Table of Contents

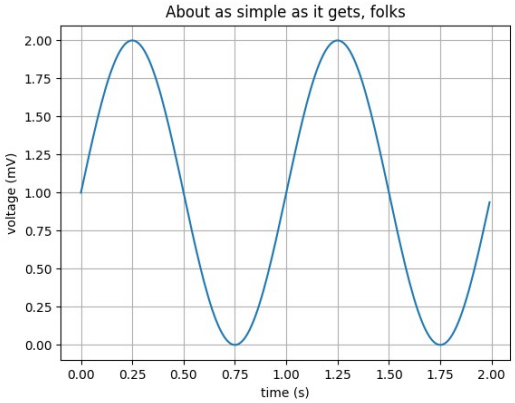
Introduction

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of formats and interactive environments.

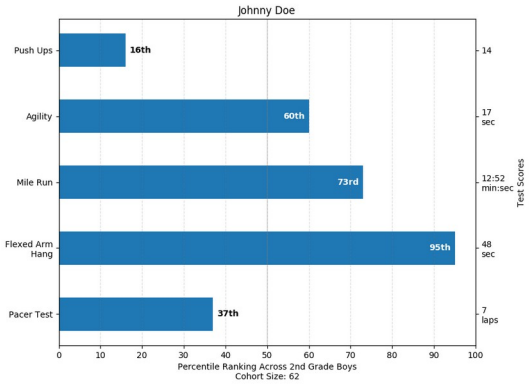
https://matplotlib.org/3.3.3/index.html

Matplotlib can create a large range of visualisations, including:

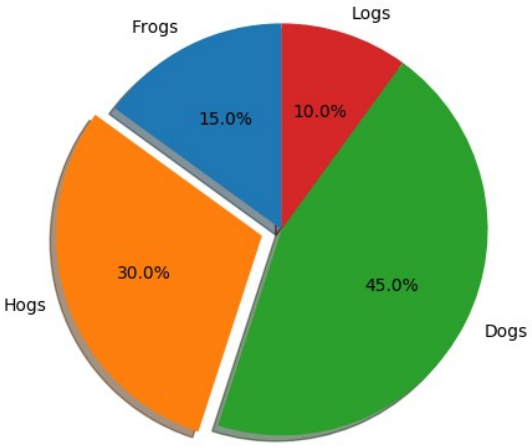
Line plots



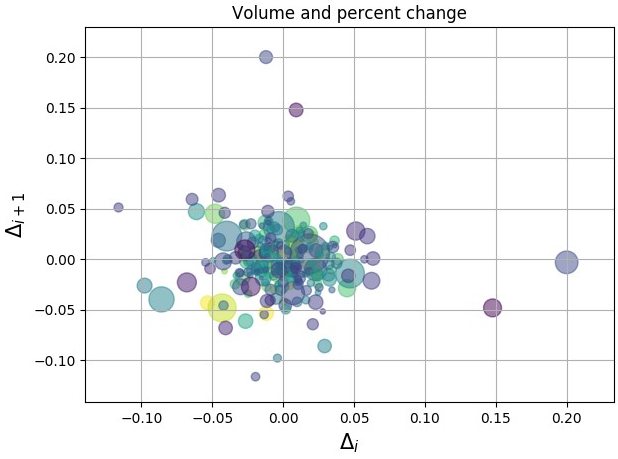
Bar charts



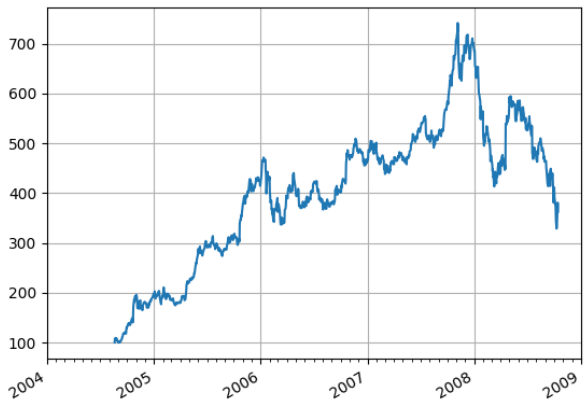
Pie Charts



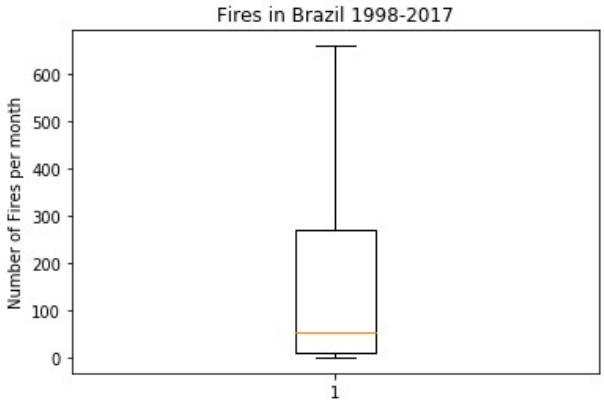
Scatter Plots



Date Plots



Box Plots



Importing Matplotlib

Matplotlib is a third-party library, which means it must be installed before you can use it. It's installed automatically with Anaconda. If you're not using Anaconda, you'll need to install it, e.g. using `pip`:

```
pip install matplotlib
```

Importing is usually done as follows:

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

which imports the `pyplot` module from the `matplotlib` library and uses the alias `plt` for it. The means, instead of referring to `pyplot` features using `matplotlib.pyplot.`*`something`*, you can simply use `plt.`*`something`*.

Object-Oriented Interface

Matplotlib has 2 interfaces:

1. pyplot, which is designed for use in interactive environments such as iPython
2. the Object-Oriented interface, which is recommended for use in Python programs

These notes will focus exclusively on the Object-Oriented interface. This interface includes the following objects:

Object	**Purpose**
Figure	Container for one or more Axes instances
Axes	Rectangular areas to hold the basic elements, such as lines, text, and so on

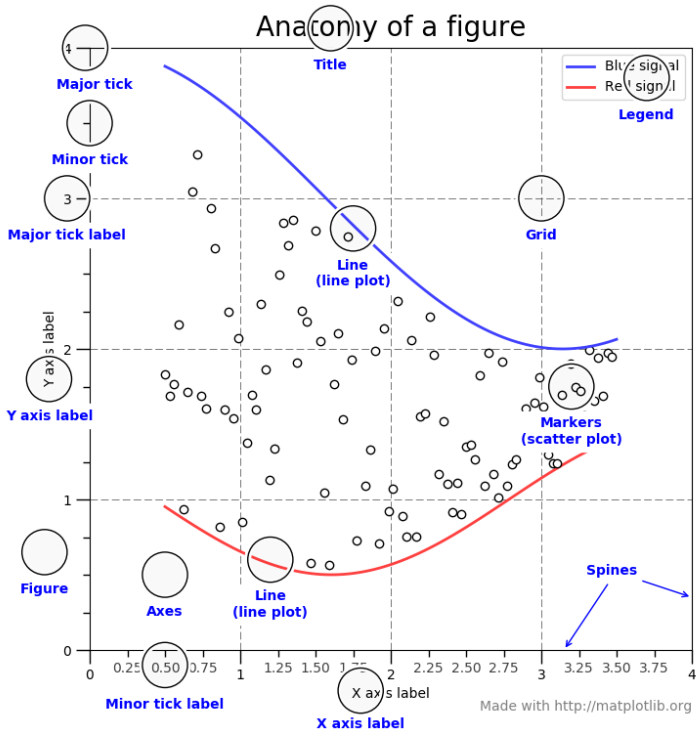
The Figure is the final image that may contain 1 or more Axes; The Axes represent an individual plot (don't confuse this with the word "axis", which refers to the x/y axis of a plot).

https://matplotlib.org/3.3.3/tutorials/introductory/lifecycle.html

Figure, Axes and components

https://matplotlib.org/tutorials/introductory/usage.html#parts-of-a-figure

Figure: The Figure keeps track of all the Axes and other components such as titles and legends. A figure can contain any number of Axes, but will typically have at least one.



Axes: This is what you think of as 'a plot'. The Axes contains two Axis objects (be aware of the difference between Axes and Axis) which take care of the data limits. Each Axes has a title, an x-label, and a y-label .

Axis: These are the number-line-like objects. They take care of setting the graph limits and generating the ticks (the marks on the axis) and ticklabels (strings labeling the ticks). An axis contains *major* and *minor* ticks. The location of the ticks is determined by a `Locator` object and the ticklabel strings are formatted by a `Formatter`. These are done by default, but can be changed if required, for example for date plots.

Other Components: You can add other graphical components such as Text and Lines.

Creating the Visualisations

To start, create an instance of `Figure` and an instance of `Axes`:

```
fig, ax = plt.subplots()
```

The `Figure` contains a canvas, and the `Axes` is a part of the `Figure` on which a visualisation (plot or chart) is displayed. `Figure`s can have multiple `Axes` on them (multiple visualisations).

You can specify the size of the `Figure` in inches, using the keyword argument figsize and specifying the width and height as a tuple, e.g.:

```
fig, ax = plt.subplots(figsize=(10,8))
```

this specifies that the Figure will be 10 inches wide and 8 inches high. (Although Spyder will automatically shrink this to fit it in the Plots window).

The visualisations are generally created using `Axes` methods, for example:

Method	Description
`ax.boxplot()`	Create a box plot, which identifies the median (middle value) and quartiles
`ax.plot_date()`	Create a date plot, which displays the values over time
`ax.bar()`	Create a (vertical) bar chart
`ax.barh()`	Create a horizontal bar chart
`ax.hist()`	Create a histogram
`ax.pie()`	Create a pie chart
`ax.scatter()`	Create a date plot, which displays the values as (x,y) pairs

https://matplotlib.org/api/axes_api.html?#plotting

You can add further information to the visualisation, such as a title, labels and text:

Method	Description
`ax.set_title()`	Set a title for the visualisation
`ax.set_xlabel()`	Set a label on the x-axis
`ax.set_ylabel()`	Set a label on the y-axis
`ax.set_xticks()`	Set the “tick” markers on the x-axis
`ax.set_yticks()`	Set the “tick” markers on the y-axis
`ax.set_xticklabels()`	Set the labels for the “tick” markers on the x-axis
`ax.set_yticklabels()`	Set the labels for the “tick” markers on the y-axis
`ax.legend()`	Display a legend
`ax.text(x,y,str)`	Display the text `str` on the Axes at location `x,y`

https://matplotlib.org/3.3.3/api/axes_api.html#axis-labels-title-and-legend
https://matplotlib.org/3.3.3/api/axes_api.html#text-and-annotations

An `Axes` also has `set()` method which uses keyword arguments to set multiple properties in one go:

```
ax.set(title="",xlabel="",ylabel="")
```

To display the visualisation(s) use: `plt.show()`
To save the figure, use: `fig.savefig(`*`filename`*`)`

Examples

The following are a selection of some of the visualisation available from matplotlib.

Sample Data

The data for the following examples is the Amazon Forest Fires `amazon.csv` file:

amazon.csv (254.82 KB) — 5 of 5 columns — Views

	# year Year when Forest Fires happen	A state Brazilian State	A month Month when Forest Fires happen	# number Number of Forest Fires reported	date Date when Forest Fires where reported
	1998 – 2017	Rio 11% Mato Grosso 7% Other (21) 81%	Janeiro 8% Fevereiro 8% Other (10) 83%	0 – 998	1Jan98 – 1Jan17
1	1998	Acre	Janeiro	0	1998-01-01
2	1999	Acre	Janeiro	0	1999-01-01
3	2000	Acre	Janeiro	0	2000-01-01
4	2001	Acre	Janeiro	0	2001-01-01
5	2002	Acre	Janeiro	0	2002-01-01
6	2003	Acre	Janeiro	10	2003-01-01
7	2004	Acre	Janeiro	0	2004-01-01
8	2005	Acre	Janeiro	12	2005-01-01
9	2006	Acre	Janeiro	4	2006-01-01
10	2007	Acre	Janeiro	0	2007-01-01

Each line of the file contains:

- the year, e.g. 1998
- the name of the State, e.g. Acre
- the month in Brazilian Portuguese, e.g. Janeiro
- the number of fires, e.g. 0
- the date, e.g. 1998-01-01

Note: The dates in the original file were incorrect, and were always set to the 1st of January irrespective of the month. I corrected these using a separate Python program and saved them in `amazon2.csv`.

Example 1: Pie Chart

The following program displays a pie chart of the total number of fires in each state. The program reads in each line of the `amazon2.csv` file, adds each state to a dictionary and associates with it the total number of fires in the state, then displays the pie chart.

```
# Program Name: plot_fires.py
# Purpose: To plot the fires per month in Brazil
# Example of: matplotlib pie chart
import matplotlib.pyplot as plt

# create an empty dictionary
data_by_state = {}

# read the data from the file
with open("amazon2.csv") as datafile:
    # for each line in the file
    for line in datafile:
        # split the line into the components
        year, state, month, fires, date = line.strip().split(",")

        # if this is the first occurence of this state
        if not state in data_by_state:
            data_by_state[state] = int(fires)
        # otherwise add to the existing value
        else:
            data_by_state[state] += int(fires)

# create a figure and an axis object
fig, ax = plt.subplots()

# set the title
ax.set_title("Fires in Acre State, Brazil 1998-2017")

# do a pie chart
ax.pie(data_by_state.values(), labels = data_by_state.keys())
plt.show()

# save the file
fig.savefig('amazon_pie_chart.png', bbox_inches='tight')
```

In general, the code to create a pie chart is:

```
ax.pie(list_of_values, labels = list_of_labels)
```

The title is set using:

```
ax.set_title("Fires in Acre State, Brazil 1998-2017")
```

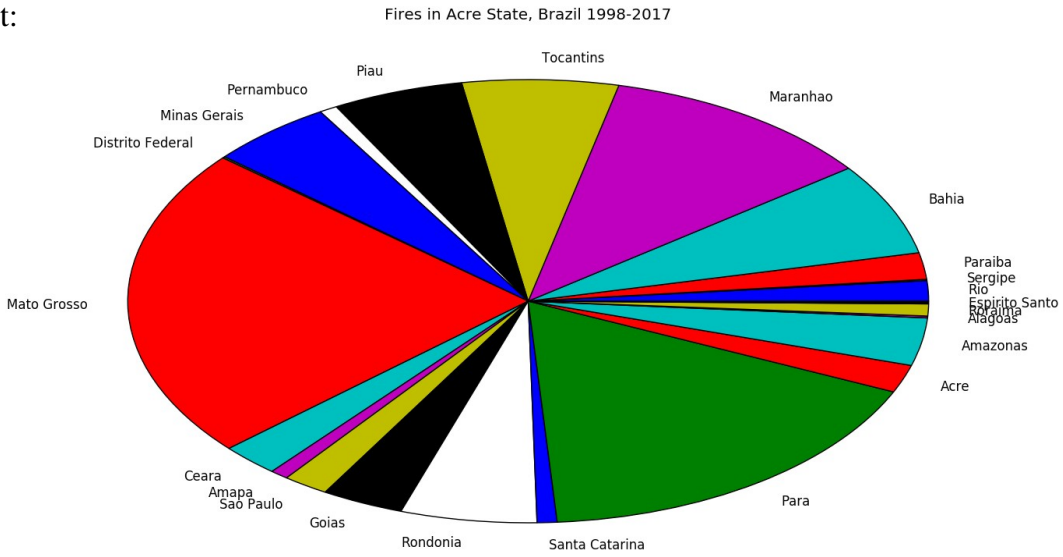
The image is saved using:

```
fig.savefig("amazon_pie_chart.png", bbox_inches="tight")
```

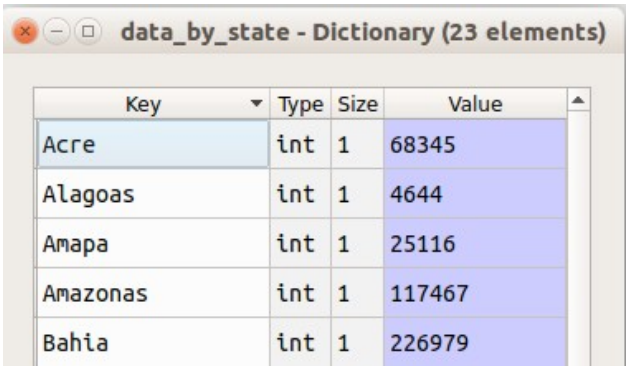
The option `bbox_inches="tight"` removes as much of the white border around the image as possible when saving the file.

Matplotlib Documentation: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.pie.html

Here is the pie chart:



The dictionary stores the name of each state as the "key" and the corresponding "value" is the total number of fires in that state.


data_by_state - Dictionary (23 elements)

Key	Type	Size	Value
Acre	int	1	68345
Alagoas	int	1	4644
Amapa	int	1	25116
Amazonas	int	1	117467
Bahia	int	1	226979

The keys (state names) provide the labels for the segments of the pie chart, the sizes of the segments are determined by the values (total fires):

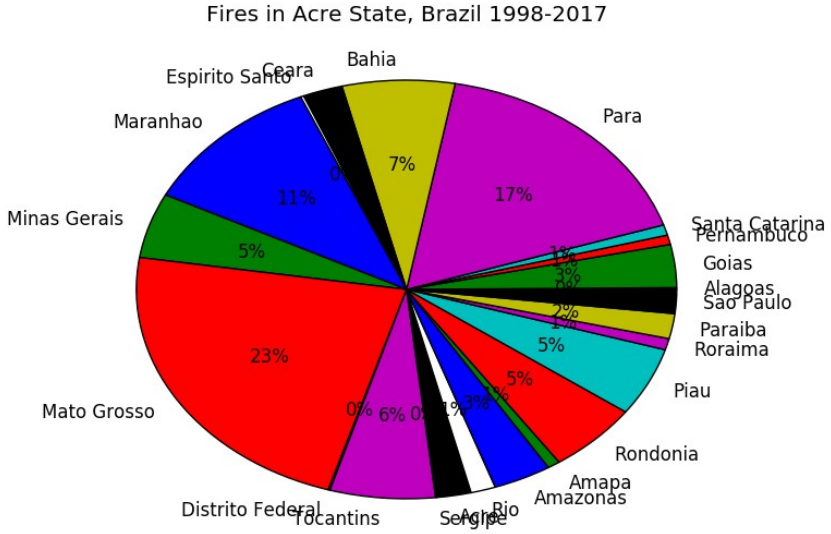
```
ax.pie(data_by_state.values(), labels = data_by_state.keys())
```

You can also add the percentage associated with each segment using the option `autopct`

```
ax.pie(data_by_state.values(), labels=data_by_state.keys(), autopct="%.0f%%")
```

where `%.0f%%` means display the percentages to the nearest whole number (zero decimal places).

For example:



Example 2: Horizontal Bar Chart

The following program displays a horizontal bar chart of the total number of fires in each state. The program reads in each line of the `amazon2.csv` file, adds each state to a dictionary and associates with it the total number of fires in the state.

```
import matplotlib.pyplot as plt

# create an empty dictionary
data_by_state = {}

# read the data from the file
with open("amazon2.csv") as datafile:
    #for each line in the file
    for line in datafile:
        # split the line into the components
        year, state, month, fires, date = line.strip().split(",")

        # if this is the first occurence of this state
        if not state in data_by_state:
            data_by_state[state] = int(fires)
        # otherwise add to the existing value
        else:
            data_by_state[state] += int(fires)
# create a figure and an axis object
fig, ax = plt.subplots()

# set the title
ax.set_title("Fires in Acre State, Brazil 1998-2017")

# set the x positions
y_pos = [ i for i in range(len(data_by_state))]

# set the y tick labels
ax.set_yticks(y_pos)
ax.set_yticklabels(data_by_state.keys())

# set the labels on the axes
ax.set_ylabel("State")
ax.set_xlabel("Total Number of Fires")

# do a horizontal bar chart
ax.barh(y_pos,data_by_state.values(), align="center")
plt.show()

# save the bar chart
fig.savefig('amazon_hbar_chart.png', bbox_inches='tight')
```

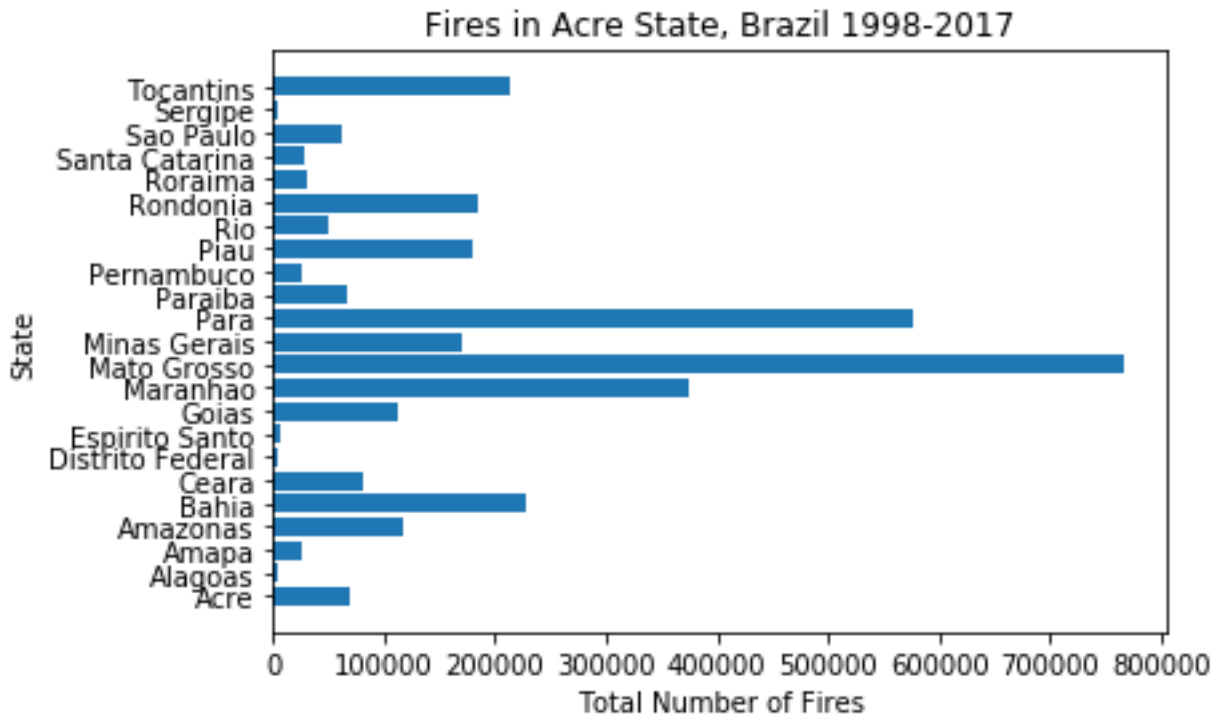
In general, the code to create a horizontal bar chart is:

```
ax.barh(list_of_positions, list_of_values)
```

The positions `y_pos` specify where the bars will be drawn.

Matplotlib Documentation: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.barh.html

Here's the bar chart:



Specifying the Positions

In general, the code to create a horizontal bar chart is:

```
ax.barh(list_of_positions, list_of_values)
```

The positions specify where the bars will be drawn. In the above program, these are specified as follows:

```
y_pos = [ i for i in range(len(data_by_state))]
```

This uses a list comprehension to generate a list `[0, 1, 2, 3, ..., 22]`, as there are 23 items in the dictionary. So the bars will be drawn at positions 0, 1, 2, ... , 22 along the y-axis.

The above program sets tick markers at each of bar positions along the y-axis, and uses the dictionary keys (the state names) as the labels for the tick markers:

```
ax.set_yticks(y_pos)
ax.set_yticklabels(data_by_state.keys())
```

The labels for the x-axis and y-axis are set using:

```
ax.set_ylabel("State")
ax.set_xlabel("Total Number of Fires")
```

Displaying the Values

You can use the `ax.text()` method to display the values at the end of the bars. The syntax is:

```
ax.text(x,y,str)
```

This will display the text `str` on the Axes at location `x,y`

The `enumerate` function returns each value and its index in a tuple (index,value), e.g.

```
for index, value in enumerate(data_by_state.values()):
    ax.text(value, index-0.25, str(value))
```

```
In [7]: list(enumerate(data_by_state.values()))
Out[7]:
[(0, 68345),
 (1, 4644),
 (2, 25116),
 (3, 117467),
 (4, 226979),
```

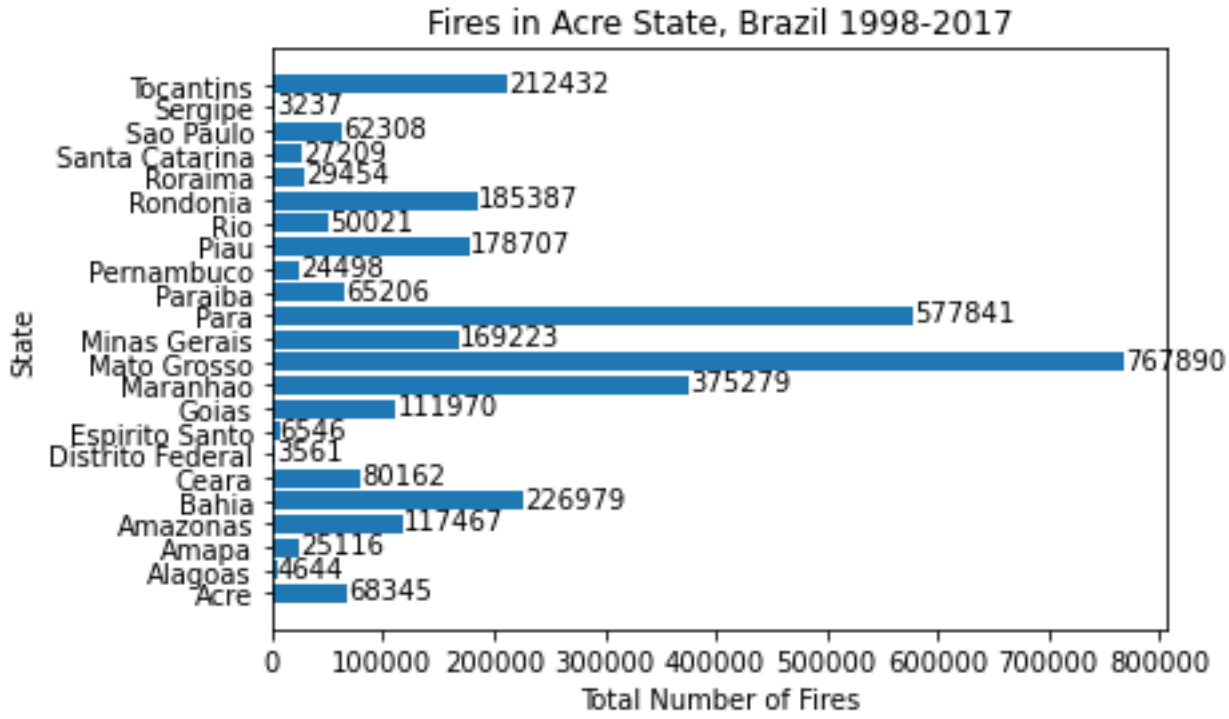
The `index` will be used as the y-coordinate (0, 1, 2) and the `value` is both the x-coordinate and the text to be displayed.

x-coordinate y-coordinate string to be displayed

```
ax.text(value, index-0.25, str(value))
```

The y-coordinate is adjusted by 0.25 to align the value with the bar.

The `value` has to be converted to a string to allowed it to be displayed as text.



Here's the modified Program:

```
# Program Name: section06_example02_bar_chart_with_values.py
# Purpose: To display a bar chart of the fires per state in Brazil
# Example of: matplotlib horizontal bar chart
import matplotlib.pyplot as plt

# create an empty dictionary
data_by_state = {}

# read the data from the file
with open("amazon2.csv") as datafile:
    #for each line in the file
      for line in datafile:
        # split the line into the components
          year, state, month, fires, date = line.strip().split(",")

          # if this is the first occurence of this state
          if not state in data_by_state:
                data_by_state[state] = int(fires)
          # otherwise add to the existing value
          else:
                data_by_state[state] += int(fires)

# create a figure and an axis object
fig, ax = plt.subplots()

# set the title
ax.set_title("Fires in Acre State, Brazil 1998-2017")

# set the x positions
y_pos = [ i for i in range(len(data_by_state))]

# set the y tick labels
ax.set_yticks(y_pos)
ax.set_yticklabels(data_by_state.keys())

# set the labels on the axes
ax.set_ylabel("State")
ax.set_xlabel("Total Number of Fires")

# display the value at the end of each bar
# enumerate returns each value and its index in a tuple (index,value)
for index, value in enumerate(data_by_state.values()):
    # the text function displays text at a specific location
    # the syntax is text(x, y, text_string)
    ax.text(value, index-0.25, str(value))

# draw the horizontal bar chart
ax.barh(y_pos,data_by_state.values(), align="center")
plt.show()

# save the bar chart
fig.savefig('amazon_hbar_chart.png', bbox_inches='tight')
```

Example 3: Date Plot

The following program displays a date plot of the number of fires per month in Acre state. The program reads in each line of the `amazon2.csv` file, adds number of fires per month to one list and adds the date to another list.

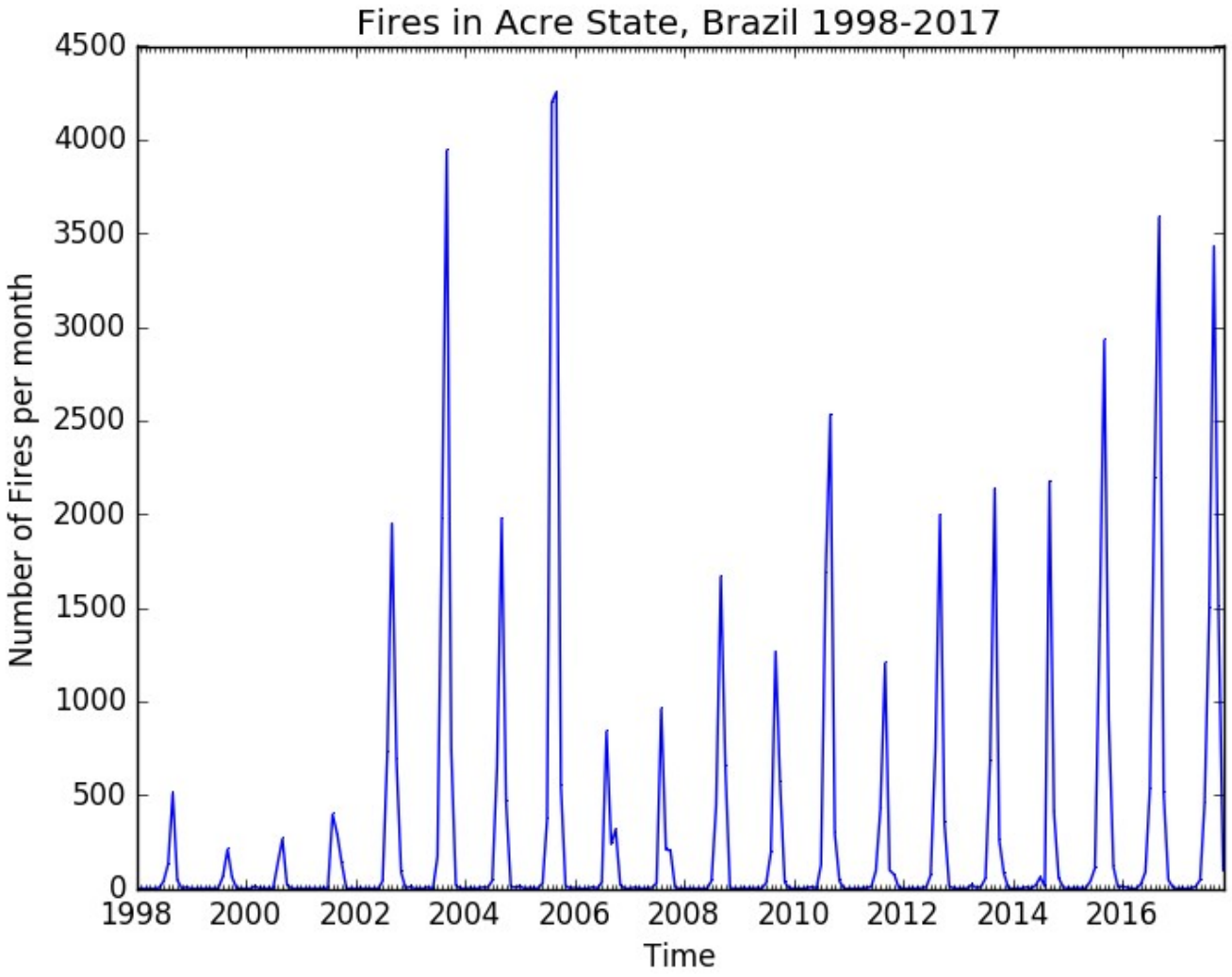
```
# Program Name: section06_example03_date_plot.py
# Purpose: To plot the fires per month in Acre, Brazil
# Example of: matplotlib date plot

import datetime

import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# create a empty lists for the data
x_dates = []
y_fires = []

# date format string
format_str = "%Y-%m-%d"

# read the data from the file
with open("amazon2_Acre.csv") as datafile:
    # for each line in the file
    for line in datafile:
        # split the line into the components
        year, state, month, fires, date = line.strip().split(",")

        # insert into lists
        # convert date in string format to a datetime object
        x_dates.append(datetime.date.fromisoformat(date))
        y_fires.append(int(fires))


# create a figure and an axis object
fig, ax = plt.subplots()

# format the ticks
months = mdates.MonthLocator()  # every month
ax.xaxis.set_minor_locator(months)

# set the labels on the axes
ax.set_xlabel("Time")
ax.set_ylabel("Number of Fires per month")

# set the title
ax.set_title("Fires in Acre State, Brazil 1998-2017")

# draw the date plot
ax.plot_date(x_dates, y_fires, marker = ",", linestyle="-")
plt.show()

# save the image
fig.savefig('acre_date_plot.png', bbox_inches='tight')
```

In general the code to create a date plot is:

```
ax.plot_date(list_of_dates, list_of_values)
```

Matplotlib Documentation: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.plot_date.html

Here's the date plot:



The date plot required additional code as follows:

1. To convert the dates in the format 1998-01-01 to `date` objects

   ```
   x_dates.append(datetime.date.fromisoformat(date))
   ```

2. To set the x-axis minor tick markers as the months:

   ```
   months = mdates.MonthLocator()  # every month
   ax.xaxis.set_minor_locator(months)
   ```

   This automatically displays the year as the major tick labels.

The code to create the date plot

```
ax.plot_date(mpl.dates.date2num(x_dates), y_fires, marker=",", linestyle="-")
```

(1 = `marker=","`, 2 = `linestyle="-"`)

also specified that

1. each value should be represented using a pixel, rather than a point, which is the default
   https://matplotlib.org/3.3.3/api/markers_api.html
2. a line should be drawn between the values (i.e. a line plot)

Example 4: Box Plot

The following program displays a Box Plot of the number of fires per month. A box plot displays:

- the median (middle value) of the data,
- a box indicating the *lower quartile* (the middle value between the minimum and the median) and the *upper quartile* (the middle value between the median and the maximum). The *interquartile range* IQR is the *upper quartile* – the *lower quartile*
- "whiskers" which indicate the values within 1.5 times the interquartile range of the upper and lower quartiles.

The program reads in each line of the `amazon2.csv` file, adds the number of fires to a list and displays the box plot of the values.

```
# Program Name: plot_fires_boxplot.py
# Purpose: To plot the fires per month in Brazil
# Example of: matplotlib boxplot
import matplotlib.pyplot as plt

# create an empty list for the data
fires_list = []

# read the data from the file
with open("amazon1.csv") as datafile:
    # for each line in the file
    for line in datafile:
        # split the line into the components
        year, state, month, fires, date = line.strip().split(",")

        # insert into the list
        fires_list.append(int(fires))

# create a figure and an axis object
fig, ax = plt.subplots()

# set the labels on the axes
ax.set_ylabel("Number of Fires per month")
# set the title
ax.set_title("Fires in Brazil 1998-2017")

# do a box plot
ax.boxplot(fires_list)
plt.show() # display

# save the image
fig.savefig('fires_boxplot.png', bbox_inches='tight')
```

Matplotlib Documentation: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.boxplot.html

Here's the box plot:

In this case, the range of values are so large (the minimum is zero and the maximum is over 25000) that the box plot isn't very useful. You can remove outliers (very large or very small values) using the option `showfliers=False`

```
ax.boxplot(fires_list,showfliers=False)
```

Here is the corresponding box plot:

The median is indicated by the orange horizontal line. The top edge of the box is the Upper Quartile and the bottom edge is the Lower Quartile. The whiskers indicate the values within 1.5 times the interquartile range of the lower and upper quartiles.

Example 5: Multiple Box Plots

Box plots are particularly useful when comparing ranges of values from different data sets. The following program uses a dictionary to store the number of fires per month for each state. The dictionary key is the state and the corresponding value is the list containing the number of fires per month in that state.

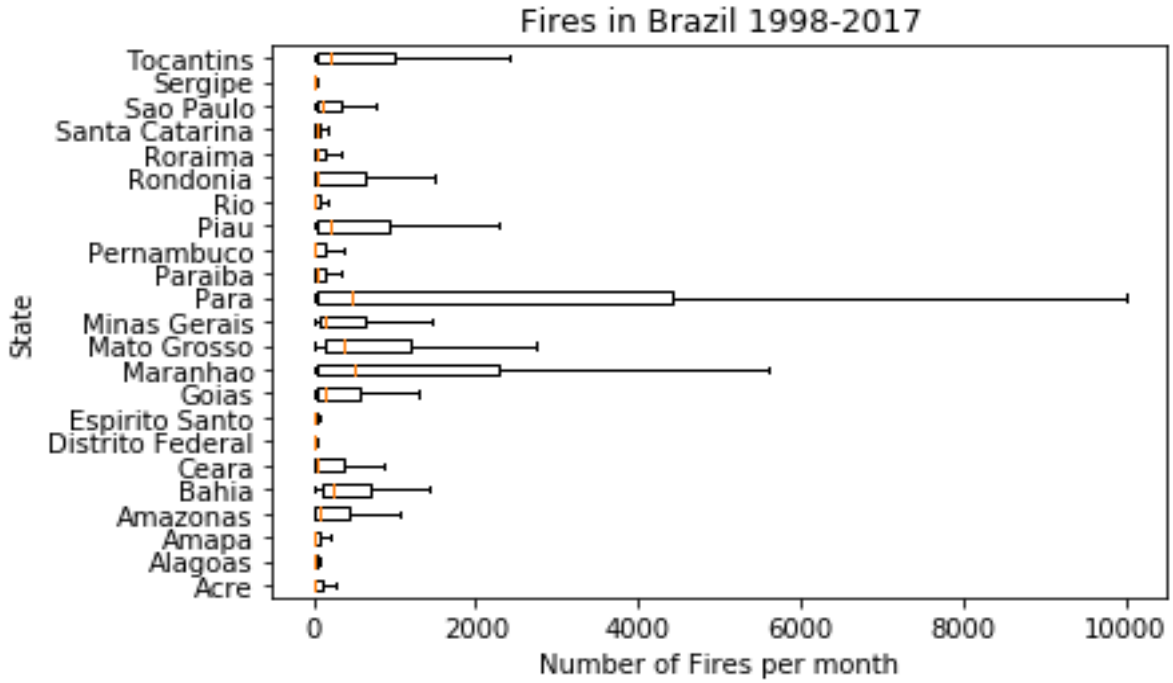
```
import matplotlib.pyplot as plt

# create an empty dictionary for the data
data_by_state = {}

# read the data from the file
with open("amazon2.csv") as datafile:
    # for each line in the file
    for line in datafile:
        # split the line into the components
        year, state, month, fires, date = line.strip().split(",")

        # if this is the first occurence of this state
        if not state in data_by_state:
            # create a list with the number of fires as the first element
            data_by_state[state] = [int(fires) ]
        # otherwise append to the existing list
        else:
            data_by_state[state].append(int(fires))

# create a figure and an axis object
fig, ax = plt.subplots()

# set the labels on the axes
ax.set_xlabel("Number of Fires per month")
ax.set_ylabel("State")
# set the title
ax.set_title("Fires in Brazil 1998-2017")

# do a box plot
ax.boxplot(data_by_state.values(),showfliers=False, vert=False, labels=data_by_state.keys())
plt.show() # display

# save the image
fig.savefig('fires_boxplot.png', bbox_inches='tight')
```

In general, to create multiple box plots, provide a list of data sets, i.e. a list of lists of values. In this case, the values of the dictionary represent the list of fires per month for each state:

data_by_state - Dictionary (23 elements)

Key	Type	Size	Value
Acre	list	239	[0, 0, 0, 0, 0, 3, 37, 130, 509, 44, ...]
Alagoas	list	240	[0, 0, 0, 0, 0, 0, 0, 1, 14, 20, ...]
Amapa	list	239	[0, 0, 0, 0, 0, 0, 0, 1, 20, 42, ...]
Amazonas	list	239	[0, 0, 0, 0, 0, 2, 71, 321, 267, 83, ...]
Bahia	list	239	[0, 0, 0, 0, 0, 55, 219, 815, 2718, 1969, ...

Here are the box plots:



The box plots are drawn horizontally, using the option `vert=False`. Labels are also applied:

```
ax.boxplot(data_by_state.values(),showfliers=False,
           vert=False, labels=data_by_state.keys())
```

Due to the number of zeros in the data for the Amazon fires, the box plots are in many cases quite compact. The following program provides another example of multiple boxplots.

Example 6: Multiple Box Plots – Triple Jump

The following program reads the results of the 2019 Women's Triple Jump Final from the file `triple_jump.csv` and uses a dictionary to store the jumps for each competitor. The dictionary key is the name of the competitor and the corresponding value is the list containing the distances jumped.

```
import matplotlib.pyplot as plt

print("This program displays multiple box plots")

# create an empty dictionary to store the results
# dictionary key is the name of the competitor
# corresponding value is the list of jumps
results ={}

print()
print("2019 Women's Triple Jump Final Results")
print()
# open the file
with open("triple_jump.csv") as data_file:
    # for each line in the file
    for line in data_file:
        # split each line into a list of components, but keep the jumps together
        values = line.strip().split(",", maxsplit=3)

        # for clarity, assigning the items in the values list to variables
        name = values[0]
        bib = values[1]
        country = values[2]

        # create a list of the jump distances, ignoring non-jumps
        jumps = [ float(x) for x in values[3].split(",") if float(x) != 0 ]

        print(f"{bib:>4} {name:18} ({country:3}) {jumps}")

        # insert into the dictionary
        # each competitor's name is associated with her list of jumps
        results[name] = jumps

# create a figure and an axis object
fig, ax = plt.subplots()

# set the labels on the axes
ax.set_xlabel("Distances")
ax.set_ylabel("Competitor")
# set the title
ax.set_title("2019 Women's Triple Jump Results")

# do a box plot
ax.boxplot(results.values(), vert=False, labels=results.keys())
plt.show() # display

# save the image
fig.savefig('triple_jump_boxplot.png', bbox_inches='tight')
```

Here's the csv file:

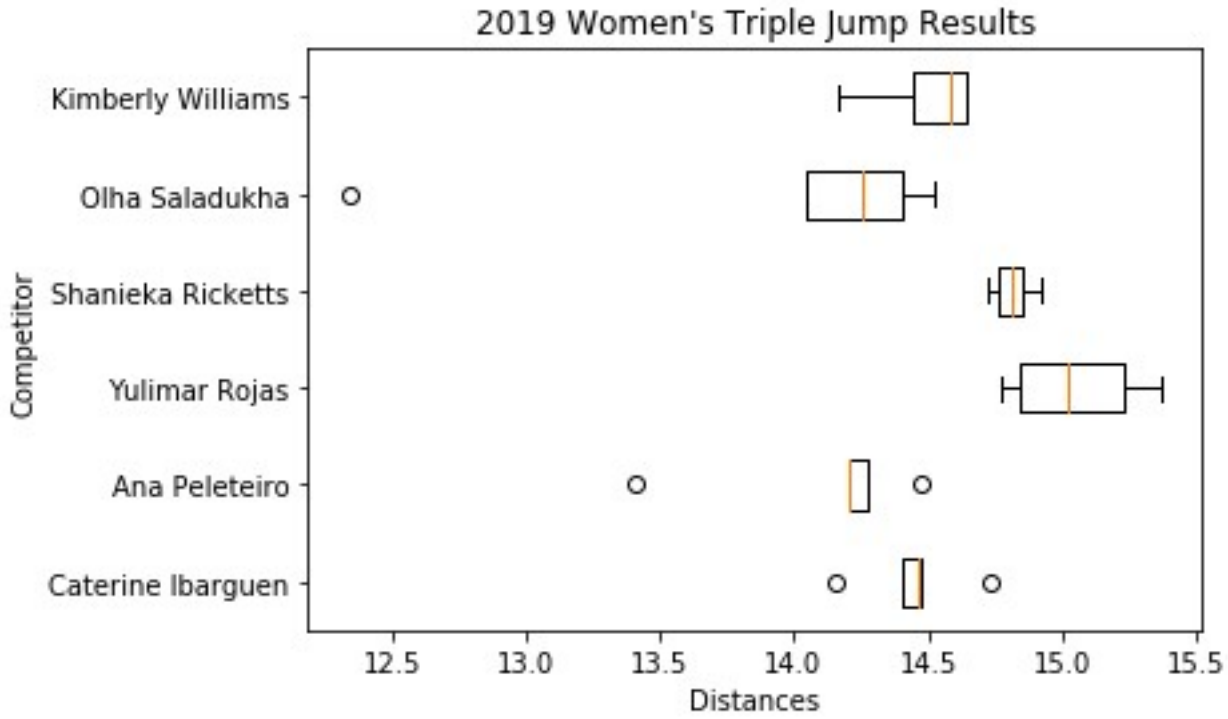
```
Caterine Ibarguen,542,COL,14.16,0.0,14.40,14.46,14.73,14.47
Ana Peleteiro,680,ESP,14.47,13.41,0.0,14.27,14.20,14.20
Yulimar Rojas,2035,VEN, 14.87, 15.37, 0, 15.18, 14.77, 0.0
Shanieka Ricketts,1199,JAM,14.81,14.76,14.92,14.72,14.85,0.0
Olha Saladukha,1842,UKR,14.52,14.40,0.0,12.34,14.25,14.05
Kimberly Williams,1205,JAM,14.64,14.64,14.53,0.0,0.0,14.17
```

Here's the dictionary:

results - Dictionary (6 elements)

Key	Type	Size	Value
Ana Peleteiro	list	5	[14.47, 13.41, 14.27, 14.2, 14.2]
Caterine Ibarguen	list	5	[14.16, 14.4, 14.46, 14.73, 14.47]
Kimberly Williams	list	4	[14.64, 14.64, 14.53, 14.17]
Olha Saladukha	list	5	[14.52, 14.4, 12.34, 14.25, 14.05]
Shanieka Ricketts	list	5	[14.81, 14.76, 14.92, 14.72, 14.85]
Yulimar Rojas	list	4	[14.87, 15.37, 15.18, 14.77]

Here are the box plots:



The box plots show that, not only did the winner Yulimar Rojas has the longest jump, she also had the highest median; that means that her overall performance was better than each of the other competitors.

Example 7: Scatter Plot – Super Computers

A Scatter Plot plots (x,y) pairs. In this example, the values are based on the Top 500 Supercomputers:

- x is the number of cores in the supercomputer (TotalCores)
- y is the supercomputer performance R_{max}

Here's a section of the data:

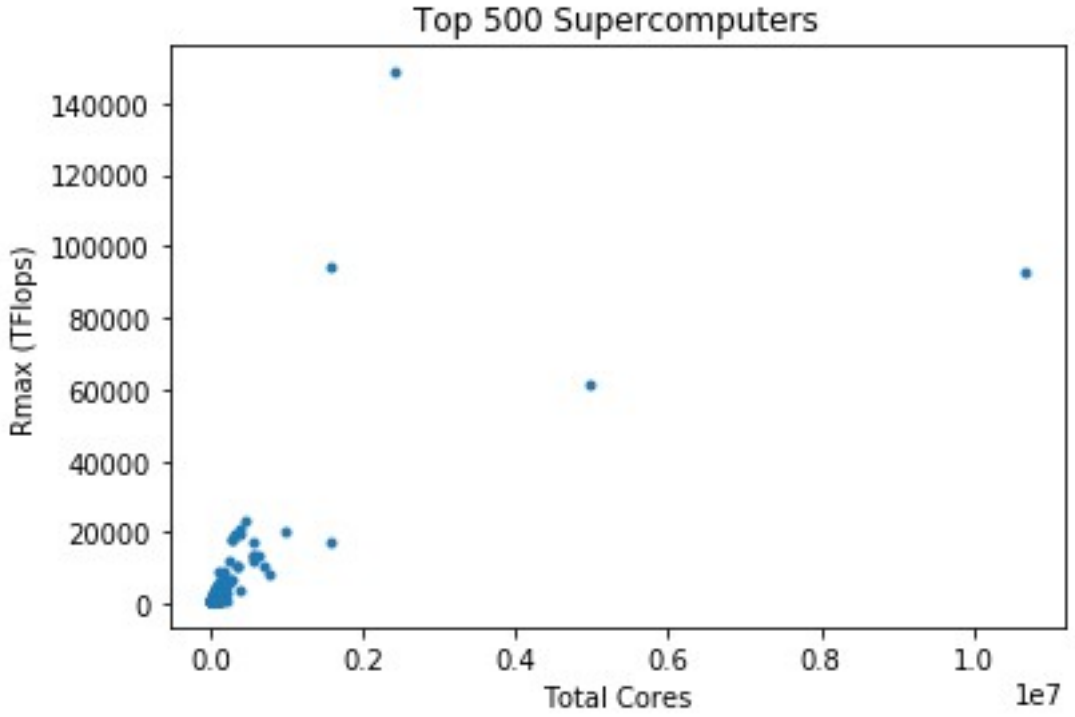
Rank	Name	Total Core	Rmax [TFlop/s]	Rpeak [TFlop/s]
1	Summit	2414592	148600	200794.88
2	Sierra	1572480	94640	125712
3	Sunway Taih	10649600	93014.59388	125435.904
4	Tianhe-2A	4981760	61444.5	100678.664
5	Frontera	448448	23516.4	38745.907

The program reads the data from the CSV file `TOP500_201906.csv` and stores the Total Cores and Rmax values in separate lists. It then does a scatter plot of the data.

```
import matplotlib.pyplot as plt

# create empty lists for the data
x_cores = []
y_rmax = []

# read the data from the file
with open("TOP500_201906.csv") as datafile:
    # read the headers line
    headers = datafile.readline()

    # for each line in the file
    for line in datafile:
        # split the line into the components
        rank, name, cores, rmax, rpeak  = line.strip().split(",")

        # insert into lists
        x_cores.append(int(cores))
        y_rmax.append(float(rmax))


# create a figure and an axis object
fig, ax = plt.subplots()

# set the labels on the axes
ax.set_xlabel("Total Cores")
ax.set_ylabel("Rmax (TFlops)")

# set the title
ax.set_title("Top 500 Supercomputers")

# do a scatter plot
ax.scatter(x_cores, y_rmax, marker=".")
plt.show()

# save the image
fig.savefig('scatterplot_supercomputers.png', bbox_inches='tight')
```

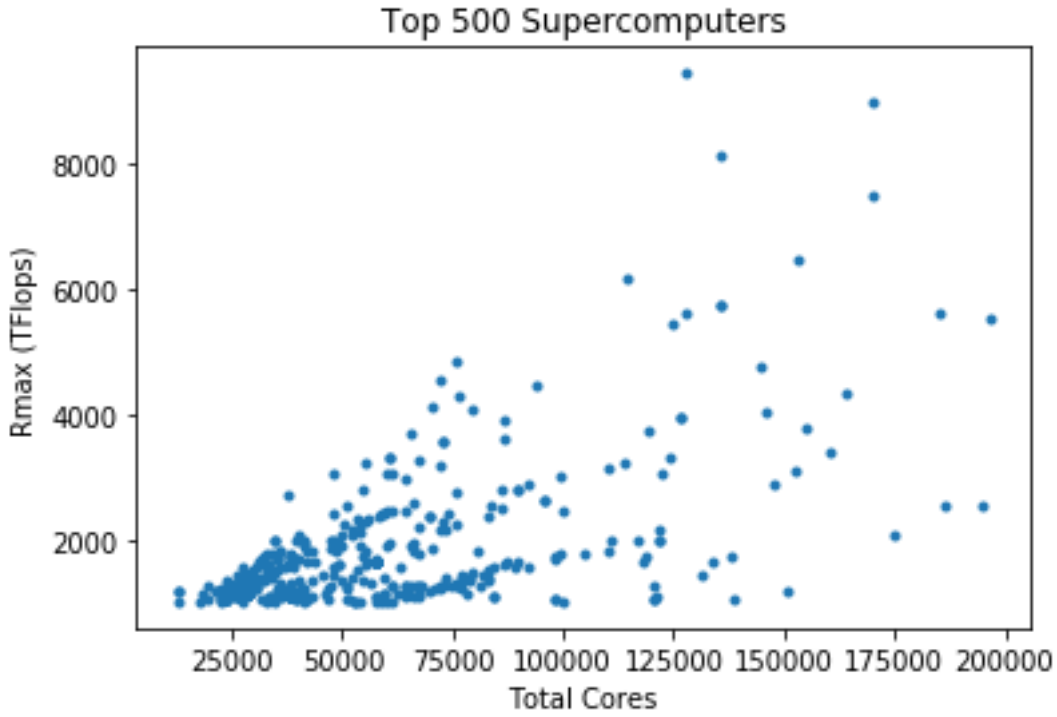
Matplotlib Documentation: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.axes.Axes.scatter.html

Here's the scatter plot:



A small number of data points in the extreme range mean that the large number of data points appear as a cluster. The following scatter plot was created by excluding points where the number of cores was over 200,000:

```
# insert into lists
if int(cores) < 200000:
    x_cores.append(int(cores))
    y_rmax.append(float(rmax))
```



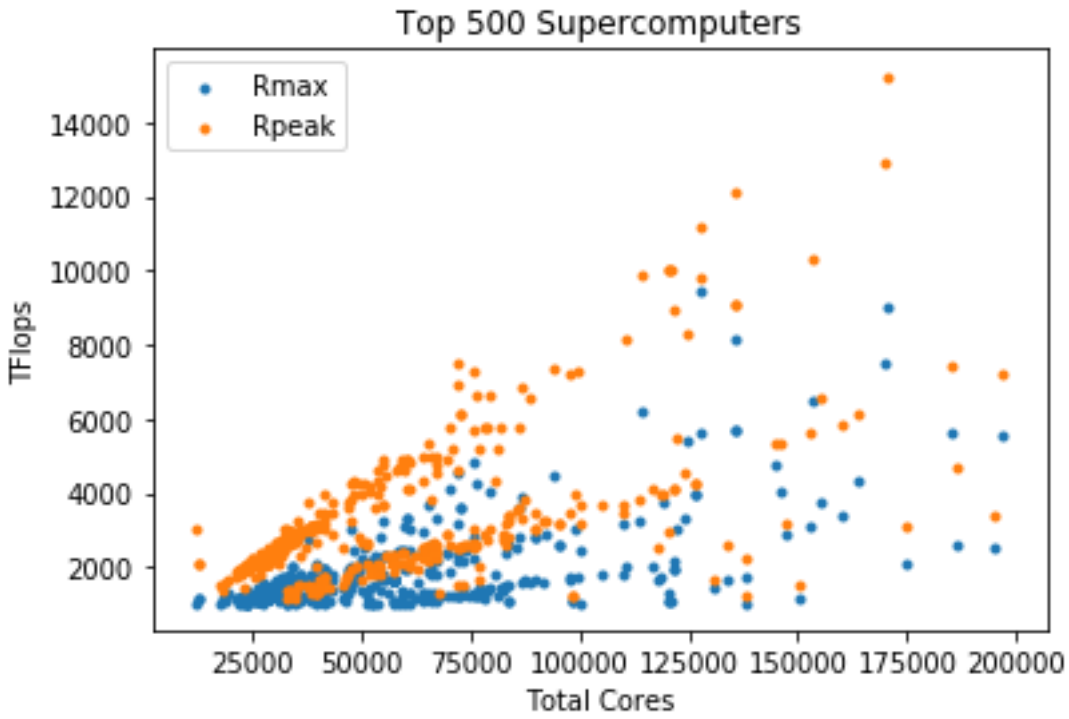
Example 8: Multiple Plots in a single Axes - Supercomputers
The following program demonstrates how you can include multiple scatter plots in a single Axes. In this case, the program also stores the R_{peak} performance values in a list and then plots the values against the number of cores.

```
import matplotlib.pyplot as plt

# create empty lists for the data
x_cores = []
y_rmax = []
y_rpeak = []

# read the data from the file
with open("TOP500_201906.csv") as datafile:
    # read the headers line
    headers = datafile.readline()

    # for each line in the file
    for line in datafile:
        # split the line into the components
        rank, name, cores, rmax, rpeak  = line.strip().split(",")

        # insert into lists
        if int(cores) < 200000:
            x_cores.append(int(cores))
            y_rmax.append(float(rmax))
            y_rpeak.append(float(rpeak))

# create a figure and an axis object
fig, ax = plt.subplots()

# set the labels on the axes
ax.set_xlabel("Total Cores")
ax.set_ylabel("TFlops")
# do a scatter plot
ax.scatter(x_cores, y_rmax, marker=".")
ax.scatter(x_cores, y_rpeak, marker=".")
ax.legend(["Rmax", "Rpeak"])
plt.show()

# save the image
fig.savefig('scatterplot_supercomputers2.png', bbox_inches='tight')
```

The key point is that to do multiple plots on the same Axes, you just call the plot method multiple times on the same Axes.

The `marker="."` argument specifies the marker style to use for the scatter plot. Matplotlib will automatically assign different colours to the 2 scatter plots.

Here is the plot:



The legend is created using the code:

```
ax.legend(["Rmax", "Rpeak"])
```

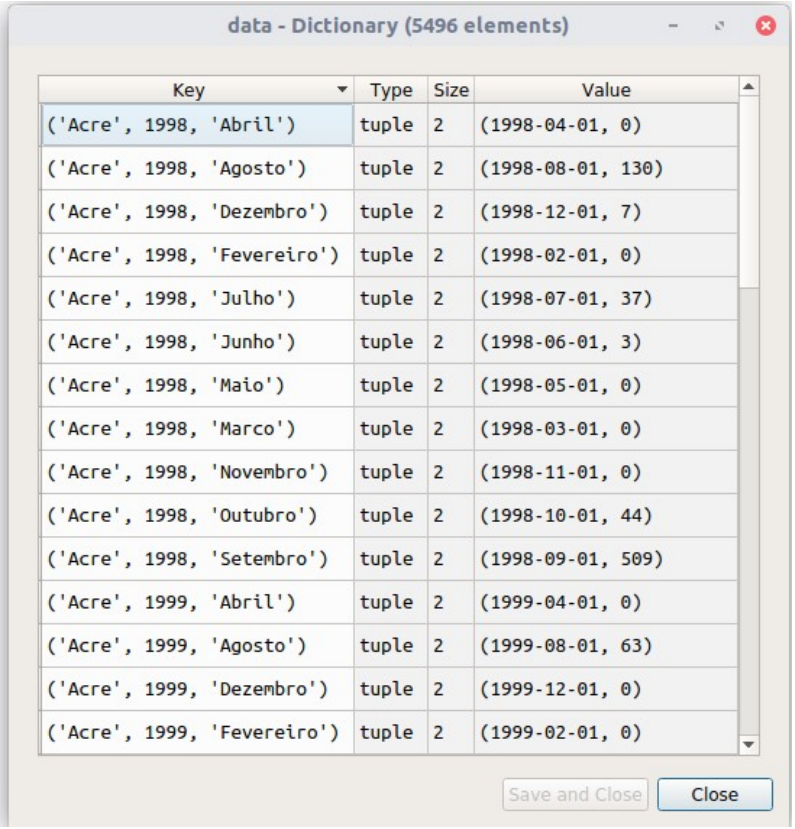
In this example, the legend is set manually, but it is possible to set it automatically (depending on the plot).

Matplotlib Documentation: https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.axes.Axes.legend.html

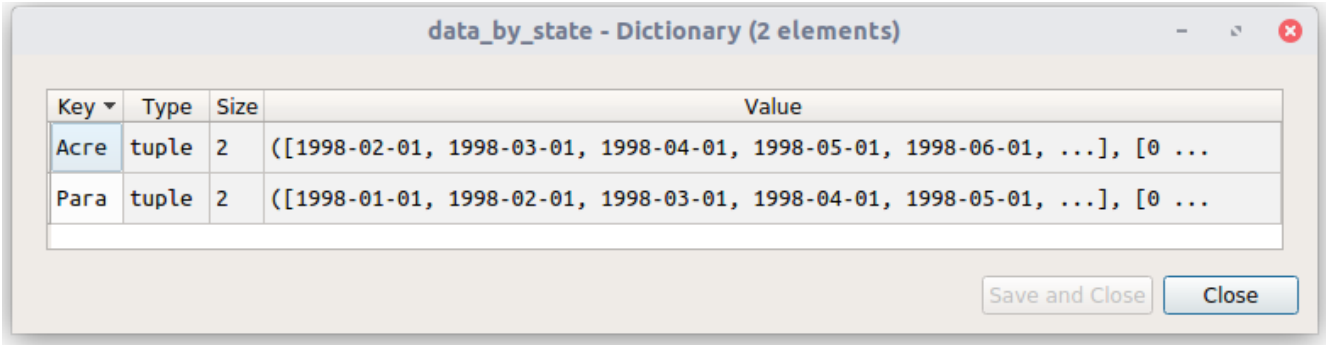
Example 9: Multiple Plots in a single Axes – Forest Fires

The following program demonstrates how you can include multiple date plots in a single Axes. The program reads in each line of the `amazon2.csv` file, and inserts each record in the dictionary as follows:

- dictionary key is a tuple containing (state, year, month)
- corresponding value is a tuple containing (date, number of fires)



The program then extracts the data for each state from the dictionary, and does a dateplot for 2 states, Acre and Para.



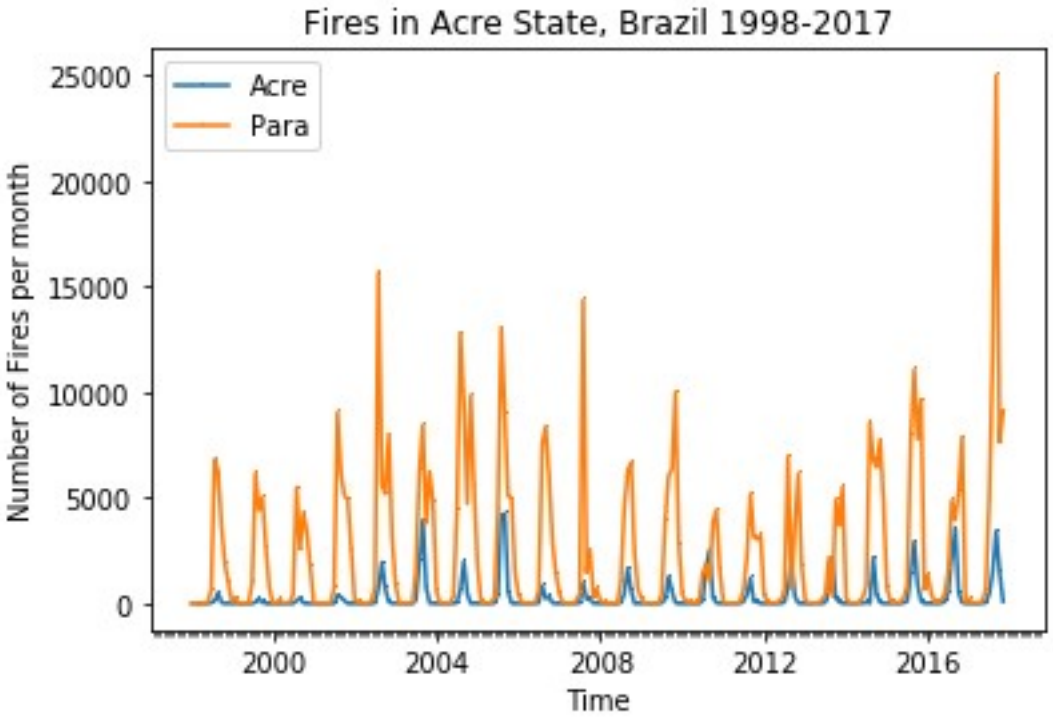
Here's the program:

```
# Program Name: section06_example09_dateplots.py
# Purpose: Plot Amazon Fires by State
# Example of: Multiple Plots on a single Axes
import datetime

import matplotlib.pyplot as plt
import matplotlib.dates as mdates

print("This program processes forest fire data")

# start with an empty dictionary
# dictionary keys will be the (state, year, month)
# corresponding values will be the (date, number of fires)
data = {}

# date format string
format_str = "%Y-%m-%d"

print()
print("Fires per month in Brazil, 1998-2017")
# open the file
with open("amazon2.csv") as data_file:
    # read in the first line containing the headers
    headers = data_file.readline()

    # for each other line in the file
    for line in data_file:
        # split each line into components (remove white space from ends of line)
        year, state, month, fires, date = line.strip().split(",")

        # insert the data into the dictionary (converting dates into datetime objects)
        data[(state, int(year), month)] = (datetime.date.fromisoformat(date),int(fires))

print()
print(f"Number of values: {len(data)}")

# create a list of states
states = []

# go through each key in the dictionary
for state,year,month in data.keys():
    # if we haven't already seen this state
    if state not in states:
        # add it to the list of states
        states.append(state)
```

(continued on the next page)

```
# create a figure and an axis object
fig, ax = plt.subplots()

# format the ticks
months = mdates.MonthLocator()  # every month
ax.xaxis.set_minor_locator(months)

# set the labels on the axes
ax.set_xlabel("Time")
ax.set_ylabel("Number of Fires per month")

# set the title
ax.set_title("Fires in Acre State, Brazil 1998-2017")


# data by state
print()
print(f"{'State':16}  Total")
# create a new empty dictionary
# the dictionary keys are the state names
# the values are the list of fires in that state
data_by_state = {}
# for each state in the list of states
for state in [ "Acre", "Para" ]:
    # insert the state along with the (list of dates and list of fires) for that state
    data_by_state[state] = [ value[0] for key,value in data.items() if key[0] == state ],[ value[1] for key,value in data.items() if key[0] == state ]

    # date plot for the current state
    ax.plot_date(data_by_state[state][0], data_by_state[state][1], marker = ",", linestyle="-")

# manually set a legend
ax.legend([ "Acre", "Para" ])
plt.show()

fig.savefig('amazon_date_plot.png', bbox_inches='tight')
```

Here is the plot:



Example 10: Multiple Axes in a Figure

You can create a figure which contains multiple plots. This is done using the method

```
plt.subplots(nrows,ncols)
```

where *nrows* is the number of rows in the subplot grid
ncols is the number of columns in the subplot grid

0,0	0,1
1,0	1,1
2,0	2,1

For example:

```
fig, ax = plt.subplots(3,2)
```

creates a subplot grid with 3 rows and 2 columns

To specify where a given plot should appear, you then use the syntax:

```
ax[row,col].plot_method()
```

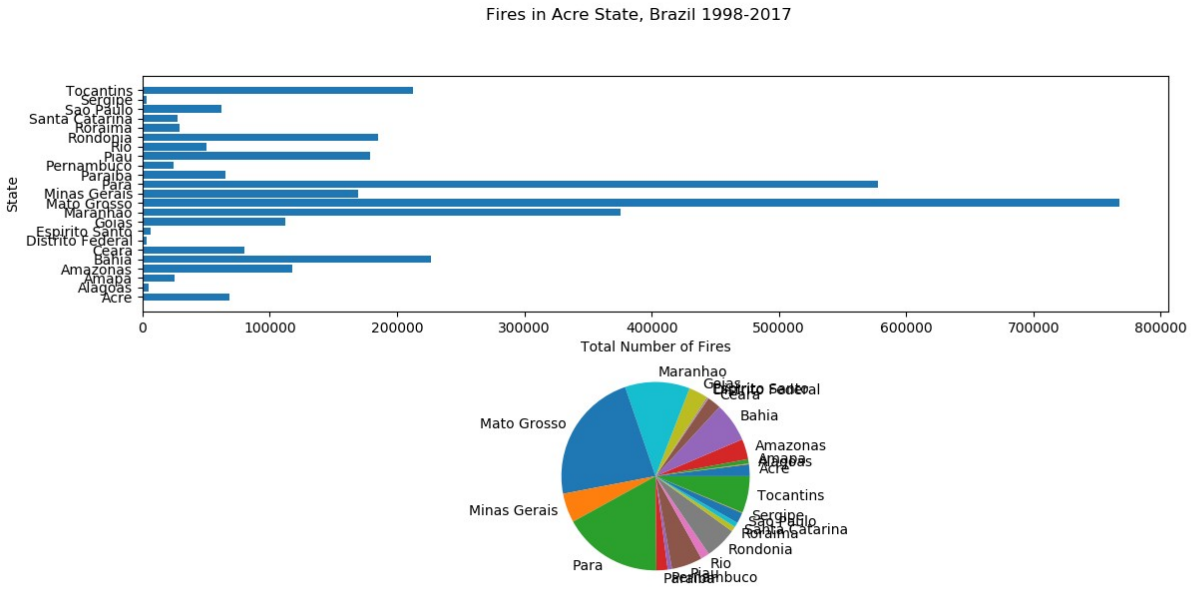
where *row* is the row index
col is the column index
and *plot_method* is the method required.

The following program does a pie chart and a bar chart for the Amazon fires data. It uses the code

```
fig, ax = plt.subplots(2)
```

to creates a subplot grid with 2 rows and 1 columns; and then the axes are `ax[0]` for the first row and `ax[1]` for the second row. The bar chart will be drawn on the first row and the pie chart on the second row.

Here is the image produced.



Unfortunately the labels on the pie chart are not very clear because there are a number of segments with very small percentages, meaning that the labels overlap.

Here is the program:

```
import matplotlib.pyplot as plt

# create an empty dictionary
data_by_state = {}

# read the data from the file
with open("amazon2.csv") as datafile:
    #for each line in the file
    for line in datafile:
        # split the line into the components
        year, state, month, fires, date = line.strip().split(",")

        # if this is the first occurence of this state
        if not state in data_by_state:
            data_by_state[state] = int(fires)
        # otherwise add to the existing value
        else:
            data_by_state[state] += int(fires)

# create a figure and an axis object using a subplot grid: 2 rows, 1 column
fig, ax = plt.subplots(2)

# set the title
fig.suptitle("Fires in Acre State, Brazil 1998-2017")

# set the x positions
y_pos = [ i for i in range(len(data_by_state))]

# set the y tick labels
ax[0].set_yticks(y_pos)
ax[0].set_yticklabels(data_by_state.keys())

# set the labels on the axes
ax[0].set_ylabel("State")
ax[0].set_xlabel("Total Number of Fires")

# do a horizontal bar chart on the first row
ax[0].barh(y_pos,data_by_state.values(), align="center")
# do a pie chart on the second row
ax[1].pie(data_by_state.values(), labels = data_by_state.keys())

plt.show()

# save the bar chart
fig.savefig('amazon_hbar_pie.png', bbox_inches='tight')
```

Matplotlib Documentation: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplots.html
https://matplotlib.org/3.3.3/gallery/subplots_axes_and_figures/subplots_demo.html

Example 11: Multiple Axes in a Figure, Sharing x-axis

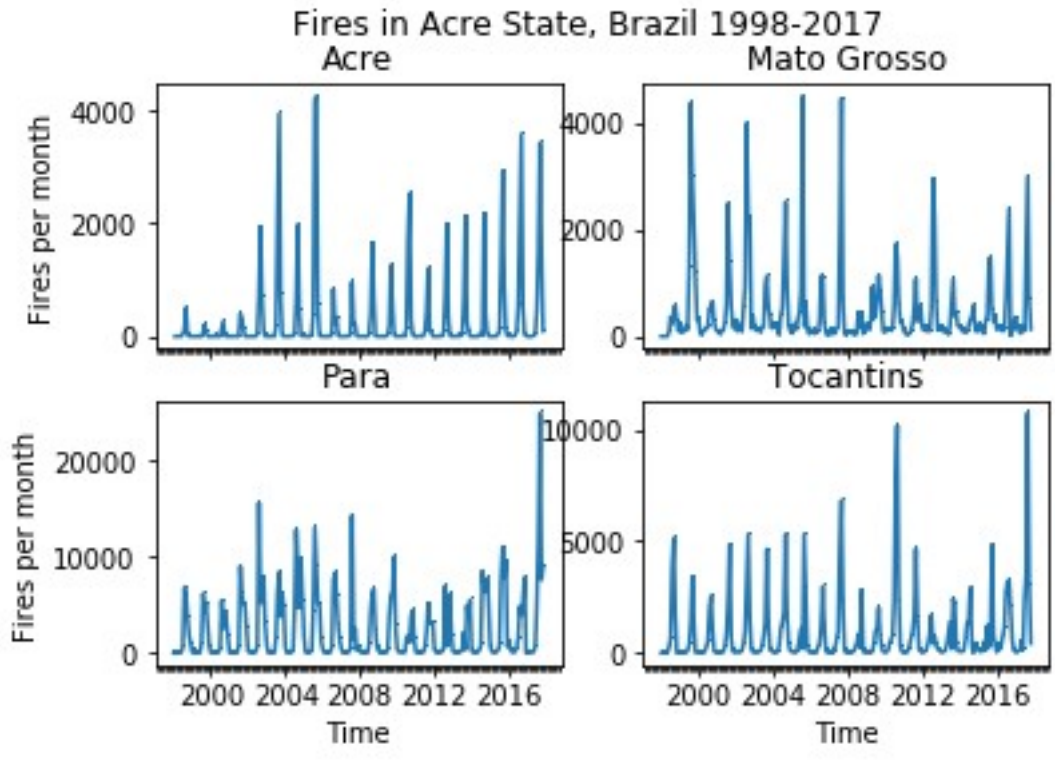
You can specify that the axes share the x-axis and y-axis, using the arguments

- `sharex=True` the x-axis will be shared
- `sharey=True` the y-axis will be shared

For example, the following code displays date plots for 4 regions in Brazil in a subplot grid using the same x-axis for the years:

```
# create a figure and an axis object
fig, ax = plt.subplots(2,2,sharex=True)

# format the ticks
months = mdates.MonthLocator()  # every month
ax[0,0].xaxis.set_minor_locator(months)

# set the title
fig.suptitle("Fires in Acre State, Brazil 1998-2017")

# date plot for each state
ax[0,0].set_title("Acre")
ax[0,0].set_ylabel("Fires per month")
ax[0,0].plot_date(data_by_state["Acre"][0], data_by_state["Acre"][1], marker = ",", linestyle="-")

ax[0,1].set_title("Mato Grosso")
ax[0,1].plot_date(data_by_state["Mato Grosso"][0], data_by_state["Mato Grosso"][1], marker = ",", linestyle="-")

ax[1,0].set_title("Para")
ax[1,0].set_xlabel("Time")
ax[1,0].set_ylabel("Fires per month")
ax[1,0].plot_date(data_by_state["Para"][0], data_by_state["Para"][1], marker = ",", linestyle="-")

ax[1,1].set_title("Tocantins")
ax[1,1].set_xlabel("Time")
ax[1,1].plot_date(data_by_state["Tocantins"][0], data_by_state["Tocantins"][1], marker = ",", linestyle="-")

# show the plots
plt.show()

fig.savefig('amazon_date_plot.png', bbox_inches='tight')
```

Here's the graph:



Example 12: Multiple Axes in a Figure, Alternative Syntax

There is an alternative syntax for creating subplots using the figure method

`add_subplot(nrows,ncols,index)`

where `rows` is the number of rows of the subplot grid
`ncols` is the columns of the subplot grid
and `index` is the index of the current plot, starting from 1

1	2
3	4
5	6

The following code uses this approach to create a date plot for the fires in each state:

```
# create a figure
fig = plt.figure(figsize=(15,15))

# create a new empty dictionary
# the dictionary keys are the state names
# the values are the list of fires in that state
data_by_state = {}
i = 1 # index for the subplots

# format the ticks
months = mdates.MonthLocator()  # every month

# set the title
fig.suptitle("Fires in Brazil 1998-2017")

# for each state in the list of states
for state in states:
    # insert the state along with the (list of dates and list of fires) for that state
    data_by_state[state] = [ value[0] for key,value in data.items() if key[0] == state ],[ value[1] for key,value in data.items() if key[0] == state ]

    # date plot for each state
    ax = fig.add_subplot(6,4,i)
    ax.plot_date(data_by_state[state][0], data_by_state[state][1], marker = ",", linestyle="-")

    i += 1

# show the plots
plt.show()

fig.savefig('amazon_date_plots.png', bbox_inches='tight')
```

The code

```
fig = plt.figure(figsize=(15,15))
```

creates a figure with dimensions 15 inches by 15 inches.

Then each subplot is added using:

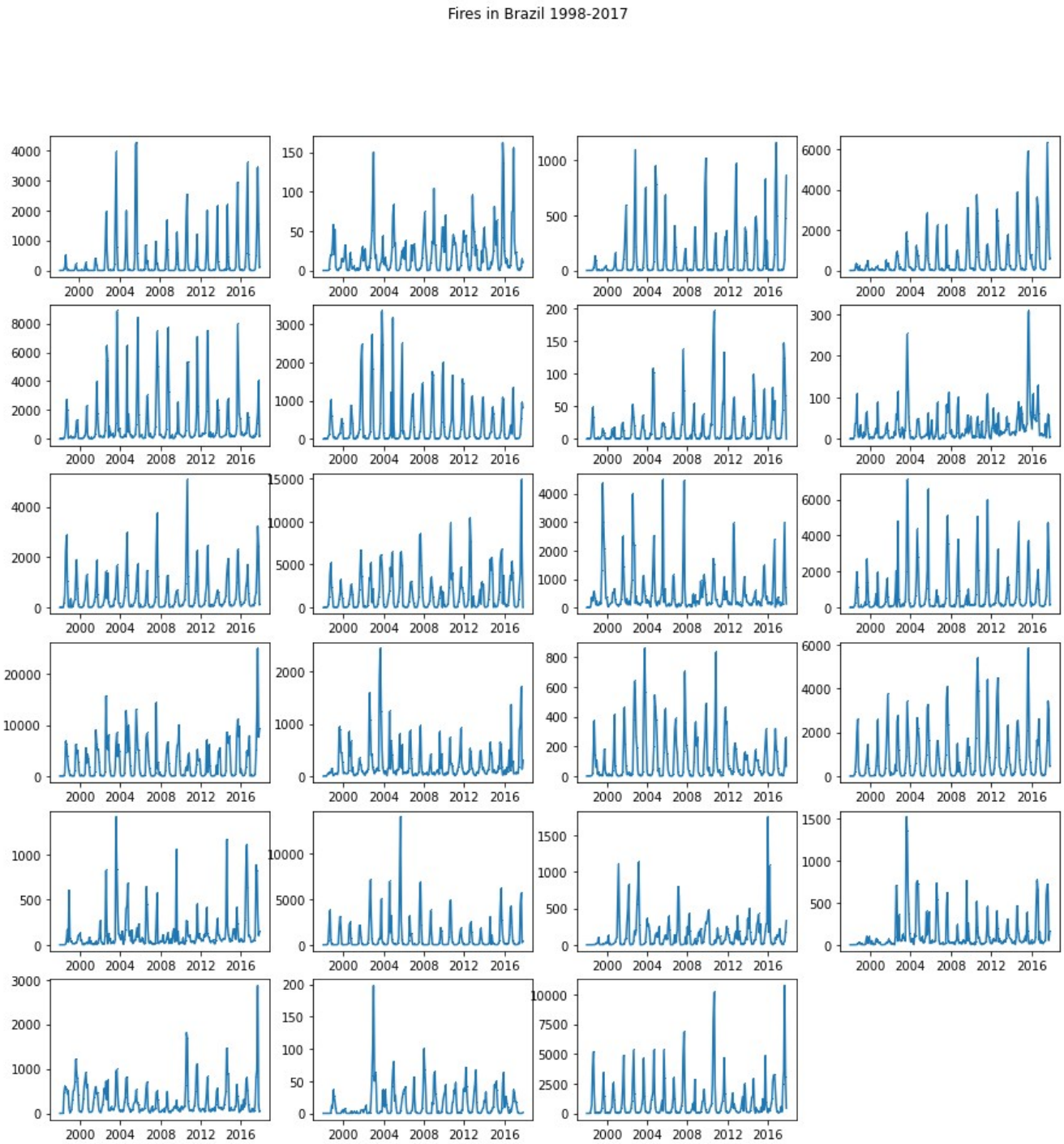
```
ax = fig.add_subplot(6,4,i)
```

where 6,4 specifies a subplot grid of 6 rows and 4 columns, and i is the index of the current subplot.

Each subplot is drawn using the same `Axes` variable `ax`:

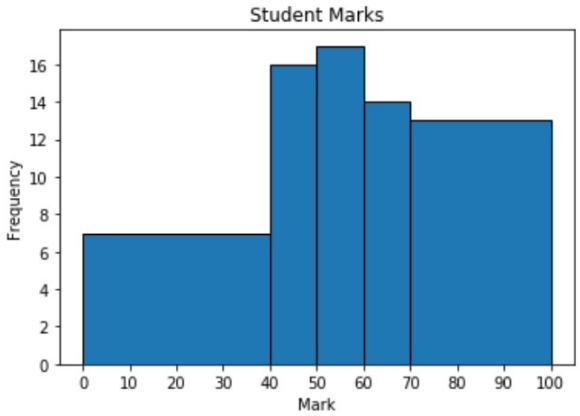
```
ax.plot_date(data_by_state[state][0]), data_by_state[state][1],
                                    marker = ",", linestyle="-")
```

Here are the graphs:

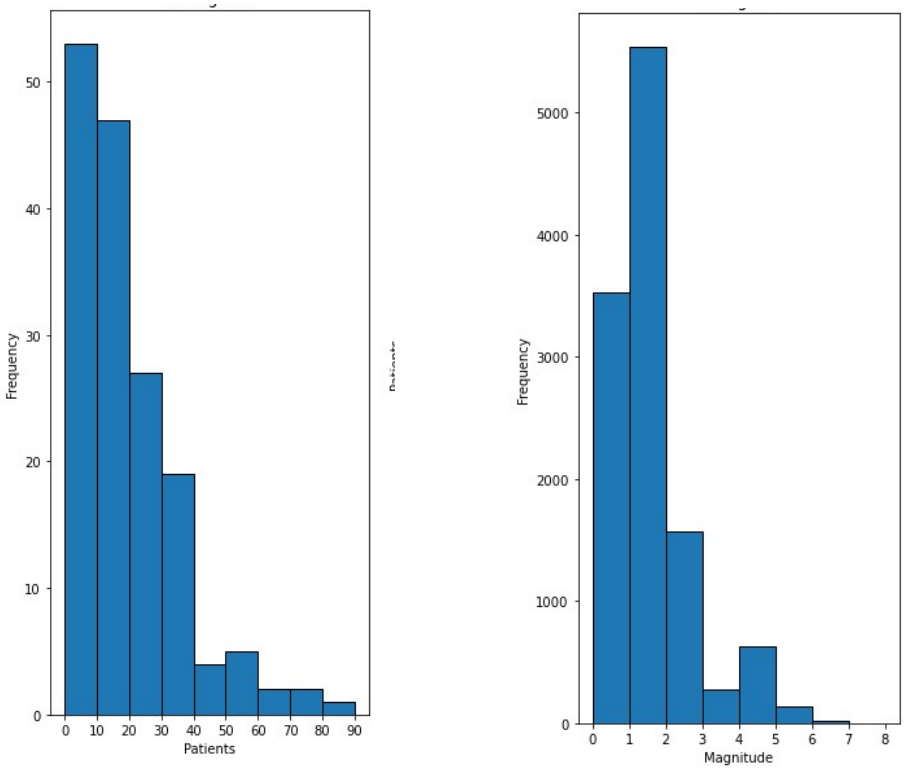


Matplotlib Documentation:
https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.figure.Figure.html#matplotlib.figure.Figure.add_subplot

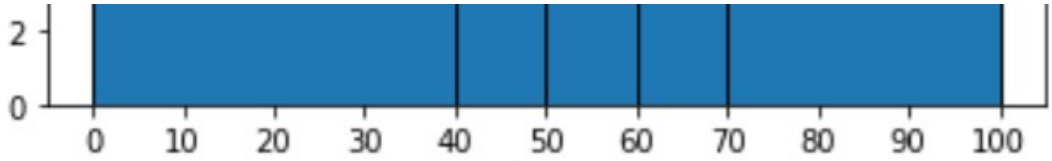
Example 13: Histogram of the Supercomputer Total Cores
A histogram is used to Display the frequency distribution of numerical data.



You need to divide the range of values into “bins”. These are the intervals corresponding to the vertical bars in the histogram. If the bins are equal width, the heights of the bars represents the frequencies:



If the bins are not of equal width, e.g. 0-<40, 40-<50, 50-<60, 60-<70, 70-100, the bars represent the frequency density:



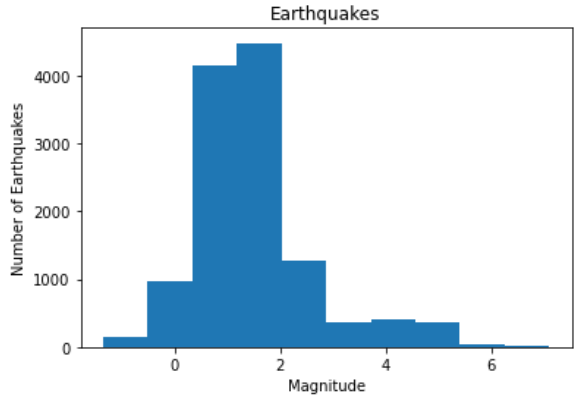
To create a histogram, use the Axes method `hist()`, which takes a parameter representing the values.

The `hist()` method will allocate the values into default bins and display the histogram.

For example, the following program displays histogram of earthquake magnitudes:

```
# Import the matplotlib
import matplotlib.pyplot as plt

# Create an empty dictionary called earthquakes
magnitudes = []
# Open the file
with open("earthquakes_2019.csv") as data_file:
    # Read in the headers line
    _ = data_file.readline()
    # For each line in the file
    for line in data_file:
        # Split the line into
        date_string, latitude, longitude, magnitude, place = line.split(",",maxsplit=4)

        # Add the magnitude value to the list
        magnitudes.append(float(magnitude))

# Create the figure and axes
fig, ax = plt.subplots()

# Set the title
ax.set_title("Earthquakes Histogram")

# set the axis labels
ax.set_xlabel("Magnitude")
ax.set_ylabel("Number of Earthquakes")

# display the histogram
ax.hist(magnitudes)

# show the figure (not required with Spyder)
plt.show()
```

Visualisation



By default, matplotlib will allocate the values into 10 “bins”. This is equivalent to the following:

```
# display the histogram, specifying the number of bins (intervals)
ax.hist(magnitudes, bins=10)
```

This uses the keyword argument `bins` to specify the number of bins. You can provide any integer value.

Alternatively, you can specify the bins using a list, e.g. `ax.hist(values, bins=bins_list)`

In the following example, the histogram uses the following intervals: 0, 0.5, 1.0, 1.5, and so on, up to 7.5 (to include the maximum magnitude, 7.1).

```
# Import the matplotlib
import matplotlib.pyplot as plt

# Create an empty dictionary called earthquakes
magnitudes = []
# Open the file
with open("earthquakes_2019.csv") as data_file:
    # Read in the headers line
    _ = data_file.readline()
    # For each line in the file
    for line in data_file:
        # Split the line into
        date_string, latitude, longitude, magnitude, place = line.split(",",maxsplit=4)

        # Add the magnitude value to the list
        magnitudes.append(float(magnitude))

# Create the figure and axes
fig, ax = plt.subplots()

# Set the title
ax.set_title("Earthquakes Histogram")
# set the axis labels
ax.set_xlabel("Magnitude")
ax.set_ylabel("Number of Earthquakes")

# specify the bins using a list
# this list will be 0, 0.5, 1.0, 1.5, ... and will include the maximum magnitude
bins_list = [ i/2 for i in range(int(max(magnitudes))*2+1)]

# set the tick marks on the x-axis to correspond with the bins
ax.set_xticks(bins_list)

# display the histogram, specifying the bins using a list
ax.hist(magnitudes, bins=bins_list, ec="black") # set the edge colour to black
```

The bins are generated using the list comprehension:

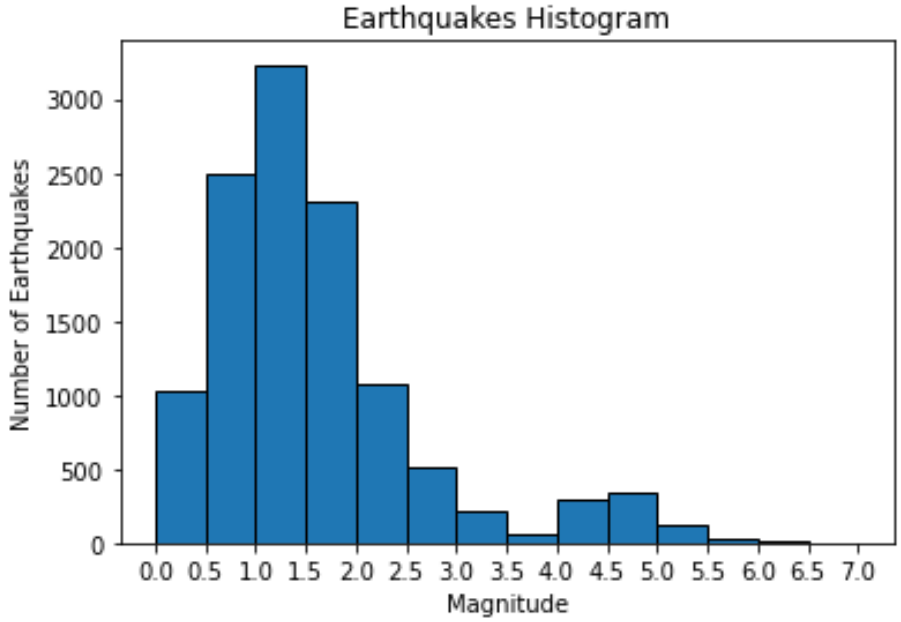
```
bins_list = [ i/2 for i in range(int(max(magnitudes))*2+1)]
```

The argument to `range` is `int(max(magnitudes))*2+1`
which in this case evaluates as `int(7.1)*2+1 = 7*2+1 = 15`
`range` then provides the sequence `0, 1, 2, 3, 4, ..., 15`
The list comprehension then takes each value in the sequence and divides it by 2, yielding
`0, 0.5, 1, 1,5, ..., 7.5`

The program uses the `bins_list` to set the tick marks on the x-axis to correspond with the bins:
`ax.set_xticks(bins_list)`

Finally, the keyword argument ec (edge colour) is used to colour the bars of the histogram black:
`ax.hist(magnitudes, bins=bins_list, ec="black")`

Visualisation

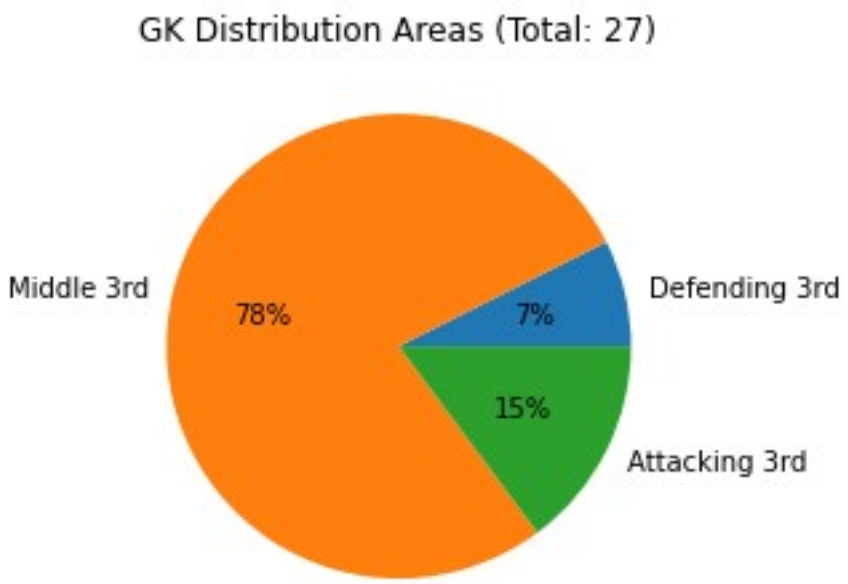


Example 14: Specifying alternative colours for a Pie Chart

The following program uses a Pie Chart to visualise the distribution areas for a Goalkeeper (i.e. the areas of the pitch where the ball was delivered to) in an international football game:

```
# program to display a pie chart of the distribution areas for an international GK
# Uses the default colours for the pie chart segments

import matplotlib.pyplot as plt

# hard-coded values for the distribution areas
areas = {"Defending 3rd":2, 'Middle 3rd': 21, "Attacking 3rd": 4}

# create the figure and axes
fig,ax = plt.subplots()

# set a title which includes the total of the values from the dictionary
ax.set_title("GK Distribution Areas (Total: " + str(sum(areas.values())) + ")")

# display the pie chart using the
ax.pie(areas.values(), labels=areas.keys(), autopct="%.0f%%")

# display the figure (not needed with Spyder)
plt.show()
```

Visualisation



The program uses matplotlib's default colours for the pie chart segments.

This version of the program specifies alternative colours to use:

```
# Program to display a pie chart of the distribution areas for an international GK
# Uses a list to specify the colours for the pie chart segments

import matplotlib.pyplot as plt

# hard-coded values for the distribution areas
areas = {"Defending 3rd":2, 'Middle 3rd': 21, "Attacking 3rd": 4}

# create the figure and axes
fig,ax = plt.subplots()

# set a title which includes the total of the values from the dictionary
ax.set_title("GK Distribution Areas (Total: " + str(sum(areas.values())) + ")")

# specify alternative colours for the pie chart segments
colours = ["lightgreen", "yellowgreen", "green"]

# display the pie chart using the
ax.pie(areas.values(), labels=areas.keys(), autopct="%.0f%%", colors=colours)

# display the figure (not needed with Spyder)
plt.show()
```

The `colours` list `["lightgreen", "yellowgreen", "green"]` uses named colours understood by matplotlib. https://matplotlib.org/stable/gallery/color/named_colors.html

The keyword argument `colors` is used to specify the alternative colours. Note the American spelling.

```
# program to display a pie chart of the distribution areas for an international GK
# Uses a list to specify the colours for the pie chart segments

import matplotlib.pyplot as plt
colours = ["lightgreen", "yellowgreen", "green"]

areas = {"Defending 3rd":2, 'Middle 3rd': 21, "Attacking 3rd": 4}

# create the figure and axes
fig,ax = plt.subplots()

# set a title which includes the total of the values from the dictionary
ax.set_title("GK Distribution Areas (Total: " + str(sum(areas.values())) + ")")

# display the pie chart using the
ax.pie(areas.values(), labels=areas.keys(), autopct="%.0f%%", colors=colours)

# display the figure (not needed with Spyder)
plt.show()
```