CMM201 Week 11: Plotting Images in Python

1 Plotting Data in Python

1.1 Aims of the Lecture

- Learn how to do basic plots using *numpy arrays* and the *matplotlib* package

1.2 Additional Reading

- Matplotlib documentation
- Matplotlib tutorial
- Matplotlib pie chart tutorial

1.3 Using Artificial Data

- First, we will install the *matplotlib* package. Maybe you already have it from the "images" lecture (week 9)

```
In [ ]: !pip install matplotlib
```

- The function that we will use from *matplotlib* is called **pyplot**.

```
In [ ]: import matplotlib.pyplot as plt
```

1.3.1 Line Plots

- You can create a line plot by defining a list

- By default, the *x*-axis will have integer values starting in 0.

```
In [ ]: plt.plot([3,6,-1,7])
```

- You can define two lists and then plot them (both have to be the same size).

```
In [ ]: plt.plot([0,1,2,3,4],[0,2,4,6,8])
```

- By default plots are made with a blue solid line, but this can be changed:

```
In [ ]:     # You can use the following:
             # blue (b), red (r), green (g)...
             # line (-), square (s), dash (--), triangle (^)...
             plt.plot([0,1,2,3,4],[0,2,4,6,8],'gs')
```

- Adding *x*- and *y*-axis labels

```
In [ ]:     plt.plot([0,1,2,3,4],[0,2,4,6,8])
            plt.xlabel('Number of shoes')
            plt.ylabel('Price')
```

- By using *plt.axis()*, you can set the minimum and maximum values of the *x*- and *y*-axis:

```
In [ ]:     plt.plot([0,1,2,3,4],[0,2,4,6,8])
            plt.axis([0,7,-9,21])
```

- You can also use *numpy* arrays to plot values.
- This is better than using lists, as you can do calculations with the values.

```
In [ ]: import numpy as np
        t=np.arange(0,5,0.2)
        plt.plot(t,t,'r--',t,t**2,'b^',t,t**3,'gs')
```

1.3.2 Scatter Plots

- Scatter plots are used to plot data along the coordinate plane.
- These are really useful when you want to analyse data trends.
- To test scatterplots, we will create a dictionary with four ranges of numbers by using *numpy* and the *random* function.

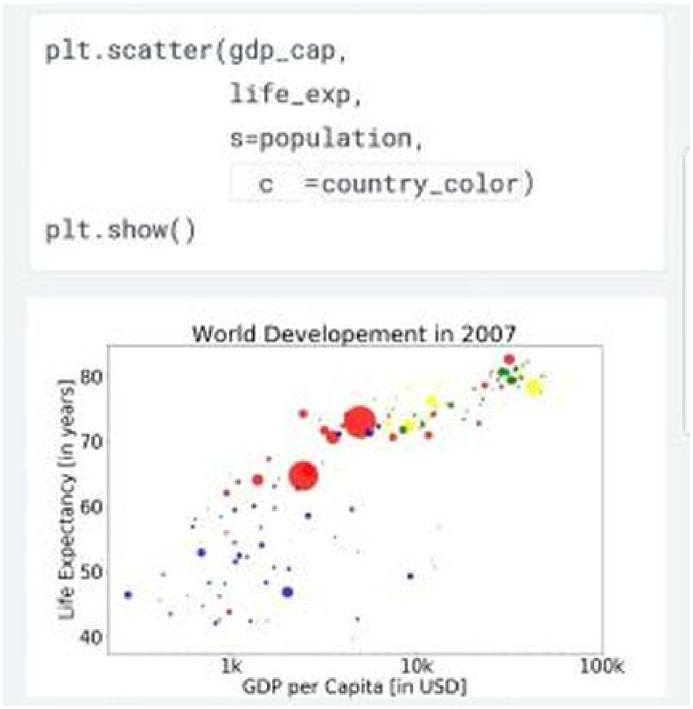
```
In [ ]: data={'a':np.arange(10),
        'b':np.arange(10)+10*np.random.randn(10),
        'c':np.random.randint(0,50,10),
        'd':np.abs(np.random.randn(10))*100}
        print(data)
```

- Now we can plot the data using
  - *a* as the x-axis values
  - *b* as the y-axis values
  - *c* as different colours for the data (these colours are random according to the number)
  - *d* as different sizes/weights

```
In [ ]: plt.scatter(data['a'],data['b'],c=data['c'],s=data['d'])
```

```
In [ ]: # Another option is to use only the keys
        # and specify the dictionary as the data to be used
        plt.scatter('a','b',c='c',s='d',data=data)
```

Can you think of any "real life" example of data where you can use this?



```
plt.scatter(gdp_cap,
            life_exp,
            s=population,
             c  =country_color)
plt.show()
```

Fig 1. An example of a scatter plot

1.3.3 Categorical Data

- Using two lists, you can produce charts with categorical data

```
In [ ]: names=["Dingos","Wild Cats","Tigers"]
        values=[1,11,111]
        plt.figure()
        plt.bar(names,values) # Create a bar chart
        plt.show() # show the figure
```

- Pie chart

```
In [ ]: plt.pie(values,labels=names)
```

```
In [ ]: c = ['gold', 'yellowgreen', 'lightcoral']
        e = (0, 0, 0.1)  # separate third slice
        plt.pie(values,labels=names,explode=e,colors=c)
```

1.4 Plotting the IRIS Dataset

- Once again, we will load the IRIS dataset and save all the contents in different variables:
- The IRIS dataset has four value columns (sepal/petal-length/width) and a class/target.

```
In [ ]: ## Load iris dataset
        from sklearn import datasets
        iris = datasets.load_iris()
```

```
data = iris['data']
header = iris['feature_names']
target = iris['target']
target_names = iris['target_names']
```

- Tally the target/class to see how many samples of each plant

```
In [ ]: unique_elements, counts_elements = np.unique(target, return_counts=True)
        print(unique_elements, counts_elements)
```

```
In [ ]: # Plot a bar chart with the tally
        plt.bar(unique_elements,counts_elements)
```

```
In [ ]: # Using the target names instead
        plt.bar(target_names,counts_elements)
```

- Plotting variables against each other (scatter)

```
In [ ]: plt.scatter(iris['data'][:,0],iris['data'][:,1])
        plt.xlabel(iris['feature_names'][0])
        plt.ylabel(iris['feature_names'][1])
```

- Using the target as a colour differentiator

```
In [ ]: plt.scatter(iris['data'][:,0],iris['data'][:,1],c=iris['target'])
        plt.xlabel(iris['feature_names'][0])
        plt.ylabel(iris['feature_names'][1])
```

- Using a third variable as a size differentiator

```
In [ ]: plt.scatter(iris['data'][:,0],iris['data'][:,1],c=iris['target'],s=iris['data'][2])
        plt.xlabel(iris['feature_names'][0])
        plt.ylabel(iris['feature_names'][1])
```

- 3D Plots

```
In [ ]: from mpl_toolkits.mplot3d import Axes3D
        fig = plt.figure(1, figsize=(4, 3))
        ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
        for name, label in [('Setosa', 0),('Versicolor', 1),('Virginica', 2)]:
            ax.text3D(data[target == label, 3].mean(),
                      data[target == label, 0].mean(),
                      data[target == label, 2].mean() + 2, name,
                      horizontalalignment='center',
                      bbox=dict(alpha=.2, edgecolor='w', facecolor='w'))
        ax.scatter(data[:, 3],data[:, 0],data[:, 2],c=target.astype(np.float),edgecolor='k')
        ax.w_xaxis.set_ticklabels([])
        ax.w_yaxis.set_ticklabels([])
        ax.w_zaxis.set_ticklabels([])
        ax.set_xlabel('Petal width')
ax.set_ylabel('Sepal length')
ax.set_zlabel('Petal length')
ax.set_title('Iris Dataset')
ax.dist = 12
plt.show()
```