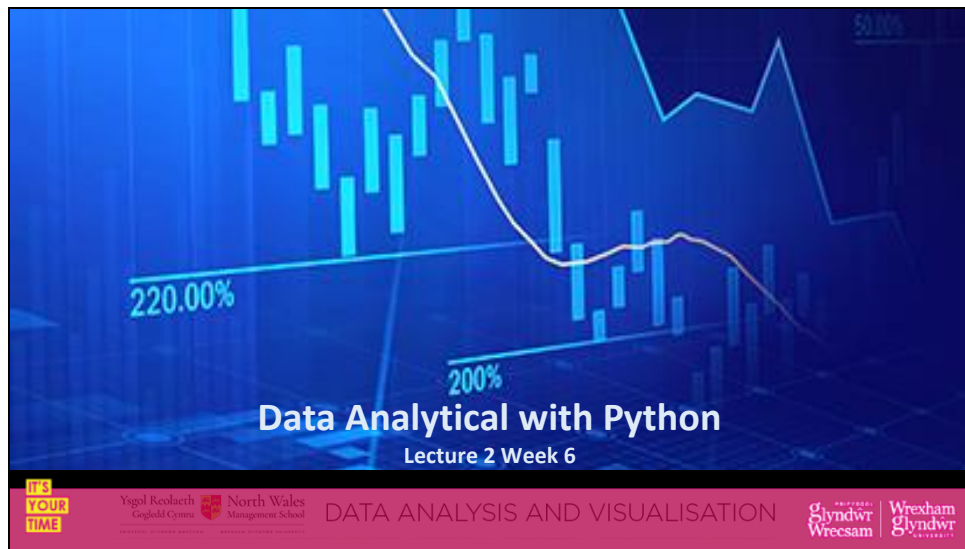


Slide 1



Welcome to lecture 2 , week 6 of data analysis and visualisation.

Python ecosystem for Data Science

- Most popular Python Libraries
 - Python Analytical libraries:
 - NumPy
 - It introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
 - It provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
 - It is fundamental as many other python libraries are built on top of NumPy
 - See the link for more information: <http://www.numpy.org/>

IT'S YOUR TIME Ysgol Rhoelach Gogledd Cymru North Wales Management School DATA ANALYSIS AND VISUALISATION Glyndwr Wrexham Wrexham Glyndwr University

The Python ecosystem of libraries, frameworks, and tools is enormous and growing. Python is used for web scraping, data analysis, web development, internet of things development (IoT), machine learning, DevOps, general scientific computing, and many other computing and scripting uses.

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Python ecosystem for Data Science

- Most popular Python Libraries
 - Python Analytical libraries:
 - NumPy
 - SciPy
 - It includes a collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
 - It is part of SciPy Stack
 - It is build on top of NumPy
 - See the link for more information: <https://www.scipy.org/scipylib/>



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

SciPy is a set of open source scientific and numerical tools for Python. It currently supports special functions, integration, ordinary differential equation (ODE) solvers, gradient optimization, parallel programming tools, an expression-to-C++ compiler for fast execution, and others

Python ecosystem for Data Science

- Most popular Python Libraries
 - Python Analytical libraries:
 - NumPy
 - SciPy
 - Pandas
 - It adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
 - It provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
 - It has limited visualisation capabilities
 - It does also allow handling missing data
 - See the link for more information: <http://pandas.pydata.org/>



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

Pandas is another software library written for the Python programming language, it's used for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Python ecosystem for Data Science

- Most popular Python Libraries
 - Python Analytical libraries:
 - NumPy
 - SciPy
 - Pandas
 - Python Visualisation libraries:
 - Matplotlib
 - It is python 2D plotting library such as line plots, scatter plots, barcharts, histograms, pie charts etc ,which produces publication quality figures in a variety of hardcopy formats
 - It is relatively low-level. It required some effort needed to create advanced visualisation
 - See the link for more information: <https://matplotlib.org/>



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits.

Python ecosystem for Data Science

- Most popular Python Libraries
 - Python Analytical libraries:
 - NumPy
 - SciPy
 - Pandas
 - Python Visualisation libraries:
 - Matplotlib
 - *Seaborn*
 - It is build on top of matplotlib and similar style to the popular ggplot2 library in R
 - It provides high level interface for drawing attractive statistical graphics
 - See the link for more information: <https://seaborn.pydata.org/>



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Hands on with Python Libraries

- Start Jupyter notebook in your system
 - Similar command if you are accessing python differently
- You need to load any Python Library you wish to use first but how?
 - In a cell in Jupyter notebook:
 - All are available in Jupyter notebook so you don't need to download & install them 😊

```
In [1]: #Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

You must import the various libraries you want to use at the start of your code.

Starting with Pandas

- Create Series() in Pandas
 - A Pandas series can be created using the constructor pandas.Series
 - First create a Series() from a dictionary, say {'X': 0., 'Y': 1., 'Z': 2.}
 - Then can create a Series() data of size say 4 from a scalar value, say 10.
 - Using index is mandatory
 - See the code snips for how to use them & output below:

```
In [1]: In [1]: M data = {'X': 0., 'Y': 1., 'Z': 2.}
#Import Python Libraries
import pandas as pd
#Create a series from a dictionary
Series1 = pd.Series(data)# pandas.Series constructor
print (Series1)
#Create series data from a scalar value
Index is mandatory.
Series2 = pd.Series(10, index=[0, 1, 2, 3])
print (Series2)
```

```
X    0.0
Y    1.0
Z    2.0
dtype: float64
0    10
1    10
2    10
3    10
dtype: int64
```



- Create Series() by using NumPy
 - NumPy does NOT have a constructor for Series()
 - A NumPy series can be created using NumPy functions
 - For example from randn() function in NumPy
 - creating a series of random values of 5 rows and 5 columns.
 - See the code snips for how to use them and its output below:

```
In [2]: import numpy as np
Series3 = np.random.randn(5, 5)
print(Series3.values, Series3)

Series3
[[ 0.25115324 -0.39602975 -1.70363325  0.23515928 -0.50743044]
 [ -0.44469959  0.64625951  0.75712661  1.10598361  0.8771257 ]
 [ 0.7365575  1.14679317 -2.3315012 -1.82787731 -0.79647379]
 [ -0.56114978  1.13796778  0.05870405 -1.85228122  0.81569823]
 [ 0.22417737  1.47138007  1.02504262  0.16153321  0.61124134]]
```

NumPy and Pandas

- Create Series() by using NumPy
 - use various function definitions in NumPy, say array()
- Then convert them to Series() using Pandas
 - You can add new index to data as demonstrated below
 - See the code snips below to see how to use them & output:

```
In [3]: # Import Python Libraries
import numpy as np
data= np.array(['B','i','g',' ', 'D', 'A', 'T','A'])
# Import Python Libraries
import pandas as pd
# without adding index
Series4 = pd.Series(data)
print ("Series4\n", Series4)
# with adding index
Series5 = pd.Series(data,index=[0,10,20,30,40,50,60,70])
print ("\nSeries5\n", Series5)
```

```
Series4
0    B
1    i
2    g
3
4    D
5    A
6    T
7    A
dtype: object

Series5
0    B
10   i
20   g
30
40   D
50   A
60   T
70   A
dtype: object
```



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

STFC Visualisation 2014 University of Leeds

To Create Series() by using NumPy we use various function definitions in NumPy , say array()
Then convert them to Series() using Pandas.

Pandas

- Accessing data in a Series
 - Index to retrieve data element using [], e.g.; Series[2]
 - See examples below for different scenarios

```
In [4]: #Import Python Libraries
import pandas as pd
#forming a character series: O, a, b, c, d, e
Series6 = pd.Series(['O','a','b','c','d','e'])
print ("Example 1:Retrieve the first element")
print (Series6[0] )
print ("Example 1:Retrieve the last element")
print (Series6[5] )
print ("Example 2:Retrieve the first three elements")
print (Series6[:3])
print ("Example 3:Retrieve the last three elements")
print (Series6[-3:])
print ("Example 4:Retrieve a single element")
print (Series6[3])
print ("Example 5:Retrieve multiple elements")
print (Series6[[1,3,0]])
```

```
Example 1:Retrieve the first element
0
Example 1:Retrieve the last element
e
Example 2:Retrieve the first three elements
0    O
1    a
2    b
dtype: object
Example 3:Retrieve the last three elements
3    c
4    d
5    e
dtype: object
Example 4:Retrieve a single element
c
Example 5:Retrieve multiple elements
1    a
3    c
0    O
dtype: object
```

Like lists, you can access a series data via its index value. examples on screen demonstrate different methods of accessing a series of data.

The first example demonstrates retrieving a specific element with index 0.

The second example retrieves indices 0, 1, and 2.

The third example retrieves the last three elements since the starting index is -3 and moves backward to -2, -1.

The fourth and fifth examples retrieve data using the series index labels.

Operations: NumPy and Pandas

- Operations on Series()

- Most of operations in lists, tuple are also valid for series
- Copy by reference using assignment sign "="
 - any changes to the series will adapt to the other one

```
In [7]: print("Series6"), print(Series6)
Series_06 = Series6
Series_06.index = ['A', 'B', 'C', 'D', 'E', 'F']
print("Series6"), print(Series6)
```

```
Series6
0    o
1    a
2    b
3    c
4    d
5    e
dtype: object
Series6
A    o
B    a
C    b
D    c
E    d
F    e
dtype: object
```

```
In [5]: 'p' in Series6
Out[5]: False
```

```
In [6]: len(Series6)
Out[6]: 6
```

- Copy by values using ".copy()" method
 - Only values will be copied not reference

```
In [8]: Series_006 = Series6.copy()
Series_006.index = [0, 1, 2, 3, 4, 5]
print("Series_006"), print(Series_006)
```

```
Series_006
A    o
B    a
C    b
D    c
E    d
F    e
dtype: object
```



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

Numerous operations can be implemented on series data. You can check whether an index value is available in a series or not.

Also, you can check all series elements against a specific condition, such as if the series value is less than 8 or not. In addition, you can perform math operations on series data directly or via a defined function.

Analysing with NumPy and Pandas

- Statistical Analysis (EDA) of a data Series
 - Various statistical methods are available:

```
In [9]: import numpy as np
Series3 = np.random.randn(10)
print ("Series3's average value is ", Series3.mean())
print ("Series3's maximum value is ", Series3.max())
print ("Series3's minimum value is ", Series3.min())
print ("Series3's standard deviation value is ", Series3.std())
```

```
Series3's average value is -0.169069604897085
Series3's maximum value is 1.8366489746223058
Series3's minimum value is -1.999786621415704
Series3's standard deviation value is 1.2408996580344474
```

- describe() is only available for Pandas (not NumPy)
 - Use pd.Series(Series3) convert it to Pandas series

```
In [10]: import pandas as pd
Series7 = pd.Series(Series3)
print("describe Series7:", print(Series7.describe()))
```

```
describe Series7:
count    10.000000
mean     -0.169070
std       1.240823
min      -1.999787
25%      -1.186528
50%      -0.203576
75%       0.886928
max       1.836649
dtype: float64
```

- If Series3 was 2d, above code won't work!

Numerous statistical methods can be applied directly on a data series. On screen we demonstrate the calculation of mean, max, min, and standard deviation of a data series. Also, the .describe() method can be used to give a data description, including quantiles.

Data Frame in Pandas

- Creating a Data Frame
 - 2-dimensional data structure with heterogeneous data types, like tabular data.
- A Pandas data frame can be created using the constructor `pandas.DataFrame(data, index, columns, dtype, copy)`
 - data can be from lists, series, dictionaries, Numpy arrays, or other data frames

```
In [11]: import pandas as pd
data = [ {'Test1': 10, 'Test2': 20}, {'Test1': 30,
'Test2': 20, 'Test3': 20}]
# With three column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['First', 'Second'])
print (df1)
```

	Test1	Test2	Test3
First	10	20	NaN
Second	30	20	20.0

Data Frame it is a two-dimensional data structure with heterogeneous data types, i.e., tabular data.

Pandas can create a data frame using the constructor `pandas.DataFrame(data, index, columns, dtype, copy)`.

A data frame can be created from lists, series, dictionaries, Numpy arrays, or other data frames.

A Pandas data frame not only helps to store tabular data but also performs arithmetic operations on rows and columns of the data frame.

You can create a data frame from dictionaries or arrays, as shown in code snip.

Also, you can set the data frame indices.

However, if you don't set the indices, then the data frame starts with 0 and goes up to n-1, where n is the length of the list.

Column names are taken by default from the dictionary keys.

However, it's possible to set labels for columns as well.

Read in CVS Data Locally

- Read in data from Excel
 - Use Pandas Data Frame and built in .read_csv() method
 - It accepts a file name (relative path) as its argument
 - Use .head() method to display its 5 first entry

```
In [12]: import pandas as pd
df = pd.read_csv("Salaries.csv")
df.head()
```

Out[12]:

	rank	discipline	phd	service	gender	salary
0	Prof	Eng	46	40	Female	200020
1	Prof	Eng	14	7	Male	110000
2	Prof	Eng	21	21	Male	120340
3	Prof	comp	48	28	Female	151000
4	Prof	comp	30	19	Male	110000

	rank	discipline	phd	service	gender	salary
1	rank	discipline	phd	service	gender	salary
2	Prof	Eng	46	40	Female	200020
3	Prof	Eng	14	7	Male	110000
4	Prof	Eng	21	21	Male	120340
5	Prof	comp	48	28	Female	151000
6	Prof	comp	30	19	Male	110000

Read in Data from clouds

- What if the excel file was in clouds?
 - You can still use Pandas Data Frame and built in `.read_csv()` method
 - It accepts URL point to the file
 - Example:

```
df = pd.read_csv("http://exempl.com/example.csv")
```

 where "example.csv" is the file name and <http://exempl.com/> is the URL pointing to the file.
you can then use all related data frame methods such as `head()` and `tail()`
 - You can also retrieve any files over HTTP/HTTPS by importing "requests"
 - Example:

```
import requests  
response = requests.get("http://marsweather.ingenology.com/v1/latest/?format=json")  
print(response.text)
```



What if the excel file was in clouds?

You can still use Pandas Data Frame and built in `.read_csv()` method

It accepts URL point to the file

Example:

```
df = pd.read_csv("http://exempl.com/example.csv")
```

where "example.csv" is the name of the file

and <http://exempl.com/> is the URL pointing to the file.

you can then use all related data frame methods such as `head()` and `tail()`

You can also retrieve any files over HTTP/HTTPS by importing "requests"

Read in Document Databases

- Store semi-structured data in the form of documents
 - Non-relational databases ("**NoSQL databases**") such as **MangoDB**
 - Key-value databases are the simplest form of NoSQL databases.
 - Different standards such as JSON, XML, BSON or YAML.
 - JSON: JavaScript Object Notation, consists of only two structures: key-value pairs
 - Very similar to Python dictionaries and ordered lists of values, i.e., arrays
 - Use the JSON loads() Method to translate to dictionary by importing Jason

```
import json
import requests
response = requests.get("http://marsweather.ingenology.com/v1/latest/?format=json")
weather = json.loads(response.text)
weather # produce text on weather
weather['report']['sol'] # produce the requested element
```



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

JSON, which stands for JavaScript Object Notation, dates to 1999. It consists of only two structures: key-value pairs, called *structures*, that are very similar to Python dictionaries; and ordered lists of values, called *arrays*, that are very much like Python lists.

JSON is so common that most languages have features to translate JSON to and from native data types. In the case of Python, that feature is the json module, which became part of the standard library with version 2.6.

When you know the URL to use, you can use the requests library to fetch data from an API and either process it on the fly or save it to a file for later processing. The simplest way to do this is exactly like retrieving a file: as already discussed.

Analysis for Data Frame

- Statistical Analysis (EDA) of a Data Frame

`DataFrame.describe(percentiles=None, include=None, exclude=None)`

Apply it to Salaries.csv
to analysis all numeric entries:

Or only to one field say salary

```
In [14]: df.salary.describe()

Out[14]: count      78.000000
         mean    108023.782051
         std     28293.661022
         min      57800.000000
         25%     88612.500000
         50%    104671.000000
         75%    126774.750000
         max    186960.000000
         Name: salary, dtype: float64
```

```
In [13]: import pandas as pd
         df = pd.read_csv("Salaries.csv")
         df.describe()

Out[13]:
```

	phd	service	salary
count	78.000000	78.000000	78.000000
mean	19.705128	15.051282	108023.782051
std	12.498425	12.139768	28293.661022
min	1.000000	0.000000	57800.000000
25%	10.250000	5.250000	88612.500000
50%	18.500000	14.500000	104671.000000
75%	27.750000	20.750000	126774.750000
max	56.000000	51.000000	186960.000000

Pandas provides various methods for analyzing data in a data frame. The `.describe()` method is used to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a data set's distribution.

`DataFrame.describe()` analyzes both numeric and object series, as well as data frame column sets of mixed data types. The output will vary depending on what is provided.

Panel in Pandas

- Creating a Panel

- It is a three-dimensional data structure like a three-dimensional array.

Constructor: `pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)`

```
In [15]: import pandas as pd
import numpy as np
data = np.random.rand(5,5,5)
Paneldata = pd.Panel(data)
print (Paneldata.describe())

<ipython-input-15-b58ac8ee7df8>:4: FutureWarning: The Panel class is
removed from pandas. Accessing it from the top-level namespace will
also be removed in the next version
Paneldata = pd.Panel(data)
```

- It will be removed from the Library soon so it won't be discussed any further.



Ysgol Rhoelach
Gogledd Cymru

North Wales
Management School

DATA ANALYSIS AND VISUALISATION

Glyndwr
Wrexham

Wrexham
Glyndwr
University

A *panel* is a three-dimensional data structure like a three-dimensional array.

Pandas creates a panel using the constructor `pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)`.

The panel can be created from a dictionary of data frames and narrays.

The data can take various forms, such as ndarray, series, map, lists, dictionaries, constants, and also another data frames.

It will be removed from the Library soon, so it won't be discussed any further.