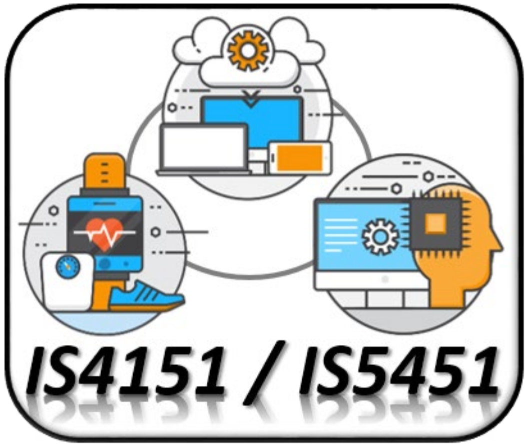


Lecture 9

IoT Data Preprocessing

IS4151/IS5451 – AIoT Solutions and Development
AY 2024/25 Semester 2

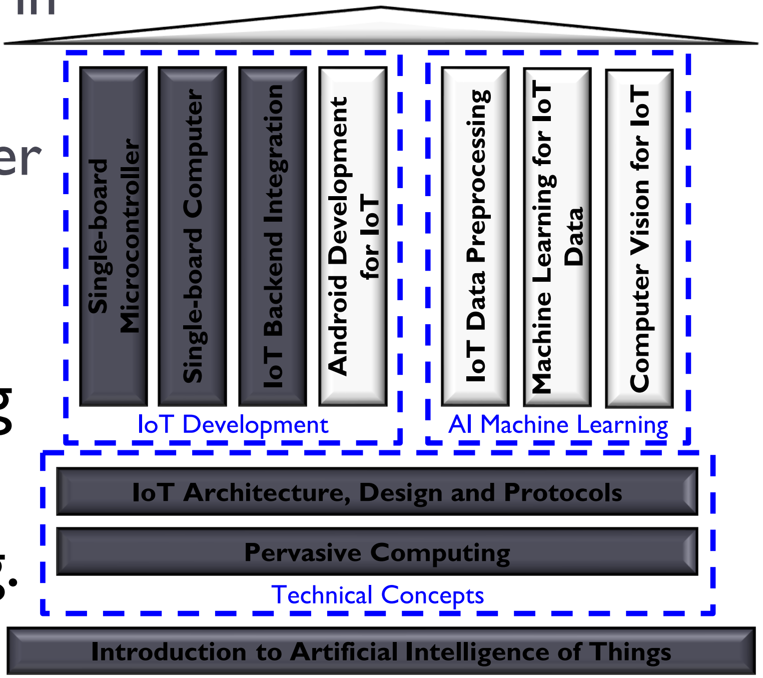
Lecturer: A/P TAN Wee Kek

Email: tanwk@comp.nus.edu.sg :: **Tel**: 6516 6731 :: **Office**: COM3-02-35

Consultation: Tuesday, 2 pm to 4 pm. Additional consultations by appointment are welcome.

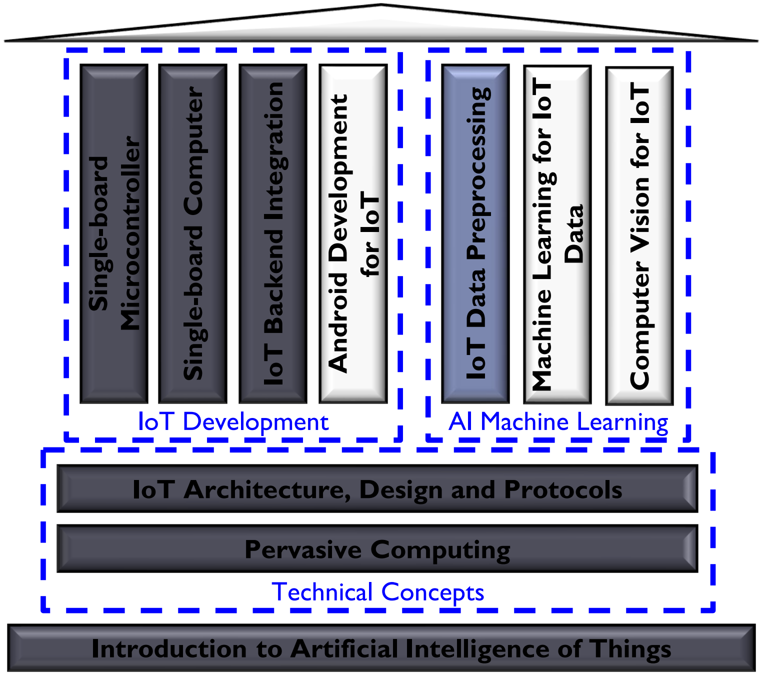
Quick Recap…

- In the previous lecture, we learnt:
  - How to connect a node device or hub to a fog processor or cloud server thereby integrating all three architectures of edge, fog and cloud.
  - RESTful web services can be created in Python with Flask and Connexion.
  - IoT sensor data sent to a cloud server can be persisted into a relational database.
- This lecture kickstarts our learning journey to find out how to use these data for AI machine learning.


Single-board Microcontroller
Single-board Computer
IoT Backend Integration
Android Development for IoT
IoT Development
IoT Data Preprocessing
Machine Learning for IoT Data
Computer Vision for IoT
AI Machine Learning
IoT Architecture, Design and Protocols
Pervasive Computing
Technical Concepts
Introduction to Artificial Intelligence of Things

Learning Objectives

- At the end of this lecture, you should understand:
  - More about machine learning.
  - How to perform data preparation with Pandas.
  - How to perform data visualisation with Matplotlib.



Single-board Microcontroller
Single-board Computer
IoT Backend Integration
Android Development for IoT
IoT Data Preprocessing
Machine Learning for IoT Data
Computer Vision for IoT
IoT Development
AI Machine Learning
IoT Architecture, Design and Protocols
Pervasive Computing
Technical Concepts
Introduction to Artificial Intelligence of Things

Readings

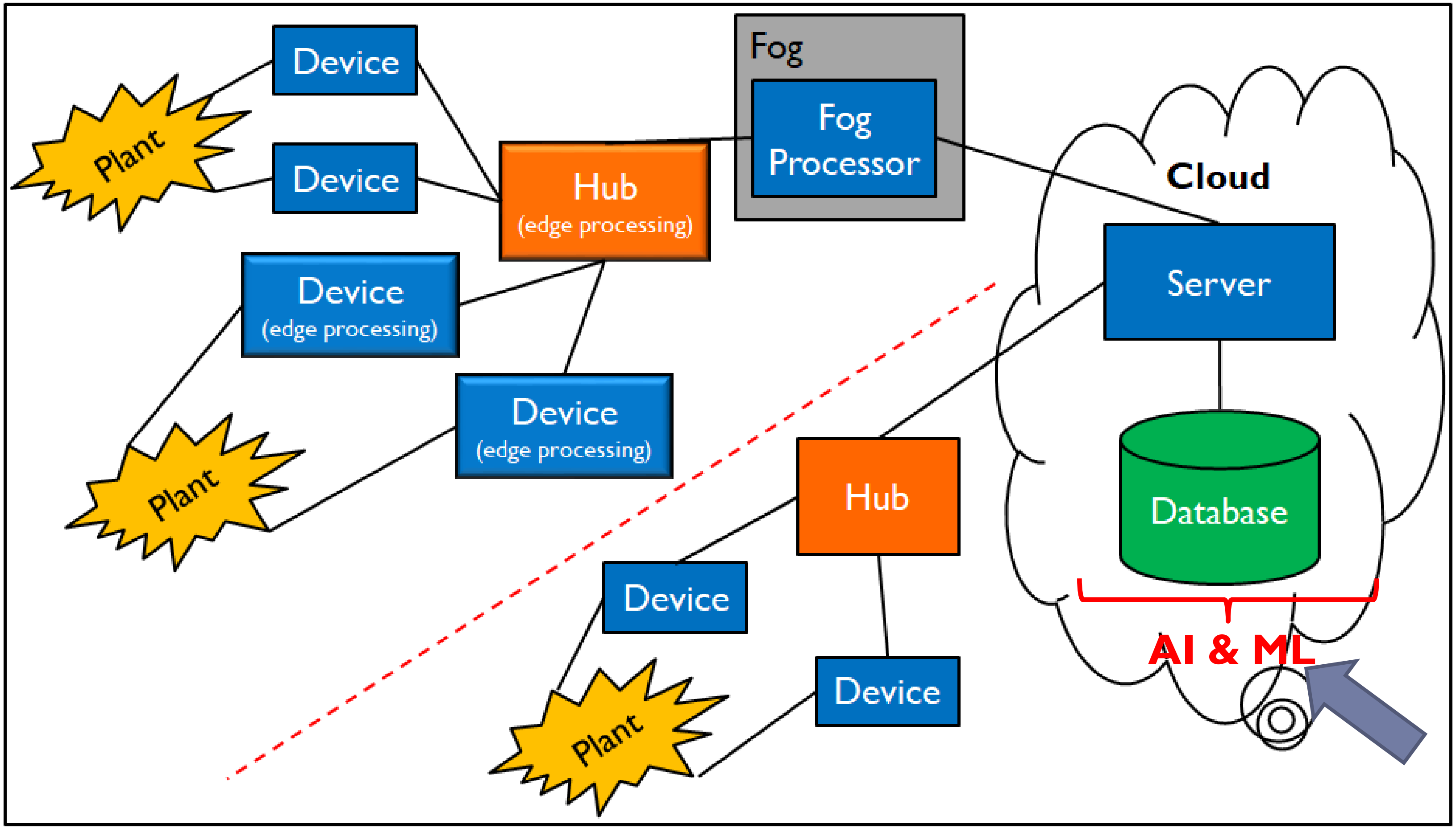
- Required readings:
  - None.
- Suggested readings:
  - None.

Technical Roadmap for IS4151/IS5451

Single-board Microcontroller
Android Wear

Single-board Computer
Android App

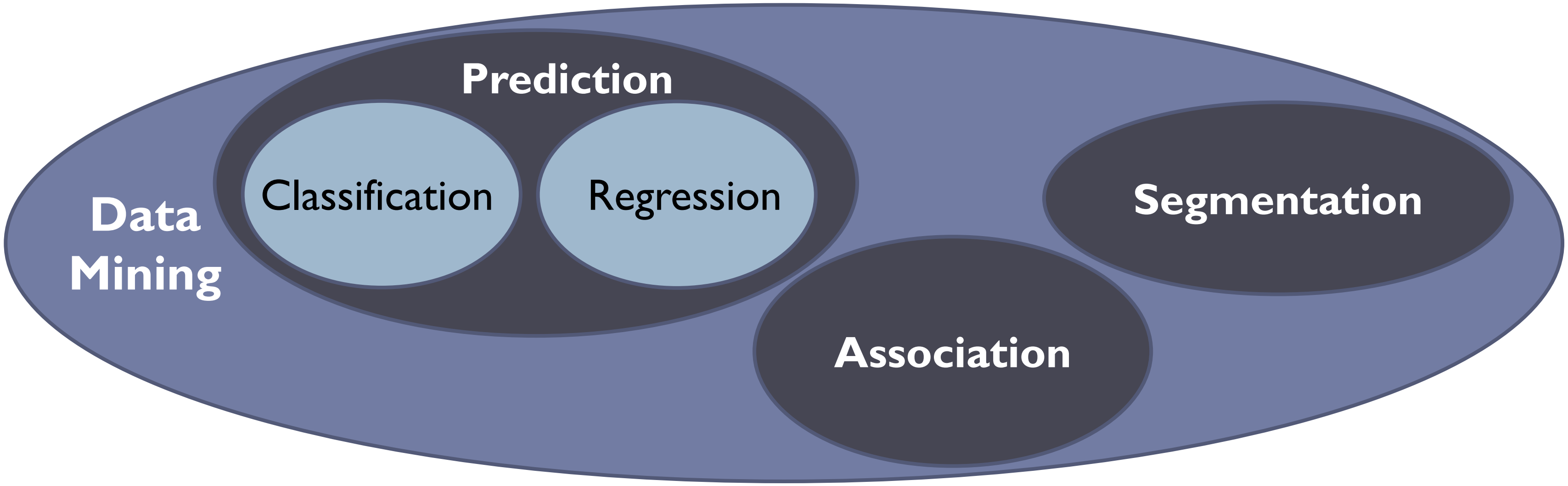
IoT Backend Integration



More About Machine Learning

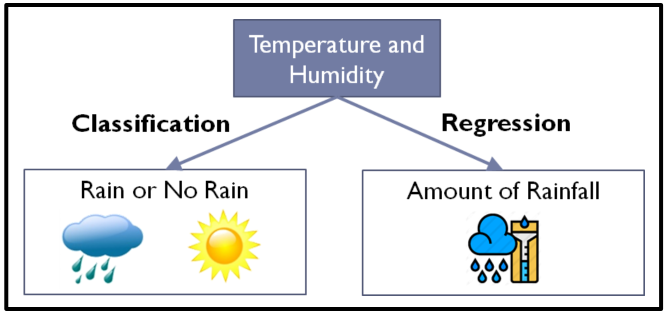
More About Machine Learning

- Recall that machine learning employs statistical and mathematical techniques to build a model based on sample data
- The objective is to identify patterns among variables in the data, i.e., data mining:
  - Data mining involves three main patterns:



More About Data Mining

- **Prediction**:
  - Forecast the outcome of future event or unknown phenomenon.
  - Intuitively, we can think of prediction as learning an **A→B** mapping, where A is the input and B is the output.
  - **Classification** – Predict weather outlook:
    - A are weather data such as temperature and humidity.
    - B is a **class label** representing the weather outlook such as "Rain" or "No Rain".



  - **Regression** – Predict rainfall:
    - A are weather data such as temperature and humidity.
    - B is a **real number** representing the amount of rainfall in millimeter.

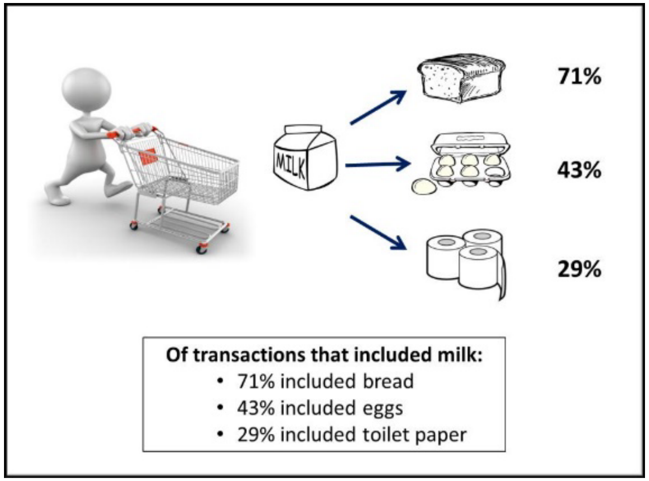
More About Data Mining (cont.)

- **Segmentation**:
  - Partition a collection of things (e.g., objects, events) in a dataset into natural groupings.
  - **Clustering**:
    - Create groups so that the members within each group have maximum similarity.
    - Members across groups have minimum similarity.
    - Examples:
      - □ Segment daily temperature data into hot day or cold day.
      - □ Segment customers based on their demographics and past purchase behaviors.

Leading Edge Shopper
Big Brand Shopper
Value Conscious Shopper
Seasonal Spender

More About Data Mining (cont.)

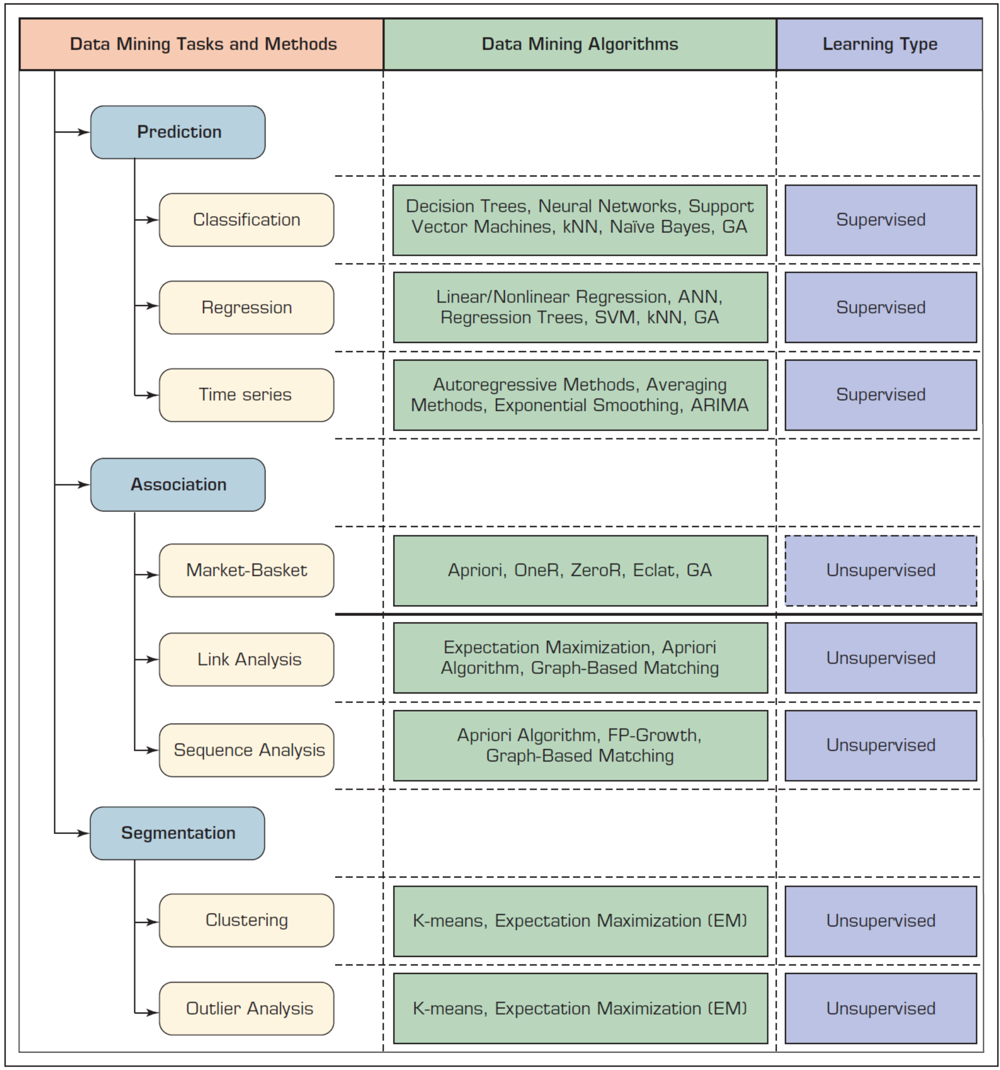
- **Association**:
  - Discover interesting relationships among variables in a large database.
  - **Market Basket Analysis**:
    - Discover regularities among products in largescale transactions recorded by point-of-sale systems in supermarkets:
      - □ I.e., each product purchased in a transaction being a variable.
    - Identify products that are commonly purchased together, e.g., beer and diaper.


MILK
71%
43%
29%
Of transactions that included milk:
• 71% included bread
• 43% included eggs
• 29% included toilet paper

More About Data Mining (cont.)

- Data mining tasks to extract the different types of patterns rely on **learning algorithms**.
- Learning algorithms can be classified according to the way patterns are extracted from historical data.
- **Supervised learning method** :
  - Training data include both independent variables and dependent variable
- **Unsupervised learning method**:
  - Training data include only the independent variables.

More About Data Mining (cont.)



Data Mining Tasks and Methods	Data Mining Algorithms	Learning Type
Prediction		
Classification	Decision Trees, Neural Networks, Support Vector Machines, kNN, Naïve Bayes, GA	Supervised
Regression	Linear/Nonlinear Regression, ANN, Regression Trees, SVM, kNN, GA	Supervised
Time series	Autoregressive Methods, Averaging Methods, Exponential Smoothing, ARIMA	Supervised
Association		
Market-Basket	Apriori, OneR, ZeroR, Eclat, GA	Unsupervised
Link Analysis	Expectation Maximization, Apriori Algorithm, Graph-Based Matching	Unsupervised
Sequence Analysis	Apriori Algorithm, FP-Growth, Graph-Based Matching	Unsupervised
Segmentation		
Clustering	K-means, Expectation Maximization (EM)	Unsupervised
Outlier Analysis	K-means, Expectation Maximization (EM)	Unsupervised

- Prediction involve A → B mapping in which we know the B to help determine the pattern.
- This is known as **supervised learning**.

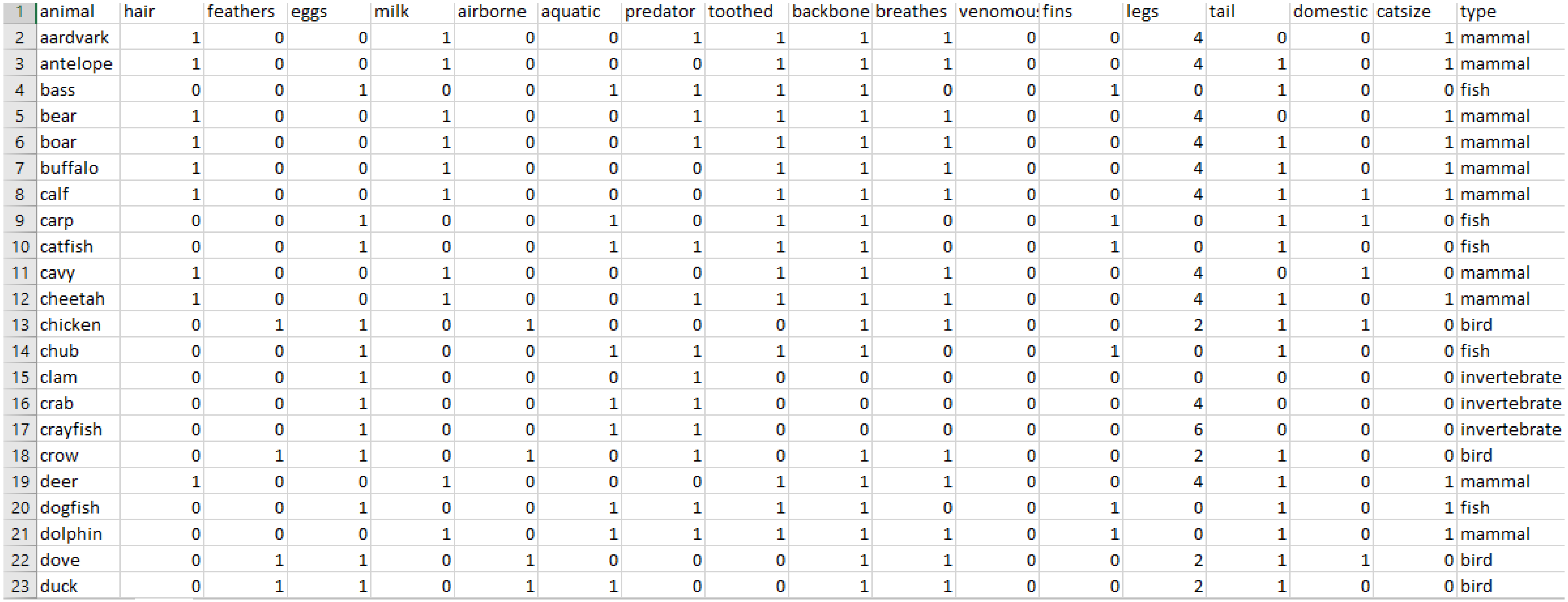
- Association and Segmentation involve just A without the B.
- We determine the pattern without the help of the B.
- This is known as **unsupervised learning**.

Source: Sharda et. al. (2020) – *Analytics, Data Science, & Artificial Intelligence: Systems for Decision Support*, pp. 206, Figure 4.2

Classification versus Clustering – A Concrete Example

- The Zoo animals dataset consists of 101 animals and 18 variables:
  - Name of the zoo animal, i.e., the identifier.
  - Type of the zoo animal, i.e., the class label:
    - 7 types – Amphibian, bird, fish, insect, invertebrate, mammal, and reptile.
  - A set of variables describing the characteristics of each animal:
    - 15 Boolean-valued variables such as hair, feathers, eggs and milk.
    - 1 numeric-valued variable, i.e., number of legs (0 to 8).

Classification versus Clustering – A Concrete Example (cont.)



1	animal	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomou	fins	legs	tail	domestic	catsize	type
2	aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	mammal
3	antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	mammal
4	bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	fish
5	bear	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	mammal
6	boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	mammal
7	buffalo	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	mammal
8	calf	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	mammal
9	carp	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	fish
10	catfish	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	fish
11	cavy	1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	mammal
12	cheetah	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	mammal
13	chicken	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	bird
14	chub	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	fish
15	clam	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	invertebrate
16	crab	0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	invertebrate
17	crayfish	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	invertebrate
18	crow	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	bird
19	deer	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	mammal
20	dogfish	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	fish
21	dolphin	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1	mammal
22	dove	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	bird
23	duck	0	1	1	0	1	1	0	0	1	1	0	0	2	1	0	0	bird

Identifier

Descriptive Attributes

Class Attribute

Classification versus Clustering – A Concrete Example (cont.)

- **Classification**:
  - Use decision tree classifier to learn the function or mapping between the characteristics of animals and their membership to each type.
  - This is a supervised learning process.
  - Use a common two-step methodology:
    - Model training and then follow by model testing.
    - In this case, we are using a simple split validation.
    - 70% of sample is used to train the model and derive a decision tree.
    - 30% of sample is used to test the model.

Classification versus Clustering – A Concrete Example (cont.)

- Compute the predictive accuracy, i.e., the model's ability to correctly predict the class label of new or previously unseen data:
  - A.k.a. testing accuracy.
- In this example, the predictive accuracy is about 90% (random stratified sampling).
- See `src01` for more details.

Classification versus Clustering – A Concrete Example (cont.)

- **Clustering**:
  - Uses **k-means clustering** algorithm, a traditional statistical analysis technique.
  - A predetermined number of clusters is defined, denoted by ***k***.
  - The algorithm assigns each data point (in this case each animal) to the cluster whose center, i.e., **centroid**, is the nearest:
    - Centroid is calculated as the average of all the points in the cluster.
    - Coordinates of the centroid are the arithmetic mean for each dimension separately over all the points in the cluster.
  - <u>Unsupervised learning process</u> – The class label, i.e., animal type, is not used by the algorithm.
  - In this example, we use *k*=2 and *k*=7 for comparison.

Classification versus Clustering – A Concrete Example (cont.)

- Cannot compute the predictive accuracy because we are not supposed to know the actual class label.
- The quality of a clustering model is evaluated using other statistical measures.
- See `src02` and `src03` for more details.

Overview of SciPy

- The Python ecosystem consists of various open-source libraries for mathematics, science, and engineering applications.

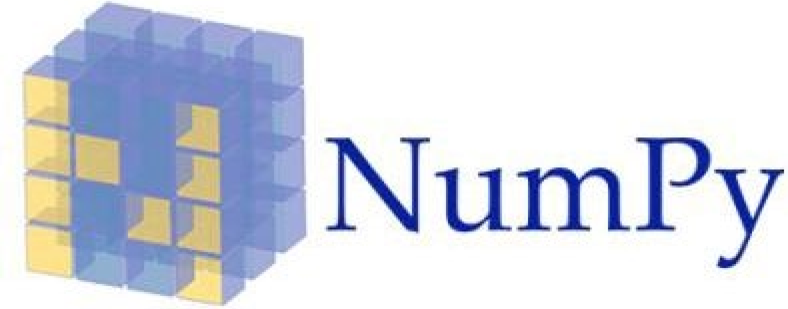
- Some of the core packages include:
  - **NumPy** – Base N-dimensional array package.
  - **Pandas** – Data structures & analysis.
  - **Matplotlib** – Comprehensive 2D Plotting.

Overview of the NumPy Library

- The built-in tools provided by the standard Python library may be too simple or inadequate for data analysis calculation.
- **NumPy** is the basic package for <u>scientific computing</u> in Python.
- It forms the basis for many other mathematical and scientific Python packages.
- NumPy library is based on one main object, i.e., **`ndarray`**, which stands for <u>N-dimensional array</u>:
  - `ndarray` is a <u>multidimensional</u> <u>homogeneous</u> array with a <u>predetermined number of items</u>.
  - All items are of the <u>same type</u> and the <u>same size</u>.

Overview of the NumPy Library (cont.)

- NumPy library operates on the `ndarray` object to support calculation of multidimensional arrays and large arrays.

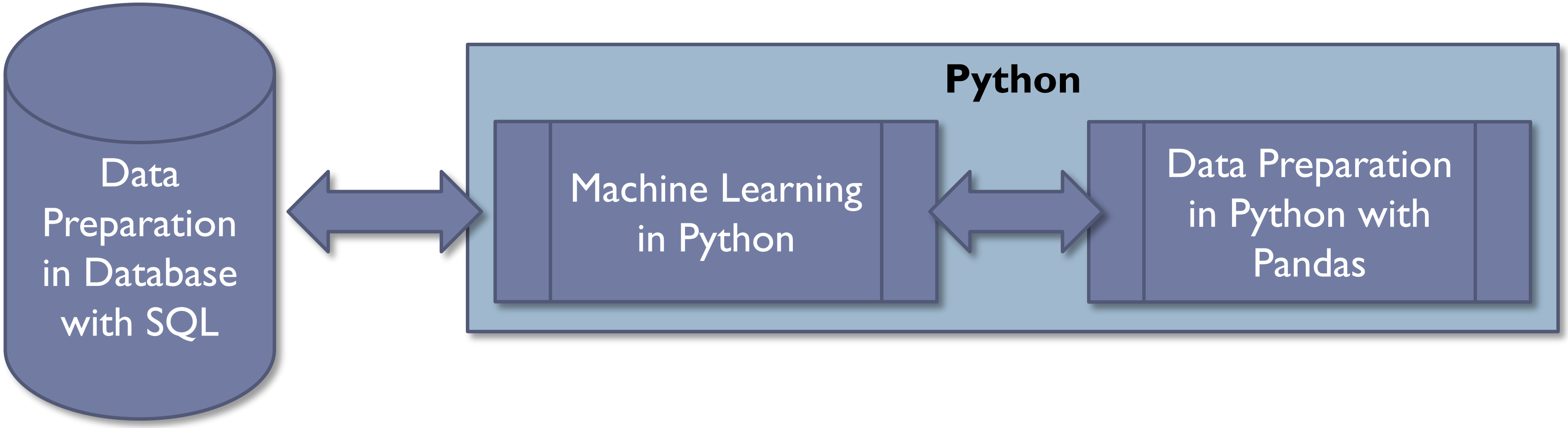


Overview of the Pandas Library

- **Pandas** is an open source Python library for performing highly specialised data analysis.
- It provides a single library for data analysts to easily process data, extract data and manipulate data, i.e., data preparation.
- Pandas is built upon the NumPy library.
- The standard data structures provided by Python and the `ndarray` provided by NumPy are not ideal for working with relational data or labelled data.

Overview of the Pandas Library (cont.)

- Pandas fills this important gap by providing two new data structures, namely `Series` and `DataFrame`.
- `Series` and `DataFrame` provide data analysts with the data manipulation capability equivalent to SQL-based relational database within Python.


Data Preparation in Database with SQL
Python
Machine Learning in Python
Data Preparation in Python with Pandas

Overview of the Matplotlib Library

- Data visualisation is an important part of data analytics as correct and efficient representation of data can improve understanding of the analysis.
- **Matplotlib** is a Python library specialising in the development of two-dimensional charts (including 3D charts).
- Key features:
  - Simplicity of usage.
  - Gradual development and interactive data visualisation.
  - Good control over graphics elements.
  - Exportable in many formats such as PNG, SVG and EPS.

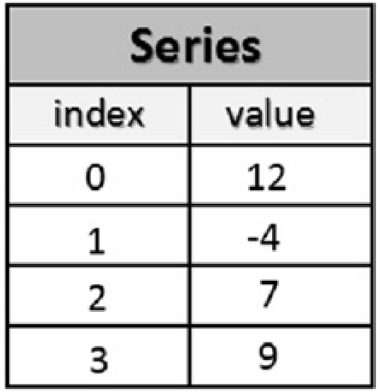
Machine Learning with Scikit-Learn

- **Scikit-Learn** is a Python library that integrates many machine learning algorithms:
  - It features various classification, regression and clustering algorithms.
  - E.g., support vector machines, random forests, gradient boosting and k-means.
- It is designed to interoperate with the Python numerical and scientific libraries NumPy and Pandas.

Data Preparation

Series

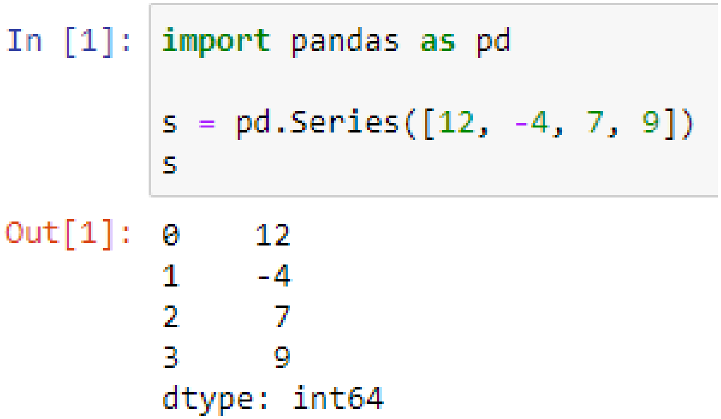
- **Series** is the Pandas object for representing one-dimensional data structures.
- Similar to an array but with some additional features.
- Internal structure is relatively easy and consists of two arrays associated with each other:
  - The main array holds the data (any NumPy type).
  - Each piece of data or element is associated with a label.
  - The label is contained within the other array.



Series	
index	value
0	12
1	-4
2	7
3	9

Creating a Series

- To create a new Series:
  - Call the Series() constructor and pass in the list containing the values to be included in the series.
  - By default, the label is a zero-based index number.
  - So, the left column is a series of labels, and the right column is the corresponding values.

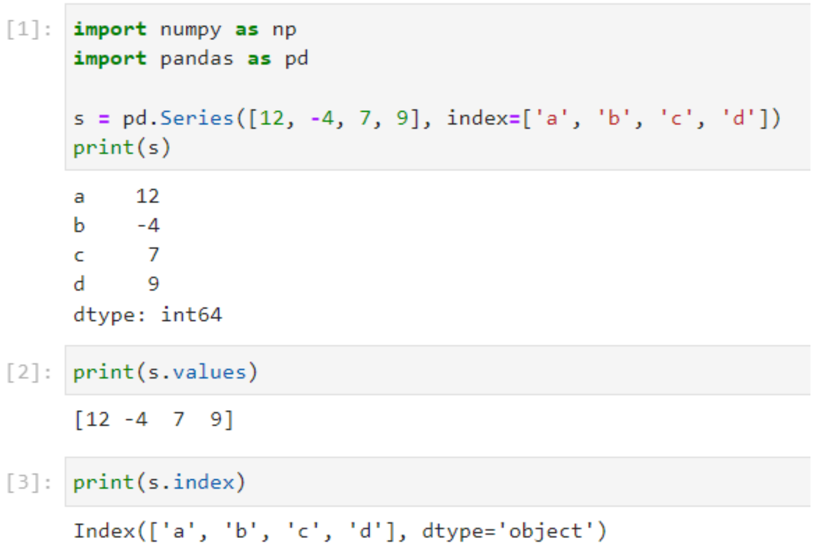


```
In [1]: import pandas as pd

        s = pd.Series([12, -4, 7, 9])
        s

Out[1]: 0    12
        1    -4
        2     7
        3     9
        dtype: int64
```

src04

Creating a Series (cont.)

- But it is preferable to create Series using meaningful labels:
  - Can distinguish and identify each item regardless of the order in which they were inserted into the Series.
  - This can be done by including the index option and assigning a list of strings containing the labels.

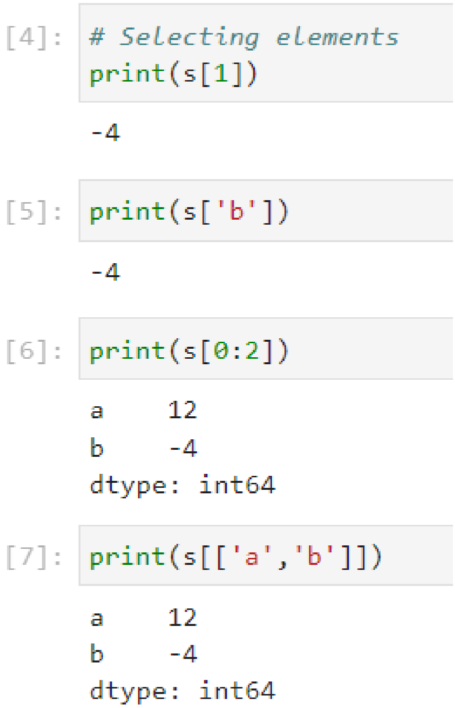


```
[1]: import numpy as np
     import pandas as pd

     s = pd.Series([12, -4, 7, 9], index=['a', 'b', 'c', 'd'])
     print(s)

     a    12
     b    -4
     c     7
     d     9
     dtype: int64

[2]: print(s.values)

     [12 -4  7  9]

[3]: print(s.index)

     Index(['a', 'b', 'c', 'd'], dtype='object')
```

src05

Selecting Elements in a `Series`

- Internal elements in the Series can be selected as:
  - Ordinary numpy array, e.g., `s[1]`
  - Label corresponding to the position of the index `s['b']`
  - It is also possible to select multiple items.

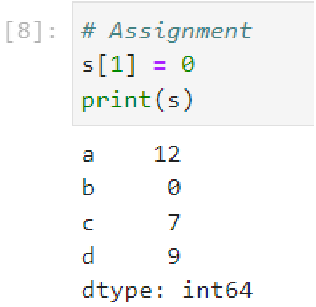


```
[4]: # Selecting elements
     print(s[1])

     -4

[5]: print(s['b'])

     -4

[6]: print(s[0:2])

     a    12
     b    -4
     dtype: int64

[7]: print(s[['a','b']])

     a    12
     b    -4
     dtype: int64
```

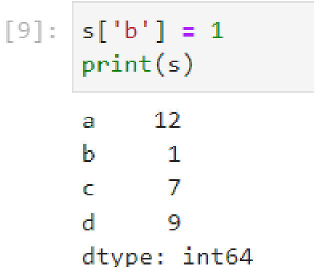
src05

Selecting Elements in a Series (cont.)

- New value can be assigned to individual elements using the assignment operator:



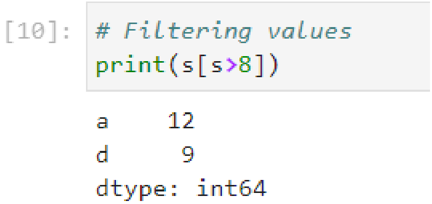
```
[8]: # Assignment
     s[1] = 0
     print(s)

     a    12
     b     0
     c     7
     d     9
     dtype: int64
```



```
[9]: s['b'] = 1
     print(s)

     a    12
     b     1
     c     7
     d     9
     dtype: int64
```

src05

- Individual elements can be filtered by their value using NumPy filter syntax:

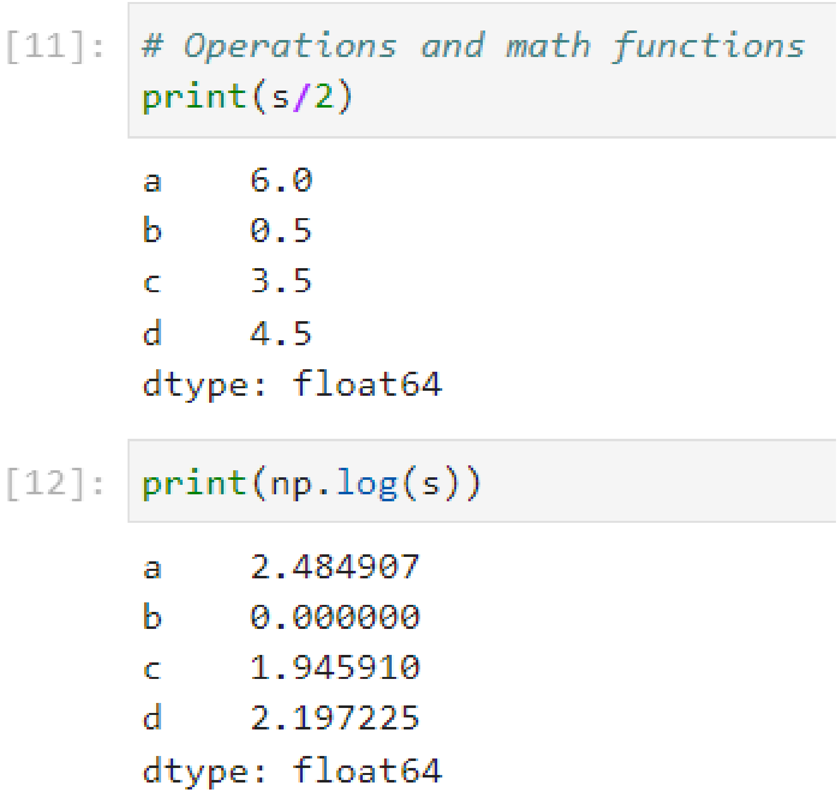


```
[10]: # Filtering values
      print(s[s>8])

      a    12
      d     9
      dtype: int64
```

src05

Basic Operations on `Series`

- We can perform <u>various operations and mathematical functions</u> that are applicable to NumPy arrays to Pandas Series:



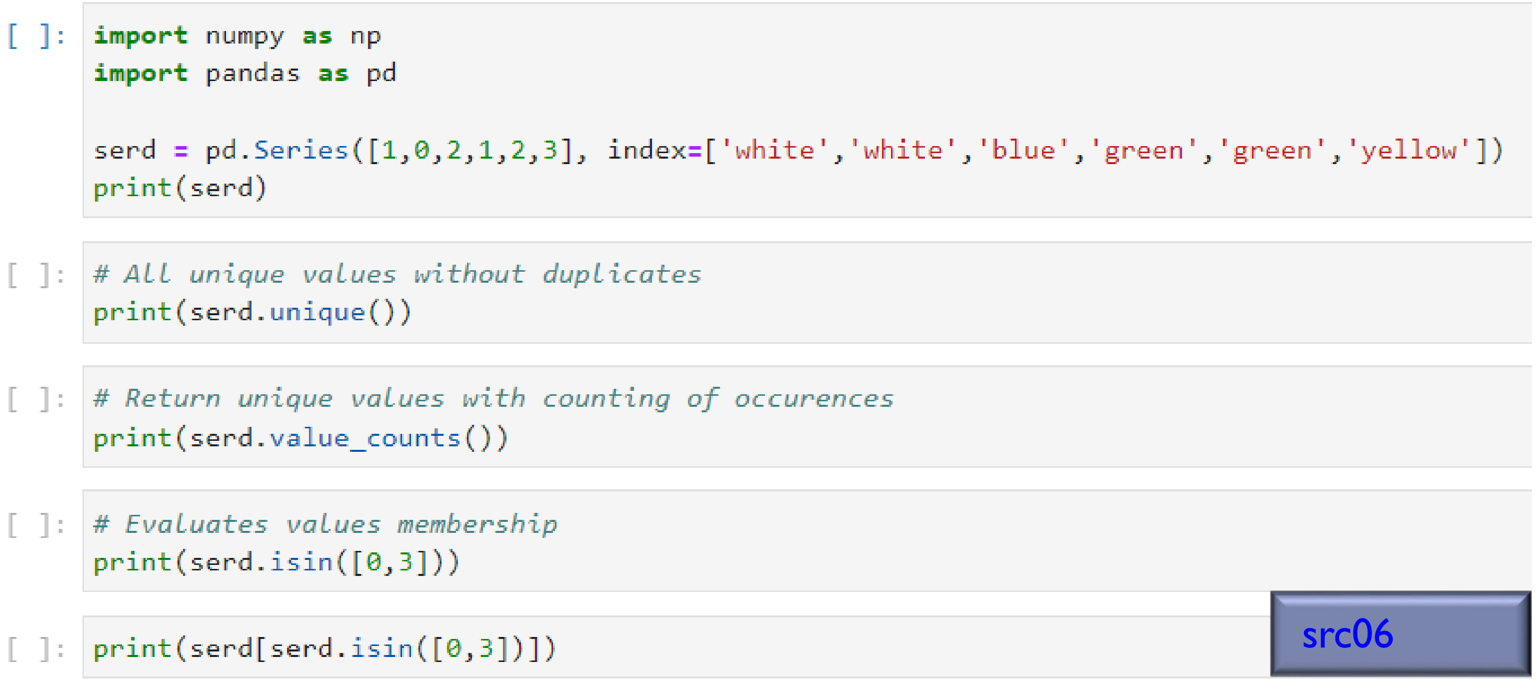
```
[11]: # Operations and math functions
      print(s/2)

a    6.0
b    0.5
c    3.5
d    4.5
dtype: float64
```

```
[12]: print(np.log(s))

a    2.484907
b    0.000000
c    1.945910
d    2.197225
dtype: float64
```

src05

Basic Operations on Series (cont.)

- Evaluating values within a Series provides information on:
  - What unique values do the samples contain – unique()
  - Counting duplicate values – value_counts()
  - Determine whether a value is present or not – isin()



```
[ ]: import numpy as np
     import pandas as pd

     serd = pd.Series([1,0,2,1,2,3], index=['white','white','blue','green','green','yellow'])
     print(serd)
```

```
[ ]: # All unique values without duplicates
     print(serd.unique())
```

```
[ ]: # Return unique values with counting of occurences
     print(serd.value_counts())
```

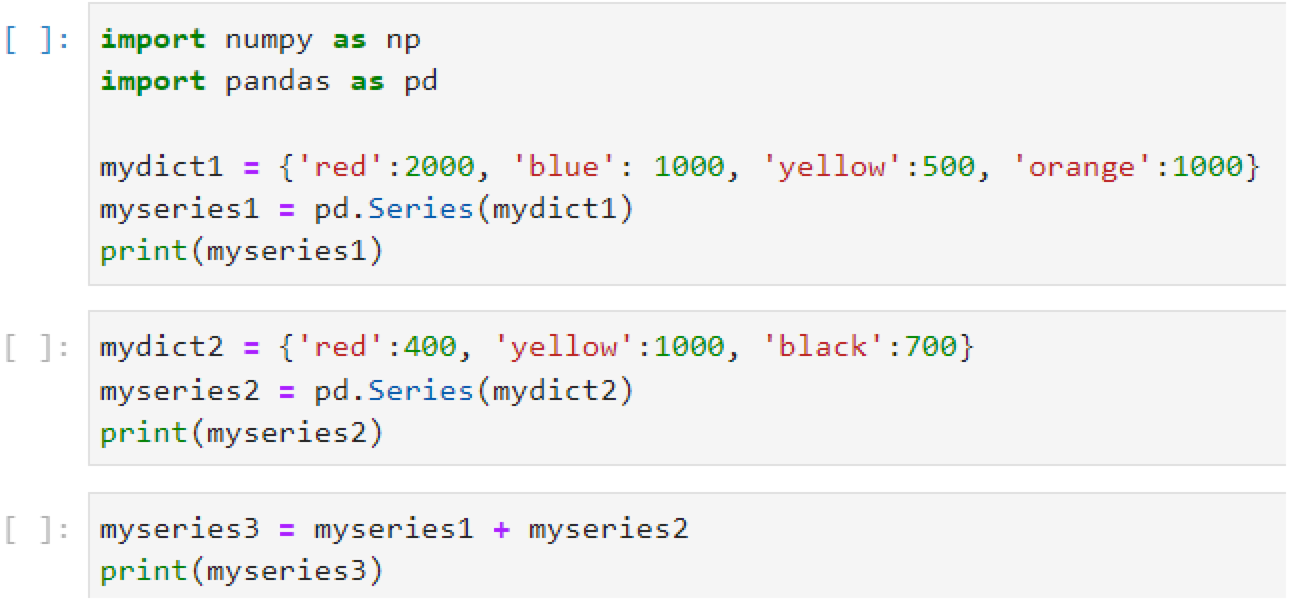
```
[ ]: # Evaluates values membership
     print(serd.isin([0,3]))
```

```
[ ]: print(serd[serd.isin([0,3])])
```

src06

Basic Operations on Series (cont.)

- Series can be created from a dict (dictionary) object as they are actually very similar in the general structure.
- Finally, since we can perform arithmetic operation between a Series and a scalar value, we can do that between two Series too.



```
[ ]: import numpy as np
     import pandas as pd

     mydict1 = {'red':2000, 'blue': 1000, 'yellow':500, 'orange':1000}
     myseries1 = pd.Series(mydict1)
     print(myseries1)
```

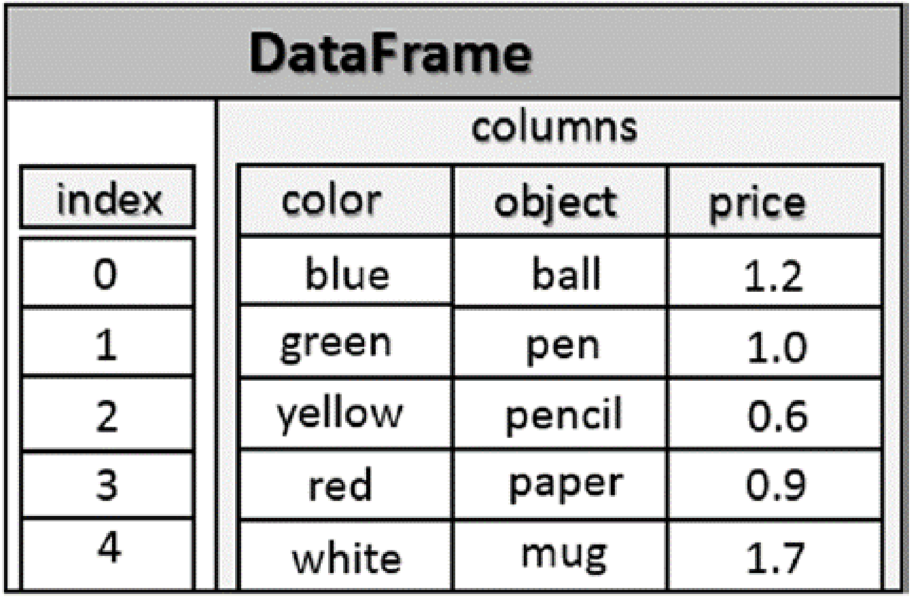
```
[ ]: mydict2 = {'red':400, 'yellow':1000, 'black':700}
     myseries2 = pd.Series(mydict2)
     print(myseries2)
```

```
[ ]: myseries3 = myseries1 + myseries2
     print(myseries3)
```

src07

DataFrame

- DataFrame is a tabular data structure very similar to a spreadsheet or relational database table.
- It extends the Series to multiple dimensions/columns.
- DataFrame is essentially an ordered collection of columns, each of which can contain values of different type.



DataFrame

	columns		
index	color	object	price
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

DataFrame (cont.)

- Recall that a Series has an index array containing labels associated with each element.
- A DataFrame has two index arrays:
  - The first array is associated with the rows and serves a similar function as the index array in a Series.
  - In fact, each label in the first array is associated with all the values in the row → Think of this as the row header in a spreadsheet or primary key in a relational database table.
  - The second array contains a series of labels, each associated with a particular column → Think of this as the column header in a spreadsheet or column/attribute name in a relational database table.

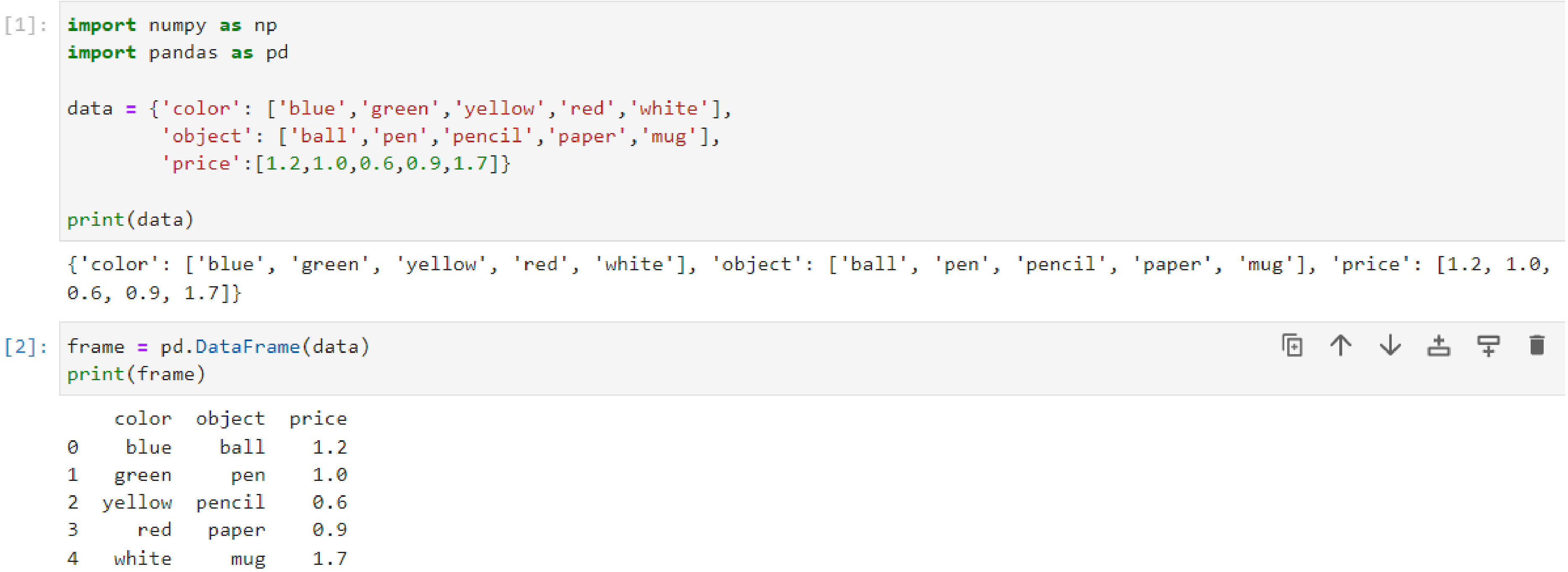
`DataFrame` (cont.)

- Another way to look at a `DataFrame` is to think of it as a `dictionary` of `Series`:
  - The keys are the column names.
  - The values are the `Series`.
  - Furthermore, all elements of each `Series` are mapped according to an array of labels called the index.

Defining a DataFrame

- The most common way to define a DataFrame is to pass a dict object to the DataFrame() constructor:
  - The dict object contains a key for each column to be defined together with a list of values.
  - This approach is quite similar to how we define a Series but now we are essentially defining multiple Series objects to make up the DataFrame.

Defining a DataFrame (cont.)



```
[1]: import numpy as np
     import pandas as pd

     data = {'color': ['blue','green','yellow','red','white'],
             'object': ['ball','pen','pencil','paper','mug'],
             'price':[1.2,1.0,0.6,0.9,1.7]}

     print(data)
```

```
{'color': ['blue', 'green', 'yellow', 'red', 'white'], 'object': ['ball', 'pen', 'pencil', 'paper', 'mug'], 'price': [1.2, 1.0,
0.6, 0.9, 1.7]}
```

```
[2]: frame = pd.DataFrame(data)
     print(frame)
```

```
    color  object  price
0    blue    ball    1.2
1   green     pen    1.0
2  yellow  pencil    0.6
3     red   paper    0.9
4   white     mug    1.7
```

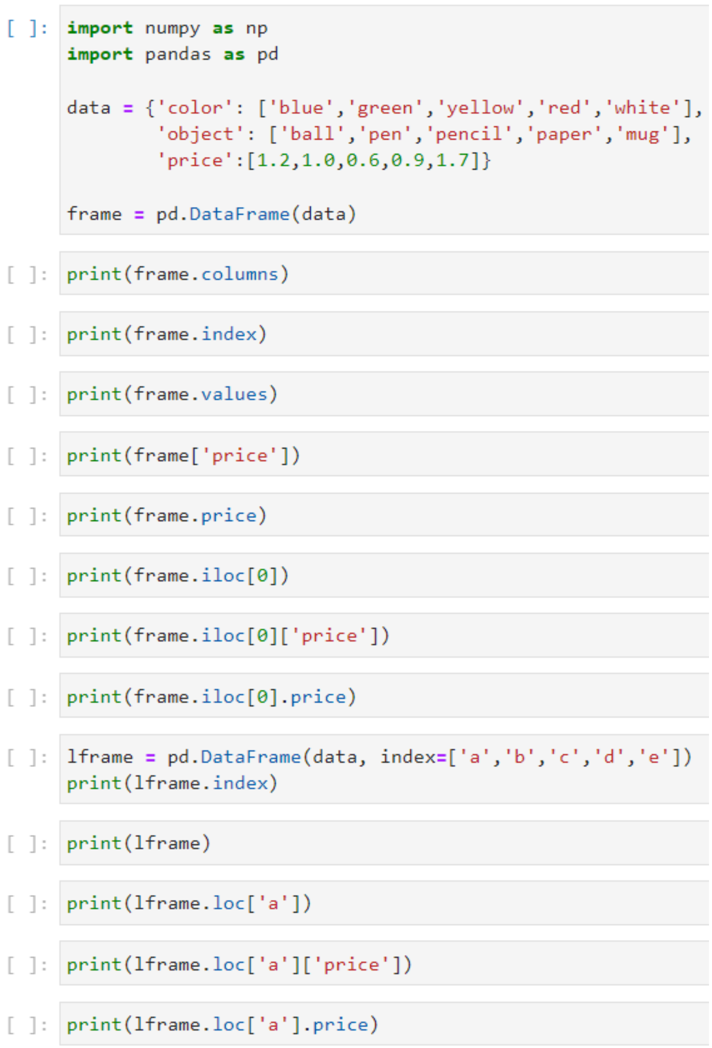
src08

Selecting Elements in a `DataFrame`

- We can obtain the name of all the columns and the list of indexes of the `DataFrame` object using its `columns` and `index` attributes, respectively.
- The entire set of data in the `DataFrame` can be retrieved using the `values` attribute.
- To select the contents of a column, we can use the column name as an index or use the column name as an attribute of the `DataFrame`.
- To select the contents of a row:
  - Use the **`iloc`** attribute with the required positional index.
  - Use the **`loc`** attribute with the required label index (if one is defined with the `index` option)

Selecting Elements in a DataFrame (cont.)

- To select the content of a cell, we combine the iloc or loc attribute with the required column name.



```
[ ]: import numpy as np
     import pandas as pd

     data = {'color': ['blue','green','yellow','red','white'],
             'object': ['ball','pen','pencil','paper','mug'],
             'price':[1.2,1.0,0.6,0.9,1.7]}

     frame = pd.DataFrame(data)
```

```
[ ]: print(frame.columns)
```

```
[ ]: print(frame.index)
```

```
[ ]: print(frame.values)
```

```
[ ]: print(frame['price'])
```

```
[ ]: print(frame.price)
```

```
[ ]: print(frame.iloc[0])
```

```
[ ]: print(frame.iloc[0]['price'])
```

```
[ ]: print(frame.iloc[0].price)
```

```
[ ]: lframe = pd.DataFrame(data, index=['a','b','c','d','e'])
     print(lframe.index)
```

```
[ ]: print(lframe)
```

```
[ ]: print(lframe.loc['a'])
```

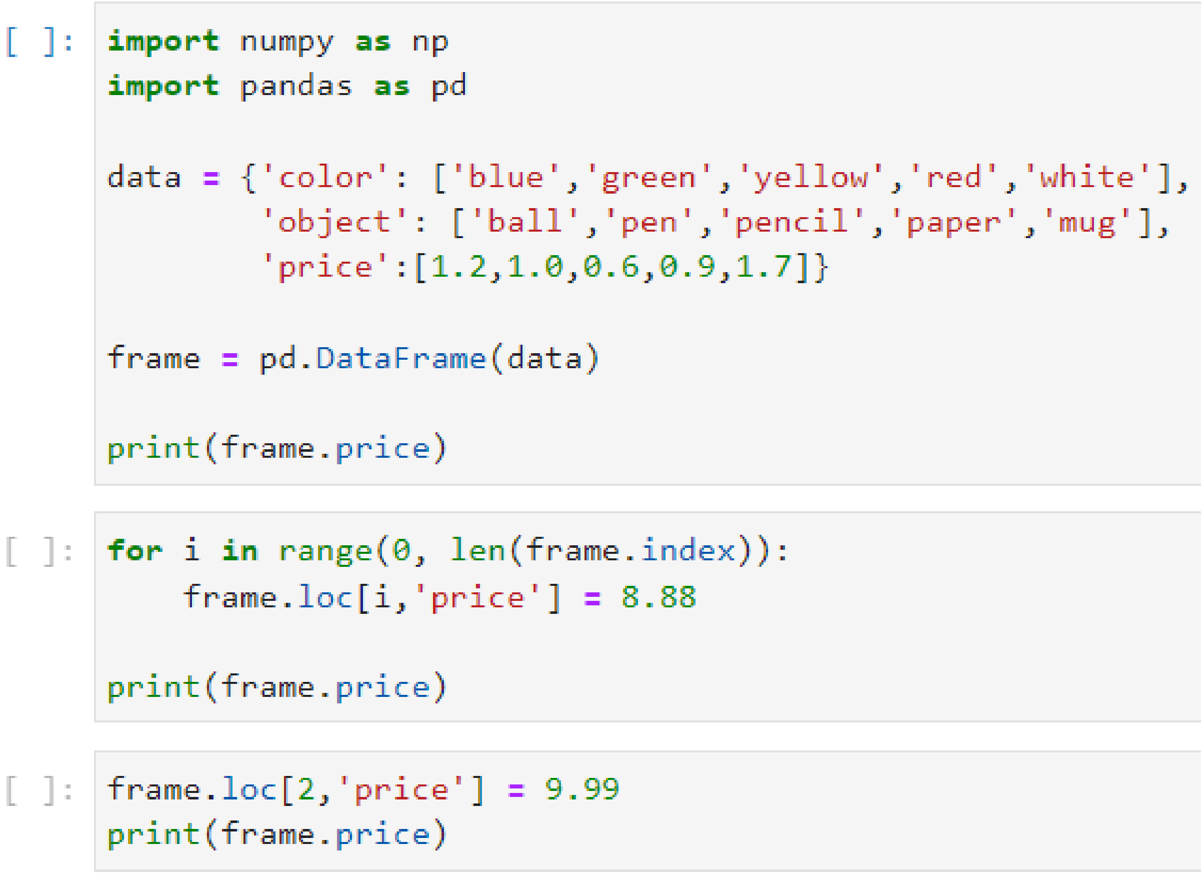
```
[ ]: print(lframe.loc['a']['price'])
```

```
[ ]: print(lframe.loc['a'].price)
```

src09

Selecting Elements in a DataFrame (cont.)

- We can assign new value to each cell with the assignment operator and the loc attribute of the DataFrame.



```
[ ]: import numpy as np
     import pandas as pd

     data = {'color': ['blue','green','yellow','red','white'],
             'object': ['ball','pen','pencil','paper','mug'],
             'price':[1.2,1.0,0.6,0.9,1.7]}

     frame = pd.DataFrame(data)

     print(frame.price)
```

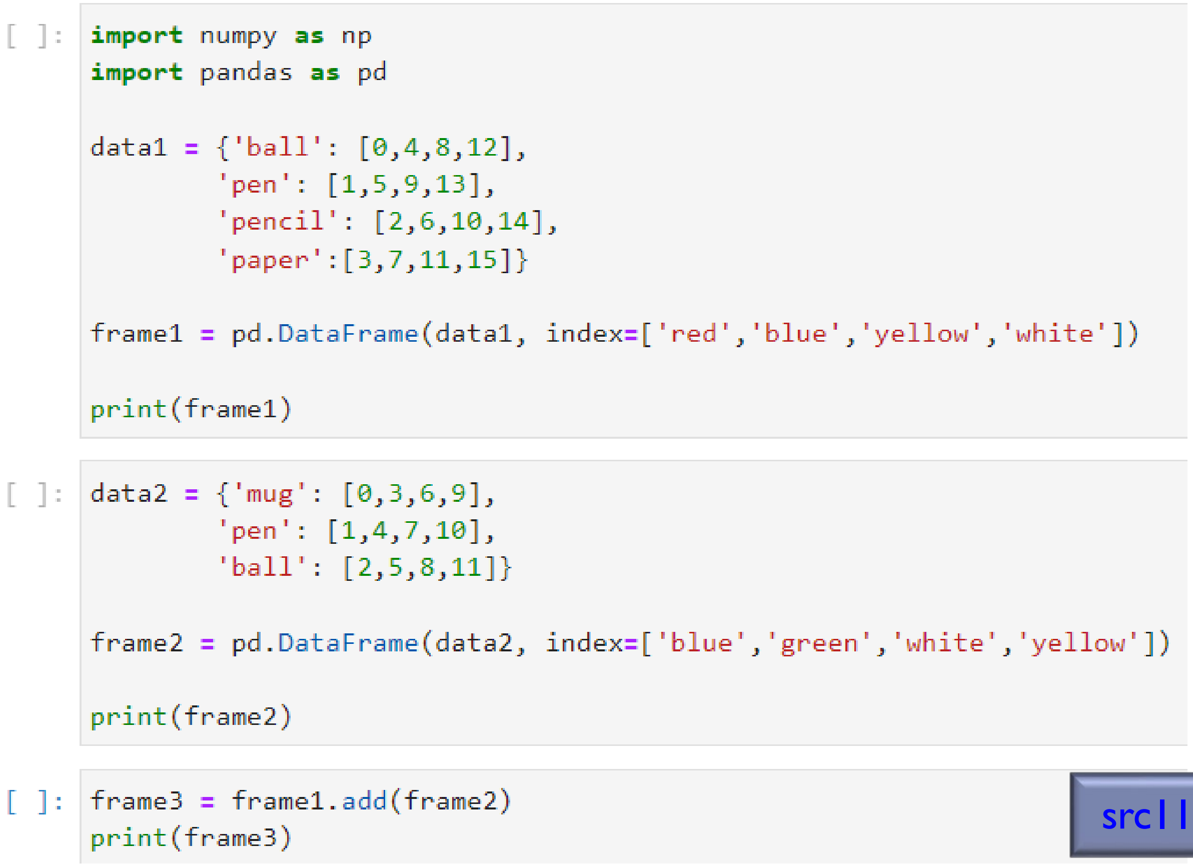
```
[ ]: for i in range(0, len(frame.index)):
         frame.loc[i,'price'] = 8.88

     print(frame.price)
```

```
[ ]: frame.loc[2,'price'] = 9.99
     print(frame.price)
```

src10

Basic Operations on DataFrame

- Arithmetic operations can be performed between two DataFrame objects using the flexible arithmetic methods:
  - add()
  - sub()
  - div()
  - mul()



```
[ ]: import numpy as np
     import pandas as pd

     data1 = {'ball': [0,4,8,12],
             'pen': [1,5,9,13],
             'pencil': [2,6,10,14],
             'paper':[3,7,11,15]}

     frame1 = pd.DataFrame(data1, index=['red','blue','yellow','white'])

     print(frame1)
```

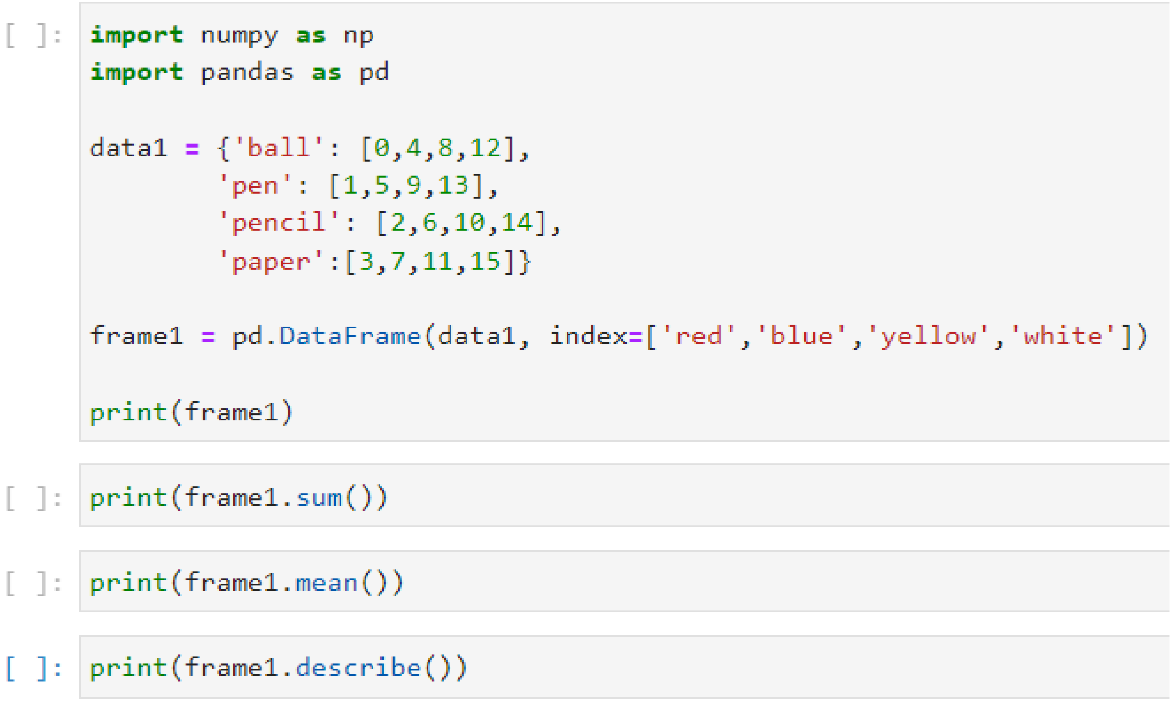
```
[ ]: data2 = {'mug': [0,3,6,9],
             'pen': [1,4,7,10],
             'ball': [2,5,8,11]}

     frame2 = pd.DataFrame(data2, index=['blue','green','white','yellow'])

     print(frame2)
```

```
[ ]: frame3 = frame1.add(frame2)
     print(frame3)
```

src11

Basic Operations on DataFrame (cont.)

- Statistic functions can be applied on the DataFrame:
  - sum()
  - mean()
  - describe()



```
[ ]: import numpy as np
     import pandas as pd

     data1 = {'ball': [0,4,8,12],
              'pen': [1,5,9,13],
              'pencil': [2,6,10,14],
              'paper':[3,7,11,15]}

     frame1 = pd.DataFrame(data1, index=['red','blue','yellow','white'])

     print(frame1)
```

```
[ ]: print(frame1.sum())
```

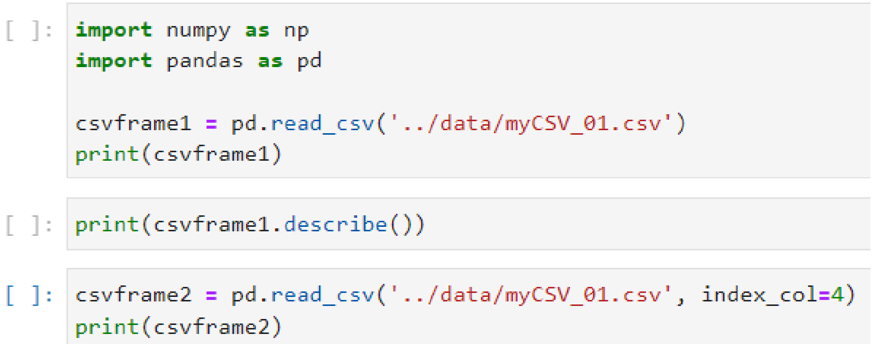
```
[ ]: print(frame1.mean())
```

```
[ ]: print(frame1.describe())
```

src12

Importing Data into DataFrame

- We can read data file in various format into a DataFrame object.
- The content of a CSV file can be read in with the read_csv() function and converted to a DataFrame object.
- The index_col attribute of read_csv() can be used to designate the row header or primary key.



```
[ ]: import numpy as np
     import pandas as pd

     csvframe1 = pd.read_csv('../data/myCSV_01.csv')
     print(csvframe1)
```

```
[ ]: print(csvframe1.describe())
```

```
[ ]: csvframe2 = pd.read_csv('../data/myCSV_01.csv', index_col=4)
     print(csvframe2)
```

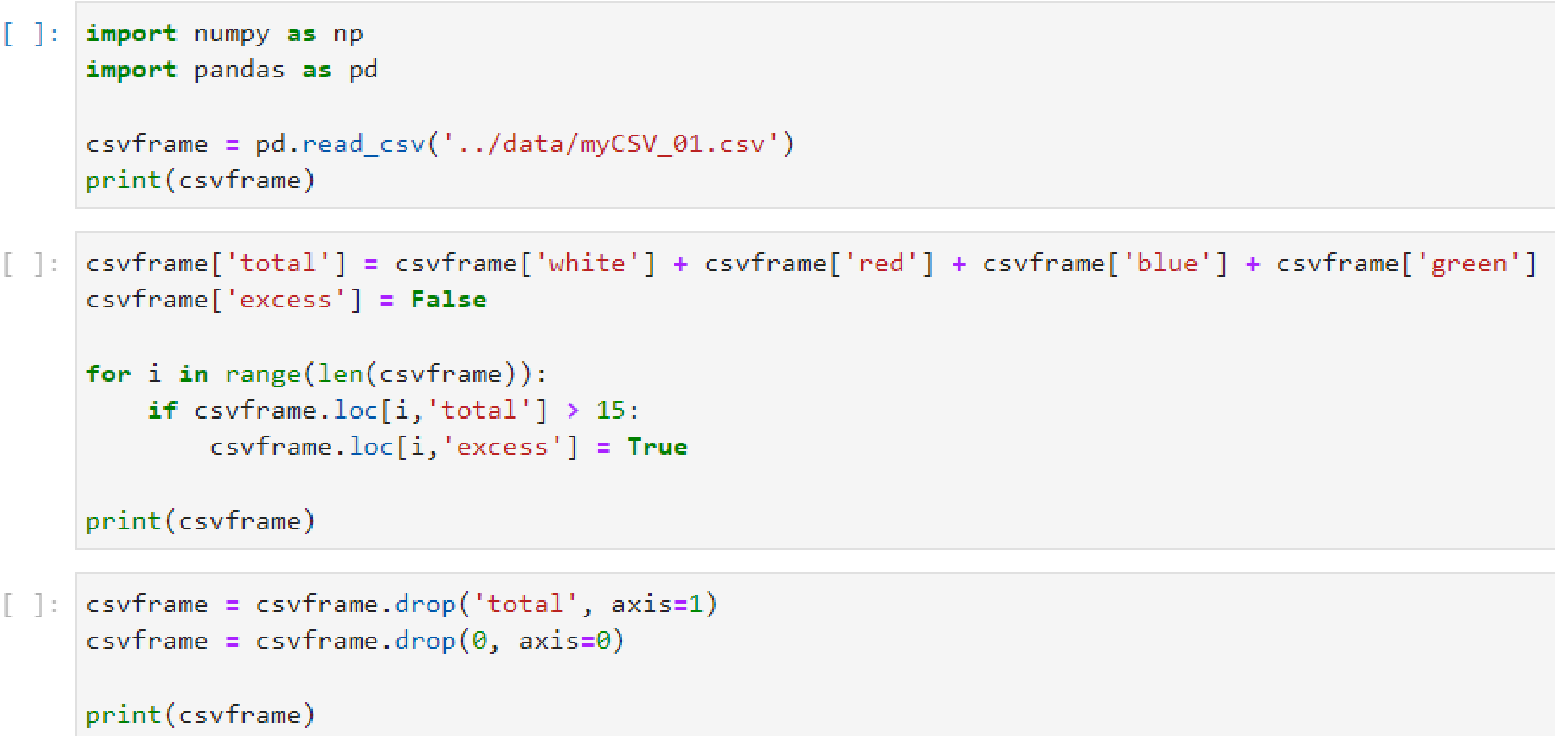
src13

- We can read other formats such as Excel, SQL and JSON.

Manipulating Columns in `DataFrame`

- Adding columns:
  - A new column can be added by setting the new column name to be equal to an initial/default value, e.g., 0.
  - We can also initialize the new column to the null value using the NumPy special value `np.nan`.
- Dropping columns:
  - We can drop a column using the `drop()` function.
  - Need to specify the column key and set the `axis` attribute to 1.
  - Setting `axis` to 0 is used to drop a row (default behaviour).

Manipulating Columns in DataFrame (cont.)



```
[ ]: import numpy as np
     import pandas as pd

     csvframe = pd.read_csv('../data/myCSV_01.csv')
     print(csvframe)
```

```
[ ]: csvframe['total'] = csvframe['white'] + csvframe['red'] + csvframe['blue'] + csvframe['green']
     csvframe['excess'] = False

     for i in range(len(csvframe)):
         if csvframe.loc[i,'total'] > 15:
             csvframe.loc[i,'excess'] = True

     print(csvframe)
```

```
[ ]: csvframe = csvframe.drop('total', axis=1)
     csvframe = csvframe.drop(0, axis=0)

     print(csvframe)
```

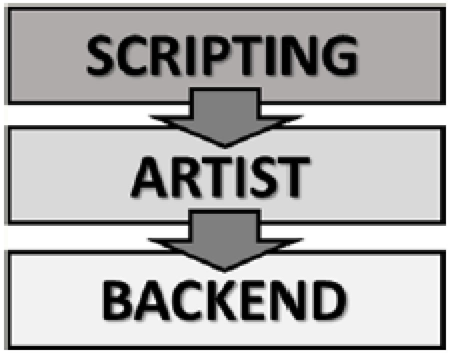
src14

Data Visualisation

Matplotlib Architecture

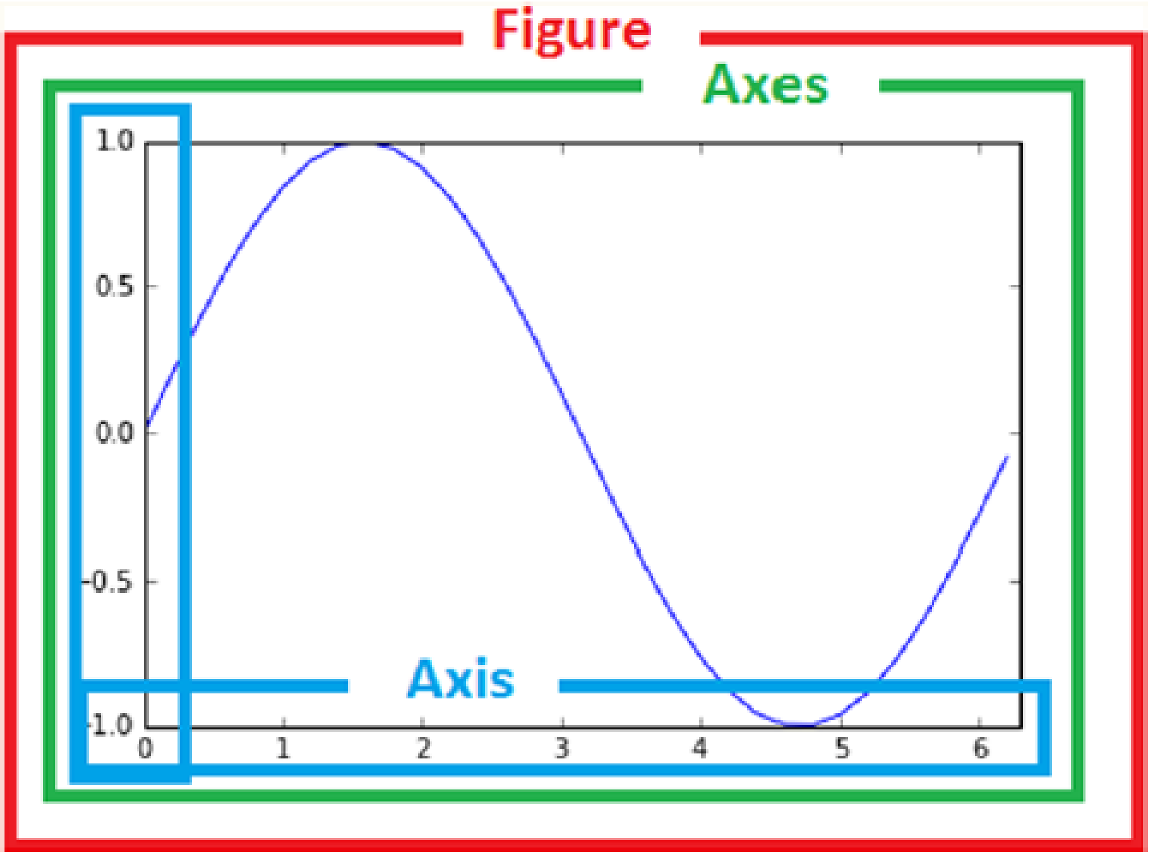
- Matplotlib provides a set of functions and tools that allow the representation and manipulation of a **Figure** (the main object) and its associated internal objects.
- Other than graphics, Matplotlib also handles the events and graphics animation.
- Thus, Matplotlib can produce interactive charts that can respond to events triggered by keyboard or mouse movement.
- The architecture of Matplotlib is structured into three layers with unidirectional communication:
  - Each layer can communicate with the underlying layer only.

Matplotlib Architecture (cont.)



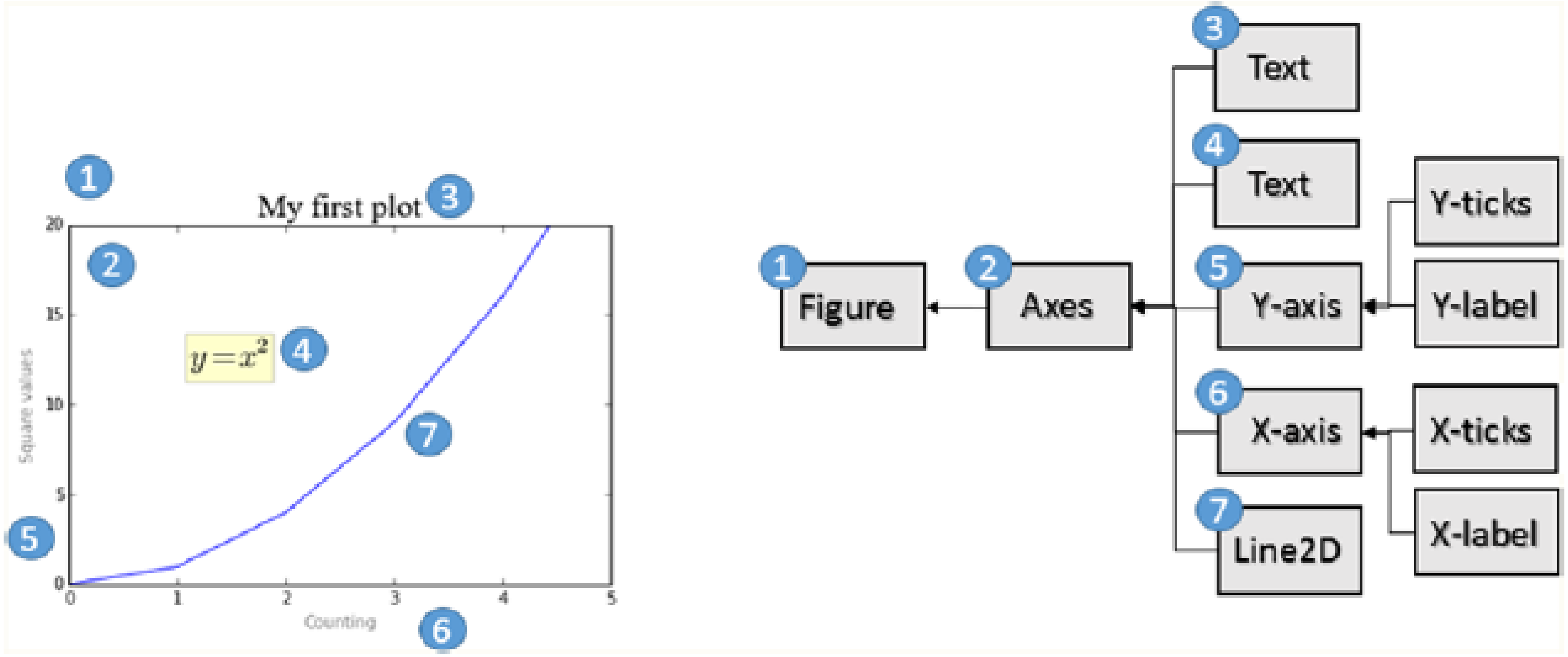
- **Scripting layer** – Consists of the **pyplot** interface for actual data calculation, analysis and visualisation.
- **Artist layer** – An intermediate layer representing all the elements that make up a chart, e.g., title, axis, labels, markers, etc.
- **Backend layer** – Matplotlib API and a set of classes to represent the graphic elements.

Matplotlib Architecture (cont.)



The three main artist objects in the hierarchy of the Artist layer.

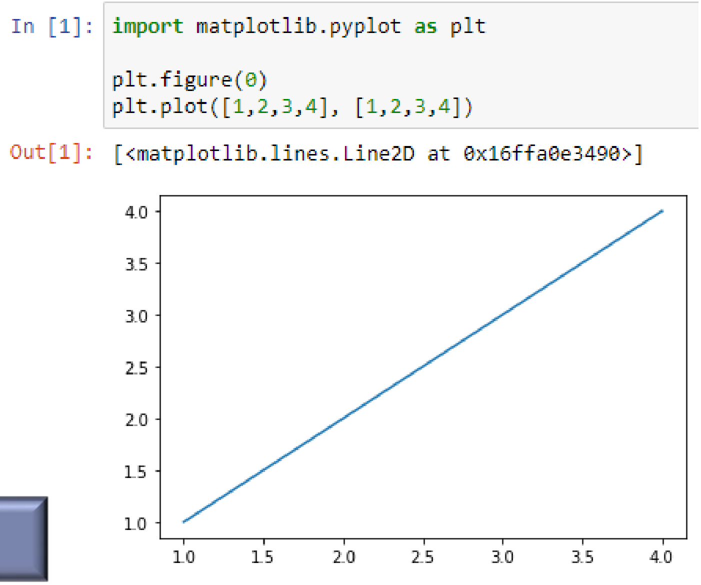
Matplotlib Architecture (cont.)



Each instance of a chart corresponds to an instance of Artist structured in a hierarchy

pyplot

- The `pyplot` module is a collection of command-style functions that allow the data analyst to operate or make changes to the Figure.
    - E.g., Create a new Figure.
- A simple interactive chart can be created using the pyplot object's `plot()` function.
- The `plot()` function uses a default configuration that does not have title, axis label, legend, etc.

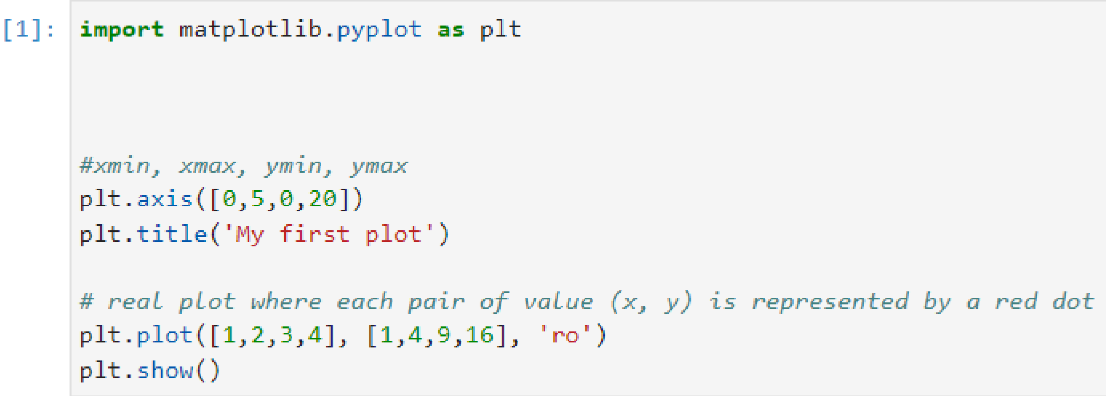


```
In [1]: import matplotlib.pyplot as plt

        plt.figure(0)
        plt.plot([1,2,3,4], [1,2,3,4])

Out[1]: [<matplotlib.lines.Line2D at 0x16ffa0e3490>]
```

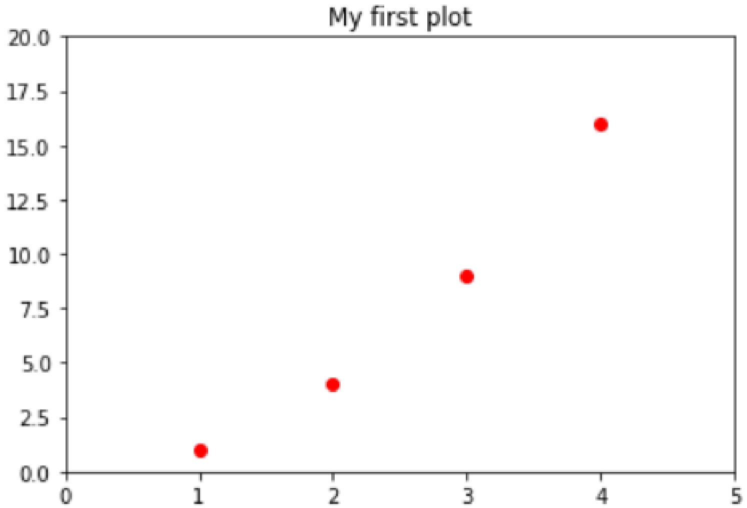
src15

pyplot (cont.)

- The default configuration can be changed to obtain the desired chart.



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



     #xmin, xmax, ymin, ymax
     plt.axis([0,5,0,20])
     plt.title('My first plot')

     # real plot where each pair of value (x, y) is represented by a red dot
     plt.plot([1,2,3,4], [1,4,9,16], 'ro')
     plt.show()
```

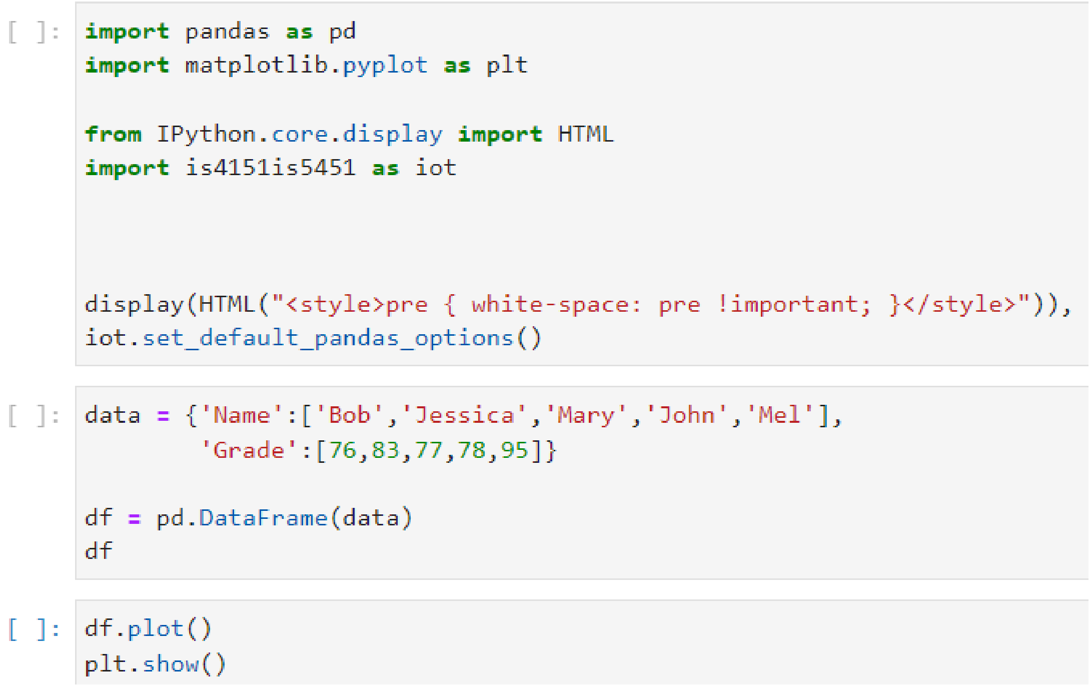


src16

Data Visualisation – Line Plot

- To create a simple line plot from data in a Pandas `DataFrame`, we can use `DataFrame.plot()`, which relies on `matplotlib`:
  - By default, you will get a line plot.

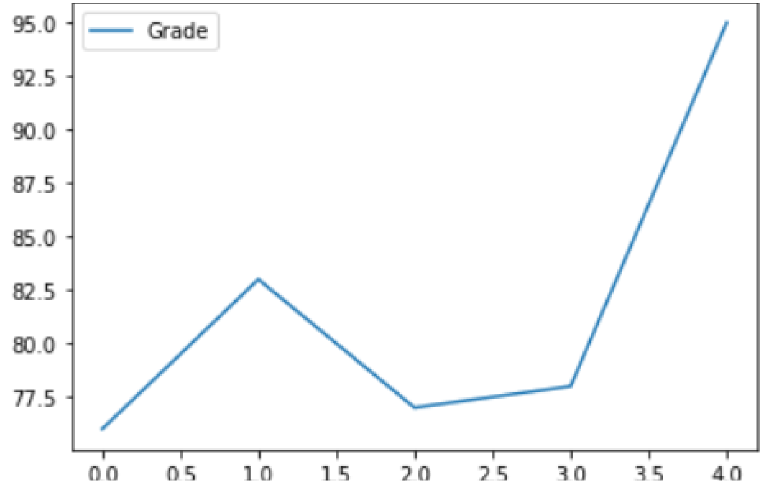
src17



```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt

     from IPython.core.display import HTML
     import is4151is5451 as iot



     display(HTML("<style>pre { white-space: pre !important; }</style>")),
     iot.set_default_pandas_options()
```

```
[ ]: data = {'Name':['Bob','Jessica','Mary','John','Mel'],
             'Grade':[76,83,77,78,95]}

     df = pd.DataFrame(data)
     df
```

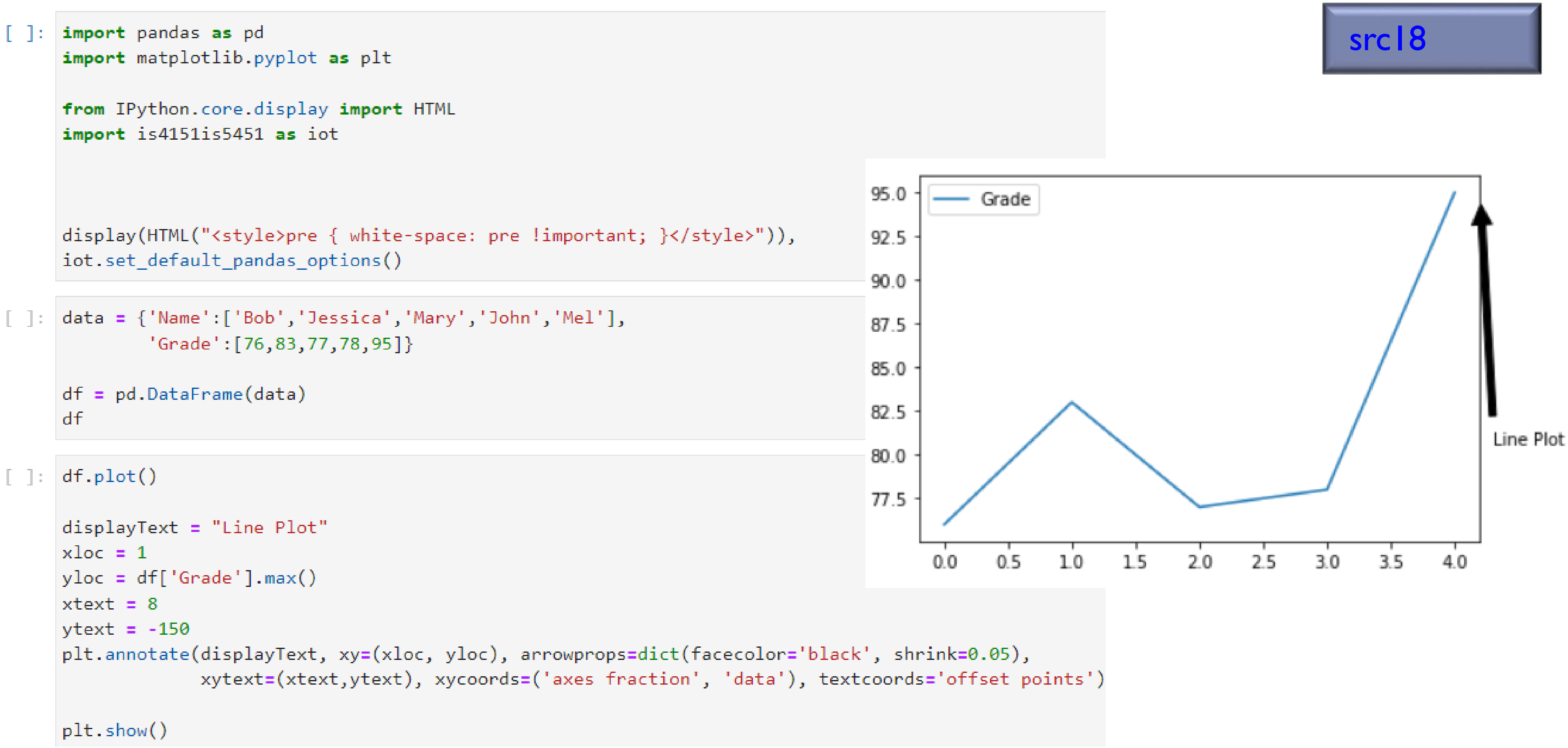
```
[ ]: df.plot()
     plt.show()
```



Data Visualisation – Line Plot (cont.)

- To customise the line plot, we need to use `matplotlib` directly:

src18



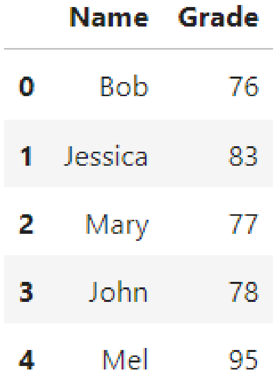
```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt

     from IPython.core.display import HTML
     import is4151is5451 as iot



     display(HTML("<style>pre { white-space: pre !important; }</style>")),
     iot.set_default_pandas_options()
```

```
[ ]: data = {'Name':['Bob','Jessica','Mary','John','Mel'],
             'Grade':[76,83,77,78,95]}

     df = pd.DataFrame(data)
     df
```

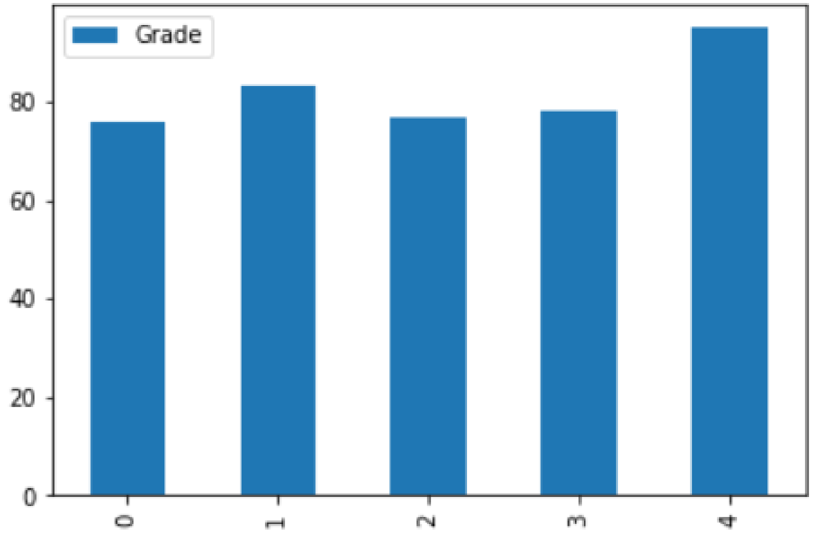
```
[ ]: df.plot()

     displayText = "Line Plot"
     xloc = 1
     yloc = df['Grade'].max()
     xtext = 8
     ytext = -150
     plt.annotate(displayText, xy=(xloc, yloc), arrowprops=dict(facecolor='black', shrink=0.05),
                  xytext=(xtext,ytext), xycoords=('axes fraction', 'data'), textcoords='offset points')

     plt.show()
```

Data Visualisation – Bar Plot

- To create a bar plot, we need to set the `kind` attribute in `DataFrame.plot()` to "`bar`".
- By default, the index of the DataFrame is a zero-based number and thus x-axis are the numbers 0 to 4.
- We can change the index of the DataFrame to the Name column and replot the bar plot.
- See the sample script in `src19`.

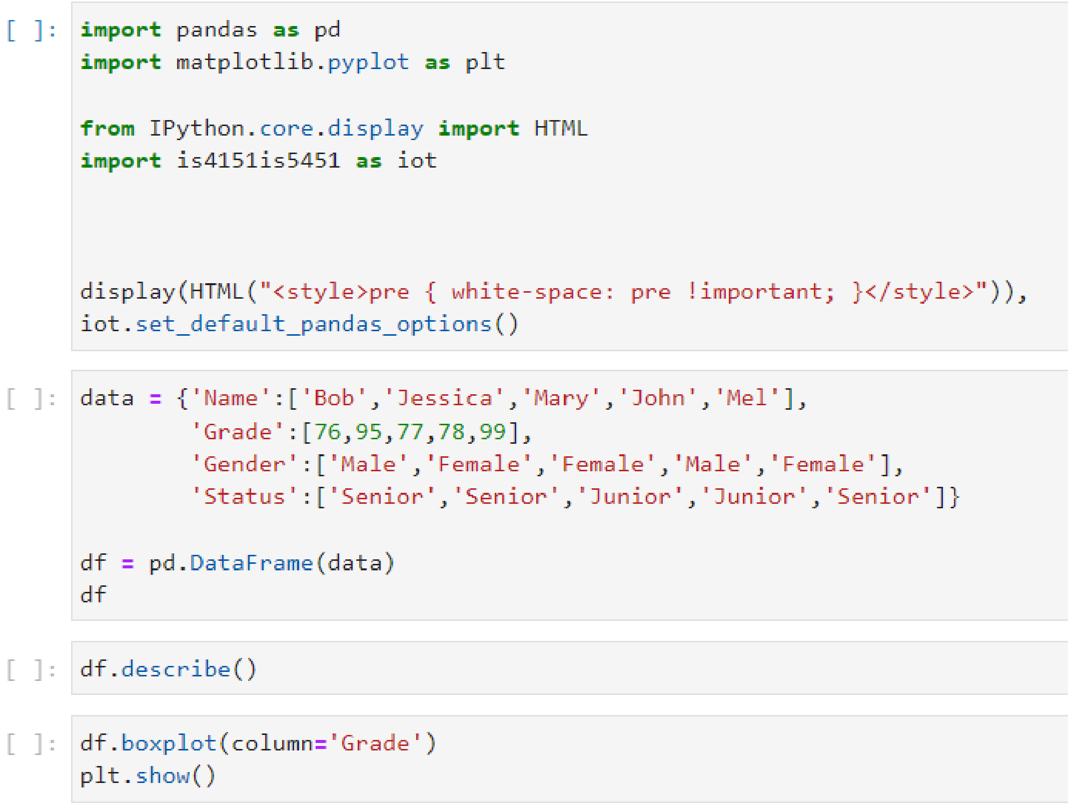


	Name	Grade
0	Bob	76
1	Jessica	83
2	Mary	77
3	John	78
4	Mel	95



Data Visualisation – Box Plot

- A box plot can be created using `DataFrame.boxplot()` and specifying the required column, in this case, "Grade".



```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt

     from IPython.core.display import HTML
     import is4151is5451 as iot



     display(HTML("<style>pre { white-space: pre !important; }</style>")),
     iot.set_default_pandas_options()
```

```
[ ]: data = {'Name':['Bob','Jessica','Mary','John','Mel'],
             'Grade':[76,95,77,78,99],
             'Gender':['Male','Female','Female','Male','Female'],
             'Status':['Senior','Senior','Junior','Junior','Senior']}

     df = pd.DataFrame(data)
     df
```

```
[ ]: df.describe()
```

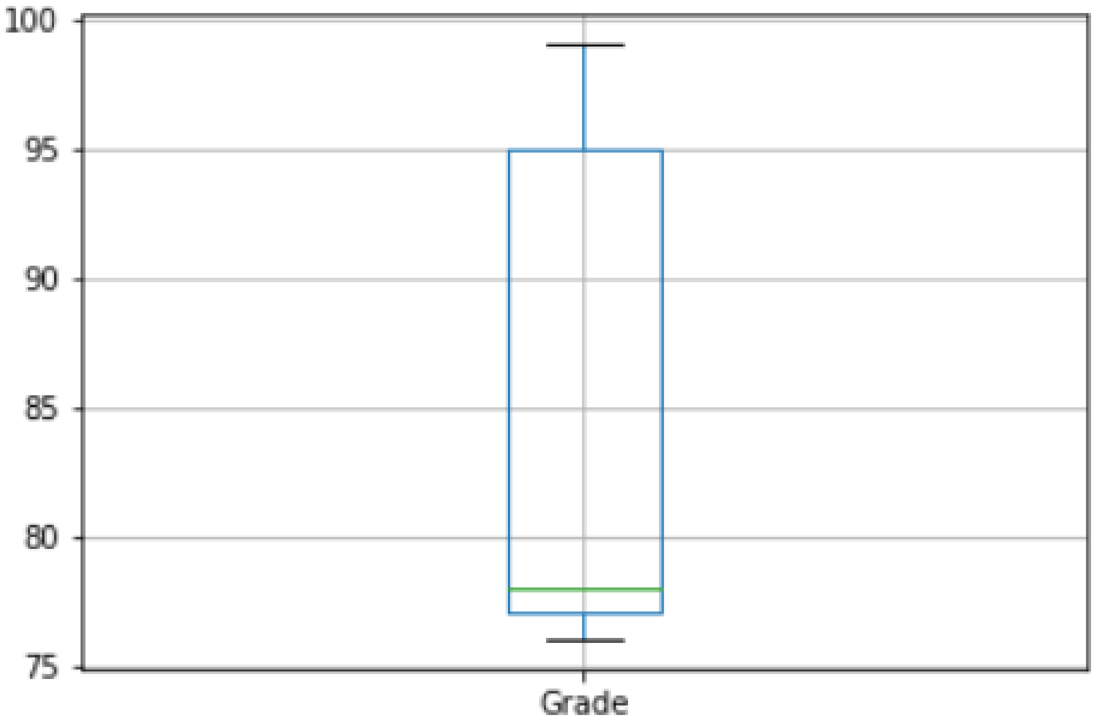
```
[ ]: df.boxplot(column='Grade')
     plt.show()
```

src20

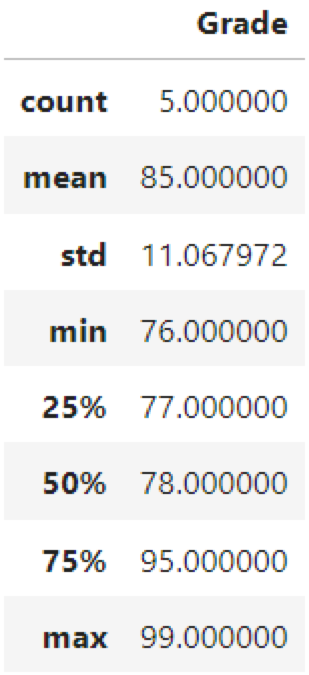
Data Visualisation – Box Plot (cont.)

- A box plot is a method for graphically depicting groups of numerical data through their quartiles:
  - The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2).
  - The whiskers extend from the edges of box to show the range of the data.
  - The position of the whiskers can represent several values:
    - If there is no outlier, the whiskers show the minimum and maximum values of the data.
    - If there are outliers, the whiskers show the:
      - □ Lowest datum still within 1.5 IQR of the lower quartile.
      - □ Highest datum still within 1.5 IQR of the upper quartile.
      - □ IQR = Q3 – Q1.
      - □ Outlier points are those past the end of the whiskers.

Data Visualisation – Box Plot (cont.)



src20



	Grade
count	5.000000
mean	85.000000
std	11.067972
min	76.000000
25%	77.000000
50%	78.000000
75%	95.000000
max	99.000000

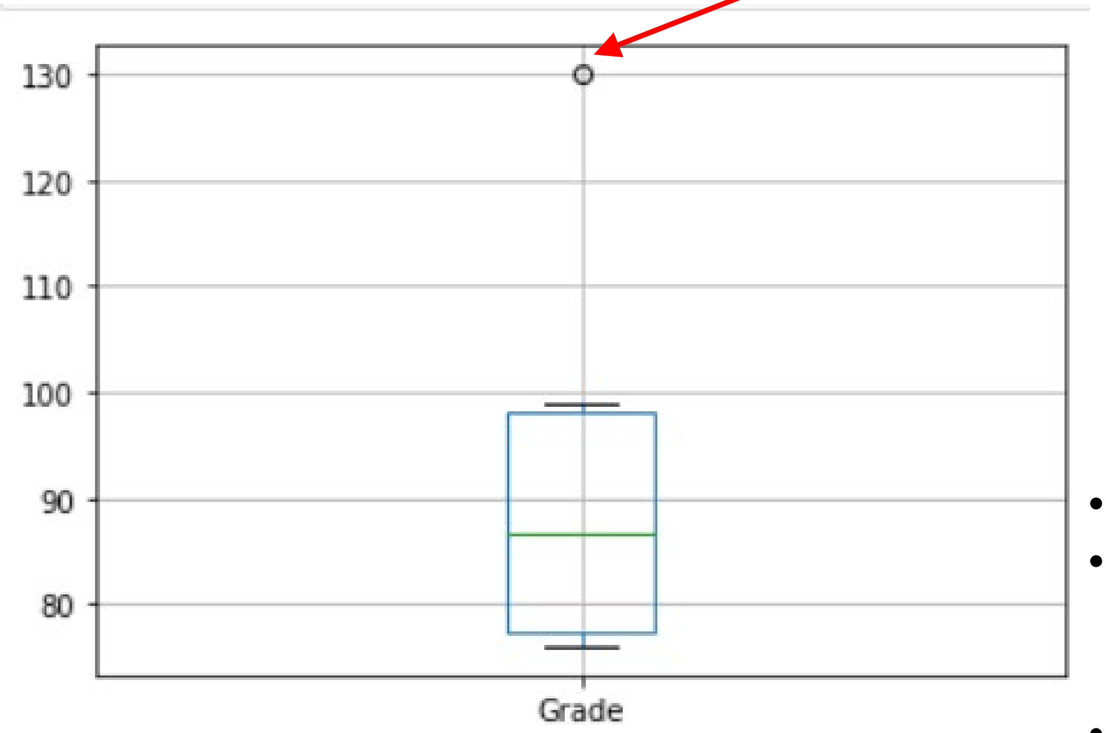
- IQR = 18
- Outliers will fall outside ±1.5 * IQR from Q1 and Q3, i.e., 50 and 122
- The min and max values are within this range.

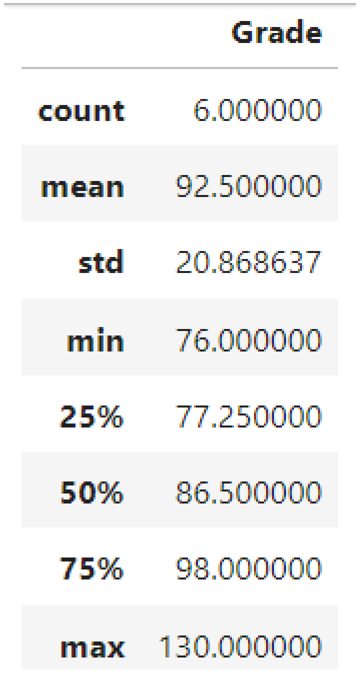
In this example, the whiskers shows the minimum and maximum values since there is no outlier.

Data Visualisation – Box Plot (cont.)

src21

Adding one extra observation with a Grade of 130 will lead to an outlier





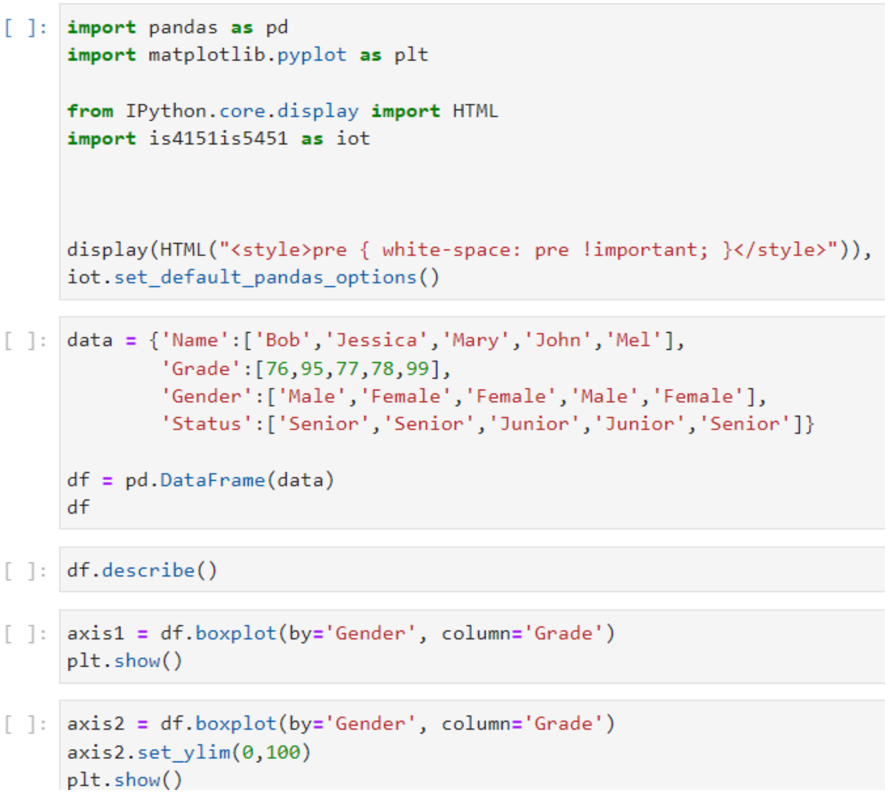
	Grade
count	6.000000
mean	92.500000
std	20.868637
min	76.000000
25%	77.250000
50%	86.500000
75%	98.000000
max	130.000000

- IQR = 20.75
- Outliers will fall outside ±1.5 * IQR from Q1 and Q3, i.e., 46.125 and 129.125
- The max value is outside this range.

In this example, the whiskers shows the lowest datum still within 1.5 IQR of the lower quartile (i.e., 76), and the highest datum still within 1.5 IQR of the upper quartile (i.e., 99).

Data Visualisation – Box Plot (cont.)

- We can create a box plot with categorisation, in this case by Gender.
- We can also adjust the y-axis so that it runs from 0 to 100.



```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt

     from IPython.core.display import HTML
     import is4151is5451 as iot



     display(HTML("<style>pre { white-space: pre !important; }</style>")),
     iot.set_default_pandas_options()
```

```
[ ]: data = {'Name':['Bob','Jessica','Mary','John','Mel'],
             'Grade':[76,95,77,78,99],
             'Gender':['Male','Female','Female','Male','Female'],
             'Status':['Senior','Senior','Junior','Junior','Senior']}

     df = pd.DataFrame(data)
     df
```

```
[ ]: df.describe()
```

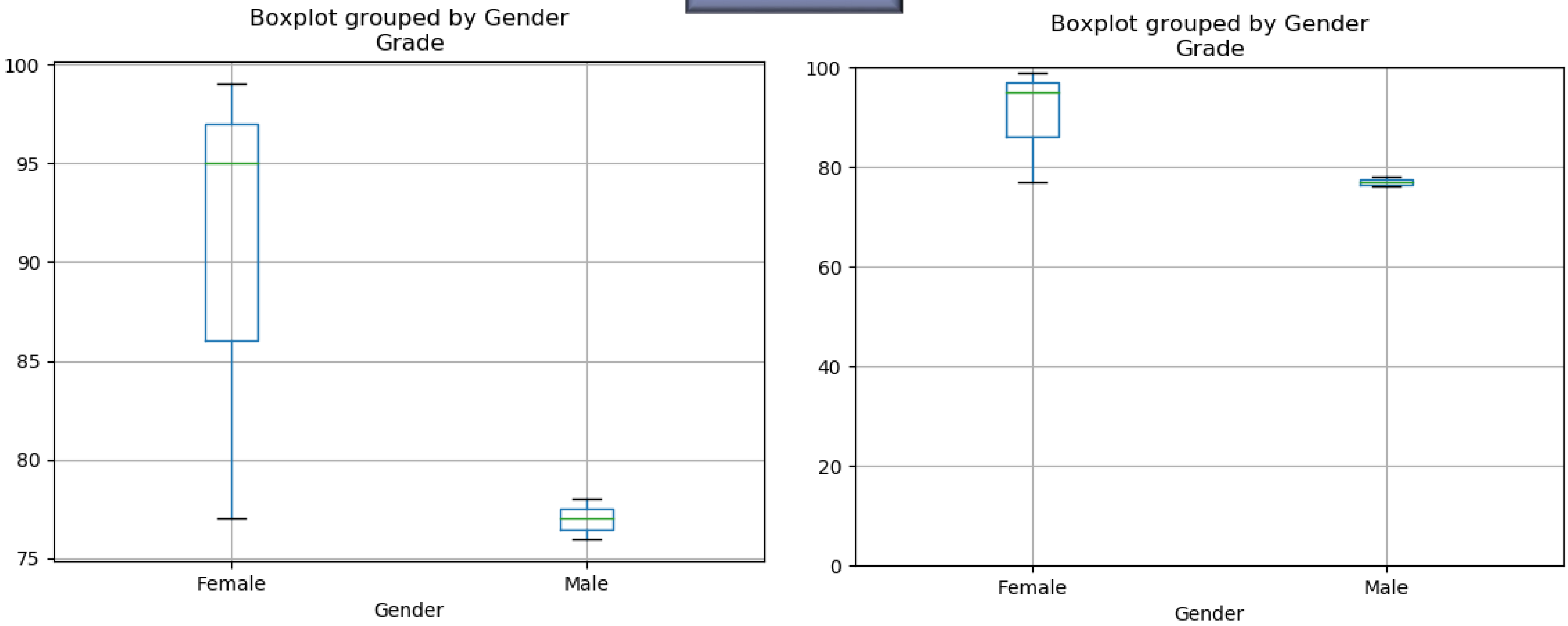
```
[ ]: axis1 = df.boxplot(by='Gender', column='Grade')
     plt.show()
```

```
[ ]: axis2 = df.boxplot(by='Gender', column='Grade')
     axis2.set_ylim(0,100)
     plt.show()
```

src22

Data Visualisation – Box Plot (cont.)

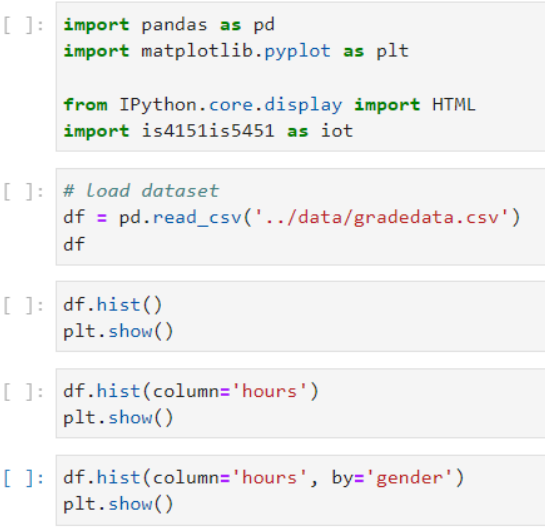
src22



Box plots with categorisation showing. The one on the right has the y-axis limit set to 0-100.

Data Visualisation – Histogram

- A histogram can be created using `DataFrame.hist()`
- By default, the histograms for the columns with numeric values will be generated.
- To generate a histogram for a particular column, you can specify the name of the required column in the `column` attribute, e.g., `df.hist(column='hours')`.



```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt

     from IPython.core.display import HTML
     import is4151is5451 as iot
```

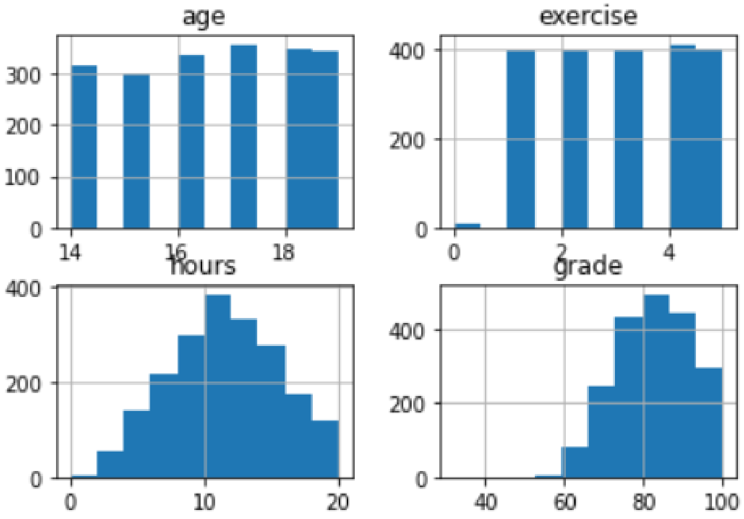
```
[ ]: # load dataset
     df = pd.read_csv('../data/gradedata.csv')
     df
```

```
[ ]: df.hist()
     plt.show()
```

```
[ ]: df.hist(column='hours')
     plt.show()
```

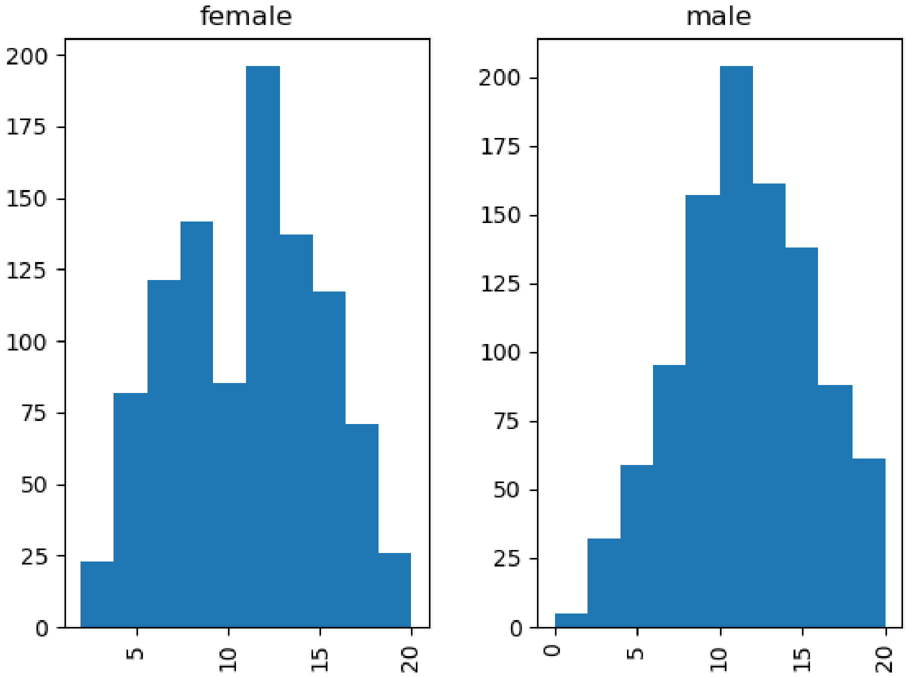
```
[ ]: df.hist(column='hours', by='gender')
     plt.show()
```

src23



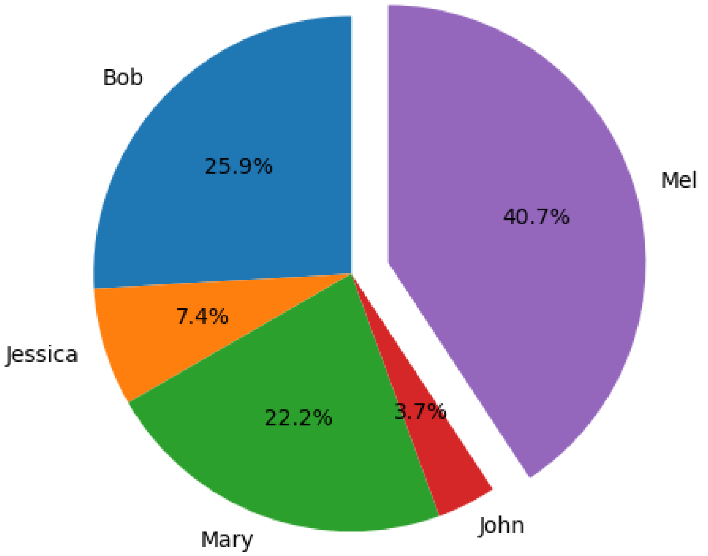
Data Visualisation – Histogram (cont.)

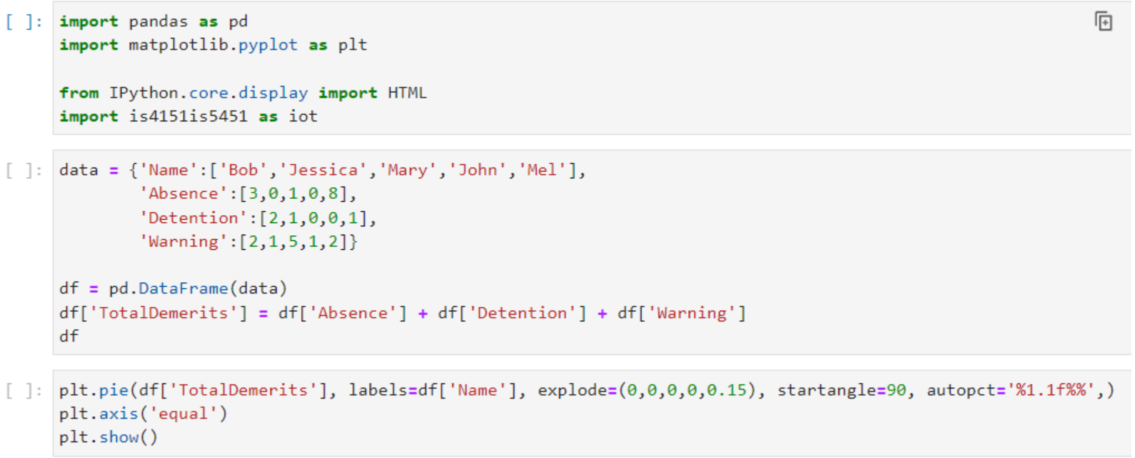
- We can also generated histogram for a numerical column grouped by another column, typically a categorical column, using the by attribute.
- See src23 for a sample script grouped by "gender".



Data Visualisation – Pie Chart

- A pie chart can be created using `pyplot.pie()`.
- Customisations can be made to the pie chart to make it more meaningful.





```
import pandas as pd
import matplotlib.pyplot as plt

from IPython.core.display import HTML
import is4151is5451 as iot
```

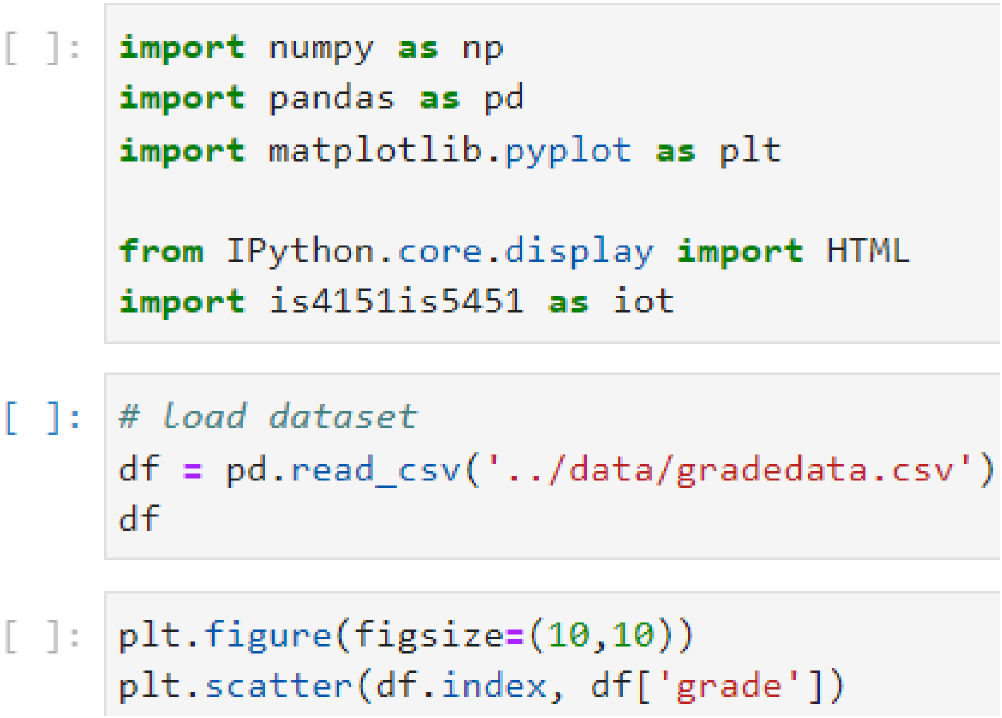
```
data = {'Name':['Bob','Jessica','Mary','John','Mel'],
        'Absence':[3,0,1,0,8],
        'Detention':[2,1,0,0,1],
        'Warning':[2,1,5,1,2]}

df = pd.DataFrame(data)
df['TotalDemerits'] = df['Absence'] + df['Detention'] + df['Warning']
df
```

```
plt.pie(df['TotalDemerits'], labels=df['Name'], explode=(0,0,0,0,0.15), startangle=90, autopct='%1.1f%%',)
plt.axis('equal')
plt.show()
```

src24

Data Visualisation – Scatter Plot

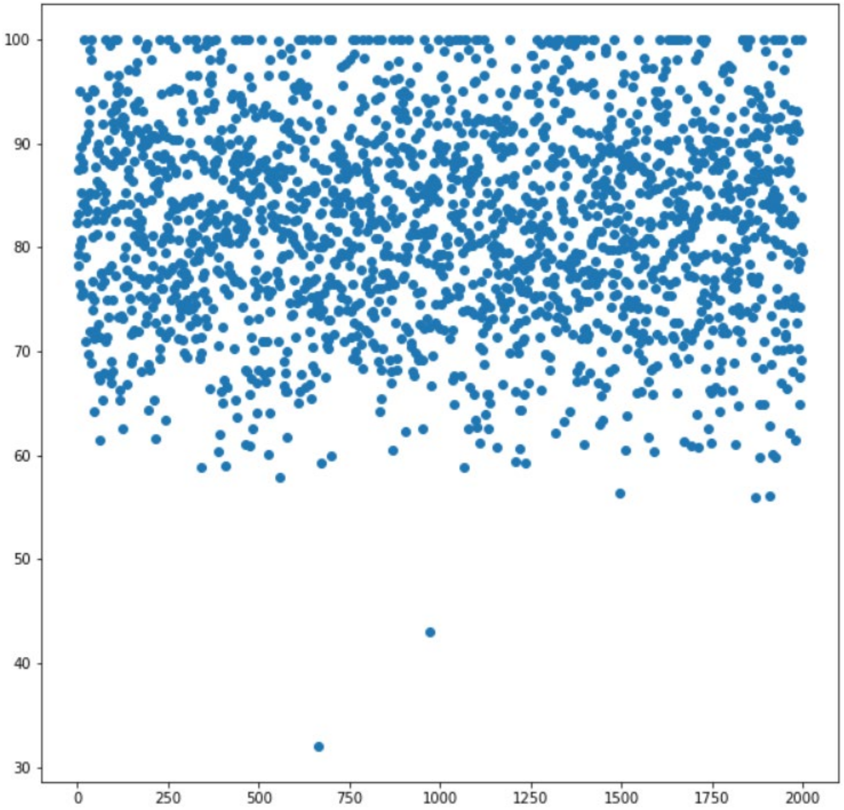
- A scatter plot can be created using `pyplot.scatter()`.



```
[ ]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt

     from IPython.core.display import HTML
     import is4151is5451 as iot
```

```
[ ]: # Load dataset
     df = pd.read_csv('../data/gradedata.csv')
     df
```

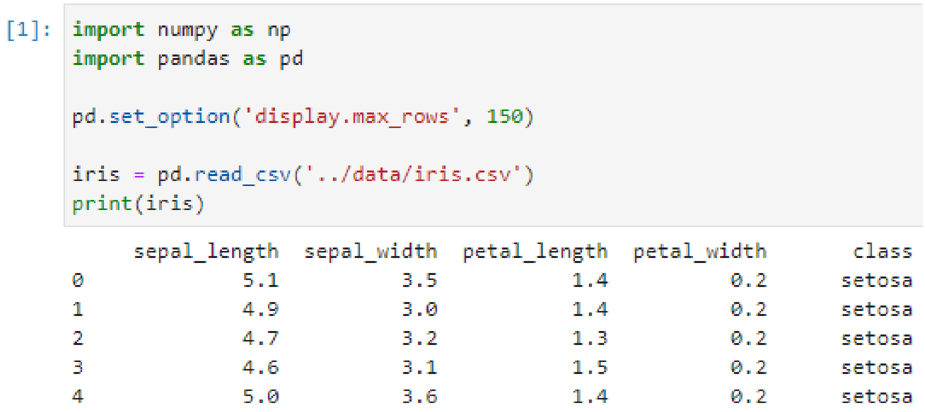
```
[ ]: plt.figure(figsize=(10,10))
     plt.scatter(df.index, df['grade'])
```



src25

Data Visualisation – Scatter Plots of the Iris Flower Dataset

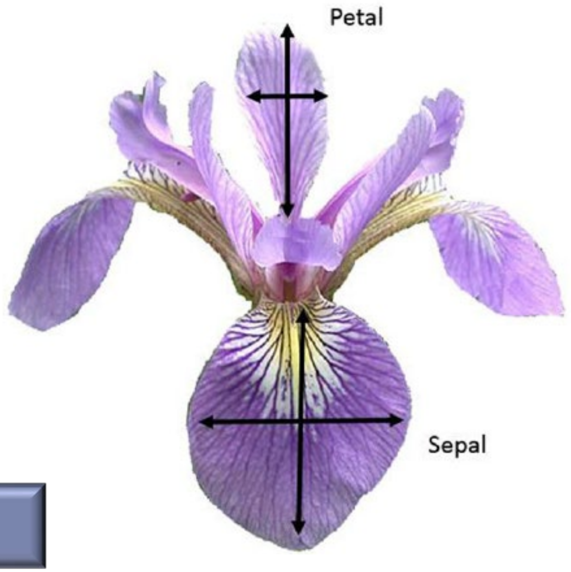
- This dataset set contains data from three different species of iris (Iris silky, virginica Iris and Iris versicolor).
- The variables include the length and width of the sepals, and the length and width of the petals.
- This dataset is widely used for classification problems.
- 150 observations with 4 independent attributes and one target attribute.



```
[1]: import numpy as np
     import pandas as pd

     pd.set_option('display.max_rows', 150)

     iris = pd.read_csv('../data/iris.csv')
     print(iris)
```

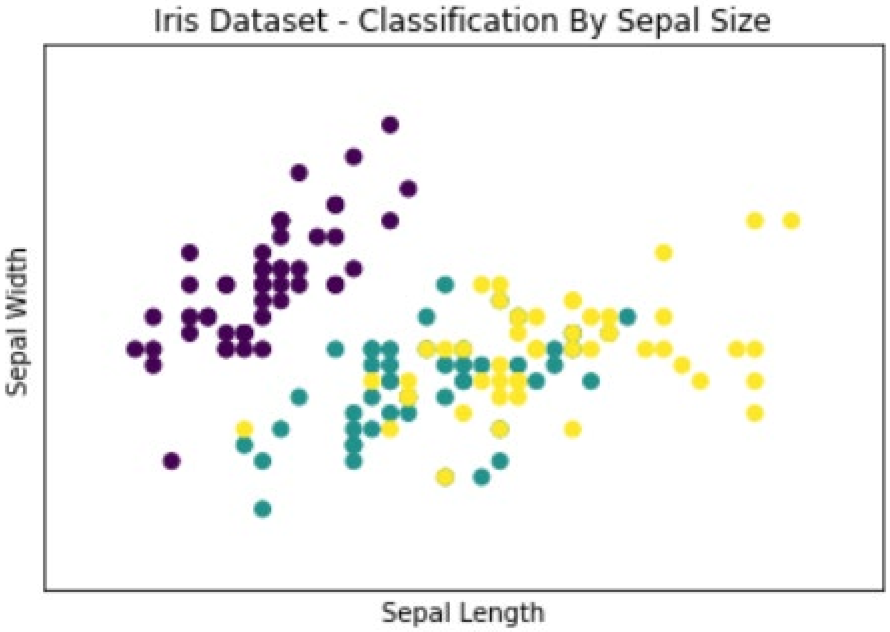
```
     sepal_length  sepal_width  petal_length  petal_width   class
0             5.1          3.5           1.4          0.2  setosa
1             4.9          3.0           1.4          0.2  setosa
2             4.7          3.2           1.3          0.2  setosa
3             4.6          3.1           1.5          0.2  setosa
4             5.0          3.6           1.4          0.2  setosa
```

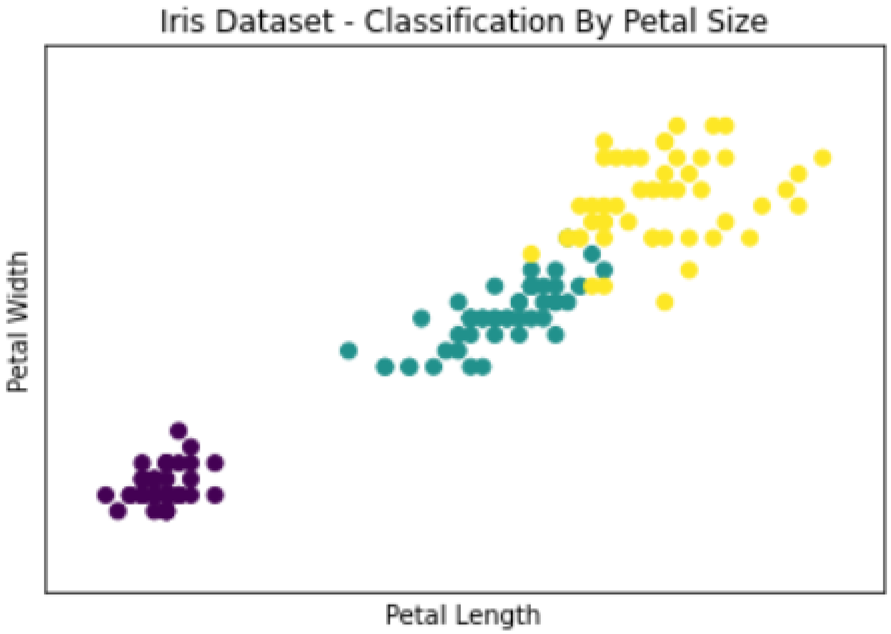
src26



Data Visualisation – Scatter Plots of the Iris Flower Dataset (cont.)

- Which variables are better for predicting iris species?
  - src27 – Scatterplot of sepal sizes.
  - src28 – Scatterplot of petal sizes.

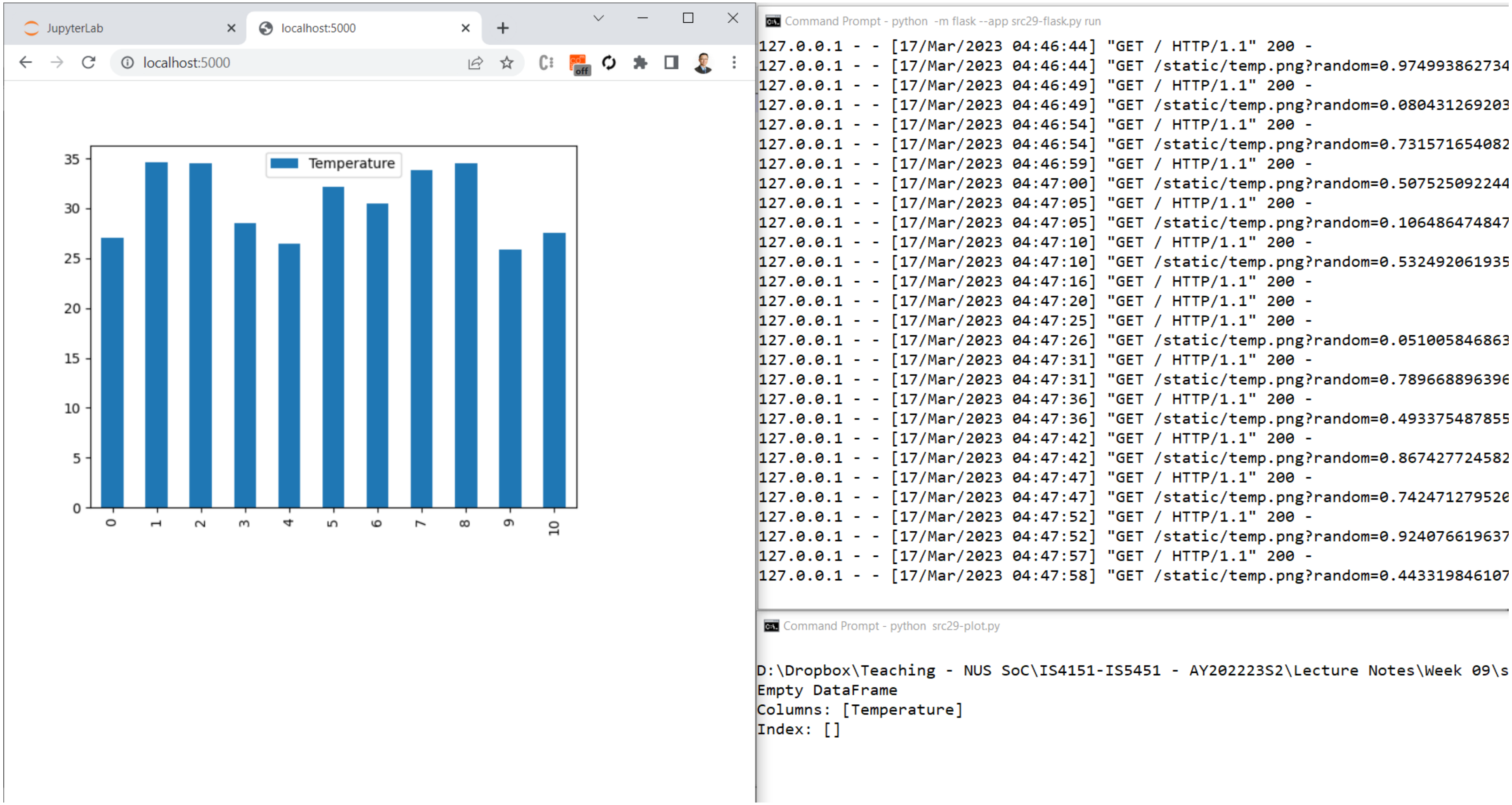




Integrating Matplotlib into an IoT Application

- We can use Matplotlib to generate data visualisations for our IoT sensor data:
  - Export the visualisations to image files.
  - Serve the image files via a Flask web application.
  - This approach can be used to create a simple data dashboard.
- Refer to the following sample source files:
  - `src29-plot.py`
  - `src29-flask.py`

Integrating Matplotlib into an IoT Application (cont.)


JupyterLab
localhost:5000
localhost:5000
Temperature
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
Command Prompt - python -m flask --app src29-flask.py run
Command Prompt - python src29-plot.py

```
127.0.0.1 - - [17/Mar/2023 04:46:44] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:46:44] "GET /static/temp.png?random=0.974993862734
127.0.0.1 - - [17/Mar/2023 04:46:49] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:46:49] "GET /static/temp.png?random=0.080431269203
127.0.0.1 - - [17/Mar/2023 04:46:54] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:46:54] "GET /static/temp.png?random=0.731571654082
127.0.0.1 - - [17/Mar/2023 04:46:59] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:00] "GET /static/temp.png?random=0.507525092244
127.0.0.1 - - [17/Mar/2023 04:47:05] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:05] "GET /static/temp.png?random=0.106486474847
127.0.0.1 - - [17/Mar/2023 04:47:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:10] "GET /static/temp.png?random=0.532492061935
127.0.0.1 - - [17/Mar/2023 04:47:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:20] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:26] "GET /static/temp.png?random=0.051005846863
127.0.0.1 - - [17/Mar/2023 04:47:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:31] "GET /static/temp.png?random=0.789668896396
127.0.0.1 - - [17/Mar/2023 04:47:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:36] "GET /static/temp.png?random=0.493375487855
127.0.0.1 - - [17/Mar/2023 04:47:42] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:42] "GET /static/temp.png?random=0.867427724582
127.0.0.1 - - [17/Mar/2023 04:47:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:47] "GET /static/temp.png?random=0.742471279526
127.0.0.1 - - [17/Mar/2023 04:47:52] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:52] "GET /static/temp.png?random=0.924076619637
127.0.0.1 - - [17/Mar/2023 04:47:57] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [17/Mar/2023 04:47:58] "GET /static/temp.png?random=0.443319846107
```

```
D:\Dropbox\Teaching - NUS SoC\IS4151-IS5451 - AY202223S2\Lecture Notes\Week 09\s
Empty DataFrame
Columns: [Temperature]
Index: []
```

Summary

- Data mining is a process that extracts useful knowledge (or patterns) from data using statistical, mathematical and artificial intelligence techniques.
- Data mining tasks may be classified into prediction, association and segmentation.
- Each data mining task can be implemented with various algorithms.
- Python provides many data science libraries that can be used for IoT data analytics.
- Data preparation helps us to pre-process the data for the subsequent analysis.

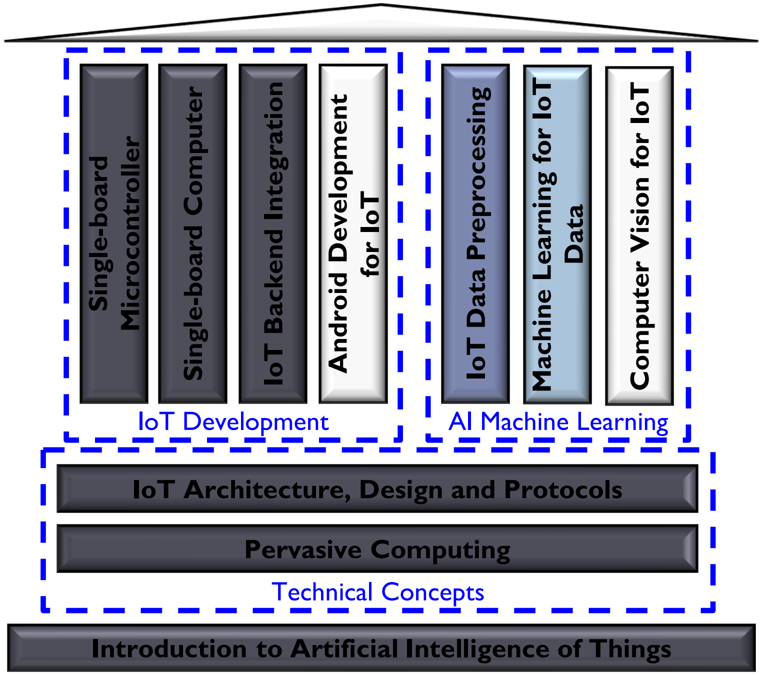
Summary (cont.)

- Data visualisation helps us to better understand the data and also function as a data analytics output by itself.

Q&A

Next Lecture…

- Learn about:
  - How to perform prediction with regression analysis.
  - How to perform prediction with classification.


Single-board Microcontroller
Single-board Computer
IoT Backend Integration
Android Development for IoT
IoT Data Preprocessing
Machine Learning for IoT Data
Computer Vision for IoT
IoT Development
AI Machine Learning
IoT Architecture, Design and Protocols
Pervasive Computing
Technical Concepts
Introduction to Artificial Intelligence of Things