Chapter 5 AI Technologies and Data Analytics

1. AI in the Lives of Today's Youth

- As children grow up using AI applications like Siri, Alexa, and Google Assistant, AI becomes second nature to them.
- They trust and rely on AI more deeply than we might expect. In some cases, their reliance on AI is so ingrained that they seldom question its processes or outcomes.
- **Key Point**: It is critical to teach young students not only how to use AI but also how it works. This demystifies the "black box" of AI, helping them understand that AI is not inherently objective or perfect.

2. Challenges in Teaching AI and Machine Learning (ML) to Younger Students

- When discussing AI with educators, two common challenges arise:
    1. **Abstraction of Large Datasets**: Young learners often find it difficult to grasp the scale of datasets required for training AI models.
    2. **Lack of Hands-On Examples**: It's challenging to teach concepts like bias and data quality without tangible, interactive experiences.

Solution: Tools like **Google's Teachable Machine** offer a hands-on experience, allowing students to interact with ML models in real-time, making abstract concepts more accessible.

3. The Unplugged Approach: Parallel Learning with Students and Machines

- In a unique teaching method, we engage students in a classification task where they and the AI "learn" simultaneously.
- **Unplugged Game**:
    - Students are shown two sets of images (cats and dogs).
    - Their task is to use **pattern matching** to identify common physical traits that signify cats and dogs, constructing a set of classification rules (e.g., a decision tree).
    - **Outcome**: This allows them to think critically about how humans generalize patterns based on logical reasoning.

4. Using Teachable Machine to Train AI Models

- Once students have devised their own classification rules, they apply the same task to Teachable Machine.
- By uploading data and training a model, students observe how machine learning algorithms process information. This provides a tangible and interactive way for them to see AI in action.
- **Excitement and Empowerment**: Students feel like they are "training their own AI model," creating a sense of ownership and enthusiasm.

5. Human vs Machine: Classification Results

- Surprisingly, the machine sometimes misclassifies a cat as a dog.
- **Discussion**:
    - This error opens up discussions on **AI subjectivity**. The machine, unlike the students, struggles with the small sample size and fails to generalize as well as human reasoning does.
    - **Key Insight**: Machine learning is not as objective as it seems. The quality and representativeness of the input data play a critical role in determining the outcome of the model.

6. Real-World Applications and Bias in AI

- To extend the lesson, students explore real-world examples where data quality and bias affect outcomes. A common example is **facial recognition** technology.
- Students are encouraged to brainstorm positive-impact AI applications:
    - A plant identification app.
    - Tumor identification software to assist doctors.

These discussions highlight the transformative potential of AI while reinforcing the importance of data quality and algorithmic fairness.

Step-by-Step Guide to Install a Batch Image Downloader Extension in Chrome

Step 1: Open Google Chrome
Make sure you are using Google Chrome on your computer.

Step 2: Open Chrome Web Store
1. In the Chrome browser, navigate to the Chrome Web Store by entering this URL in the address bar: https://chrome.google.com/webstore.
2. This is where you can find all the extensions available for Chrome.

Step 3: Search for an Image Downloader Extension
1. In the Chrome Web Store, there is a search bar at the top left.
2. Type "Image Downloader" or "Batch Image Downloader" into the search bar and press Enter.

Some recommended extensions for downloading images in bulk include:
- Image Downloader
- Fatkun Batch Download Image

Step 4: Install the Extension
1. In the list of results, find the extension you want to use (e.g., Image Downloader or Fatkun Batch Download Image).
2. Click on the extension name to view details about it.
3. On the extension's page, click the Add to Chrome button in the top right corner.

Step 5: Confirm Installation
1. A pop-up will appear asking you to confirm the installation of the extension.
2. Click Add Extension to proceed.
3. Chrome will now download and install the extension. Once installed, you'll see the extension's icon in the top-right corner of your browser (next to the address bar).

Step 6: Use the Extension to Download Images in Bulk
1. Go to Google Images or any webpage where you want to download images.
   - For example, search for "poisonous fish" in Google Images.
2. Click the extension's icon in the top-right corner of Chrome (next to the address bar).
3. The extension will scan the page and list all the images it has found.
4. Select the images you want to download, or use the Select All option if you want to download all the images.
5. Click the Download or similar button to save all the selected images to your computer.

Depending on the extension, you might be prompted to choose a download location for the images, or they may automatically be saved to your default downloads folder.

Step 7: Check Your Downloaded Images
- Once the download is complete, check your Downloads folder or the specified directory to ensure all the images have been saved correctly.

Step-by-Step Guide for Classifying Cats and Dogs Using Teachable Machine

---

Step 1: Downloading Animal Images in Bulk

To train the Teachable Machine model, you'll need a set of labeled images for each animal class (e.g., cats and dogs). Here's how you can download these images in bulk:

A. Download Images Using a Chrome Extension

1. Install a Batch Image Downloader Extension:
    - Go to the Chrome Web Store at chrome.google.com/webstore.
    - Search for "Image Downloader" or "Fatkun Batch Download Image".
    - Click on Add to Chrome for one of the extensions and confirm the installation.
2. Search for Images in Google Images:
    - Open Google Images and search for the first animal class, such as "cats".
    - Scroll down to load more images to increase the size of the dataset.
3. Use the Extension to Download Images:
    - Click on the extension icon next to the Chrome address bar.
    - Select the images you want to download or click Select All to grab all visible images.
    - Click Download, and the images will be saved to your computer.

Repeat the above steps for the second class (e.g., "dogs").

B. Organize the Images:

- Create a folder for each class of images, such as:
    - Cats/
    - Dogs/

Save the downloaded images into their respective folders for easier upload to Teachable Machine.

Step 2: Building the Model Using Teachable Machine

1. Open Teachable Machine:
    - Go to Teachable Machine in your browser.
2. Create a New Project:
    - On the homepage, click Get Started.
    - Select the Image Project option (since you're working with image classification).
3. Set Up Your Classes:
    - You'll see two default classes on the left, labeled as "Class 1" and "Class 2".
    - Rename the classes to "Cats" and "Dogs" (or the animals you are classifying).
4. Upload Training Data (Images):
    - Click Upload under each class to upload your pre-downloaded images.
    - For "Cats", upload all the images in the Cats/ folder.
    - For "Dogs", upload all the images in the Dogs/ folder.

You can add more classes if you want to classify additional animals.

5. Adjust Training Settings (Optional):
    - By default, Teachable Machine will use its recommended settings. However, you can explore Advanced Settings if you want to tweak things like Epochs or Batch Size.
    - Usually, the default settings work well for most basic classification tasks.

Step 3: Training the Model

1. Train the Model:
    - Once you have uploaded your images, click Train Model.
    - The training process might take a few seconds to a couple of minutes, depending on the number of images.

During this time, Teachable Machine will use your images to train a machine learning model that can distinguish between the animal classes (e.g., cats and dogs).

2. Wait for the Training to Complete:
    - Once training is complete, you will be able to test your model directly in the browser.

Step 4: Testing the Model

1. Upload Test Images or Use the Webcam:
    - You can test the model with additional images of cats and dogs to see how well it performs.
    - Either upload test images or use your webcam to show the model real-time images of a

cat or a dog (or another animal).

2. Check the Model's Predictions:
    - The model will provide a confidence score indicating how sure it is that the image belongs to a certain class (e.g., how sure it is that the image is of a cat or a dog).

Step 5: Exporting or Sharing the Model

1. Export the Model:
    - After testing, you can export your trained model.
    - Teachable Machine provides options to export your model as:
        - TensorFlow.js (to use it on websites).
        - TensorFlow Lite (to use it on mobile devices).
        - Download Model (to use it offline).
2. Share the Model (Optional):
    - You can also generate a link to share the model so that others can try it in their browser.

Tips for Improving Model Accuracy

- Increase Image Variety: Ensure you have a variety of images for each class (e.g., different breeds of dogs, different lighting conditions, and angles).
- More Data: The more images you upload for each class, the better the model's accuracy.
- Clean Data: Make sure the images clearly represent the category you're classifying. Remove irrelevant or low-quality images.

Data visualization refers to the representation of data through use of graphics. Making informative visualization is an important task in data analysis, no matter as a part of the exploratory process or as a way of generating ideas for models.

Python has many add-on libraries for making static or dynamic visualizations. In this chapter, we will introduce a library called **matplotlib** which stands for mathematics-plot-library and the techniques of making various graphs for presenting statistical data and also the result of artificial intelligence.

In fact, matplotlib is a desktop plotting package designed for creating publication quality plots. There are many modules under it including **pyplot**. We usually import it by:

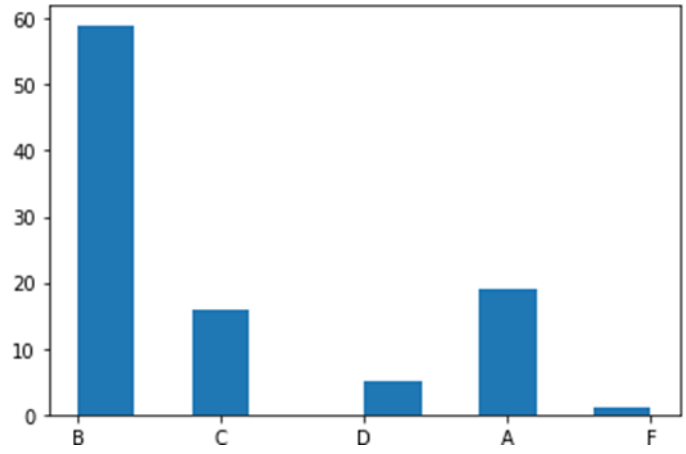
```
import matplotlib.pyplot as plt
```

Consider a set of one dimensional data. The values can be either numerical values (e.g. marks of students) or non-numerical values (e.g. letter grades of students). **Frequency plot** is based on the number of occurrence of each unique value. Let's use an example to illustrate different plots. The file "ama1234.csv" contains the marks and letter grades of 100 students. We can first read the file as a single DataFrame, then extract the two columns as two Series.

```
import pandas as pd
result = pd.read_csv("ama1234.csv")
marks = result["marks"]
grades = result["grades"]
```

Based on the grades, we can directly plot a **histogram** using `hist()` to show the frequency of students obtaining each grades. The `show()` method displays the plot.

```
plt.hist(grades)
plt.show()
```

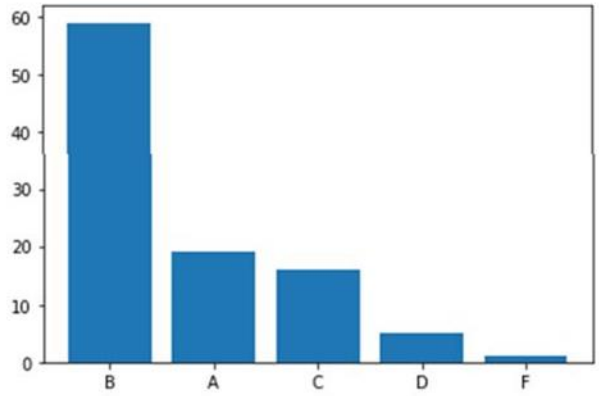


In the previous chapter, we have introduced the `value_counts()` method in Pandas which gives a frequency table of a set of data by counting the frequency of each unique value. The result is a Series with the index being each unique value and the values being the frequency of each unique value. We might first store it as two arrays:

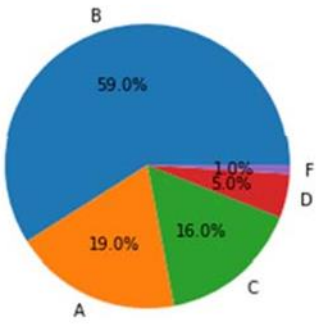
```
x = grades.value_counts().index
y = grades.value_counts().values
```

Using these two arrays x(index) and y(values), we can plot a **bar chart** or a **pie chart** to illustrate frequency of the set of data. For bar chart, two arrays are required for the x-axis and y-axis. For pie chart, only an array of value is necessary. We might also put the labelling and auto-percentage optionally.

```
plt.bar(x,y)
plt.show()
```




```
plt.pie(y,labels=x,autopct='%1.1f%%')
plt.show()
```



If we look at the Series of marks in the example problem, we can see that the entries are float point numerical values ranged between 0 to 100, corrected to 1 decimal point. It would be unrealistic to make a frequency plot of each unique mark. Instead, we might group similar marks together. In statistics, **data binning** is a way to group numbers of more-or-less continuous values into a smaller number of "bins". After binning, we might plot a histogram or a density plot based on the frequency of binned values.

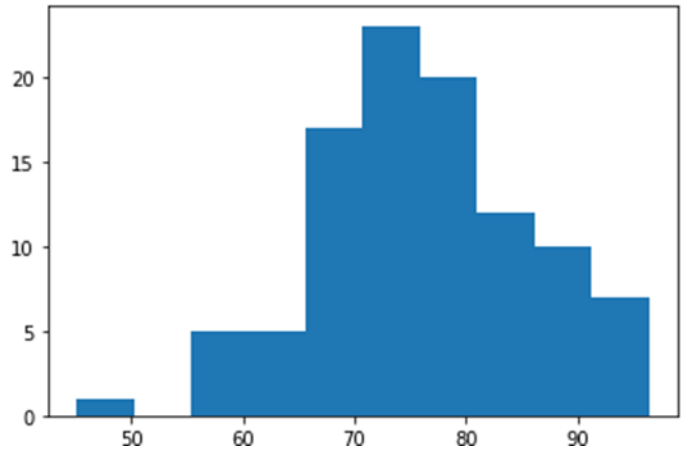
In pyplot, when we plot a histogram of an array of values from a continuous variable, it will be automatically binned. We can also specify the binning criteria by the number of even width intervals, or by the end points between intervals by the following syntax respectively:

```
plt.hist(data_set, bins=number_of_bins)

plt.hist(data_set, bins=[point_0,point_1,...,point_n])
```

```
plt.hist(marks)
```

```
(array([ 1.,  0.,  5.,  5., 17., 23., 20., 12., 10.,  7.]),
 array([45.1 , 50.23, 55.36, 60.49, 65.62, 70.75, 75.88, 81.01, 86.14,
        91.27, 96.4 ]),
 <a list of 10 Patch objects>)
```



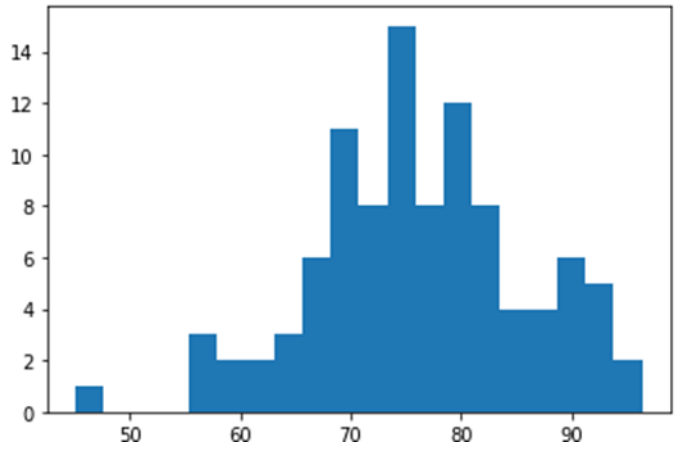
In the example, without putting any parameter, the data is automatically put into 10 bins of equal width

along the real number line. The end points of the intervals and the frequency of data in each interval are shown in the array.

Suppose we would like to use 20 bins with narrower interval, simply set the parameter 20.

```
plt.hist(marks,bins=20)
```

```
(array([ 1.,  0.,  0.,  0.,  3.,  2.,  2.,  3.,  6., 11.,  8., 15.,  8.,
        12.,  8.,  4.,  4.,  6.,  5.,  2.]),
 array([45.1  , 47.665, 50.23 , 52.795, 55.36 , 57.925, 60.49 , 63.055,
        65.62 , 68.185, 70.75 , 73.315, 75.88 , 78.445, 81.01 , 83.575,
        86.14 , 88.705, 91.27 , 93.835, 96.4  ]),
 <a list of 20 Patch objects>)
```



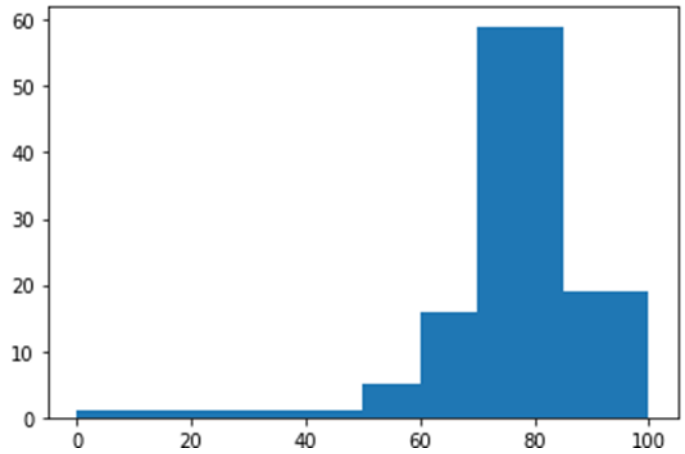
Suppose the rubric of ama1234 is as follows:

A: [85,100], *B*: [70,85], *C*: [60,70], *D*: [50,59], *F*: [0,50]

We may form an array with the threshold marks together with 0 mark and full mark (100) to divide the interval for data binning. Notice that the width of each interval is not even. This is illustrated by width of rectangles on the histogram.

```
plt.hist(marks,bins=[0,50,60,70,85,100])
```

```
(array([ 1.,  5., 16., 59., 19.]),
 array([  0,  50,  60,  70,  85, 100]),
 <a list of 5 Patch objects>)
```



Some set of data is dependent to a continuous variable. For example, stock price is dependent on the time variable. To present such data, we may use a **line graph**. This is good for showing the trend and local extreme values. As an example, we will try to study the stock price of two companies, Apple (AAPL) and Microsoft (MSFT). The files "AAPL.csv" and "MSFT.csv" contains the historical data in 5 years. The column of adjusted close price is stored as a Series.

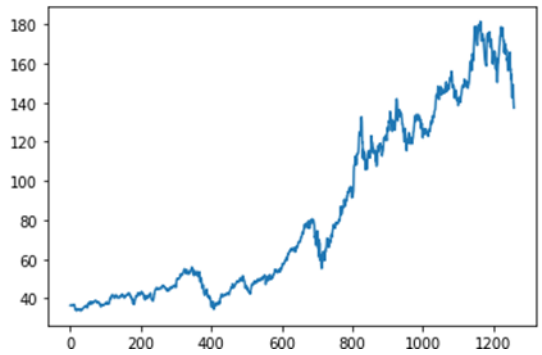
```
apple_data = pd.read_csv("AAPL.csv")
apple_price = apple_data["Adj Close"]
msft_data = pd.read_csv("MSFT.csv")
msft_price = msft_data["Adj Close"]
```

In pyplot, we can directly use `plot()` to make a line graph of a Series or array. Moreover, the colour and style of the line can be adjusted by the commands inside the brackets.

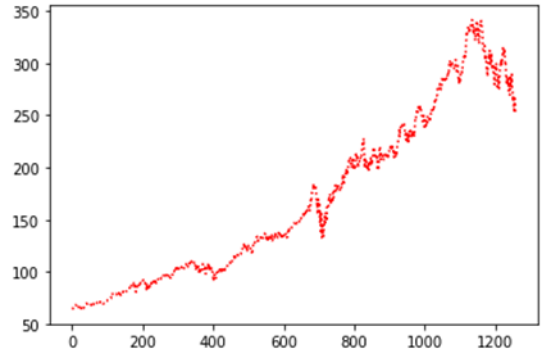
command	colour
`'k'`	black
`'g'`	green
`'r'`	red
`'b'`	blue
`'y'`	yellow

command	style
`'--'`	dashed line
`':'`	dotted line
`'*'`	points with stars
`'o'`	points with circle
`'+'`	points with plus sign

```
plt.plot(apple_price)
plt.show()
```

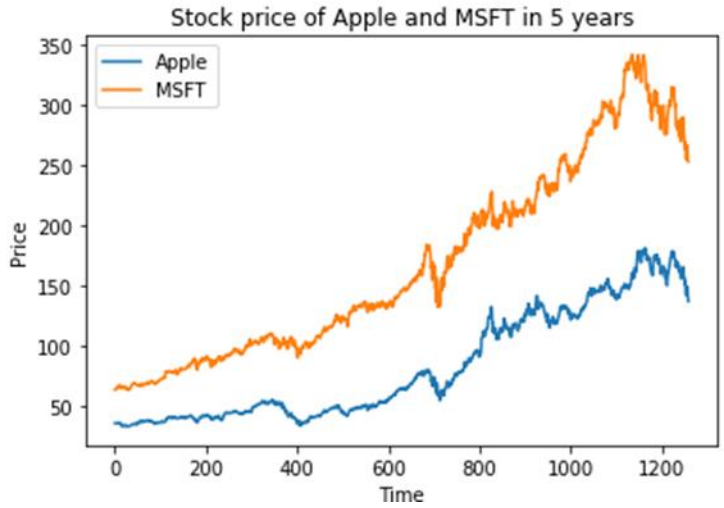



```
plt.plot(msft_price,'r:')
plt.show()
```



However, this is hard for comparison. In matplotlib, we can plot several lines on the same figure. To show information of the figure and distinguish the lines, we can also add title, labels and legend. This applies not only on line graph but also on the graphs we have introduced before. Upon executing the `show()` command, all these graphs and information before will be displayed on the same figure.

```
plt.plot(apple_price)
plt.plot(msft_price)
plt.xlabel("Time")
plt.ylabel("Price")
plt.title("Stock price of Apple and MSFT in 5 years")
plt.legend(["Apple","MSFT"])
plt.show()
```

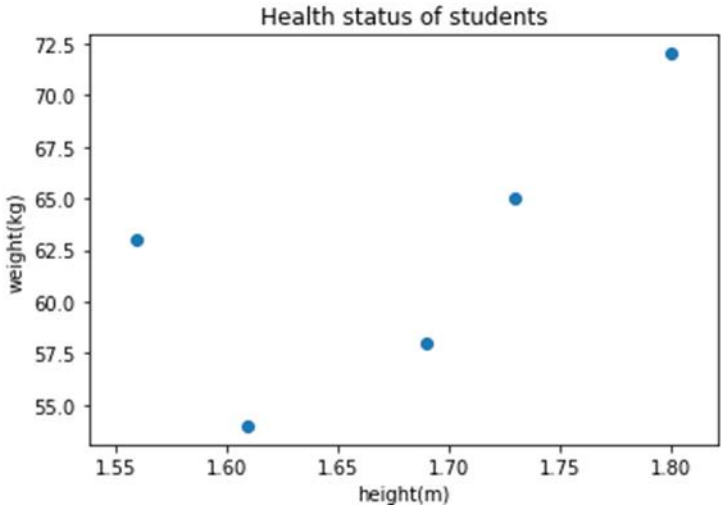


A set of data might consist of more than one variables. An example is the health data from the previous chapter which contains two variables height and weight. For such data, the purpose of visualization is to show the relationship between the two variables. This can be illustrated by a **scatter plot**. More details will be discussed in a later chapter related to correlation and regression.

For example, we might want to study the relation between height and weight. After reading the csv file "health.csv" into a DataFrame, extract the columns representing the two variables (height and weight) into two Series. For each data index, a point with x-coordinate being its height and y-coordinate being its weight is plot on the figure. As a result, a scatter plot should contain as many points as the number of rows of the original DataFrame.

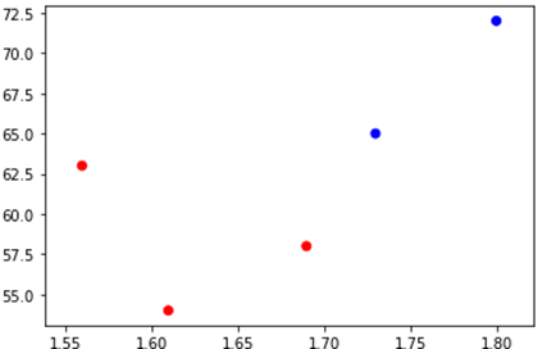
```
df1 = pd.read_csv("health.csv")
x = df1["height"]
y = df1["weight"]
```

```
plt.scatter(x,y)
plt.title("Health status of students")
plt.xlabel("height(m)")
plt.ylabel("weight(kg)")
plt.show()
```

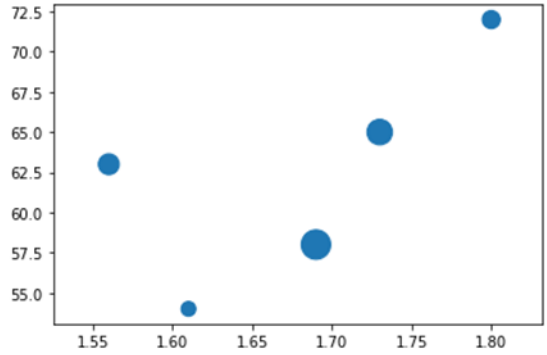


In case the data points can be classified into different categories (e.g. male and female), we might assign colours to each point with `c=[colour0,colour1,...]`. If the data points are weighted, we might show each point with a different size using `s=[size0,size1,...]`.

```
plt.scatter(x,y,c=['b','r','b','r','r'])
plt.show()
```




```
plt.scatter(x,y,s=[300,100,150,200,400])
plt.show()
```



References:

https://medium.com/readyai-org/teachable-machine-course-eb925c2c370a