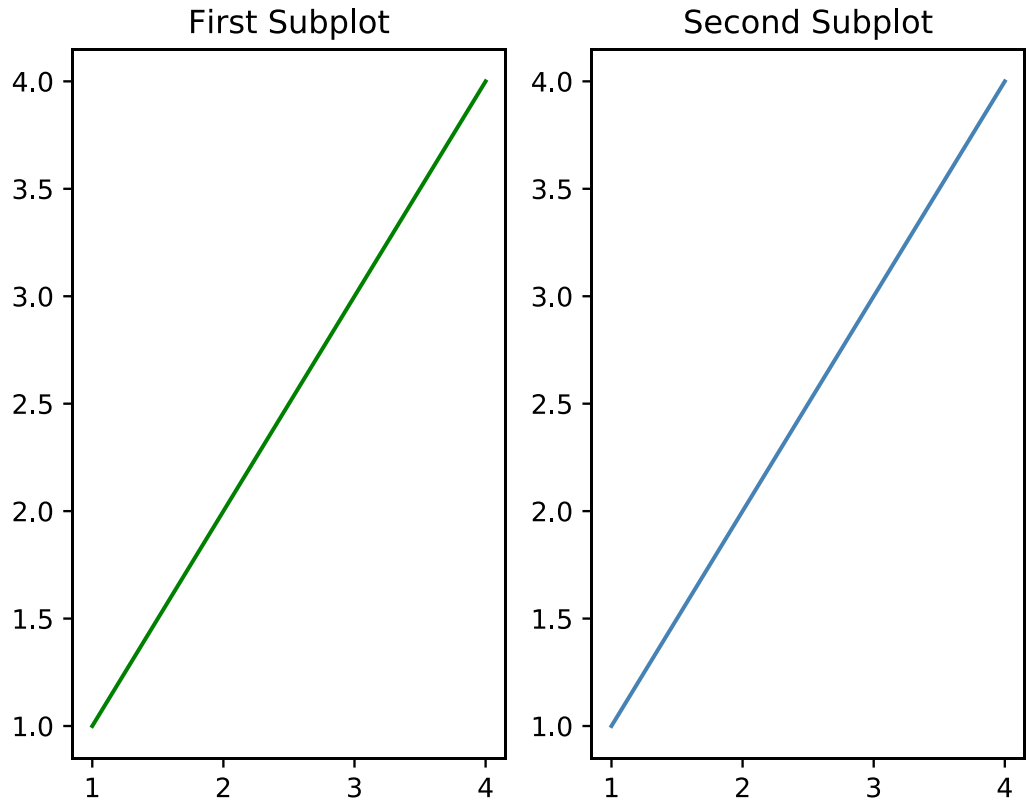
- The number of rows of subplots
- The number of columns of subplots
- The index of the subplot we want to create

For instance, the command plt.subplot(2, 3, 2) would create "Subplot 2" from the figure above.

Any plt.plot() that comes after plt.subplot() will create a line plot in the specified subplot.

Line Graphs in Matplotlib : Subplots

```
# Data sets
x = [1, 2, 3, 4]
y = [1, 2, 3, 4]

# First Subplot
plt.subplot(1, 2, 1)
plt.plot(x, y, color='green')
plt.title('First Subplot')

# Second Subplot
plt.subplot(1, 2, 2)
plt.plot(x, y, color='steelblue')
plt.title('Second Subplot')

# Display both subplots
plt.show()
```



Line Graphs in Matplotlib: Subplots

Lab Task 5:

We have defined the lists months, temperature, and flights_to_hawaii for you.

a) Using the plt.subplot command, plot temperature vs months in the left box of a figure that has 1 row with 2 columns.

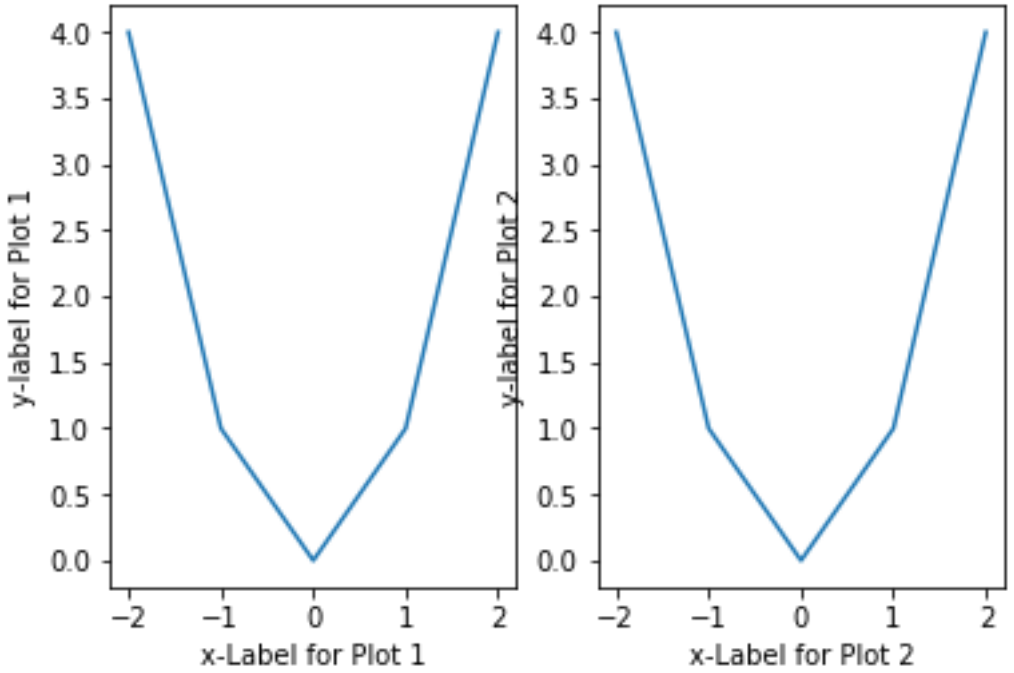
```
from matplotlib import pyplot as plt


months = range(12)
temperature = [36, 36, 39, 52, 61, 72, 77, 75, 68, 57, 48, 48]
flights_to_hawaii = [1200, 1300, 1100, 1450, 850, 750, 400, 450, 400, 860, 990, 1000]
```

b) Plot flights_to_hawaii vs temperature in the same figure, to the right of your first plot.

Add the parameter "o" to the end of your call to plt.plot to make the plot into a scatterplot, if you want!

Line Graphs in Matplotlib: Subplots

Sometimes, when we're putting multiple subplots together, some elements can overlap and make the figure unreadable:



- We can customize the spacing between our subplots to make sure that the figure we create is visible and easy to understand
- To do this, we use the plt.subplots_adjust() command

Line Graphs in Matplotlib: Subplots

.subplots_adjust() has some keyword arguments that can move your plots within the figure:

- left — the left-side margin, with a default of 0.125. You can increase this number to make room for a y-axis label
- right — the right-side margin, with a default of 0.9. You can increase this to make more room for the figure, or decrease it to make room for a legend
- bottom — the bottom margin, with a default of 0.1. You can increase this to make room for tick mark labels or an x-axis label
- top — the top margin, with a default of 0.9
- wspace — the horizontal space between adjacent subplots, with a default of 0.2
- hspace — the vertical space between adjacent subplots, with a default of 0.2

Line Graphs in Matplotlib: Subplots

For example:

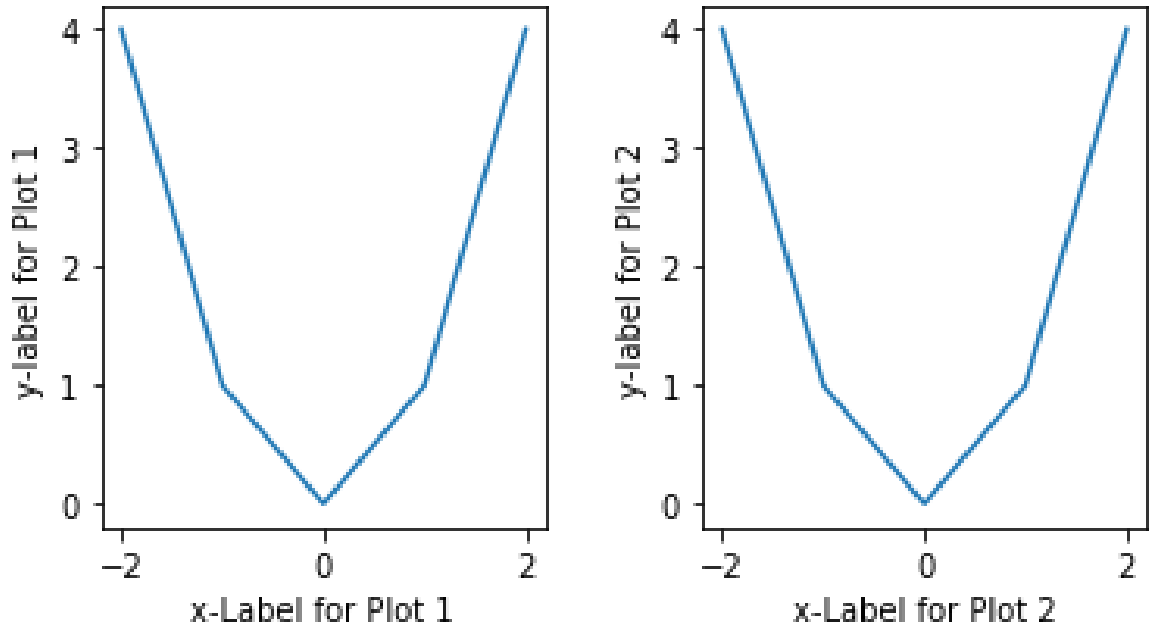
if we were adding space to the bottom of a graph by changing the bottom margin to 0.2 (instead of the default of 0.1), we would use the command:

```
plt.subplots_adjust(bottom=0.2)
```

To adjust both the top and the hspace:

```
plt.subplots_adjust(top=0.95, hspace=0.25)
```

```
# Left Plot
plt.subplot(1, 2, 1)
plt.plot([-2, -1, 0, 1, 2], [4, 1, 0, 1, 4])

# Right Plot
plt.subplot(1, 2, 2)
plt.plot([-2, -1, 0, 1, 2], [4, 1, 0, 1, 4])

# Subplot Adjust
plt.subplots_adjust(wspace=0.35)
plt.show()
```

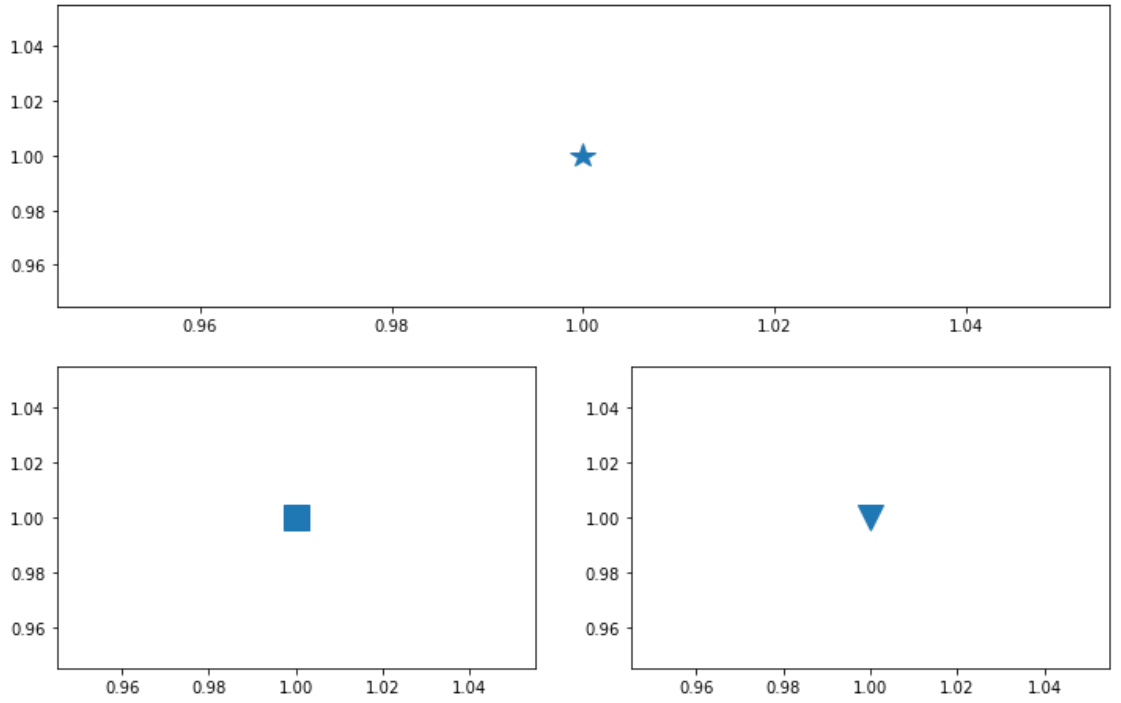


Line Graphs in Matplotlib: Subplots

Lab Task 6:

(a) Create a figure that has two rows of subplots. It should have:

- one subplot in the top row
- two subplots in the bottom row



(b) Plot straight_line vs x in this subplot you've selected.

Now, use the plt.subplot() command to select the box in the first column of the second row (the one with a square in it). Plot parabola vs x in this box.

(c) Now, use the plt.subplot() command to select the box in the second column of the second row (the one with a triangle in it). Plot cubic vs x in this box.

(d) Increase the spacing between horizontal subplots to 0.35 and the bottom margin to 0.2.

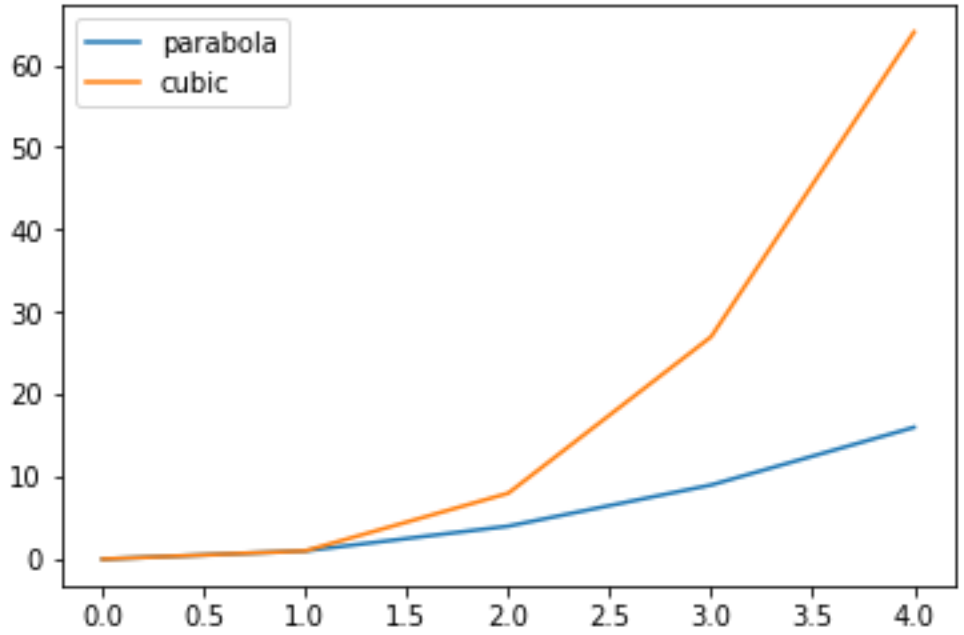
Line Graphs in Matplotlib: Legends

When we have multiple lines on a single graph we can label them by using the command plt.legend().

The legend method takes a list with the labels to display.

```
plt.plot([0, 1, 2, 3, 4], [0, 1, 4, 9, 16])
plt.plot([0, 1, 2, 3, 4], [0, 1, 8, 27, 64])
plt.legend(['parabola', 'cubic'])
plt.show()
```

which would display a legend on our graph, labeling each line:

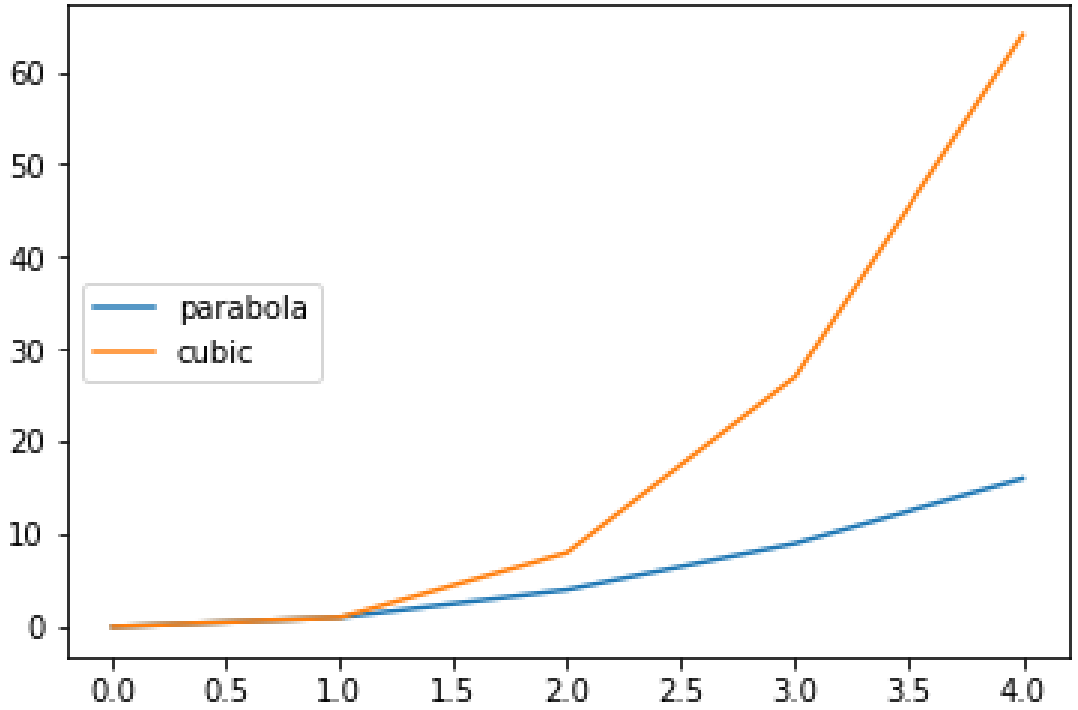


Line Graphs in Matplotlib: Legends

plt.legend() can also take a keyword argument loc, which will position the legend on the figure.

For, example, we can call plt.legend() and set loc to 6, which would move the legend to the left side of the graph:

```
plt.legend(['parabola', 'cubic'], loc=6)
plt.show()
```



These are the position values loc accepts:

Number Code	String
0	best
1	upper right
2	upper left
3	lower left
4	lower right
5	right
6	center left
7	center right
8	lower center
9	upper center
10	center

Note: If you decide not to set a value for loc, it will default to choosing the “best” location.

Line Graphs in Matplotlib: Legends

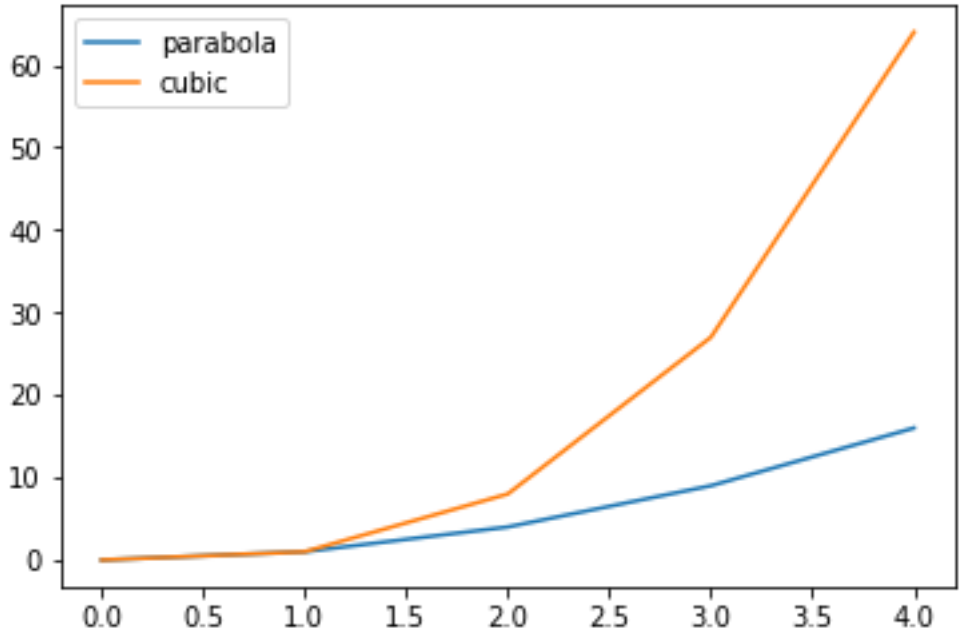
Sometimes, it's easier to label each line as we create it.

If we want, we can use the keyword label inside of plt.plot().

If we choose to do this, we don't pass any labels into plt.legend().

For example:

```
plt.plot([0, 1, 2, 3, 4], [0, 1, 4, 9, 16], label="parabola")
plt.plot([0, 1, 2, 3, 4], [0, 1, 8, 27, 64], label="cubic")
plt.legend() # Still need this command!
plt.show()
```



Line Graphs in Matplotlib: Legends

Lab Task 6:

Consider the three plotted lines. They represent the temperatures over the past year in Hyrule (hyrule), Kakariko (kakariko), and the Gerudo Valley (gerudo).

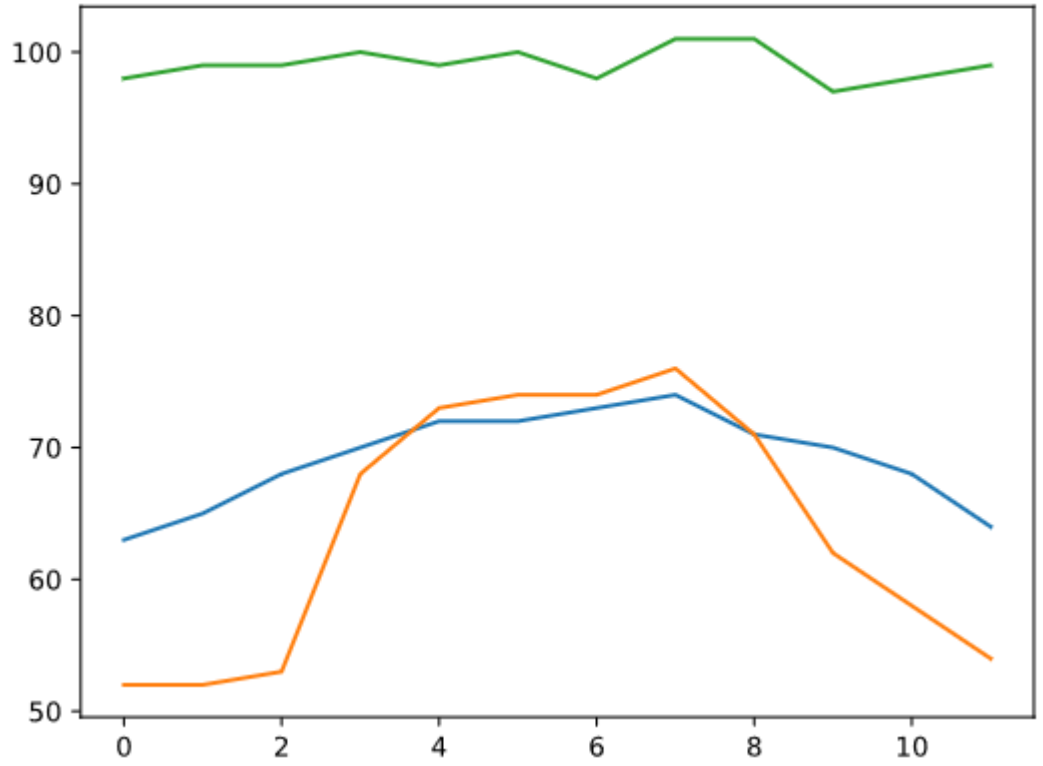
(a) Create a list of strings containing "Hyrule", "Kakariko", and "Gerudo Valley", and store it in a variable called legend_labels.

(b) Create a legend for the graph by feeding in legend_labels into plt.legend().

(c) Set the legend to be at the lower center of the chart.

Line Graphs in Matplotlib: Legends

```
from matplotlib import pyplot as plt

months = range(12)
hyrule = [63, 65, 68, 70, 72, 72, 73, 74, 71, 70, 68, 64]
kakariko = [52, 52, 53, 68, 73, 74, 74, 76, 71, 62, 58, 54]
gerudo = [98, 99, 99, 100, 99, 100, 98, 101, 101, 97, 98, 99]

plt.plot(months, hyrule)
plt.plot(months, kakariko)
plt.plot(months, gerudo)

#create your legend here

plt.show()
```



Line Graphs in Matplotlib: Modify Ticks

Why modify tick marks?

Because our plots can have multiple subplots, we have to specify which one we want to modify.

In order to do that, we call plt.subplot() as:

```
ax = plt.subplot(1, 1, 1)
```

ax is an axes object, and it lets us modify the axes belonging to a specific subplot.

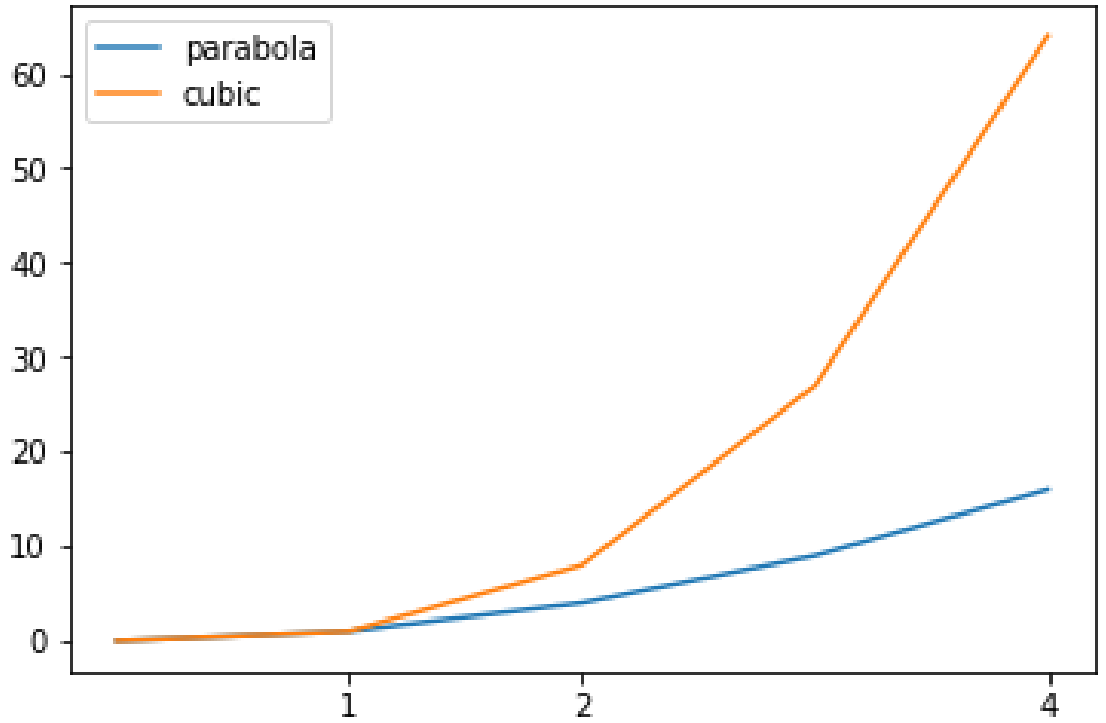
Even if we only have one subplot, when we want to modify the ticks, we will need to start by calling either ax = plt.subplot(1, 1, 1) or ax = plt.subplot() in order to get our axes object.

Line Graphs in Matplotlib

Suppose we wanted to set our x-ticks to be at 1, 2, and 4. We would use the following code:

```
ax = plt.subplot()
plt.plot([0, 1, 2, 3, 4], [0, 1, 4, 9, 16])
plt.plot([0, 1, 2, 3, 4], [0, 1, 8, 27, 64])
ax.set_xticks([1, 2, 4])
```

We can also modify the y-ticks by using ax.set_yticks().



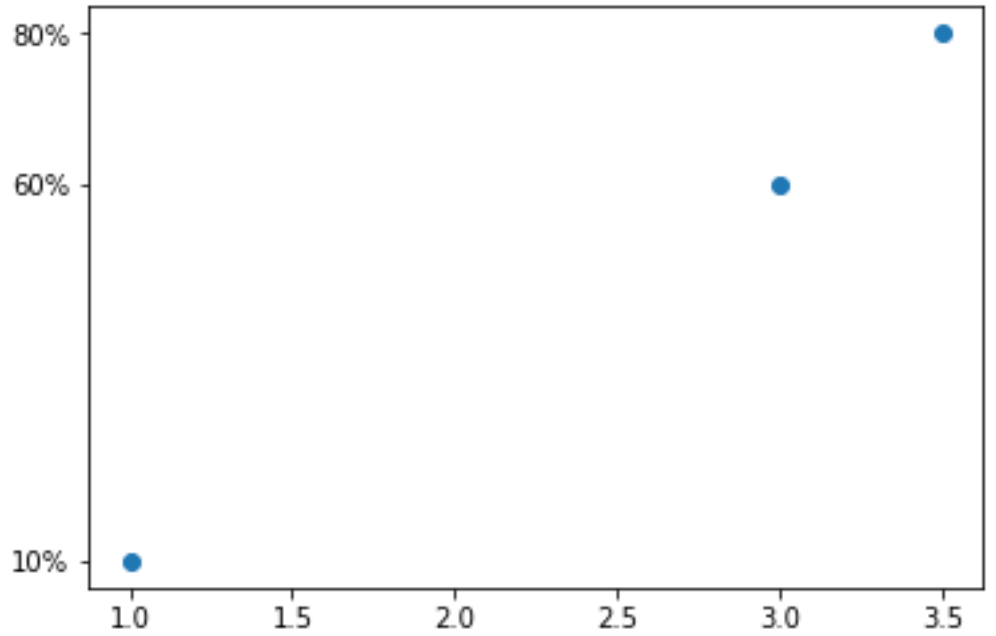
Line Graphs in Matplotlib

If we want special labels (such as strings), we can use the command ax.set_xticklabels() or ax.set_yticklabels().

For example, we might want to have a y-axis with ticks at 0.1, 0.6, and 0.8, but label them 10%, 60%, and 80%, respectively.

To do this, we use the following commands:

```
ax = plt.subplot()

plt.plot([1, 3, 3.5], [0.1, 0.6, 0.8], 'o')

ax.set_yticks([0.1, 0.6, 0.8])

ax.set_yticklabels(['10%', '60%', '80%'])
```



Lab Task 7:

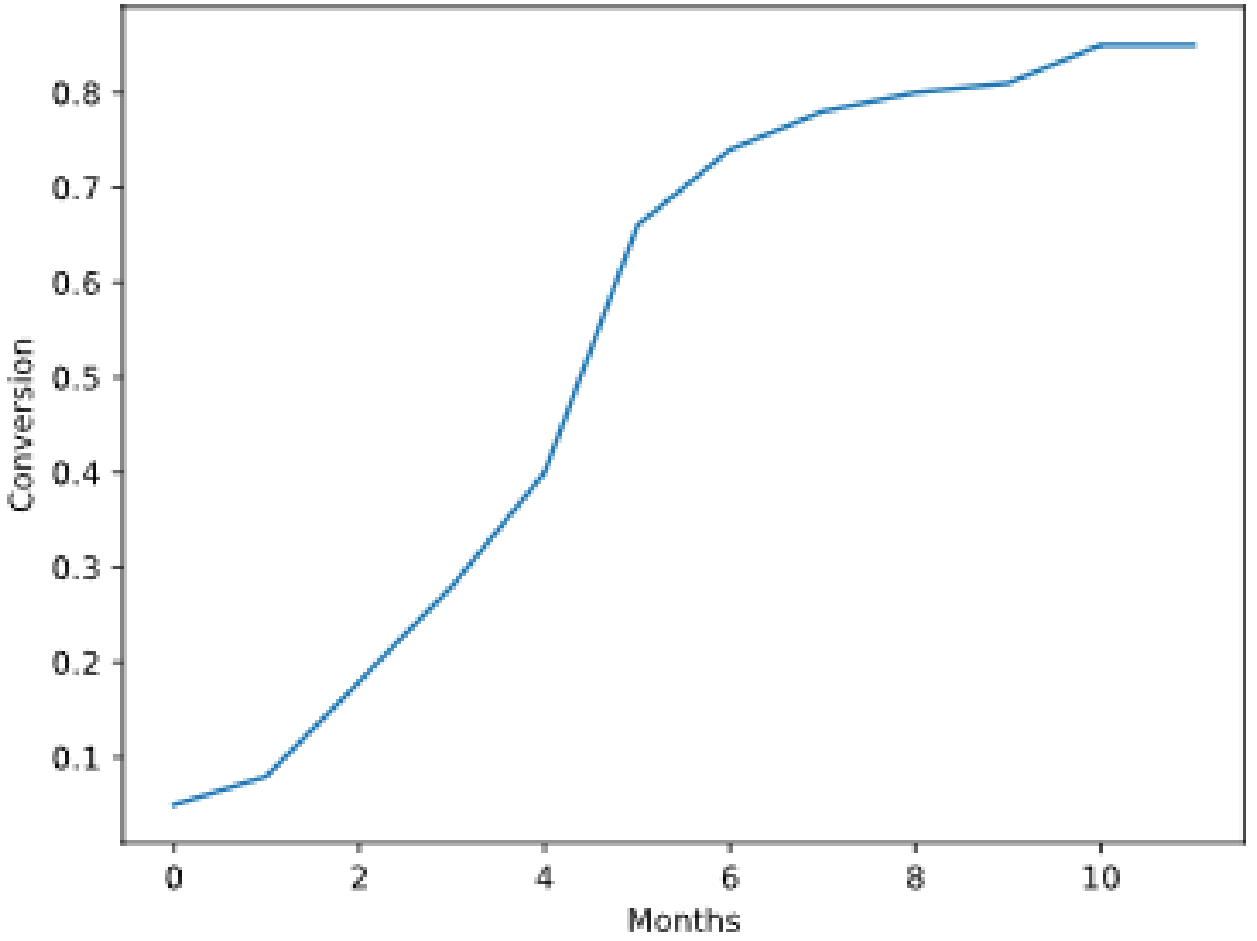
- Let's imagine we are working for a company called Dinnersaur, that delivers dinners to people who don't want to cook. Dinnersaur recently started a new service that is subscription-based, where users sign up for a whole year of meals, instead of buying meals on-demand.

- We have plotted a line for you in the editor representing the proportion of users who have switched over to this new service since it was rolled out in January.

    - First, save the set of axes in a variable called ax. We will use ax to set the x- and y-ticks and labels to make this graph easier to read.
    - Using ax, set the x-ticks to be the months list.
    - Set the x-tick labels to be the month_names list.
    - Set the y-ticks to be [0.10, 0.25, 0.5, 0.75].
    - Label the y-ticks to be the percentages that correspond to the values [0.10, 0.25, 0.5, 0.75], instead of decimals.

Lab Task 7:

```
from matplotlib import pyplot as plt

month_names = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul
"Aug", "Sep","Oct", "Nov", "Dec"]

months = range(12)
conversion = [0.05, 0.08, 0.18, 0.28, 0.4, 0.66, 0.74, 0.78, 0.8
0.81, 0.85, 0.85]

plt.xlabel("Months")
plt.ylabel("Conversion")

plt.plot(months, conversion)
```



Line Graphs in Matplotlib: Figures

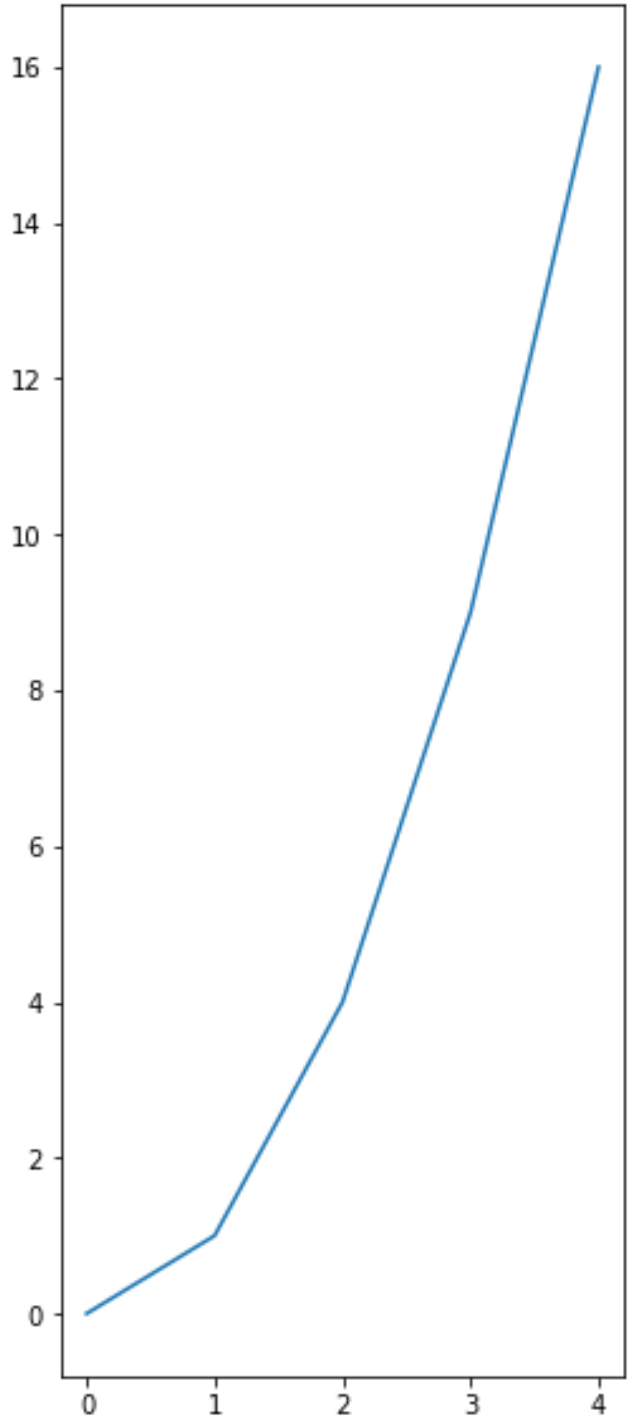
To create a figure with a width of 4 inches, and height of 10 inches, we would use:

```
plt.figure(figsize=(4, 10))
```

To save it, we can use the command plt.savefig() to save out to many different file formats, such as png, svg, or pdf.

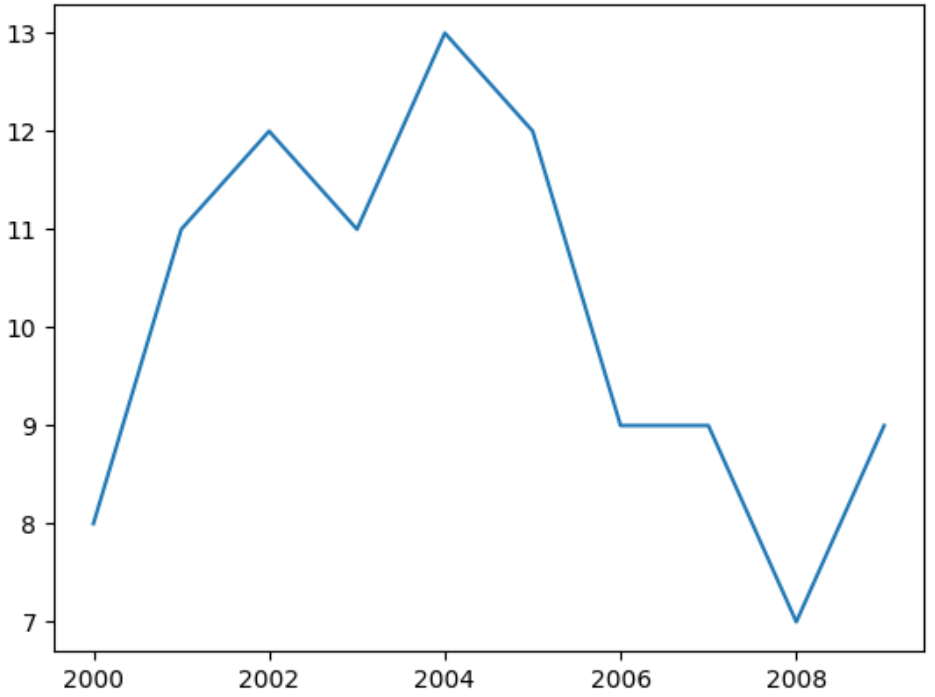
After plotting, we can call plt.savefig('name_of_graph.png')

```
# Figure 2

plt.figure(figsize=(4, 10))

plt.plot(x, parabola)

plt.savefig('tall_and_narrow.png')
```



Line Graphs in Matplotlib: Figures

```
from matplotlib import pyplot as plt

word_length = [8, 11, 12, 11, 13, 12, 9, 9, 7, 9]
power_generated = [753.9, 768.8, 780.1, 763.7, 788.5, 782, 787.2, 806.4,
806.2, 798.9]
years = [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009]

plt.close('all')
plt.figure()
plt.plot(years, word_length)
plt.savefig('winning_word_lengths.png')
plt.figure(figsize=(7, 3))
plt.plot(years, power_generated)
plt.savefig('power_generated.png')
```



Final Exercise on Line Graphs in Matplotlib

- Define three lists, x, y1, and y2 and fill them with integers. These numbers can be anything you want, but it would be neat to have them be actual metrics that you want to compare.
- Plot y1 vs x and display the plot.
- On the same graph, plot y2 vs x (after the line where you plot y1 vs x)
- Make the y1 line a pink line and the y2 line a gray line. Give both lines round markers.
- Give your graph a title of “Two Lines on One Graph”, and label the x-axis ”Amazing X-axis” and y-axis ”Incredible Y-axis”.
- Give the graph a legend and put it in the lower right.

Solution:

```
from matplotlib import pyplot as plt

x = range(6)
y1 = [1, 2, 3, 4, 5, 6]
y2 = [-1, 1, 3, 4, 4, 4]



plt.show()
```

Simple Bar Chart

- The plt.bar function allows you to create simple bar charts to compare multiple categories of data.

- Some possible data that would be displayed with a bar chart:

    - x-axis — famous buildings, y-axis — heights
    - x-axis — different planets, y-axis — number of days in the year
    - x-axis — programming languages, y-axis — lines of code written by you

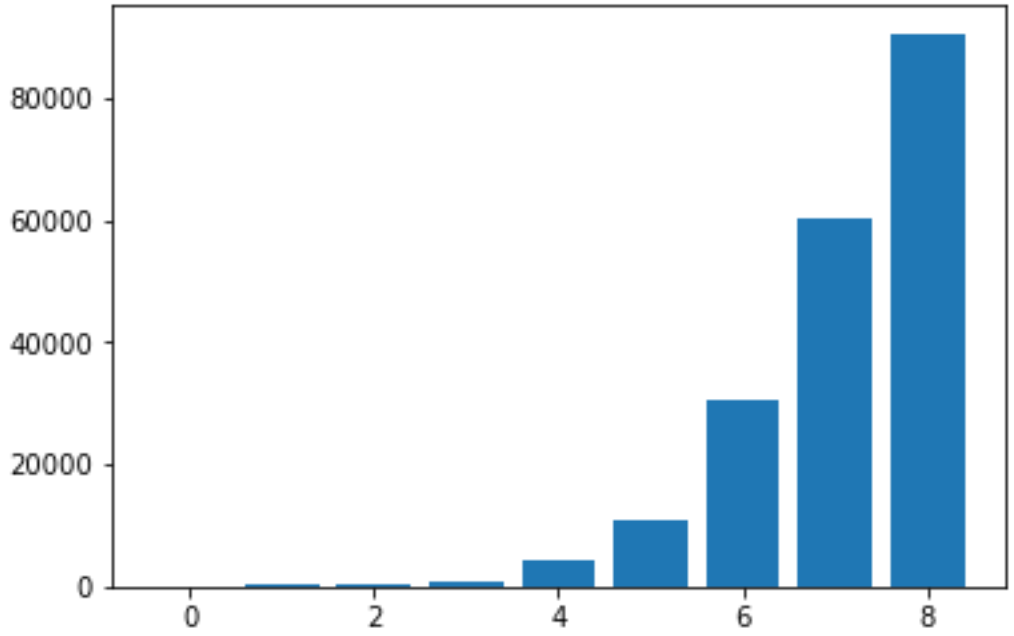
- We call plt.bar with two arguments:

```
heights = [88, 225, 365, 687, 4333, 10756, 30687, 60190, 90553]
x_values = range(len(heights))
```

- the x-values — a list of x-positions for each bar
- the y-values — a list of heights for each bar

Simple Bar Chart

```
days_in_year = [88, 225, 365, 687, 4333, 10756, 30687,
60190, 90553]
plt.bar(range(len(days_in_year)), days_in_year)
plt.show()
```



Exercise 1

- We are going to help the cafe MatplotSip analyze some of the sales data they have been collecting. In script.py, we have included a list of drink categories and a list of numbers representing the sales of each drink over the past month.

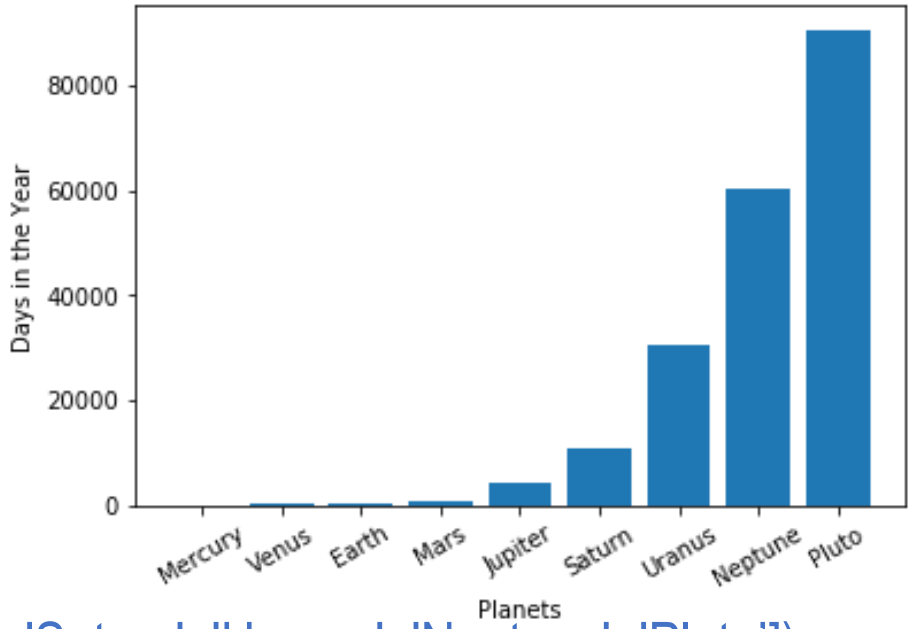
- Use plt.bar to plot numbers of drinks sold on the y-axis. The x-values of the graph should just be the list [0, 1 ... , n-1], where n is the number of categories (drinks) we are plotting. So at x=0, we'll have the number of cappuccinos sold.

- Show the plot and examine it. At this point, we can't tell which bar corresponds to which drink, so this chart is not very helpful. We'll fix this in the next section.

```
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha", "espresso"]
sales =  [91, 76, 56, 66, 52, 27]
```

Simple Bar Chart II

- In the drinks chart from the last exercise, we could see that sales were different for different drink items, but this wasn't very helpful to us, since we didn't know which bar corresponded to which drink.



To customize the tick marks on the x-axis:

(a) Create an axes object

```
ax = plt.subplot()
```

(b) Set the x-tick positions using a list of numbers

```
ax.set_xticks([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

(c) Set the x-tick labels using a list of strings

```
ax.set_xticklabels(['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune', 'Pluto'])
```

(d) If your labels are particularly long, you can use the rotation keyword to rotate your labels by a specified number of degrees:

```
ax.set_xticklabels(['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune', 'Pluto'],
rotation=30)
```

Exercise:

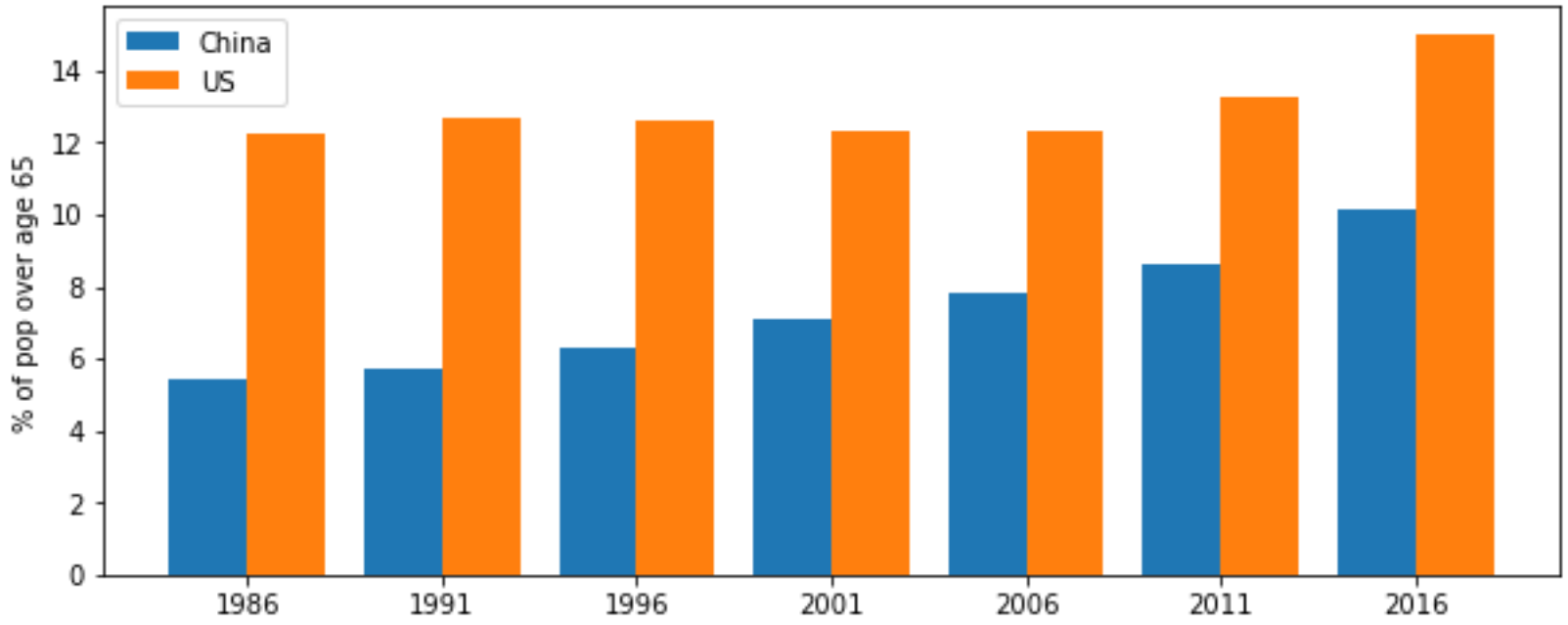
- The list drinks represents the drinks sold at MatplotSip. We are going to set x-tick labels on the chart you made with plt.bar in the last exercise.

- First, create the axes object for the plot and store it in a variable called ax.
- Set the x-axis ticks to be the numbers from 0 to the length of drinks.
- Use the strings in the drinks list for the x-axis ticks of the plot you made with plt.bar.

```
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha", "espresso"]
sales =  [91, 76, 56, 66, 52, 27]

plt.bar(range(len(drinks)), sales)

#create your ax object here
plt.show()
```

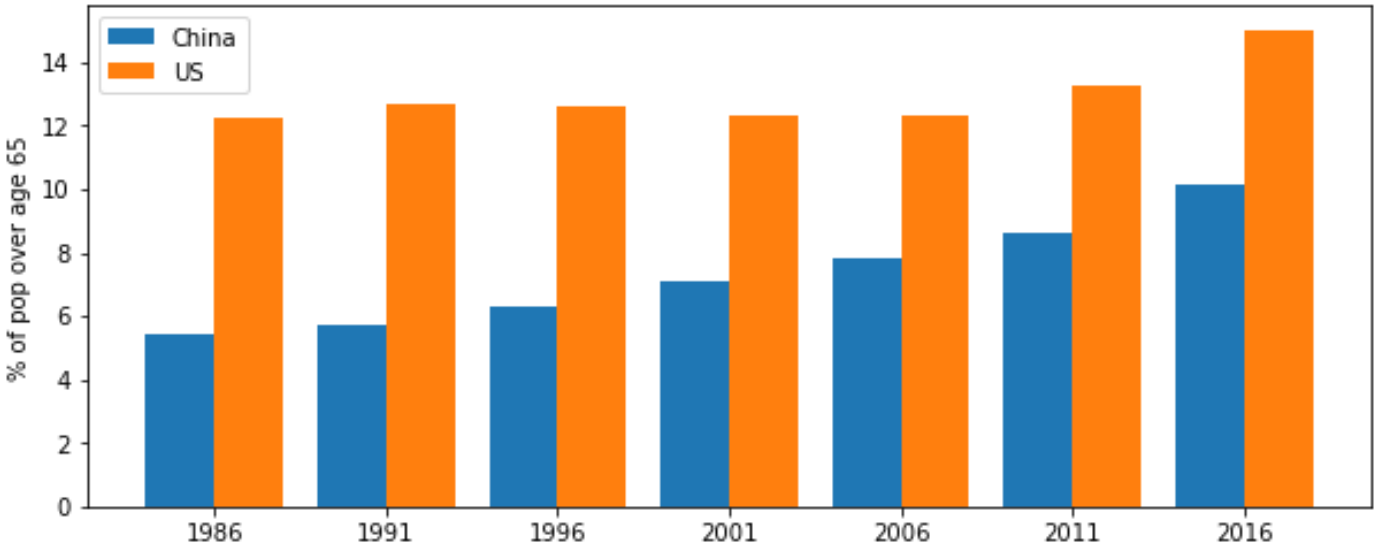
Side-By-Side Bars

- We can use a bar chart to compare two sets of data with the same types of axis values.
- To do this, we plot two sets of bars next to each other, so that the values of each category can be compared.
- For example, here is a chart with side-by-side bars for the populations of the United States and China over the age of 65 (in percentages):



```
# China Data (blue bars)
n = 1  # This is our first dataset (out of 2)
t = 2 # Number of datasets
d = 7 # Number of sets of bars
w = 0.8 # Width of each bar
x_values1 = [t*element + w*n for element in range(d)]
```

```
# US Data (orange bars)
n = 2  # This is our second dataset (out of 2)
t = 2 # Number of datasets
d = 7 # Number of sets of bars
w = 0.8 # Width of each bar
x_values2 = [t*element + w*n for element in range(d)]
```



Exercise

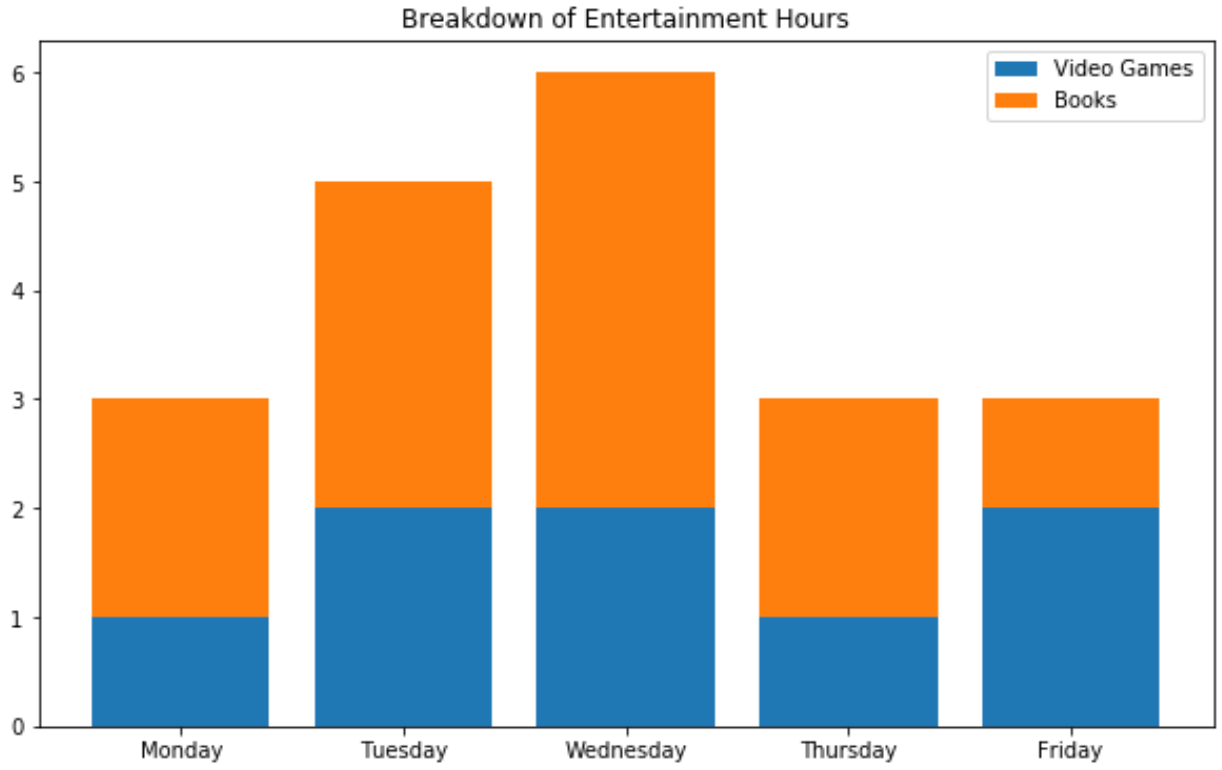
- The second location of MatplotSip recently opened up, and the owners want to compare the drink choices of the clientele at the two different locations.
- To do this, it will be helpful to have the sales of each drink plotted on the same axes. We have provided sales2, a list of values representing the sales of the same drinks at the second MatplotSip location.
- Use the plt.bar to position the bars corresponding to sales1 on the plot. The x-values for plt.bar should be the store1_x list that you just created.

```
from matplotlib import pyplot as plt


drinks = ["cappuccino", "latte", "chai", "americano", "mocha",

"espresso"]

sales1 = [91, 76, 56, 66, 52, 27]

sales2 = [65, 82, 36, 68, 38, 40]
```

Stacked Bars

- If we want to compare two sets of data while preserving knowledge of the total between them, we can also stack the bars instead of putting them side by side.
- For instance, if someone was plotting the hours they've spent on entertaining themselves with video games and books in the past week and the second set of bars has bottom specified:

```
video_game_hours = [1, 2, 2, 1, 2]

plt.bar(range(len(video_game_hours)),
video_game_hours)


book_hours = [2, 3, 4, 2, 1]

plt.bar(range(len(book_hours)), book_hours,
bottom=video_game_hours)
```



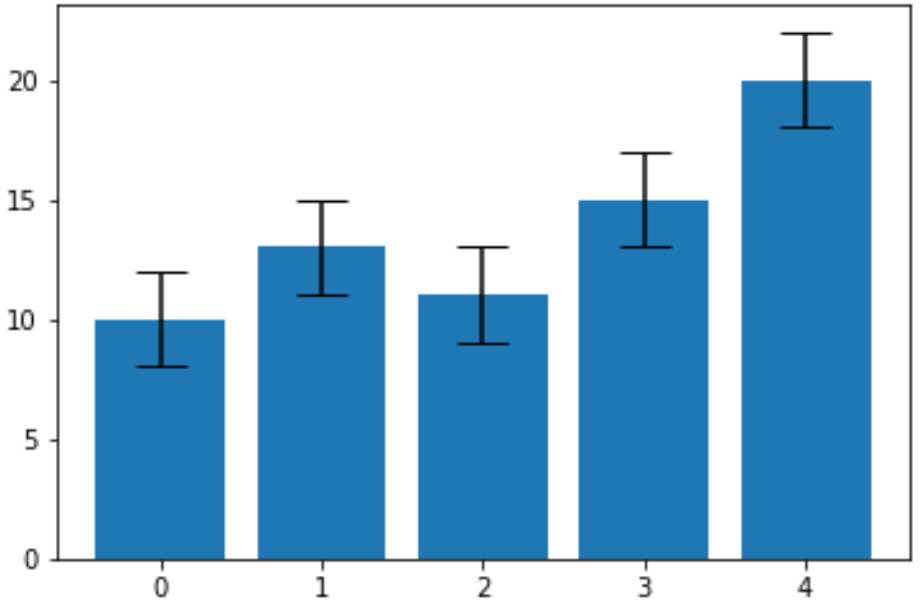
Exercise

- You just made a chart with two sets of sales data plotted side by side. Let's instead make a stacked bar chart by using the keyword bottom.
- Put the sales1 bars on the bottom and set the sales2 bars to start where the sales1 bars end.
- We should add a legend to make sure we know which set of bars corresponds to which location.
- Label the bottom set of bars as "Location 1" and the top set of bars as "Location 2" and add a legend to the chart.

```
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha", "espresso"]
sales1 =  [91, 76, 56, 66, 52, 27]
sales2 = [65, 82, 36, 68, 38, 40]

plt.show()
```

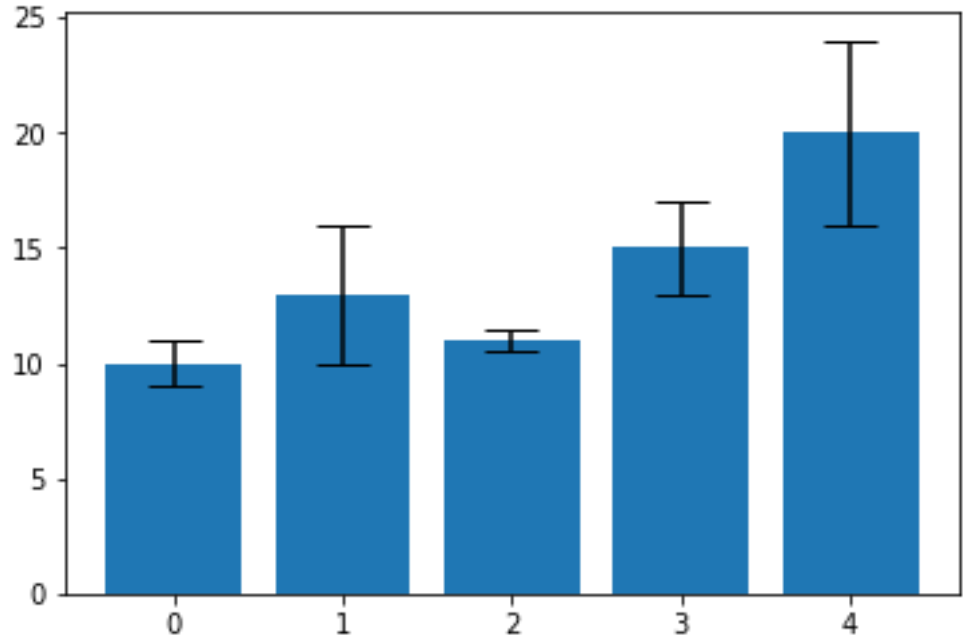
Error Bars

- Sometimes, we need to visually communicate some sort of uncertainty in the heights of those bars. Here are some examples:
    - The average number of students in a 3rd grade classroom is 30, but some classes have as few as 18 and others have as many as 35 students.
    - We measured that the weight of a certain fruit was 35g, but we know that our scale isn't very precise, so the true weight of the fruit might be as much as 40g or as little as 30g.
    - The average price of a soda is $1.00, but we also want to communicate that the standard deviation is $0.20.
- To display error visually in a bar chart, we often use error bars to show where each bar could be, taking errors into account.



- Each of the black lines is called an error bar.
- The taller the bar is, the more uncertain we are about the height of the blue bar.
  - If we wanted to show an error of +/- 2, we would add the keyword yerr=2 to our plt.bar command.
- The horizontal lines at the top and bottom are called caps. They make it easier to read the error bars.
  - To make the caps wide and easy to read, we would add the keyword capsize=10

```
values = [10, 13, 11, 15, 20]
yerr = 2
plt.bar(range(len(values)), values, yerr=yerr, capsize=10)
plt.show()
```



If we want a different amount of error for each bar, we can make yerr equal to a list rather than a single number:

```
values = [10, 13, 11, 15, 20]
yerr = [1, 3, 0.5, 2, 4]
plt.bar(range(len(values)), values, yerr=yerr, capsize=10)
plt.show()
```

Exercise

- For someone who is learning about the different drink types at MatplotSip, a bar chart of milk amounts in each drink may be useful. We have provided the ounces_of_milk list, which contains the amount of milk in each 12oz drink in the drinks list. Plot this information as a bar chart.
- According to different barista styles and measurement errors, there might be variation on how much milk actually goes into each drink. We've included a list error, with an error of 10% on each amount of milk. Display this error as error bars on the bar graph.
- Add caps of size 5 to your error bars.

```
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha", "espresso"]
ounces_of_milk = [6, 9, 4, 0, 9, 0]
error = [0.6, 0.9, 0.4, 0, 0.9, 0]

# Plot the bar graph here

plt.show()
```

Pie Chart

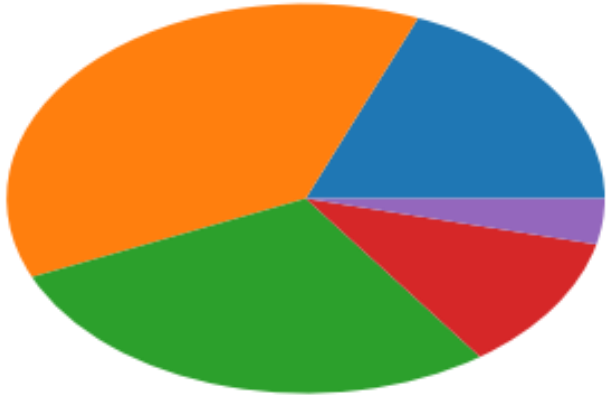
- If we want to display elements of a data set as proportions of a whole, we can use a pie chart.

- Pie charts are helpful for displaying data like:

    - Different ethnicities that make up a school district
    - Different macronutrients (carbohydrates, fat, protein) that make up a meal
    - Different responses to an online poll

- We can make a pie chart with the command plt.pie, passing in the values you want to chart:

```
budget_data = [500, 1000, 750, 300, 100]

plt.pie(budget_data)
plt.show()
```



Exercise:

- MatplotSip keeps track of how many people pay by credit card, cash, Apple pay, or other methods. This is given to you in the payment_method_names and payment_method_freqs lists.

  - Display the payment_method_freqs list as a pie chart.
  - Now, set the axes to be equal.

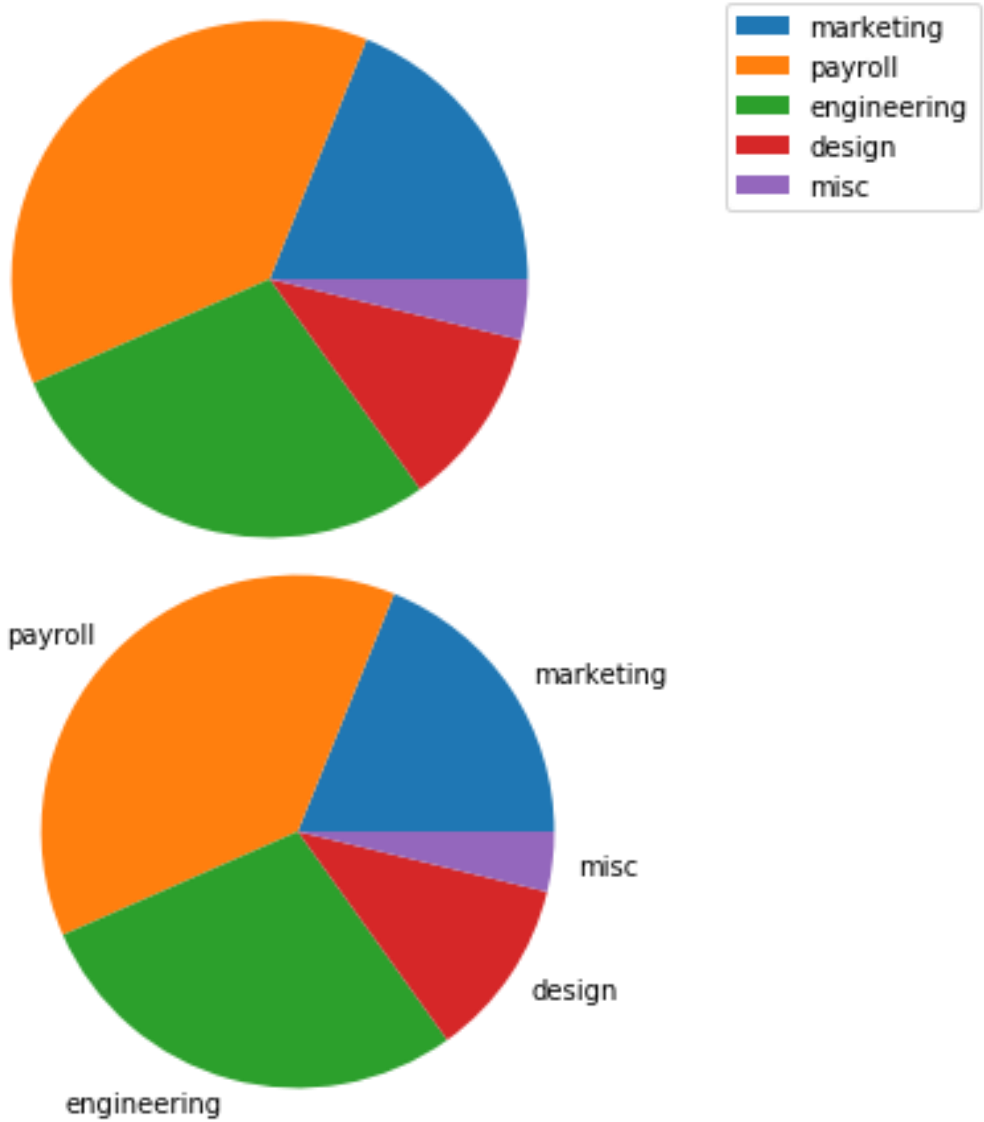
```
from matplotlib import pyplot as plt
import numpy as np

payment_method_names = ["Card Swipe", "Cash", "Apple Pay", "Other"]
payment_method_freqs = [270, 77, 32, 11]

#make your pie chart here

plt.show()
```

Pie Chart Labeling

- We also want to be able to understand what each slice of the pie represents. To do this, we can either:
  - use a legend to label each color, or
  - put labels on the chart itself.

```
budget_data = [500, 1000, 750, 300, 100]
budget_categories = ['marketing', 'payroll', 'engineering', 'design', 'misc']

plt.pie(budget_data)
plt.legend(budget_categories)
```

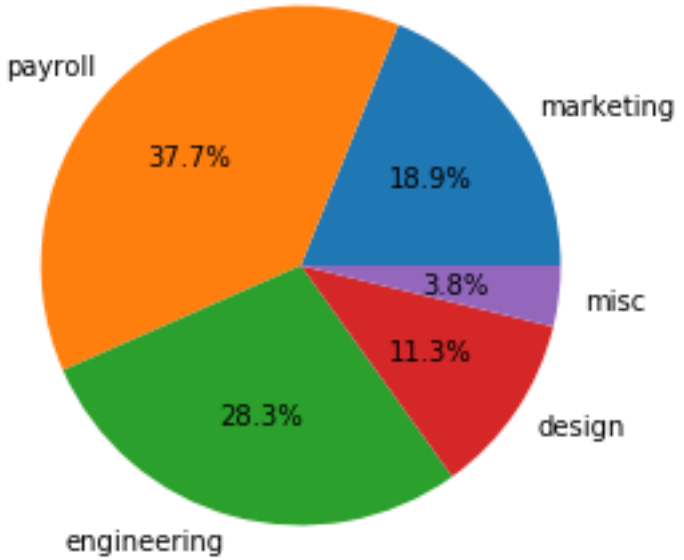
```
#option 2
plt.pie(budget_data, labels=budget_categories)
```



Pie Chart Labeling

- Pie charts is added with the percentage of the total that each slice occupies.
- Matplotlib can add this automatically with the keyword autopct.
- Some common formats are:
  - '%0.2f' — 2 decimal places, like 4.08
  - '%0.2f%%' — 2 decimal places, but with a percent sign at the end, like 4.08%.
  - '%d%%' — rounded to the nearest int and with a percent sign at the end, like 4%.

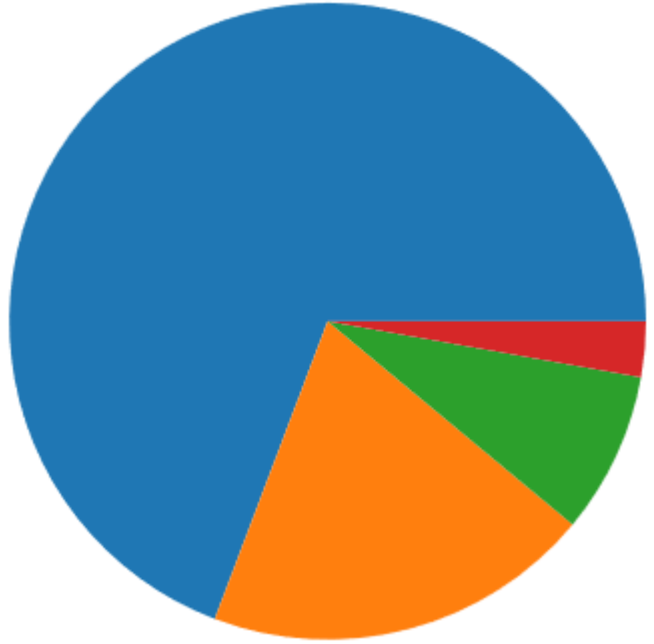
```
plt.pie(budget_data, labels=budget_categories,  autopct='%0.1f%%')
```



Excercise

- Add a legend to the chart you made in the previous exercise by passing in a list of labels to plt.legend. For the labels, use the list payment_method_names.
- Add a percentage to each slice using Matplotlib's autopct parameter. Go to one decimal point of precision.

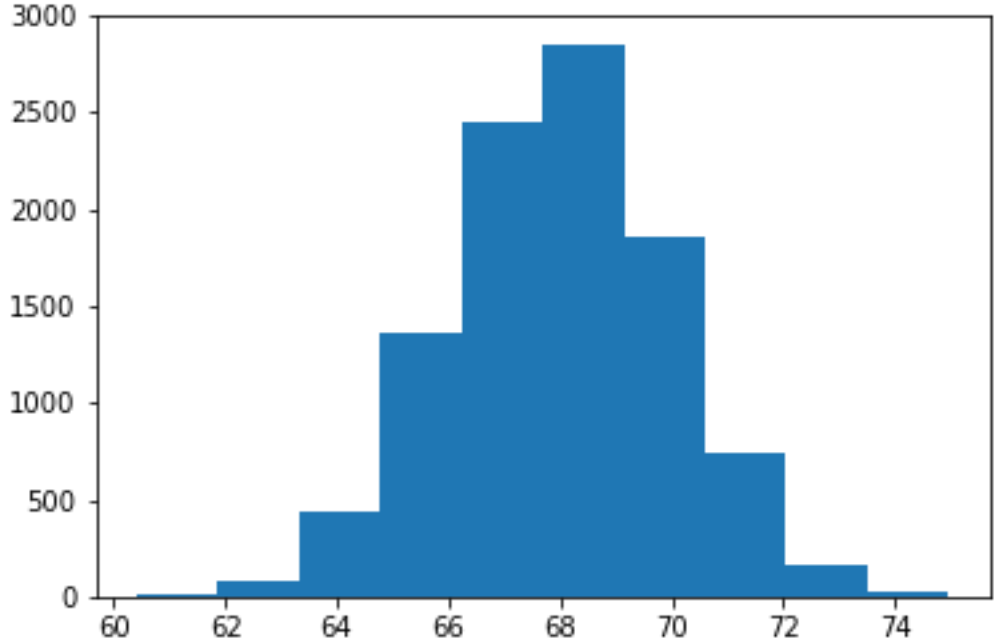
```
import codecademylib
from matplotlib import pyplot as plt

payment_method_names = ["Card Swipe", "Cash", "Apple Pay", "Other"]
payment_method_freqs = [270, 77, 32, 11]

plt.pie(payment_method_freqs)
plt.axis('equal')

plt.show()
```



Histogram

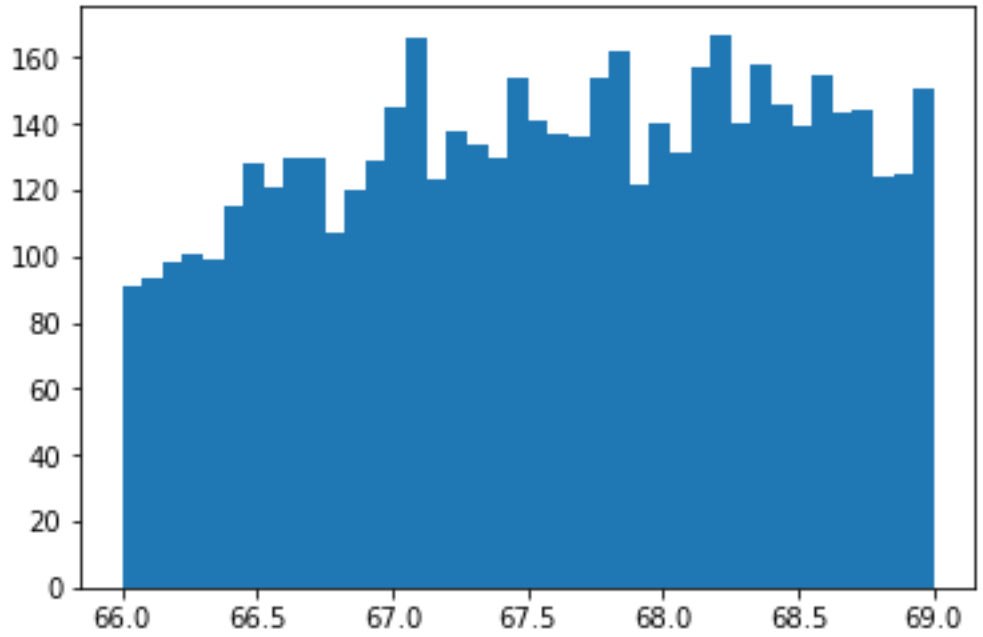
- A histogram tells us how many values in a dataset fall between different sets of numbers (i.e., how many numbers fall between 0 and 10? Between 10 and 20? Between 20 and 30?).
- Each of these questions represents a bin, for instance, our first bin might be between 0 and 10.
  - All bins in a histogram are always the same size.
  - The width of each bin is the distance between the minimum and maximum values of each bin.
  - In our example, the width of each bin would be 10.
- To make a histogram in Matplotlib, we use the command plt.hist
- plt.hist finds the minimum and the maximum values in your dataset and creates 10 equally-spaced bins between those values.

```
plt.hist(dataset)
plt.show()
```



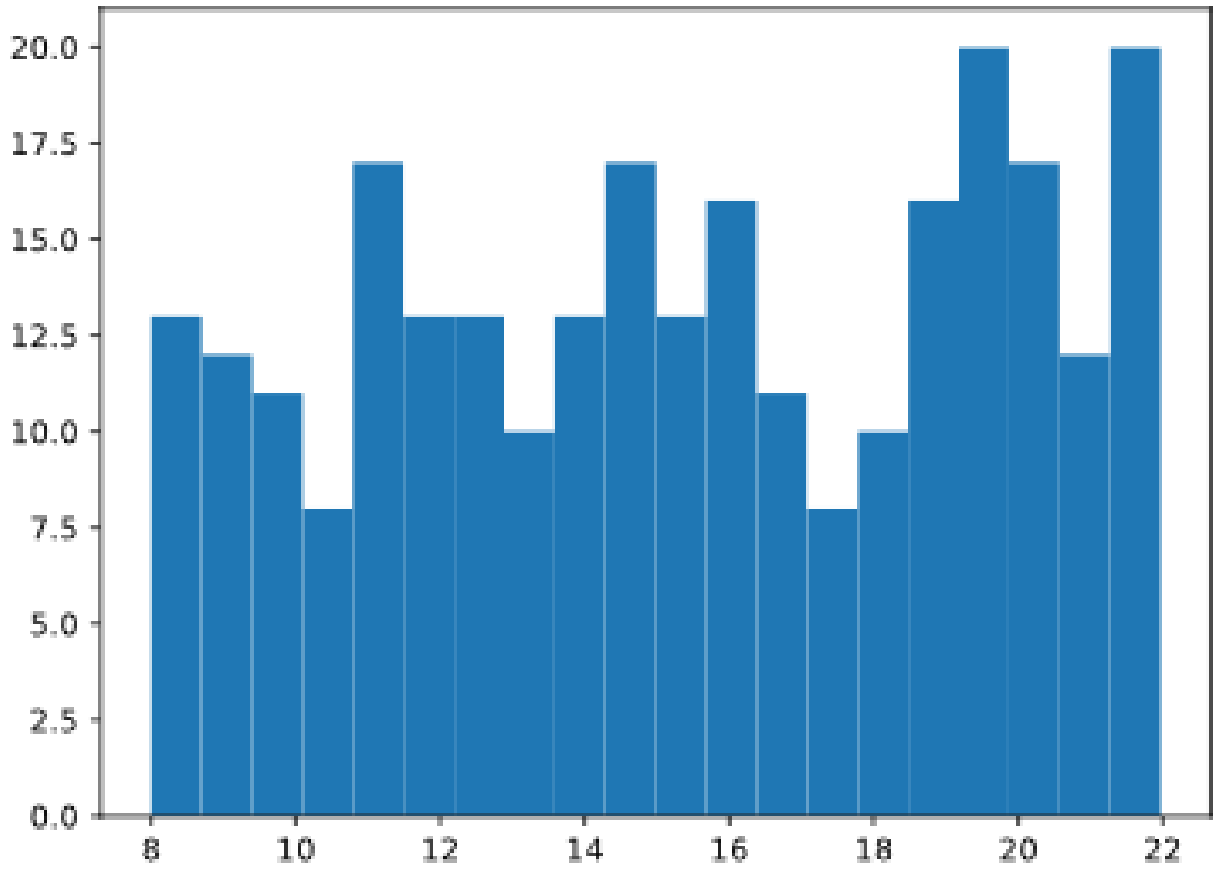
- If we want more than 10 bins, we can use the keyword bins to set how many bins we want to divide the data into.
- The keyword range selects the minimum and maximum values to plot.

```
plt.hist(dataset, range=(66,69), bins=40)
```



```
from matplotlib import pyplot as plt

def convert_time_to_num(time):
  mins = int(time[-2:])
  frac_of_hour = mins/60.0
  hour = int(time[:-3])
  time = hour + frac_of_hour
  return time

sales_times_raw = []
with open('sales_times.csv') as csvDataFile:
  csvReader = csv.reader(csvDataFile)
  for row in csvReader:
    sales_times_raw.append(row[2])
  sales_times_raw = sales_times_raw[1:]

sales_times = []
for time in sales_times_raw:
  sales_times.append(convert_time_to_num(time))
#create the histogram here
plt.hist(sales_times, bins=20)
plt.show()
```

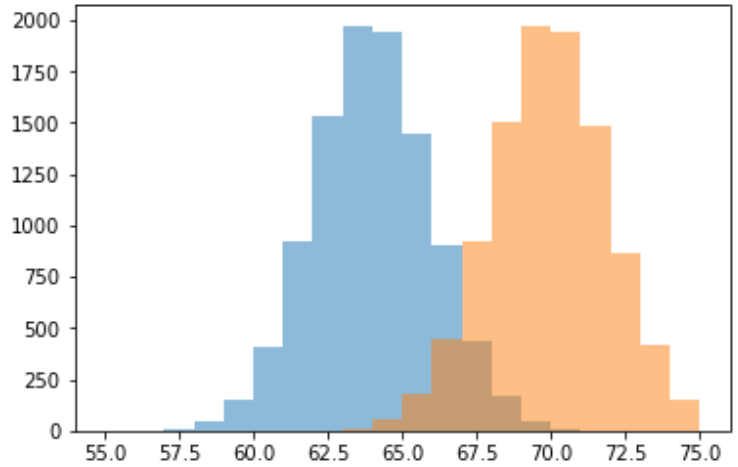
Sample Data of the Dataset
id,card_no,time
1,3531391693620796,19:21



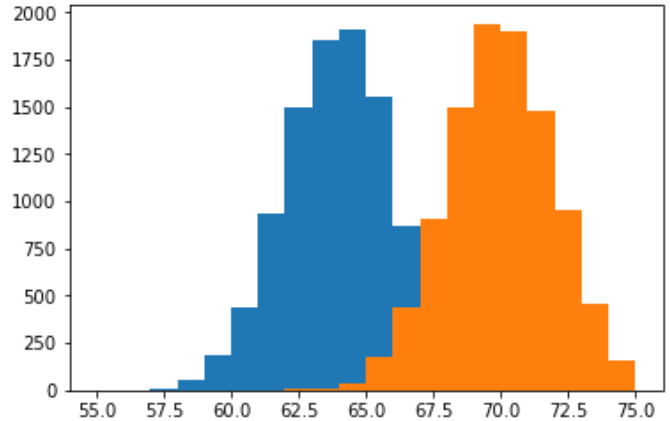
Multiple Histograms

```
plt.hist(a, range=(55, 75), bins=20, alpha=0.5)
plt.hist(b, range=(55, 75), bins=20, alpha=0.5)
```

- If we want to compare two different distributions, we can put multiple histograms on the same plot.
- This could be useful, for example, in comparing the heights of a bunch of men and the heights of a bunch of women.
- However, it can be hard to read two histograms on top of each other.
  - For example, in this histogram, we can't see all of the blue plot, because it's covered by the orange one:



- Use the keyword alpha, which can be a value between 0 and 1. This sets the transparency of the histogram.
  - A value of 0 would make the bars entirely transparent.
  - A value of 1 would make the bars completely opaque.



- use the keyword histtype with the argument 'step' to draw just the outline of a histogram:

```
plt.hist(a, range=(55, 75), bins=20, histtype='step')
plt.hist(b, range=(55, 75), bins=20, histtype='step')
```