QBUS6850: Tutorial 2 – Linear Algebra, Data Handling and Plotting

Objectives

- To handle data using Numpy, Pandas libraries;
- To plot data using matplotlib library;

1. Libraries

For this tutorial you will need to use external libraries. Libraries are groups of useful functions for a particular domain.

The import statement is used to import code libraries.

```
import numpy as np
```

Numpy contains useful linear algebra functions. Numpy borrows many function names and concepts from MATLAB

2. Vectors

Creating a vector is straight forward with numpy. To create a vector you can either use either

```
a = np.array([1, 2, 3])
A = np.matrix([1, 2, 3])
```

where the contents of the [ ] (square brackets) are the contents of the vector.

You can check the shape of a vector by

```
print(a.shape)
print(A.shape)
```

Note that the shape of vector a is different from vector A. This is because numpy array dimensions are ambiguous. It can be interpreted as a row or column vector. If you want to exactly specify the shape then use the matrix function instead to create your vector.

Be careful with arrays! Products of arrays are elementwise, not vector multipliciation!

```
print(a * a)
```

Instead you should use

```
print(np.dot(a,a))
```

or

```
print(A * A.transpose())
```

The type of data that a vector (and matrix) can store is fixed to a single type. You cannot mix integers, floats etc in the same vector. Numpy infers the data type from the data used to create the vector.

```
print(a.dtype)
c = np.array([1.0, 2.0, 3.0])
print(c.dtype)
```

Here are some useful shortcuts for creating common special vectors

```
# Length 10 zero vector
zeros = np.zeros((10,1))

# Length 5 ones vector
ones = np.ones((5,1))

# You can create a single valued vector as the product
# of a vector of ones and your desired value
twos = ones * 2

# 11 numbers from 0 to 100, evenly spaced
lin_spaced = np.linspace(0, 100, 11)

# All numbers from 0 up to (not including) 100 with gaps
of 2
aranged = np.arange(0, 100, 2)
```

Vector transposition is straight forward

```
A_transposed = A.transpose()
```

Checking for vector/array equality is a little trickier than checking for equality of other types. Here is how I suggest you check if two vectors are equal

```
print(np.array_equal(A, A))
print(np.array_equal(A, A_transposed))

# You may wish to check equality up to a tolerance
# This is useful since floating points aren't perfect
print(np.allclose(A, A, rtol=1e-05))
```

Vectors can be summed together as expected, or linear combinations can be formed

```
c = a + a
d = 3*a * 1.5*c
```

Taking the norm of the vector is taken care of by numpy

```
norm2 = np.linalg.norm(a, ord=2)
print(norm2)
```

We can test if two vectors are orthogonal by the dot product. If vectors are orthog then their inner product (dot product) is 0.

```
# Generate an orthogonal vector
x = [1, 2, 3]
y = [4, 5, 6]
orthog = np.cross(x, y)
print("Orthogonal") if np.dot(x, orthog) == 0 else print
("Not Orthogonal")
```

3. Matrices

Numpy Matrices behave in the same way as vectors

```
b = np.array([[1, 4, 7],
              [2, 5, 8],
              [3, 6, 9]])
B = np.matrix([[1, 4, 7],
              [2, 5, 8],
              [3, 6, 9]])
print(B)

# Multiplication of matrices is as normal
print(B * B)

# Scalar Product
print(5 * B)

# Summing
print(B + B)

# Transpose
print(B.transpose())
```

Some useful shortcuts for special matrices

```
# 10 x 10 zero matrix
m_zeros = np.zeros((10,10))
print(m_zeros)

# 5 x 5 ones matrix
m_ones = np.ones((5,5))
print(m_ones)

# Matrix of all twos
m_twos = m_ones * 2
print(m_twos)
```

Often we are interested in the diagonal entries of a matrix (e.g. covariance matrices) or creating a diagonal matrix (e.g. identity matrix)

```
# Identity matrix
eye = np.identity(3)
print(eye)

# Get the diagonal entries
diag_elements = np.diag(B)
print(diag_elements)

# Create a matrix with values along the diagonal
m_diag = np.diag([1,2,3])
print(m_diag)
```

Other useful functions of matrices include the rank, trace and determinant. Th e rank tells us how many linearly independent cols or rows there are, the trace is used in calculating norms and the determinant tells us if the matrix is inverti ble.

```
# Matrix Rank i.e. n linearly independent cols or rows
print(np.linalg.matrix_rank(B))

# Sum of diagonal entries
print(np.trace(B))

# Determinant
e = np.array([[1, 2], [3, 4]])
print(np.linalg.det(e))
print("Matrix is invertible") if np.linalg.det(e) != 0
else print ("Matrix not invertible")
```

4. Pandas

Pandas is a library for data manipulation. The key feature of Pandas is that the data structures it uses can hold multiple different data types. For example it can create an array with integers, strings and floating point numbers all at once. You could consider this capability as similar to an excel spreadsheet. Whereas you cannot mix and match data types in Python Lists or Numpy Arrays.

Pandas is already installed and available in Anaconda/Spyder. To begin using it we first import it as follows

```
import pandas as pd
```

Download the **drinks.csv** file from Blackboard and place it in the same folder as your Python file.

Then load the CSV by using

```
drinks = pd.read_csv('drinks.csv')
```

Check that the DataFrame was loaded correctly by viewing some basic information by using

```
drinks.dtypes
drinks.info()
```

View summary statistics of the DataFrame drinks by using the *describe* function

```
drinks.describe()
```

Now you can begin to manipulate the data. Lets begin by extracting the beer_servings column. Extracting a column will return a variable with the Series type, not DataFrame. Series is for 1D data and DataFrame is for 2D data.

Here I have used the head function to show the first 5 rows.

```
drinks['beer_servings'].head()
```

Just like a DataFrame you can get statistics on a Series. Try it

```
drinks['beer_servings'].describe()
```

You can get single statistics from a series

```
drinks['beer_servings'].mean()
```

You can search or query the DataFrame. For example you can get the rows where the continent is Europe. Try the following

```
euro_frame = drinks[drinks['continent'] == 'EU']
```

Again you can query this Series for it's own statistics.

```
euro_frame['beer_servings'].mean()
```

Queries can be compounded. In the following example you will get the countries in Europe where the number of wine servings is greater than 300. Try it.

```
euro_heavywine = drinks[(drinks['continent'] == 'EU') &
(drinks['wine_servings'] > 300)]
euro_heavywine
```

DataFrames can be sorted. Try the following code to sort by litres of alcohol and return the last 10 entries using the tail function

```
top_drinkers =
drinks.sort_values(by='total_litres_of_pure_alcohol').tai
l(10)
top_drinkers
```

4.1. Handling Missing Data

The drinks_corrupt.csv actually contains lots of missing entries. Pandas filters out any rows with missing data automatically for you. However sometimes you may want to filter them out manually or specify special codes to ignore etc.

In this case we want to do two things:

- Keep contintent code "NA" since this is valid and means "North America"
- Remove all other rows containing legitimate missing values

By default Pandas will convert "NA" to NaN data type. We should prevent this.

```
default_na_values = ['', '#N/A', '#N/A N/A', '#NA', '-
1.#IND', '-1.#QNAN', '-NaN', '-nan', '1.#IND', '1.#QNAN',
'N/A', 'NA', 'NULL', 'NaN', 'nan']

our_na_values = default_na_values
our_na_values.remove('NA')

drinks_dirty = pd.read_csv('drinks_corrupt.csv',
keep_default_na = False, na_values = our_na_values)
```

Check the number of NaN values per column

```
drinks_dirty.isnull().sum()
```

To remove rows that contain any missing entries try the dropna function

```
drinks_clean = drinks_dirty.dropna()
```

If you want to remove rows where every column has a missing entry use the how parameter

```
drinks_clean_byrows = drinks_dirty.dropna(how='all')
```

You can avoid the creation of a new variable by doing an in place replacement. Try the following

```
drinks_dirty.dropna(inplace = True)
```

Alternative

Instead of deleting corrupt or missing entries you can replace them with a useful value. If we use the drinks.csv file we know there are no corrupt or missing entries. In this case we can simply use **replacement** to fix the "NA" (North America) issue.

```
drinks = pd.read_csv('drinks.csv')
```

To replace the data in place use the fillna function. Try the following to replace missing continent values with the string ‘NA’

```
drinks['continent'].fillna(value='NA', inplace=True)
```

4.2. Creating Columns

You can create new columns as a function of (i.e. based on) the existing columns. For example try creating a total_servings column

```
drinks['total_servings'] = drinks.beer_servings +
drinks.spirit_servings + drinks.wine_servings
```

and create the total litres column

```
drinks['alcohol_mL'] =
drinks.total_litres_of_pure_alcohol * 1000
```

Then check your changes using the head function

```
# Check changes
drinks.head()
```

4.3. Renaming Data

Renaming columns is straightforward

```
drinks.rename(columns={'total_litres_of_pure_alcohol':'al
cohol_litres'}, inplace=True)
```

4.4. Deleting Data

Deleting data is easy. Pandas provides two options. The first is with the drop function. Axis 1 refers to columns and axis 0 refers to rows.

```
drinks_wout_ml = drinks.drop(['alcohol_litres'], axis=1)
```

5. Plotting: Pyplot

Pyplot is a library for data manipulation. Pyplot follows many of the conventions of MATLAB's plotting functions. It provides a very simple interface to plotting for common tasks.

Pyplot is already installed and available in Anaconda/Spyder. To begin using it we first import it as follows

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

Pyplot requires a figure to draw each plot upon. A Figure can contain a single plot or you can subdivide it into many plots. You can think of a figure as a "blank canvas" to draw on.

By default Pyplot will draw on the las figure you created.

Creating a new figure is simple

```
my_figure = plt.figure()
```

5.1. Bar Chart

Lets create a simple bar chart comparing the number of beer servings among EU countries that drink lots of wine.

The first parameter of the bar(x, y) function is the x position to draw at and the second is the vertical heights of each bar. Here we use x = ind and y = euro_heavywine['beer_servings']

```
ind = np.arange(len(euro_heavywine))
plt.bar(ind, euro_heavywine['beer_servings'])
# This shows the figure
my_figure
```

We need to label the individual bars. We can use the xticks function to do this

```
plt.xticks(ind, euro_heavywine['country'])
my_figure
```

Next label each of the axis

```
plt.xlabel("Country")
plt.ylabel("Beer Servings")
my_figure
```

Finally give your plot a title

```
plt.title("Beer Servings of Heavy Wine Drinking EU
Countries")
my_figure
```

5.2. Line Plot

Lets create a line plot showing the total servings of top drinking countries in descending order.

Create a new figure to draw on

```
# Line Plot
my_figure2 = plt.figure()
```

Then sort the dataframe by total servings and pick the top 10 using the head function.

```
# Get the 10 countries with highest total servings
top_drinkers = drinks.sort_values(by='total_servings',
```

```
ascending=False).head(10)
```

Then plot the total servings column of top_drinkers. You should also label your line plots so that you can show a legend later. Pyplot will automatically assign a colour to each of your lines. You can choose a specific colour like so

```
# Line plot with a label and custom colour
# Other optional parameters include linestyle and
markerstyle
plt.plot(np.arange(0,10,1),
top_drinkers['total_servings'], label="Total Servings",
color="red")
my_figure2
```

Label each of the axis and title

```
plt.xlabel("Drinking Rank")
plt.ylabel("Total Servings")
plt.title("Drinking Rank vs Total Servings")
my_figure2
```

Show the legend on your Figure

```
# Activate the legend using label information
plt.legend()
my_figure2
```