Week 1 - Introduction

1. Basic Python Syntax

The first task of this tutorial is a brief revision of basic python syntax, including function and class.

□□ Hello World!

In the workspace on the right, write a Python program which displays the text *Hello World!*

The output of your program should look like this:

```
Hello World!
```

Note: *Any* `#` *followed by text is a comment. It allows you to make notes without affecting your code.*

Function Syntax

Functions are defined using the following syntax

```
def function_name(parameter_1, parameter_2, ... parameter_n):
    # your code here to complete the task
    return result
```

Take note of the following:

- `def` is a keyword which signifies that this is a function definition
- `function_name` is chosen by you and follows the same naming rules as variables
- each `parameter_i` has a name chosen by you and is available as a variable in the function
- `:` is placed after the parameter list
- the function's code block must be indented
- `return` signifies that the next thing will be "given back" to the caller

Example

```
def power(x, exponent):
    y = x**exponent
    return y

z = power(2, 3)
print(z)
```

House Price Predictor

Write a Python **function** that returns predicted house prices from a learned linear model (we will talk about for example linear regression models in the next week).

Function Specification

- name: `predict`
- parameters:
  - `X` (numpy array): an array containing the house features with shape (n_samples, n_features)
- returns:
  - `y` (numpy array): an array of predicted values

Example Usage

```
data = np.array([[4500, 4, 2, 18],
                 [1588, 3, 2, 5],
                 [1714, 2, 1, 4]])

print(predict(data))
```

```
[323126.62 104713.47 104063.75]
```

Background

Suppose that you generated a linear regression model to predict house prices.

You used the following 4 features: `SQFT`, `Bedrooms`, `Baths` and `Age`

The corresponding linear regression formula is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon$$

The learned β's by e.g. linear regression model are:

- β_0: -26733.12
- β_1: 87.50
- β_2: -29192.65
- β_3: 41420.77
- β_4: -553.40

We've collected these into an array for your convenience

```
betas = np.array([-26733.12, 87.50, -29192.65, 41420.77, -553.40])
```

Hints

To make your prediction, you will need to do the following things:

- add a column of `ones` to the start of your `data` matrix
- make your prediction using a matrix multiplication: $\mathrm{data} \times \mathrm{betas}.T$

It will be helpful to test your function by printing the returned value using the example data provided earlier.

Python Class Syntax

Classes are defined using the following syntax

```
class class_name():

    def instance_method1(self, parameter_1, ..., parameter_n):
        # do something
        return # optional

    def instance_method2(self, parameter_1, ..., parameter_n):
        # do something
        return # optional
```

Take note of the following:

- `class` is a keyword which signifies that this is a class definition
- `class_name` is chosen by you and follows the same naming rules as variables
- `:` is placed after the name
- each `instance_method1` has a name chosen by you and is a function that gets applied to instances
- instance methods must be indented underneath the class definition
- the first parameter of each instance method must be `self`, which is a variable available inside the method scope which references the current object on which the function is operating

Example

You have likely used scikit-learn's LinearRegression class in the next week. For example we can create an instance of the class as follows:

```
from sklearn.linear_model import LinearRegression

model_instance = LinearRegression()
```

Constructor Methods

Often you want to initialise an instance with some pre-set values. This can be accomplish with a "constructor" function which is called when we create an instance.

The constructor function is denoted in Python with the `__init__` name

```
class class_name():

    def __init__(self, parameter_1, ..., parameter_n):
        # Set values here
```

Example

```
class Point():
    def __init__(self, x, y):
        self.x = x
        self.y = y

point = Point(10, 20)

print(point.x)
```

□□ Create a Student Class

Now let's create a Student **class** object to store and update student information in a database.

Class Specification:

- class name: `Student` (notice for class object, we usually capitalize the first letter)
- initialization: in the `__init__` function, store `name` (student name), `sid` (student id), `email` (student email) as attributes.
- Implement an `update_email` function that updates student email with a new one.

Example Usage

```
student1 = Student('Jack', 412345678, 'jack123@gmail.com')
print('Student name: {}, sid: {}, email {}'.format(student1.name, student1.sid, student1.email))

student1.update_email('jack456@gmail.com')
print('Student name: {}, sid: {}, email {}'.format(student1.name, student1.sid, student1.email))
```

```
Student name: Jack, sid: 412345678, email jack123@gmail.com
Student name: Jack, sid: 412345678, email jack456@gmail.com
```

2. Python Linear Algebra

The second task is to introduce python syntax for linear algebra operations.

Linear algebra operations

There are some important linear algebra operations that you will need to know. These operations can all be done using functions from the `numpy` library.

Transpose

Taking the transpose of a matrix means you **switch** all the rows and columns - the first row becomes the first column and vice versa for all rows and columns. You can also think of the transpose as flipping all the matrix elements around the diagonal.

The transpose is denoted with an uppercase **T**.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

To transpose a matrix in Python, you can use the `.T` attribute:

```
matrix.T
```

```
import numpy as np

A = np.array([[1, 4, 5],
              [2, 8, 6],
              [3, 1, 4]])

print(A.T)
```

Vector Product

The product of vectors is calculated using the **inner product** which is a way of multiplying them together so that the output is a scalar (not a vector). If you only have two vectors, the inner product becomes the **dot product** which you have already seen before:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

In Python, you can use the `np.dot` function to calculate the dot product of two vectors:

```
np.dot(vector_1, vector_2)
```

```
import numpy as np
```

```
a = np.array([10, 2, 5])

b = np.array([1, 2, 3])

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

Matrix Product

To multiply two matrices together, there are two methods you can use:

- `np.matmul` function

```
np.matmul(matrix_1, matrix_2)
```

- `@` symbol

```
matrix_1 @ matrix_2
```

Let us try them!

```
import numpy as np

A = np.array([[1, 2],
              [3, 4]])

print('Matmul')
print(np.matmul(A, A))

print('@')
print(A @ A)
```

Note: *If matrix A has dimensions (m, n) and matrix B has dimensions (n, p) then A* $\times$ *B will have dimensions (m, p). Note that the number of* ***columns*** *in the first matrix* ***must match*** *the number of* ***rows*** *in the second matrix - if not then the dimensions of the matrix won't agree and multiplying them together won't make sense!*

Matrix Inverse

The inverse of a matrix is denoted using the $^{-1}$ symbol i.e. the inverse of A is A^{-1}.

To calculate the inverse of a matrix we use the function `np.linalg.inv` :

```
np.linalg.inv(matrix)
```

```
import numpy as np

A = np.array([[4, 5, 3],
              [3, 9, 0],
              [2, 7, 9]])
```

```
Ainv = np.linalg.inv(A)

print(Ainv)
```

Note: *Note that some matrices do not have any inverse. If the matrix is not square, or if the determinant of a matrix is 0, then it is* ***not invertible****. You can calculate the determinant of your matrix using* `np.linalg.det`

□□ Transpose and multiply

Consider the matrices A and B defined below:

```
A = np.array([[3, 6, 12],
              [11, 2, 2],
              [1, 4, 18],
              [2, 8, 10]])

B = np.array([[1, 4, 7],
              [3, 4, 8]])
```

Calculate the transpose of B and then multiply that result with A. In other words, calculate $\mathbf{A} \times \mathbf{B}^T$.

`Print` your answer. Your output should look like this:

```
[[XXX XXX]
 [ XX  XX]
 [XXX XXX]
 [XXX XXX]]
```

Question 1! Why did we need to calculate the transpose of B before multiplying?

Question 2! Have you tried $A * B$? Does this work? Why?

□□ Dot product

Use `numpy` to evaluate the dot product below and `print` the result of x

$$x = \begin{bmatrix} 2 & 4 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Concatenating arrays

There are four ways you can concatenate:

- `np.vstack` - joins array together in the **vertical (up, down)** axis. You must make sure the arrays to be stack have the same number of columns.

```
np.vstack((array_1, array_2))
```

```
import numpy as np

a = np.array([[1, 2],
              [3, 4]])

b = np.array([[5, 6],
              [7, 8]])

print(np.vstack((a, b)))
```

- `np.hstack` - joins arrays together in the **horizontal (left, right)** axis. You must make sure the arrays to be stack have the same number of rows.

```
np.hstack((array_1, array_2))
```

```
import numpy as np

a = np.array([[1, 2],
              [3, 4]])

b = np.array([[5, 6],
              [7, 8]])

print(np.hstack((a, b)))
```

- `np.concatenate` - joins arrays together along a particular axis direction. You must make sure the arrays to be stack have the same other dimensions.

```
np.concatenate((array_1, array_2))
```

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(np.concatenate((a, b)))
```

```
import numpy as np
```

```
a = np.array([[1, 2],
              [3, 4]])

b = np.array([[5, 6],
              [7, 8]])

print(np.concatenate((a, b), axis = 0))
print(np.concatenate((a, b), axis = 1))
```

Note: *The `np.concatenate` function as another argument called **`axis`** that allows you to specify the joining axis - if `axis = 0` the function is equivalent to `np.vstack` (this is the default) and if `axis = 1` the function is equivalent to `np.hstack`*

- `np.append` - adds values to the end of the array

```
np.append(array_1, array_2)
```

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(np.append(a, b))
```

Example Errors:

Question! Why does this code fail?

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(np.concatenate(a, b))
```

Answer! The first argument (input) for each of these functions must be a tuple of the arrays you want to concatenate.

Question! Where and why does this code have an error?

```
import numpy as np

a = np.array([[1, 2, 3],
              [4, 5, 6]])

b = np.array([[1],
              [4]])

print(np.hstack((a, b)))
```

```
print(np.vstack((a,b)))
```

Answer!

When using `np.hstack` and `np.vstack` the dimensions must agree along the axis of joining

- For `np.hstack` the dimensions of the two arrays must agree along the horizontal axis - **number of rows must match**
- For `np.vstack` the dimensions must agree along the vertical axis - **the number of columns must match**

Ones

To create an array of ones we use the `np.ones` function:

```
np.ones(shape)
```

```
import numpy as np

print(np.ones((2, 3)))
```

Hint: *Remember to pass in the shape of your array as a tuple i.e. using the extra () brackets*

Column of ones:

Later, we will need you to create a column of ones. To do this you need to change the dimensions of the `shape` you input.

Recall that `shape = (rows, columns)` and so to create a column we want `shape = (rows, 1)` where the number of rows will be equivalent to the number of ones we want.

```
import numpy as np

print(np.ones((5, 1)))
```

□□ Column of ones

Concatenate a column of `ones` to the **beginning** of this matrix:

```
data = np.array([[5, 7, 10, 4, 1],
                 [5, 12, 7, 5, 2],
                 [4, 2, 9, 18, 7],
                 [4, 1, 10, 0, 8],
                 [9, 10, 4, 5, 1]])
```

`Print` out the new matrix.

Hint: *The number of ones in your column must match the dimensions of the matrix. Recall that you can obtain the dimensions using the* `.shape` *attribute i.e. data.shape*

□□ Row of ones

Concatenate a row of `ones` to the **top row** of this matrix:

```
data = np.array([[5, 7, 10, 4, 1],
                 [5, 12, 7, 5, 2],
                 [4, 2, 9, 18, 7],
                 [4, 1, 10, 0, 8],
                 [9, 10, 4, 5, 1]])
```

`Print` out the new matrix.

Note: *You did something very similar in the Column of ones challenge - you can use this to help you*

□□ Matrix multiplication

Consider the matrices `A`, `B` and `C` defined below:

```
A = np.array([[0.625, 0.125],
              [-0.25, 0.75]])

B = np.array([[1, 4],
              [3, 4]])

C = np.array([[1, 3],
              [2, 2]])
```

Multiply B with the inverse of C and then multiply this result with A. In other words calculate the following: $\mathbf{A} \times \mathbf{B} \times \mathbf{C}^{-1}$

`Print` your answer. Your output should look like this:

```
[[X. X.]
 [X. X.]]
```

Question! What does your result tell you about the relationship between A and the result of $B \times C^{-1}$?

□□ Predict one house price

Suppose that you generated a linear regression model to predict house prices.

You used the following 4 features: `SQFT`, `Bedrooms`, `Baths` and `Age`

The corresponding linear regression formula is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon$$

Your estimates for β are:

- β_0: -26733.12
- β_1: 87.50
- β_2: 29192.65
- β_3: 41420.77
- β_4: -553.40

Using these values, predict the price of a house with the following features:

- sqft area: 4500
- 4 bedrooms
- 2 bathrooms
- 18 years old

Hint: *Use a dot product!*

Your output should look like this:

```
xxxxxx.xx
```

□□ Error Message

Write a function called `error_message`, which will be used to display a message to the user when they don't have permission to perform the command.

The message should say

```
I'm sorry, Dave. I'm afraid I can't do that.
```

where **Dave** is a placeholder that must be replaced by the username.

Function specification

- name: `error_message`
- parameters: `username` (`str`)
- return: `None`

Marking note

You don't need to print anything to pass this Challenge. But it might be helpful to check your result e.g.

```
> error_message("Dave")
I'm sorry, Dave. I'm afraid I can't do that.
```

□□ Energy

Write a function to calculate the **energy** of an object in its rest frame given the **mass**.

The formula is

$$e = mc^2$$

where $c = 299792458$ m/s

Function specification

- name: `energy`
- parameters: mass (`float`)
- return: energy (`float`)

Marking note

You don't need to print anything to pass this Challenge. But it might be helpful to check your result e.g.

```
> print(energy(100))
8.987551787368176e+18
```

3. Python data manipulation

This task shows you how you can read, extract, modify data with Pandas package.

Country alcohol consumption

In this exercise, you will work with a dataset describing alcohol drinking pattern in each country.

Feedback Survey

Please give the teaching team some feedback for this week!

Question 1 *Submitted Feb 25th 2022 at 4:00:22 pm*

How did you feel after the tutorial?

- [x] Great!
- [ ] Satisfied
- [ ] Disappointed
- [ ] Confused

Question 2 *Submitted Feb 25th 2022 at 4:00:48 pm*

How do you feel about the speed of the tutorial?

- [ ] Too fast
- [x] Just right
- [ ] Too slow

Question 3 *Submitted Feb 25th 2022 at 4:00:50 pm*

How do you feel about the difficulty of concepts?

- [ ] Too hard
- [x] Just right

- ( ) Too easy

Question 4 *Submitted Feb 25th 2022 at 4:00:29 pm*

The material was clear and easy to understand

- (•) True
- ( ) False

Question 5 *Submitted Feb 25th 2022 at 4:00:27 pm*

The tutor was clear and easy to understand

- (•) True
- ( ) False

Question 6 *Submitted Feb 23rd 2022 at 12:51:57 pm*

Please write any general comments, feedback or elaborate on your previous answers.

Good Job !!!