Python Primer Sept 2025-Part2

Simon Tomlinson

simon.tomlinson@ed.ac.uk

Using Libraries in Python

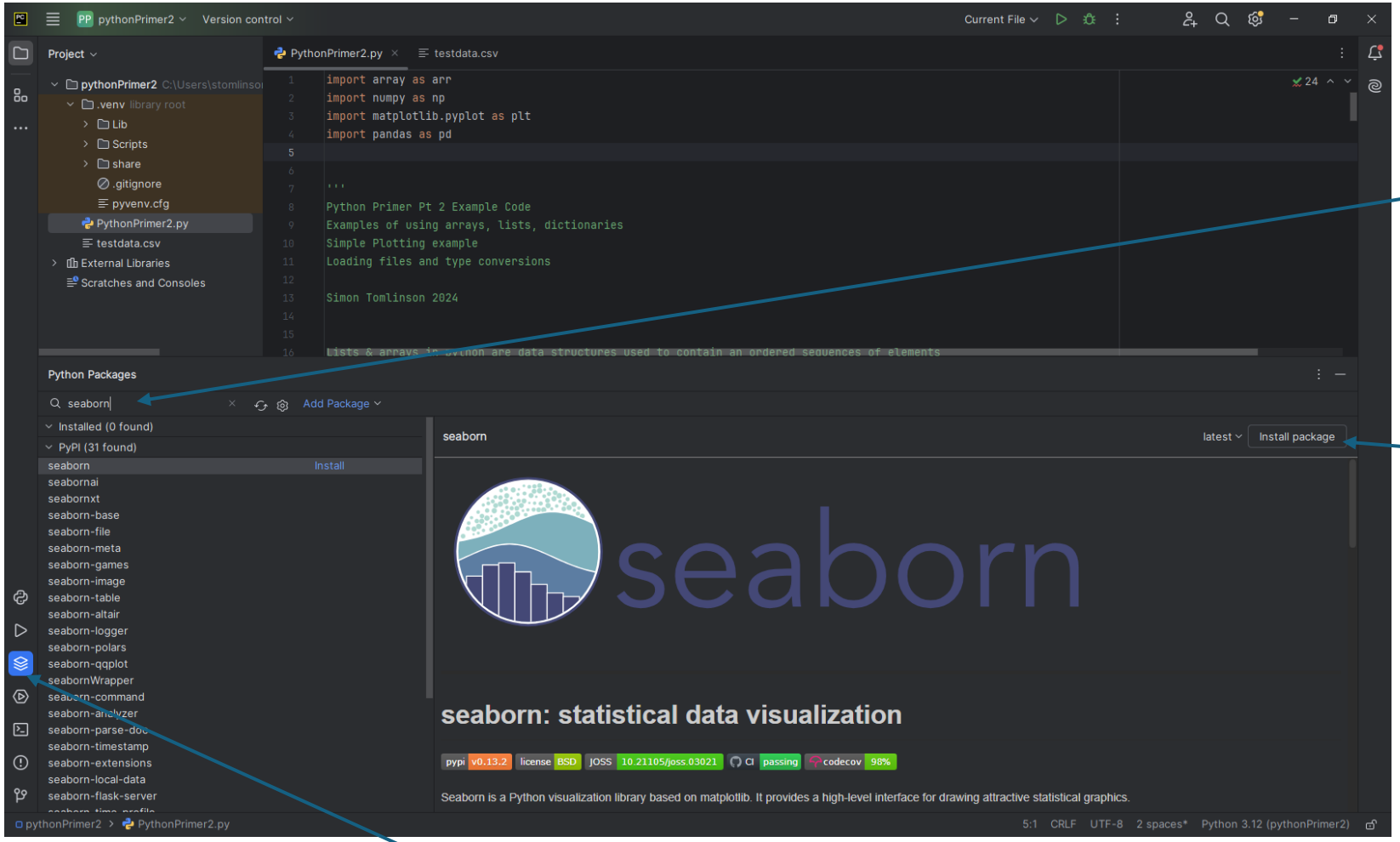
```
import array as arr

import numpy as np

import matplotlib.pyplot as plt

import pandas
```

- We can import functionality from existing libraries into our python script using an “import” statement
- There are different forms example are shown here
  - `import pandas` – imports all the functionality provided by the library called `pandas`
  - `import array as arr` – imports all the functionality from the array library and access it through `arr`
  - `import matplotlib.pyplot as plt` – import just `pyplot` from `matplotlib` and access it as `plt`
- Some libraries come built-in to Python, like array, others need installing
- Importing subsets of functionality is more efficient, using “`as`” stops the same named element provided by different packages conflicting

Installing Libraries

- We can install libraries by downloading the library and installing individually- but this is quick tricky to do in some cases!
- We can use an online repository (a collection of packages)
- https://pypi.python.org/simple is the URL for the Pypi repository, a commonly used package repository for Python
- We can then use an automated tool to connect to this repository and update packages- the most common tool used is PIP
- PyCharm uses PIP to provide an easy interface for installing packages (into your Python virtual environment)

Find the Packages Tool In PyCharm

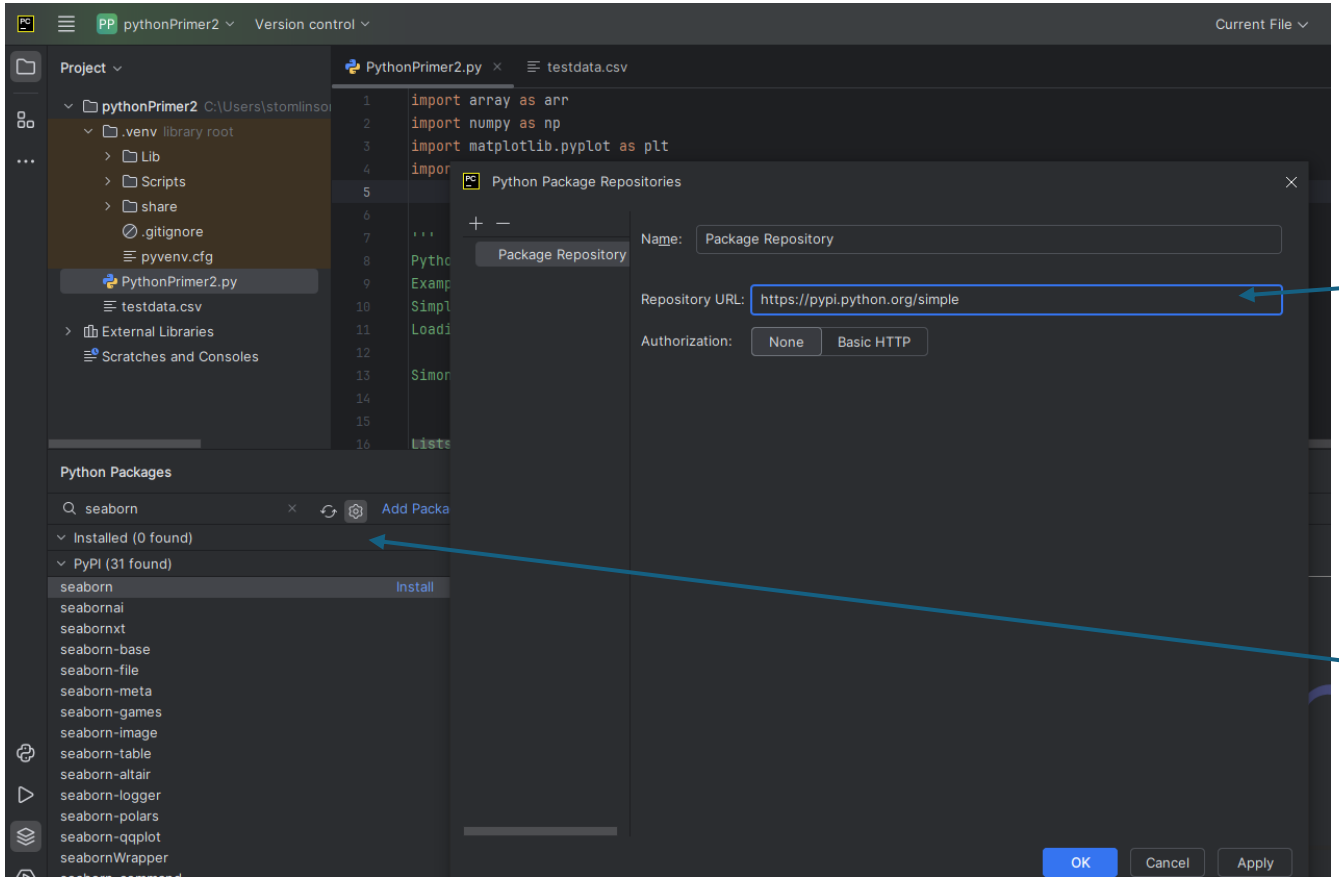


Search here eg Seaborn

Click install

Packages tool

Sometimes a Package Repository is Note Linked in the IDE

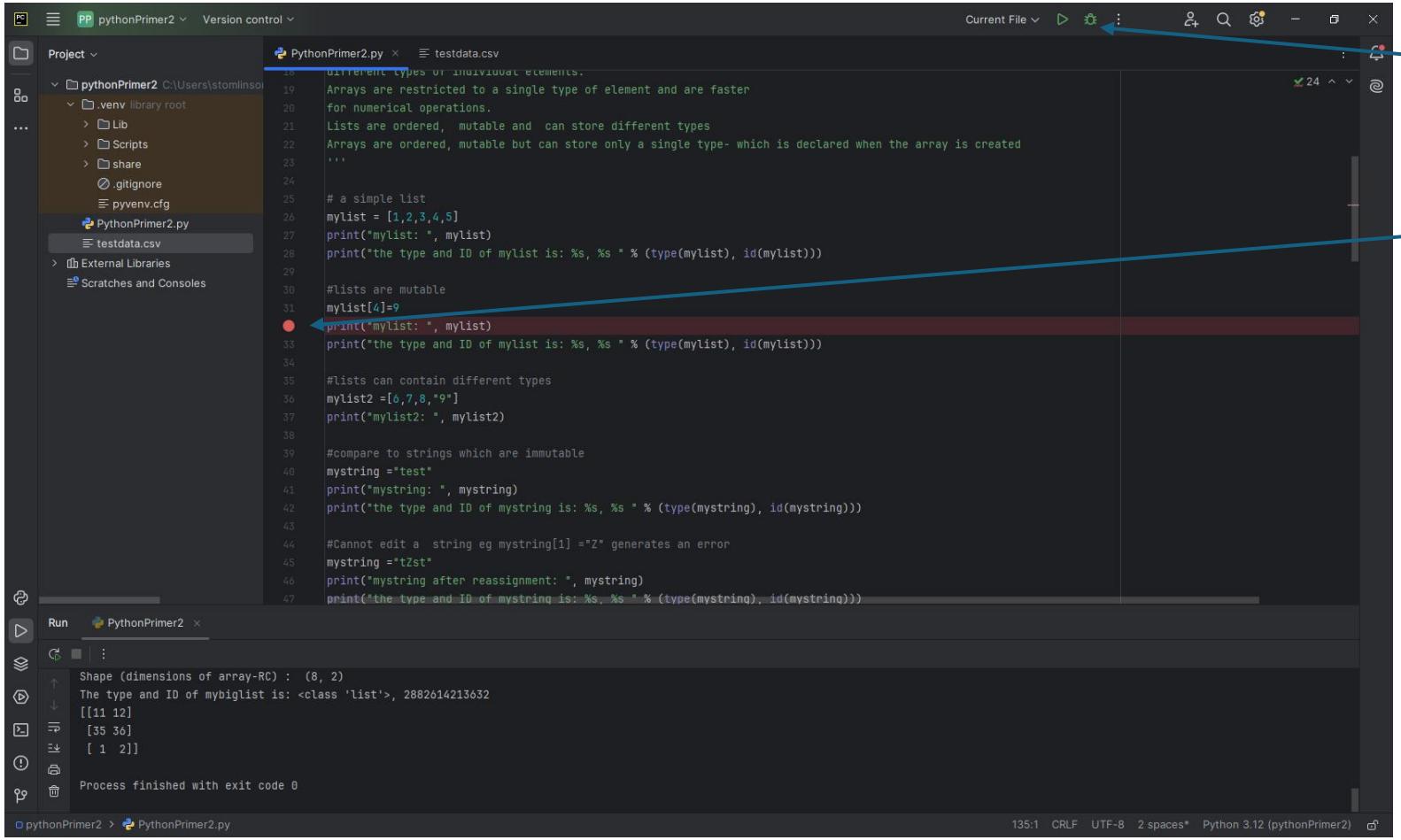


Then add repository

Click here

- URL is https://pypi.python.org/simple
- If you have conda/anaconda do not select this
- Conda is another way of managing packages

Using The Debugger Tool…



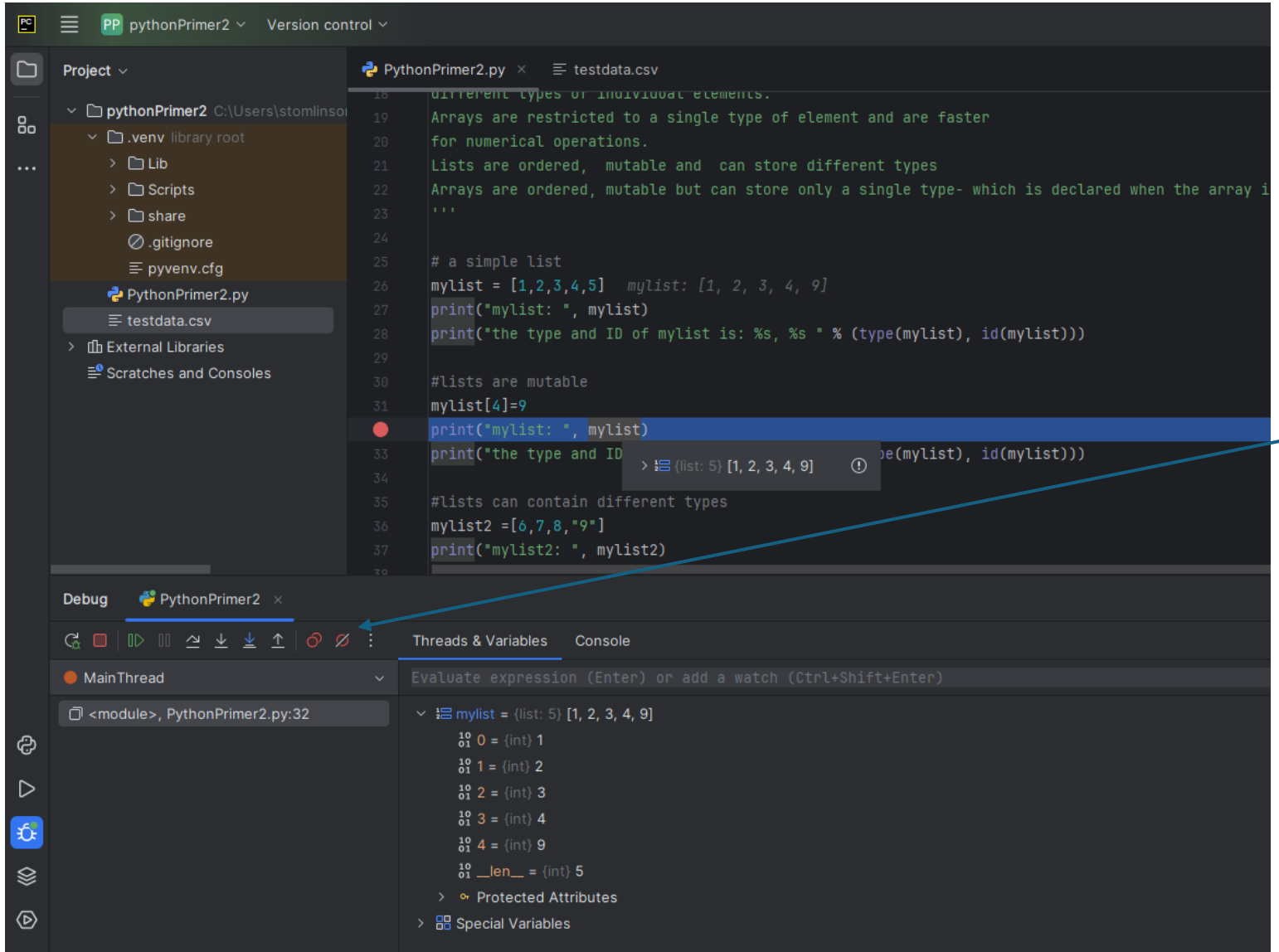
Run code –click the bug icon

Click to add breakpoint (red circle)

- Code execution will stop at the breakpoint and you can go through line by line

This helps find where you code is having problems..

Stepping Through Code



- Execution stops at the breakpoint
- We can see the current value of variables (etc)
- We can step through or stop using the menu here

- You can step through code, follow loops etc and see how the value of variables changes over time…
- Press the red stop square when finished

Required Packages for today

- `numpy`
- `matplotlib`
- `pandas`

- We will need to install these before running the class code
- Code for today's class is in the file `PythonPrimer2.py`
- There is also a small file, `testdata.csv` that is available to be used
- For today, create a new project and then copy these two files to the project folder

Data Structures-Containers for Data

- We often need to work with arrays (ID array, vectors) or 2D arrays (matrices) or other types of tabular data
- We might consider a string as an array of characters
- But strings are immutable- so operations that change a string create a new string object, referenced with the same variable name
- Imagine arrays and vectors worked like this-every time you edit an element a whole new array would need to be created- not very efficient!!

Reminder

```
mystring:  test
the type and ID of mystring is: <class 'str'>, 1590170336640
mystring after reassignment:  tZst
the type and ID of mystring is: <class 'str'>, 1590173686288
mystring2 is a new string:  tZst
the type and ID of mystring2 is: <class 'str'>, 1590173686288
```

```
mystring ="test"
print("mystring: ", mystring)
print("the type and ID of mystring is: %s, %s " % (type(mystring), id(mystring)))

#Cannot edit a  string eg mystring[1] ="Z" generates an error
mystring ="tZst"
print("mystring after reassignment: ", mystring)
print("the type and ID of mystring is: %s, %s " % (type(mystring), id(mystring)))

#You can see why you cannot edit strings... all strings with the same initial value
address the same object
mystring2 ="tZst"
print("mystring2 is a new string: ", mystring2)
print("the type and ID of mystring2 is: %s, %s " % (type(mystring2), id(mystring2)))
```

Data Structures

- Build-in Lists
- Built-in Arrays
- Numpy numerical arrays
- Pandas for import of tabular data

Lists

```
mylist:  [1, 2, 3, 4, 5]
the type and ID of mylist is: <class 'list'>, 1590170407296
mylist:  [1, 2, 3, 4, 9]
the type and ID of mylist is: <class 'list'>, 1590170407296
mylist2:  [6, 7, 8, '9']
```

- Lists are ordered, mutable and can store different types

```
# a simple list
mylist = [1,2,3,4,5]
print("mylist: ", mylist)
print("the type and ID of mylist is: %s, %s " % (type(mylist), id(mylist)))

#lists are mutable
mylist[4]=9
print("mylist: ", mylist)
print("the type and ID of mylist is: %s, %s " % (type(mylist), id(mylist)))

#lists can contain different types
mylist2 =[6,7,8,"9"]
print("mylist2: ", mylist2)
```

Arrays

```
myarray:  [1, 2, 3, 4, 5, 6]
the type and ID of myarray is: <class 'array.array'>, 1590173719328
myarray:  [1, 2, 10, 4, 5, 6]
the type and ID of myarray is: <class 'array.array'>, 1590173719328
```

- Similar to simple list- but we specify a type – all the elements must be of that type…

```
#here "i" specifies the type: "i"- integer,  "f"- float, "u"- unicode #etc
#arrays are mutable
myarray = arr.array("i",[1,2,3,4,5,6])
print("myarray: ", myarray.tolist())
print("the type and ID of myarray is: %s, %s " % (type(myarray), id(myarray)))

#Edit an element in the array- still the same location in memory
myarray[2]=10
print("myarray: ", myarray.tolist())
print("the type and ID of myarray is: %s, %s " % (type(myarray), id(myarray)))
```

- Note the ID memory address of the object does not change after editing the contents!

We Can Also Make Matrices From 2D Lists

```
#We can also make multidimensional lists eg
mymd_list =[[11, 12, 13, 14],[15, 16, 17, 18],[4, 12, 3, 2]]
print("mymd_list: ", mymd_list)

#Note that Python indexing  is [R][C]- so first we index the row and then
the column. Note the position in the container, indexed using the R=0, C=2
in this example.
mymd_list[0][2]=0
print("mymd_list edit: ", mymd_list)
```

```
[[11, 12, 13, 14],
 [15, 16, 17, 18],
 [4, 12, 3, 2]]
```

- Really just a list-of-lists

We Can Also Make Matrices From 2D Lists

mymd_list: [[11, 12, 13, 14], [15, 16, 17, 18], [4, 12, 3, 2]]
mymd_list edit: [[11, 12, 0, 14], [15, 16, 17, 18], [4, 12, 3, 2]]

```
#We can also make multidimensional lists eg
mymd_list =[[11, 12, 13, 14],[15, 16, 17, 18],[4, 12, 3, 2]]
print("mymd_list: ", mymd_list)

#Note that Python indexing  is [R][C]- so first we index the row and then
the column. Note the position in the container, indexed using the R=0, C=2
in this example.
mymd_list[0][2]=0
print("mymd_list edit: ", mymd_list)
```

But

- We cannot easily use build-in arrays to make matrices
- If you can guarantee to store the same kind of thing in a container object then allowing the storage of multiple types in one data structure
- So performance of arrays generally is better than lists because of how these components are implemented
- Many Python programmers working with numerical data use the package Numpy to provide convenient matrices and other arrays

Numpy

- A mutable matrix using Numpy

```
mynp_array:  [[11, 12, 13, 44], [35, 36, 4, 8], [1, 2, 3, 4]]
the type and ID of mynp_array is: <class 'numpy.ndarray'>, 1590469307696
mynp_array:  [[11, 12, 0, 44], [35, 36, 4, 8], [1, 2, 3, 4]]
the type and ID of mynp_array is: <class 'numpy.ndarray'>, 1590469307696
Shape (dimensions of array-RC) :  (3, 4)
```

```
mynp_array = np.array([[11, 12, 13, 44], [35, 36, 4, 8], [1, 2, 3, 4]])

print("mynp_array: ", mynp_array.tolist())
print("the type and ID of mynp_array is: %s, %s " % (type(mynp_array),id(mynp_array)))
mynp_array[0][2]=0
print("mynp_array: ", mynp_array.tolist())
print("the type and ID of mynp_array is: %s, %s " % (type(mynp_array),id(mynp_array)))
print("Shape (dimensions of array-RC) : ",mynp_array.shape)
```

More details https://numpy.org/doc/stable/index.html

Dictionaries

- Dictionaries- versatile data structures that associate a key with a value
- Look a value up using a key
- Performance is (more or less) independent of the number of elements in the dictionary

```
mydict =      {
  "gene1": "TRP53",
  "gene2": "BRCA1",
  "genex": "GAPDH"
}
print("my dictionary: ",mydict)
print("genex: ", mydict["genex"] )
```

my dictionary: {'gene1': 'TRP53', 'gene2': 'BRCA1', 'genex': 'GAPDH'}
genex: GAPDH

Pandas- Nice Ways to Load and Manage Tabular Data

```
0   1
1   2
2   3
3   4
4   5
5   6
6   7
7   8
Name: first_column, dtype: int64
first_column     2
second_column    2
Name: 1, dtype: int64
```

testdata.csv

first_column	second_column
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

```
## Read the CSV file into a DataFrame
df = pd.read_csv('testdata.csv')

# Access data in the DataFrame using column names or indexing
print(df["first_column"])
print(df.iloc[1])  # Access the second row (remember Python uses zero indexing)
```

See https://pandas.pydata.org/

Converting Data Types

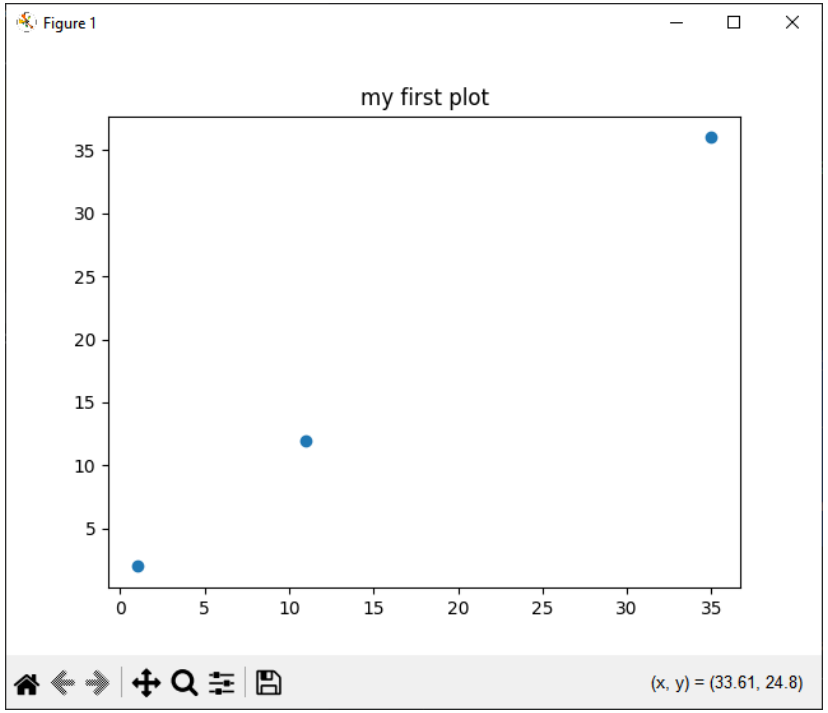
- Starts off as a Panda and ends as a Numpy array or a simple 2D list…

The type and ID of mybignp is: <class 'numpy.ndarray'>, 2647553580880
Shape (dimensions of array-RC) : (8, 2)
The type and ID of mybiglist is: <class 'list'>, 2647227513920

```
df = pd.read_csv('testdata.csv')

#we can then convert our data frame df to a list of to a numpy matrices
mybignp =df.to_numpy()
print("The type and ID of mybignp is: %s, %s " % (type(mybignp),
id(mybignp)))
print("Shape (dimensions of array-RC) : ",mybignp.shape)

#we can even convert back to a list
mybiglist =mybignp.tolist()
print("the type and ID of mybiglist is: %s, %s " % (type(mybiglist),
id(mybiglist)))
```

Plotting Data

- This code generates a popup window with your plot that you can save for future use…




```
#Python import matplotlib.pyplot as plt

#take first columns to plot using np.r_[0:3, 0:2]
print(np.r_[mynp_array[0:3, 0:2]])

plt.scatter(np.r_[mynp_array[0:3,
0]],np.r_[mynp_array[0:3, 1]])
plt.title('my first plot')
plt.show()
```

https://matplotlib.org/

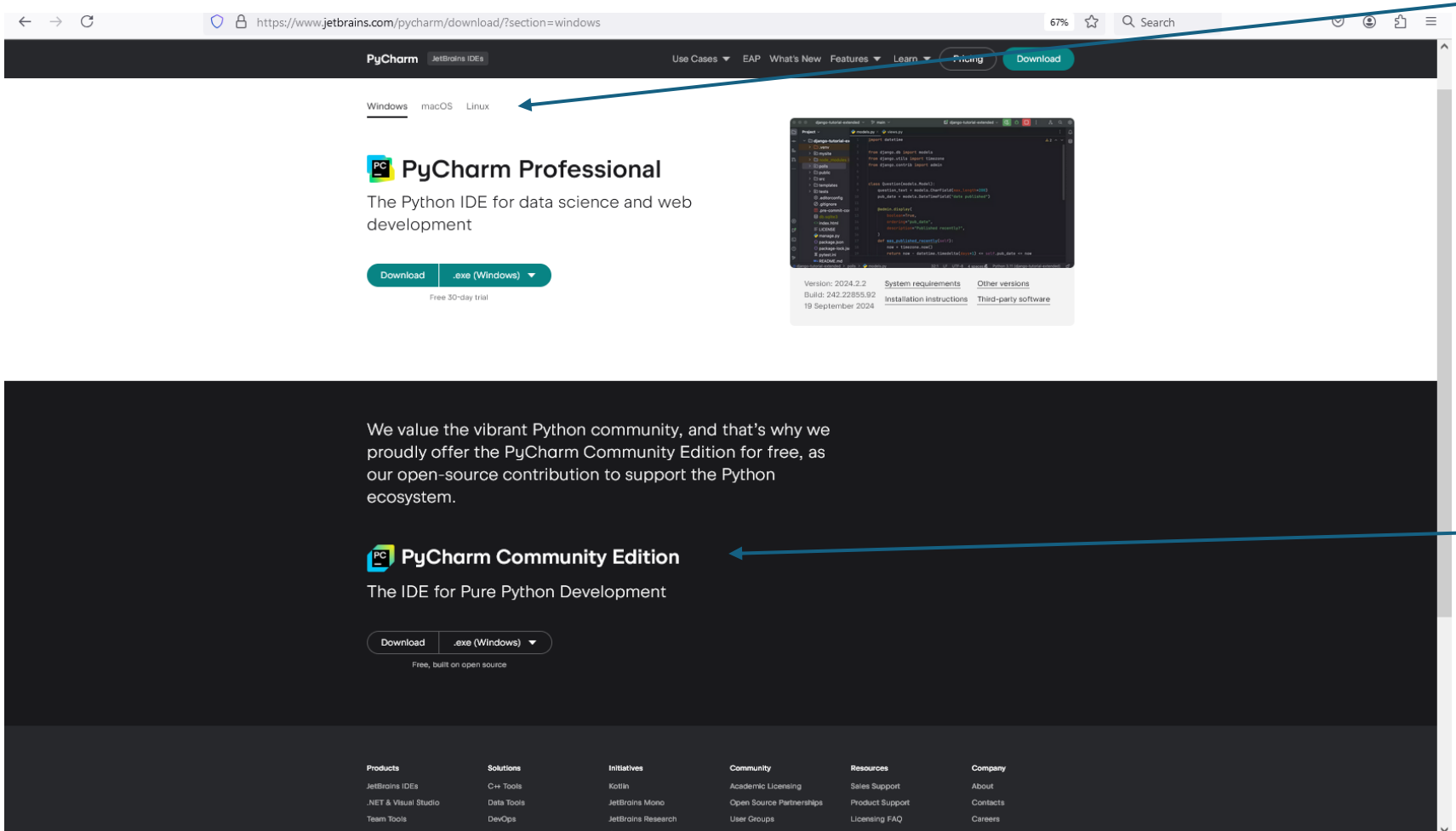
Some Final Challenges..

```
#Load the example csv file (testdata.csv) and make a line plot showing rows against columns
#Label the axis of this plot and give it an appropriate caption

#Create a small matrix containing numbers 1 to 10 and use a for loop to go through
#the values in matrix adding the constant value of 2 to each element

#Seaborn is another useful graphics package for Python- install this package and using the documentation
#try to generate  a line plot from the example data
```

- Keep practicing!!!

Download Page for PyCharm-

- If you want to install PyCharm on your own machine
- We use the Community edition for classes-this is available to install for free



Select OS

Use this installer!