Module: CET 313 ARTIFICIAL INTELLIGENCE

Name: Dimple Kadayat

Student ID: 219465005

Programme: Bsc. (Hons) Computer System Engineering

Centre: University of Sunderland

E-Portfolio
Documentation

My E-portfolio link is given below:

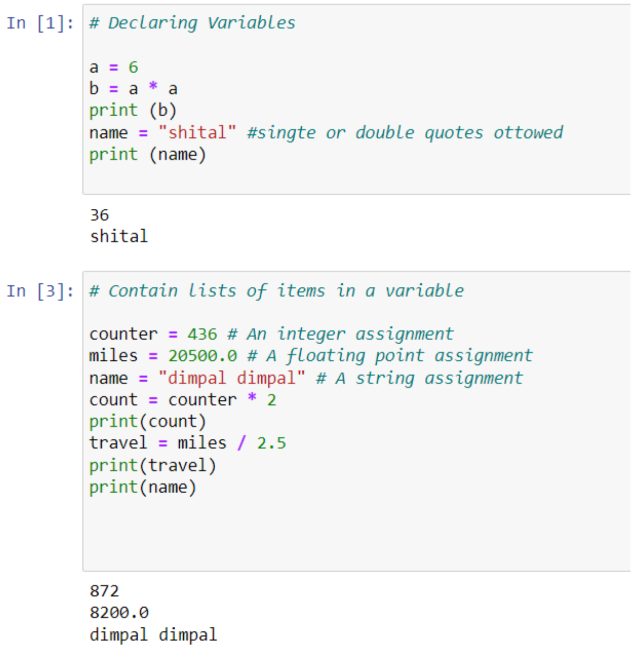
https://canvas.sunderland.ac.uk/eportfolios/7730?verifier=IRv5vhOHhyZX8ykWMfJ1ctLnaY62WawtFOExxNAe

Introduction

This was my first trimester at the University of Sunderland, London. It was pretty hard for me this trimester to understand how to use the canvas service, e-library provided by the university along with assignment submission portal of the units chosen. This trimester, everything was new for me as I'm a new comer and have to deal with many new things. I got to learn so many things which I haven't learnt in my past years and in my past courses. Talking about Artificial Intelligence and machine learning, I got to learn many new things whereas some units were quite similar with what I did in my past years. I personally felt luckier as I got chance to study the course material created by Sunderland University which was easy to understand and helpful as well. It is the one behind successful completion of this unit as it made even easier to understand the whole chapter. I'm really grateful towards the course material for all the achievements and knowledge I have gained. This unit is more about Artificial Intelligence, machine learning, neural network and many more. Few parts of the unit were easy for me as I was familiar with some of the terms whereas some of them like: natural language processing, relationships and some searching algorithms. All the summary about these things and the weekly tasks is mentioned below.

WEEK 1:

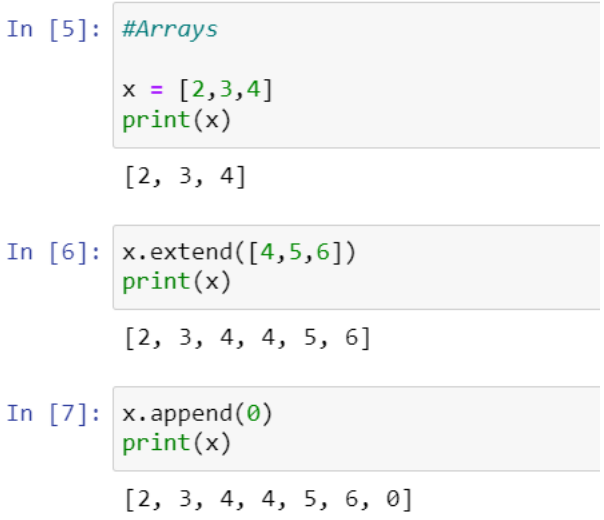
In the first week of our session, we learned about the introduction to AI and the programs used in AI. In this week, we learned the basic knowledge about python programming and its development. We got chance to learn about making comments in python, and running codes on Jupyter notebook IDE. Also, we were taught about using variables in python which is used for storing the information. The screen shot below shows the using of variables in python.



```
In [1]: # Declaring Variables

        a = 6
        b = a * a
        print (b)
        name = "shital" #singte or double quotes ottowed
        print (name)

        36
        shital

In [3]: # Contain lists of items in a variable

        counter = 436 # An integer assignment
        miles = 20500.0 # A floating point assignment
        name = "dimpal dimpal" # A string assignment
        count = counter * 2
        print(count)
        travel = miles / 2.5
        print(travel)
        print(name)

        872
        8200.0
        dimpal dimpal
```

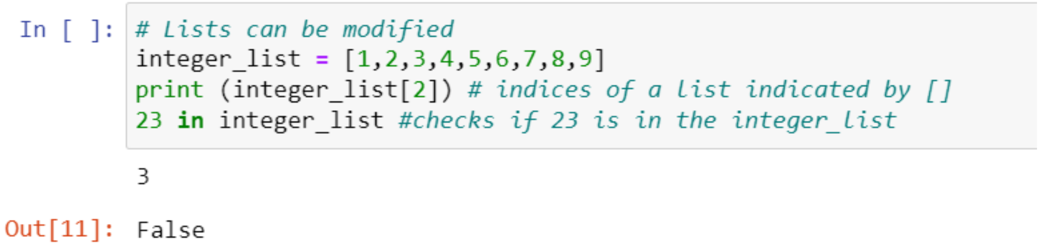
I learned how to use the following using python.

1. Arrays



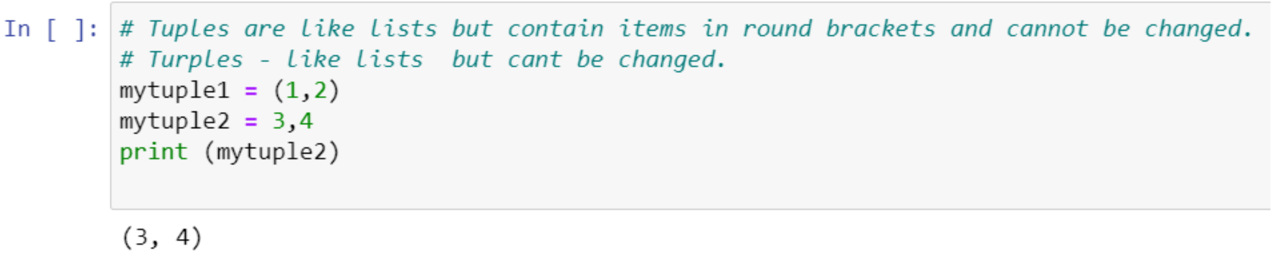
```
In [5]: #Arrays

        x = [2,3,4]
        print(x)

        [2, 3, 4]

In [6]: x.extend([4,5,6])
        print(x)

        [2, 3, 4, 4, 5, 6]

In [7]: x.append(0)
        print(x)

        [2, 3, 4, 4, 5, 6, 0]
```

2. Lists



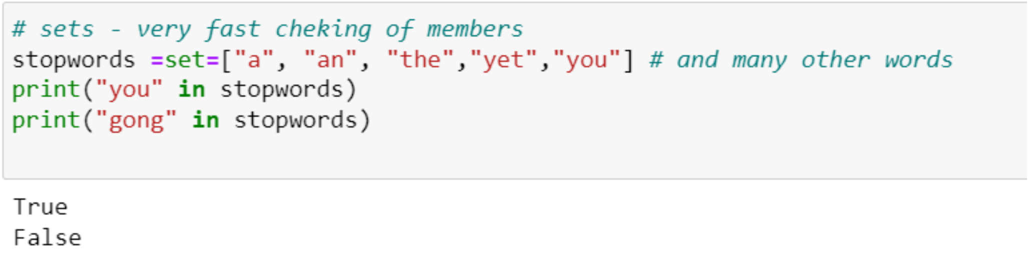
```
In [ ]: # Lists can be modified
        integer_list = [1,2,3,4,5,6,7,8,9]
        print (integer_list[2]) # indices of a list indicated by []
        23 in integer_list #checks if 23 is in the integer_list

        3

Out[11]: False
```

3. Tuples



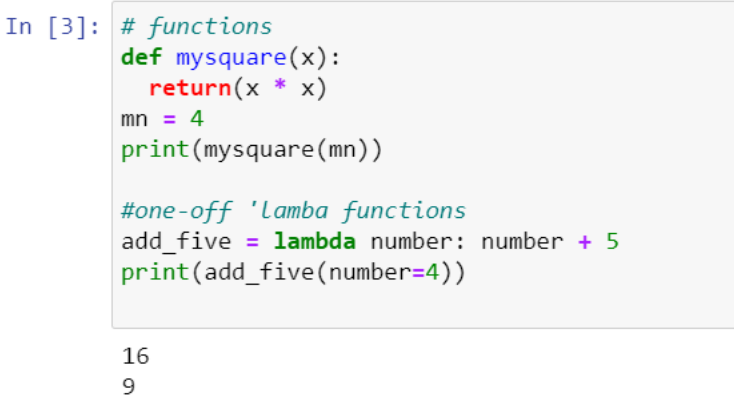
```
In [ ]: # Tuples are like lists but contain items in round brackets and cannot be changed.
        # Turples - like lists  but cant be changed.
        mytuple1 = (1,2)
        mytuple2 = 3,4
        print (mytuple2)

        (3, 4)
```

4. Dictionaries



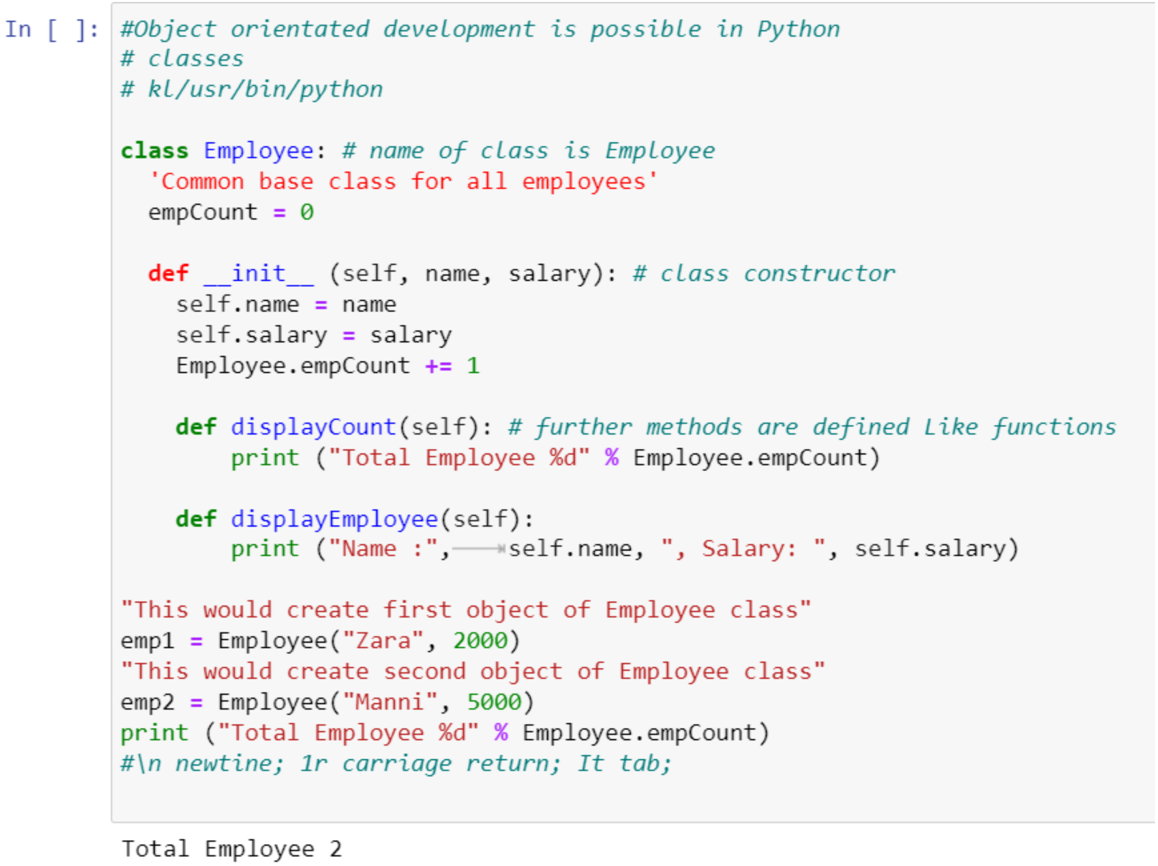
```
# sets - very fast cheking of members
stopwords =set=["a", "an", "the","yet","you"] # and many other words
print("you" in stopwords)
print("gong" in stopwords)

True
False
```

5. Functions



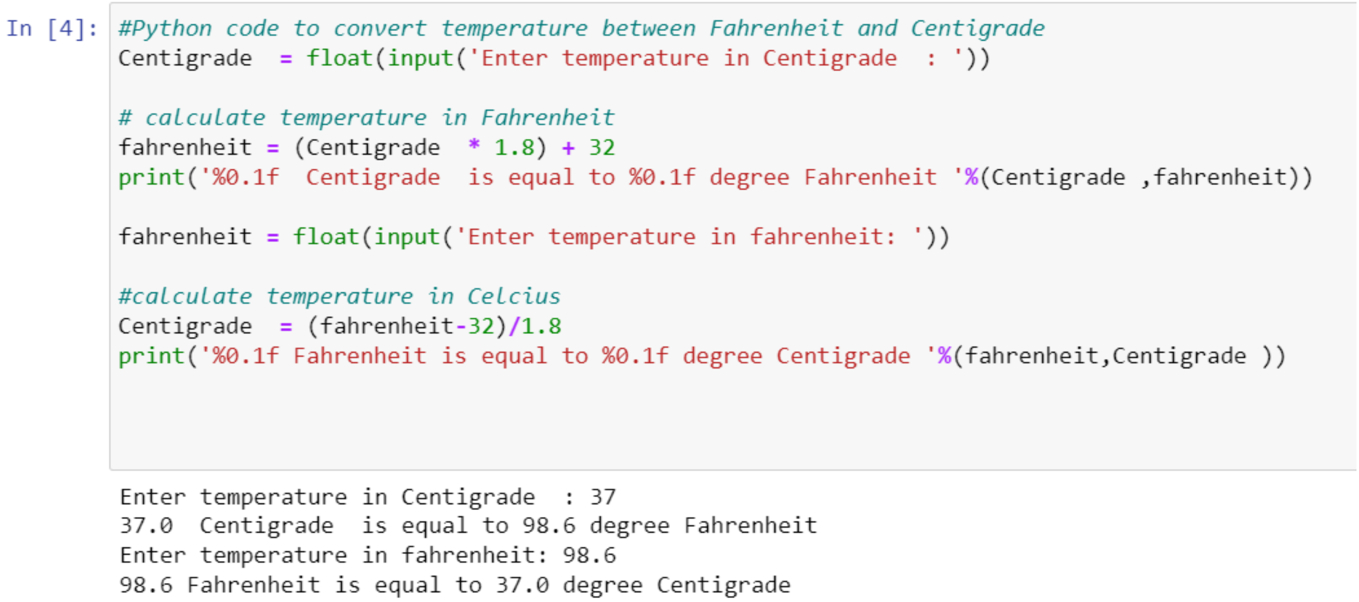
```
In [3]: # functions
        def mysquare(x):
          return(x * x)
        mn = 4
        print(mysquare(mn))

        #one-off 'lamba functions
        add_five = lambda number: number + 5
        print(add_five(number=4))

        16
        9
```

6. Classes



```
In [ ]: #Object orientated development is possible in Python
        # classes
        # kl/usr/bin/python

        class Employee: # name of class is Employee
          'Common base class for all employees'
          empCount = 0

          def __init__ (self, name, salary): # class constructor
            self.name = name
            self.salary = salary
            Employee.empCount += 1

            def displayCount(self): # further methods are defined Like functions
                print ("Total Employee %d" % Employee.empCount)

            def displayEmployee(self):
                print ("Name :",→self.name, ", Salary: ", self.salary)

        "This would create first object of Employee class"
        emp1 = Employee("Zara", 2000)
        "This would create second object of Employee class"
        emp2 = Employee("Manni", 5000)
        print ("Total Employee %d" % Employee.empCount)
        #\n newtine; 1r carriage return; It tab;

        Total Employee 2
```

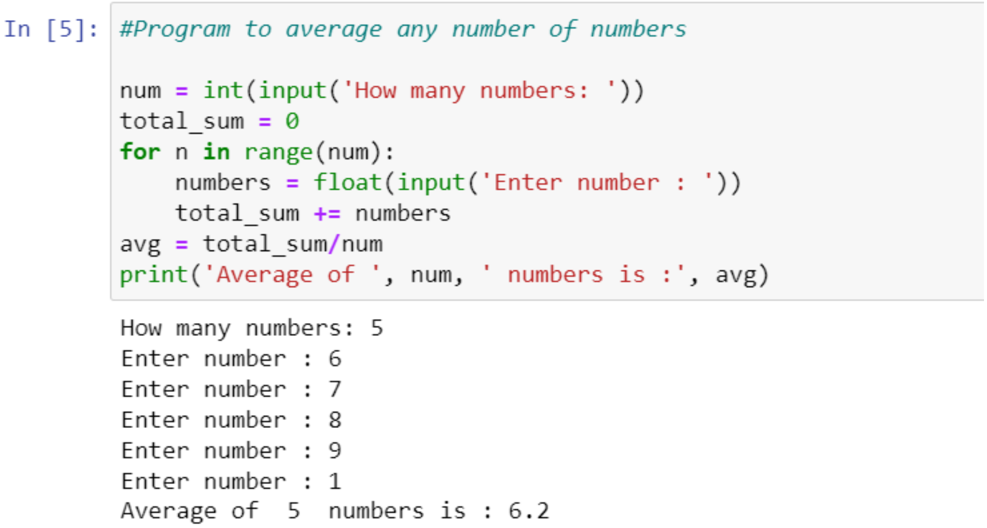
I managed to create a simple conversion between Fahrenheit and centigrade.



```
In [4]: #Python code to convert temperature between Fahrenheit and Centigrade
        Centigrade  = float(input('Enter temperature in Centigrade  : '))

        # calculate temperature in Fahrenheit
        fahrenheit = (Centigrade  * 1.8) + 32
        print('%0.1f  Centigrade  is equal to %0.1f degree Fahrenheit '%(Centigrade ,fahrenheit))

        fahrenheit = float(input('Enter temperature in fahrenheit: '))

        #calculate temperature in Celcius
        Centigrade  = (fahrenheit-32)/1.8
        print('%0.1f Fahrenheit is equal to %0.1f degree Centigrade '%(fahrenheit,Centigrade ))
```

```
Enter temperature in Centigrade  : 37
37.0  Centigrade  is equal to 98.6 degree Fahrenheit
Enter temperature in fahrenheit: 98.6
98.6 Fahrenheit is equal to 37.0 degree Centigrade
```

I managed to create a simple program to average any number of numbers.



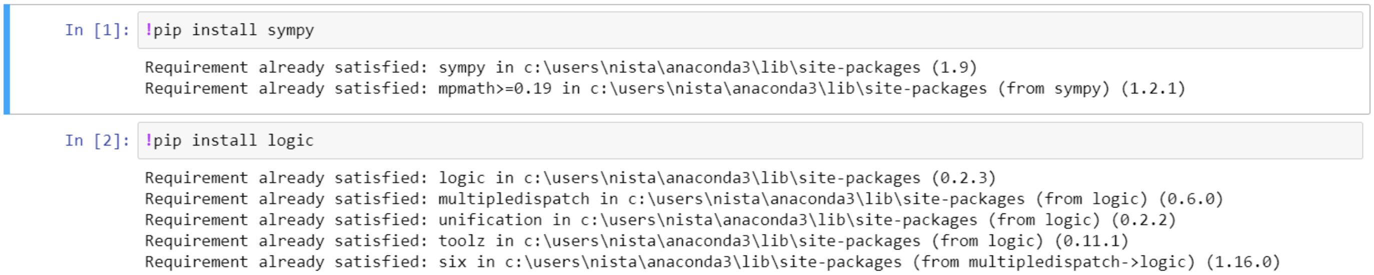
```
In [5]: #Program to average any number of numbers

        num = int(input('How many numbers: '))
        total_sum = 0
        for n in range(num):
            numbers = float(input('Enter number : '))
            total_sum += numbers
        avg = total_sum/num
        print('Average of ', num, ' numbers is :', avg)
```

```
How many numbers: 5
Enter number : 6
Enter number : 7
Enter number : 8
Enter number : 9
Enter number : 1
Average of  5  numbers is : 6.2
```

In this way I have completed the weekly task of first week.

WEEK 2:

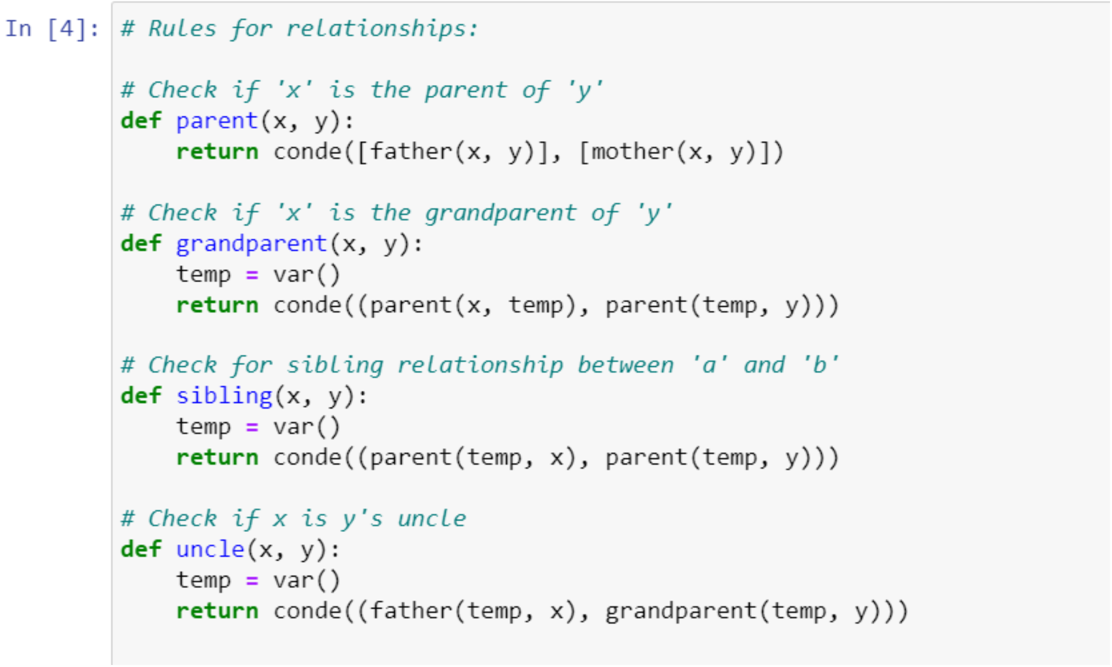
In second week we learned about the logic in artificial intelligence. Using the pip install command logpy and sympy were installed. For working with the relationship relationship.json was imported.



```
In [1]: !pip install sympy

        Requirement already satisfied: sympy in c:\users\nista\anaconda3\lib\site-packages (1.9)
        Requirement already satisfied: mpmath>=0.19 in c:\users\nista\anaconda3\lib\site-packages (from sympy) (1.2.1)

In [2]: !pip install logic

        Requirement already satisfied: logic in c:\users\nista\anaconda3\lib\site-packages (0.2.3)
        Requirement already satisfied: multipledispatch in c:\users\nista\anaconda3\lib\site-packages (from logic) (0.6.0)
        Requirement already satisfied: unification in c:\users\nista\anaconda3\lib\site-packages (from logic) (0.2.2)
        Requirement already satisfied: toolz in c:\users\nista\anaconda3\lib\site-packages (from logic) (0.11.1)
        Requirement already satisfied: six in c:\users\nista\anaconda3\lib\site-packages (from multipledispatch->logic) (1.16.0)
```

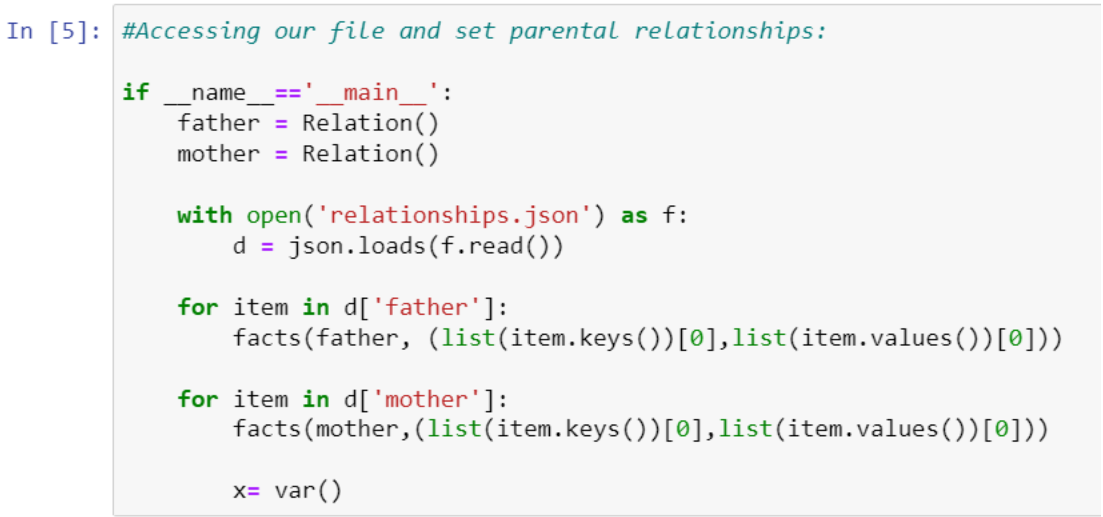
After that I:

Defined some rules for relationships.



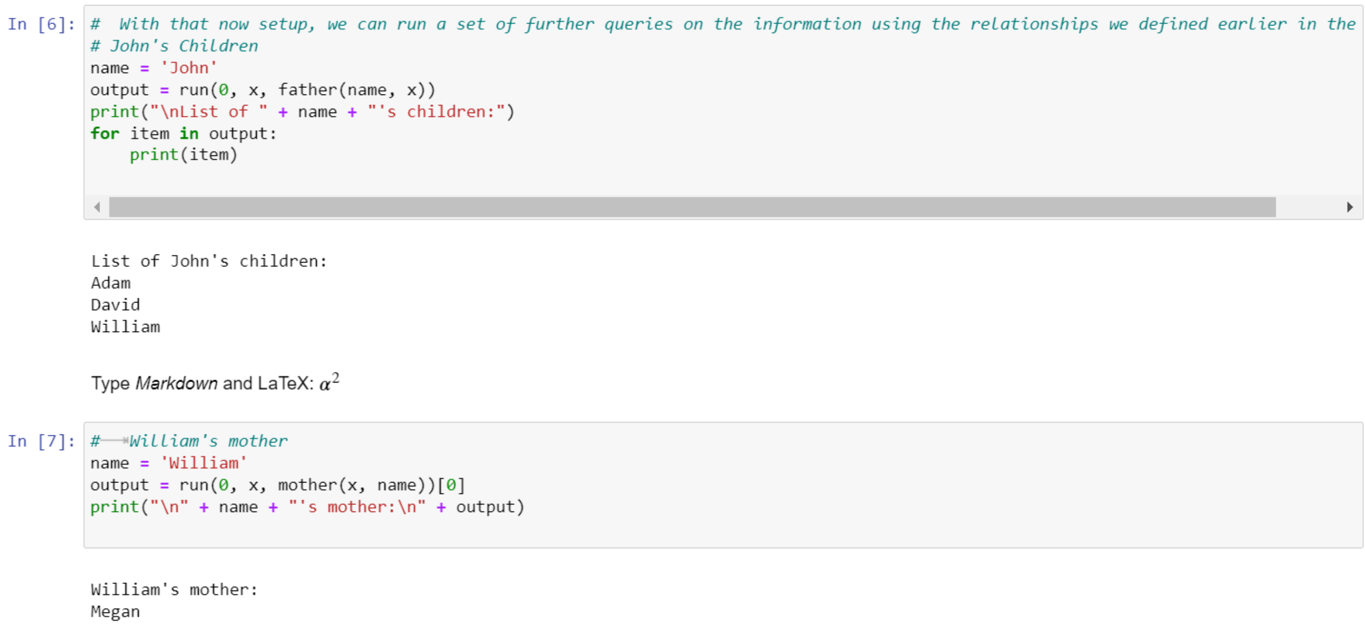
```
In [4]: # Rules for relationships:

        # Check if 'x' is the parent of 'y'
        def parent(x, y):
            return conde([father(x, y)], [mother(x, y)])

        # Check if 'x' is the grandparent of 'y'
        def grandparent(x, y):
            temp = var()
            return conde((parent(x, temp), parent(temp, y)))

        # Check for sibling relationship between 'a' and 'b'
        def sibling(x, y):
            temp = var()
            return conde((parent(temp, x), parent(temp, y)))

        # Check if x is y's uncle
        def uncle(x, y):
            temp = var()
            return conde((father(temp, x), grandparent(temp, y)))
```

access the file and set parental relationships.



```
In [5]: #Accessing our file and set parental relationships:

        if __name__=='__main__':
            father = Relation()
            mother = Relation()

            with open('relationships.json') as f:
                d = json.loads(f.read())

            for item in d['father']:
                facts(father, (list(item.keys())[0],list(item.values())[0]))

            for item in d['mother']:
                facts(mother,(list(item.keys())[0],list(item.values())[0]))

                x= var()
```

With the use of relationships that I defined earlier in the code run a set of further queries on the information.



```
In [6]: #  With that now setup, we can run a set of further queries on the information using the relationships we defined earlier in the
        # John's Children
        name = 'John'
        output = run(0, x, father(name, x))
        print("\nList of " + name + "'s children:")
        for item in output:
            print(item)
```

```

List of John's children:
Adam
David
William
```

Type *Markdown* and LaTeX: α^2

```
In [7]: #	William's mother
        name = 'William'
        output = run(0, x, mother(x, name))[0]
        print("\n" + name + "'s mother:\n" + output)
```

```

William's mother:
Megan
```

In [8]:

```python
# Adam's parents
name = 'Adam'
output = run(0, x, parent(x, name))
print("\nList of " + name + "'s parents:")
for item in output:
  print(item)
```

```
List of Adam's parents:
John
Megan
```

In [9]:

```python
# Wayne's grandparents
name = 'Wayne'
output = run(0, x, grandparent(x, name))
print("\nList of " + name + "'s grandparents:")
for item in output:
  print(item)
```

```
List of Wayne's grandparents:
Megan
John
```

In [10]:

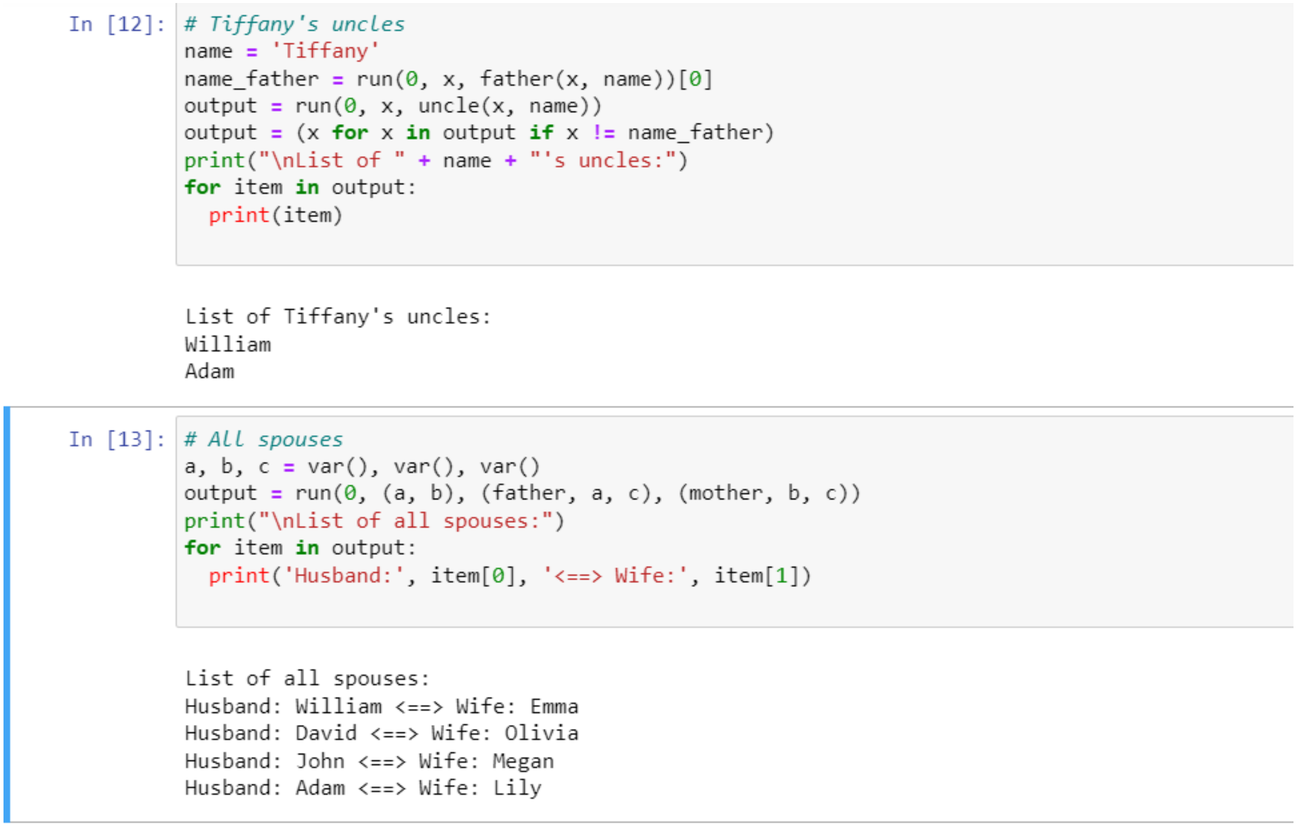
```python
#	Megan's grandchildren
name = 'Megan'
output = run(0, x, grandparent(name, x))
print("\nList of " + name + "'s grandchildren:")
for item in output:
  print(item)
```

```
List of Megan's grandchildren:
Chris
Sophia
Tiffany
Stephanie
Peter
Neil
Wayne
Julie
```

In [11]:

```python
# David's siblings
name = 'David'
output = run(0, x, sibling(x, name))
siblings = (x for x in output if x != name)
print("\nList of " + name + "'s siblings:")
for item in siblings:
  print(item)
```

```
List of David's siblings:
William
Adam
```



In [12]:

```
# Tiffany's uncles
name = 'Tiffany'
name_father = run(0, x, father(x, name))[0]
output = run(0, x, uncle(x, name))
output = (x for x in output if x != name_father)
print("\nList of " + name + "'s uncles:")
for item in output:
  print(item)
```

```
List of Tiffany's uncles:
William
Adam
```

In [13]:

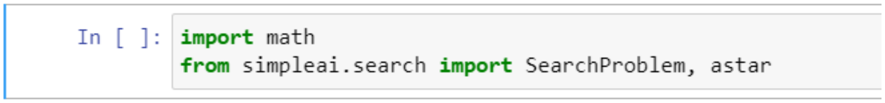
```
# All spouses
a, b, c = var(), var(), var()
output = run(0, (a, b), (father, a, c), (mother, b, c))
print("\nList of all spouses:")
for item in output:
  print('Husband:', item[0], '<==> Wife:', item[1])
```

```
List of all spouses:
Husband: William <==> Wife: Emma
Husband: David <==> Wife: Olivia
Husband: John <==> Wife: Megan
Husband: Adam <==> Wife: Lily
```

Week 3:

Here, I am going to use the the algorithm to solve a maze.

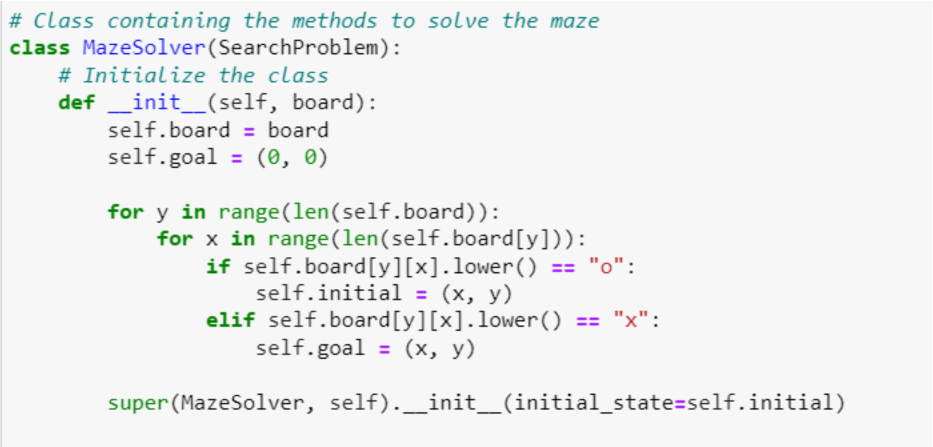
The first thing, I did was importing the necessary libraries as shown below:



In [ ]:

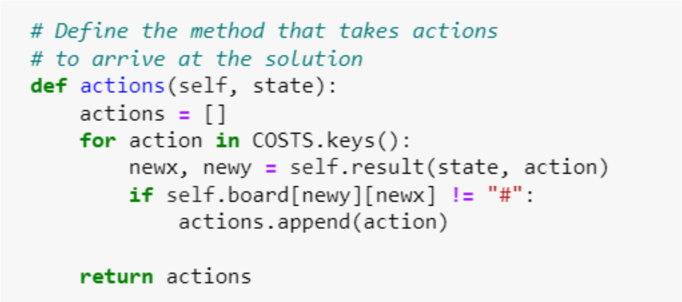
```
import math
from simpleai.search import SearchProblem, astar
```

Then, I created a class containing the method that is needed for solving the issue.



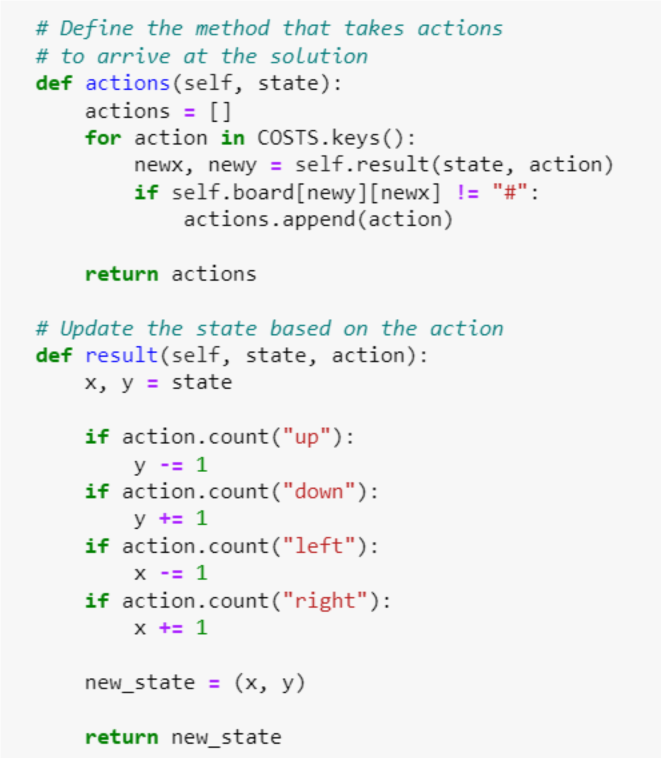
```
# Class containing the methods to solve the maze
class MazeSolver(SearchProblem):
    # Initialize the class
    def __init__(self, board):
        self.board = board
        self.goal = (0, 0)

        for y in range(len(self.board)):
            for x in range(len(self.board[y])):
                if self.board[y][x].lower() == "o":
                    self.initial = (x, y)
                elif self.board[y][x].lower() == "x":
                    self.goal = (x, y)

        super(MazeSolver, self).__init__(initial_state=self.initial)
```

I then used the code below to define the method that takes action to reach to the solution.



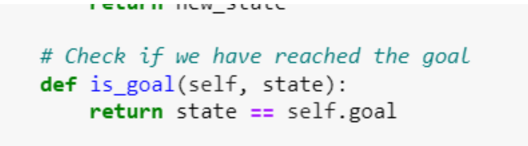
```
# Define the method that takes actions
# to arrive at the solution
def actions(self, state):
    actions = []
    for action in COSTS.keys():
        newx, newy = self.result(state, action)
        if self.board[newy][newx] != "#":
            actions.append(action)

    return actions
```

The state was then defined and updated based on the activity.



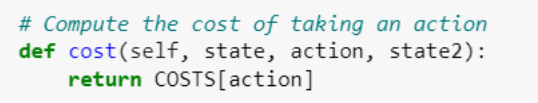
```
# Define the method that takes actions
# to arrive at the solution
def actions(self, state):
    actions = []
    for action in COSTS.keys():
        newx, newy = self.result(state, action)
        if self.board[newy][newx] != "#":
            actions.append(action)

    return actions

# Update the state based on the action
def result(self, state, action):
    x, y = state

    if action.count("up"):
        y -= 1
    if action.count("down"):
        y += 1
    if action.count("left"):
        x -= 1
    if action.count("right"):
        x += 1

    new_state = (x, y)

    return new_state
```

I used the code below to see if I had met my aim.



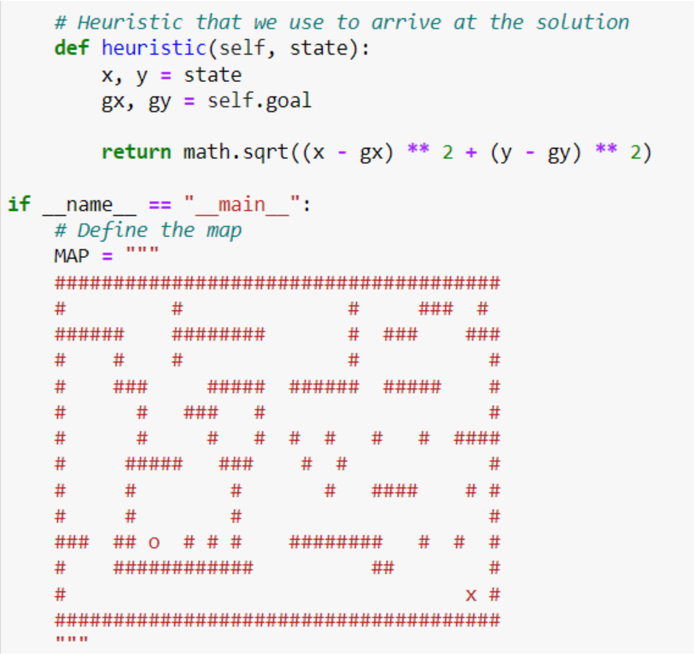
```
        return new_state

    # Check if we have reached the goal
    def is_goal(self, state):
        return state == self.goal
```

I used the code below to calculate the cost of doing the action.



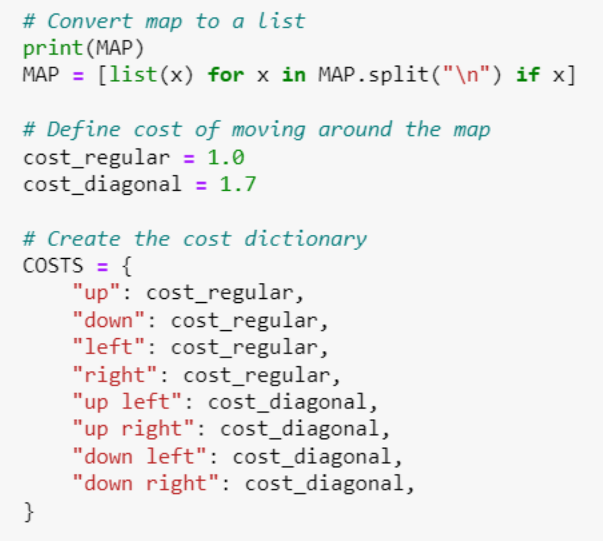
```
# Compute the cost of taking an action
def cost(self, state, action, state2):
    return COSTS[action]
```

The heuristic used to obtain the result answer is listed below.



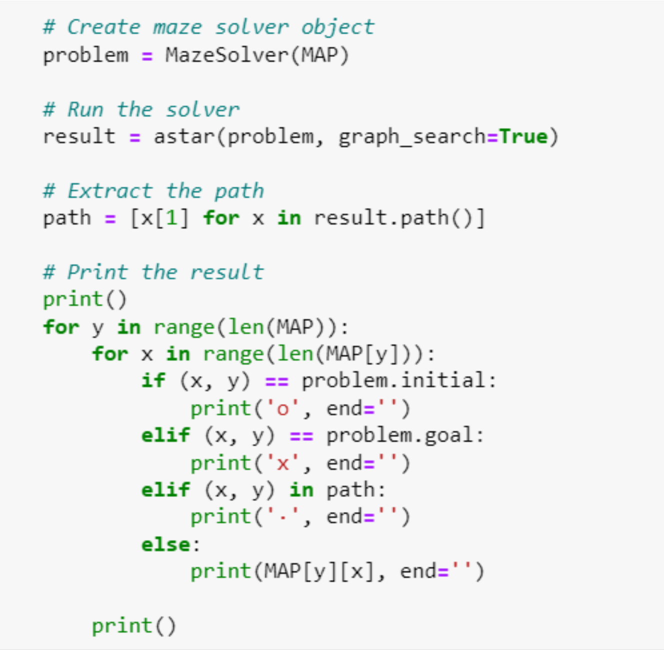
```
    # Heuristic that we use to arrive at the solution
    def heuristic(self, state):
        x, y = state
        gx, gy = self.goal

        return math.sqrt((x - gx) ** 2 + (y - gy) ** 2)

if __name__ == "__main__":
    # Define the map
    MAP = """
    ######################################
    #         #              #     ###  #
    ######    ########       #  ###    ###
    #    #    #              #           #
    #    ###     #####  ######  #####    #
    #      #   ###   #                   #
    #      #     #   #  #  #   #   #  ####
    #     #####   ###    #  #            #
    #     #        #       #   ####    # #
    #     #        #                     #
    ###  ## o  # # #    ########   #  #  #
    #    ############          ##        #
    #                                  x #
    ######################################
    """
```

Creating the convert map to list



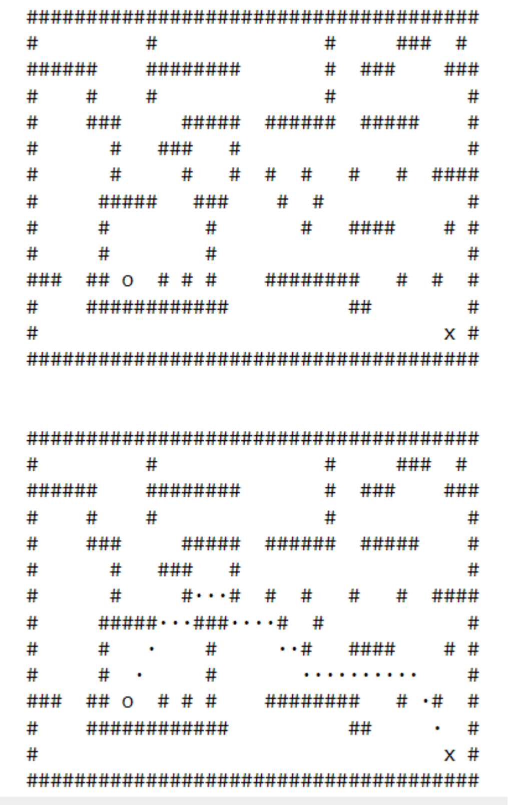
```
# Convert map to a list
print(MAP)
MAP = [list(x) for x in MAP.split("\n") if x]

# Define cost of moving around the map
cost_regular = 1.0
cost_diagonal = 1.7

# Create the cost dictionary
COSTS = {
    "up": cost_regular,
    "down": cost_regular,
    "left": cost_regular,
    "right": cost_regular,
    "up left": cost_diagonal,
    "up right": cost_diagonal,
    "down left": cost_diagonal,
    "down right": cost_diagonal,
}
```

Creating the maze solver object



```
# Create maze solver object
problem = MazeSolver(MAP)

# Run the solver
result = astar(problem, graph_search=True)

# Extract the path
path = [x[1] for x in result.path()]

# Print the result
print()
for y in range(len(MAP)):
    for x in range(len(MAP[y])):
        if (x, y) == problem.initial:
            print('o', end='')
        elif (x, y) == problem.goal:
            print('x', end='')
        elif (x, y) in path:
            print('·', end='')
        else:
            print(MAP[y][x], end='')

    print()
```

Code output:



```
######################################
#         #              #     ###  #
######    ########       #  ###    ###
#    #    #              #           #
#    ###     #####  ######  #####    #
#      #   ###   #                   #
#      #     #   #  #  #   #   #  ####
#     #####   ###    #  #            #
#     #        #       #   ####    # #
#     #        #                     #
###  ## o  # # #    ########   #  #  #
#    ############          ##        #
#                                  x #
######################################


######################################
#         #              #     ###  #
######    ########       #  ###    ###
#    #    #              #           #
#    ###     #####  ######  #####    #
#      #   ###   #                   #
#      #     #···#  #  #   #   #  ####
#     #####···###····#  #            #
#     #   ·    #     ··#   ####    # #
#     #  ·     #       ··········    #
###  ## o  # # #    ########   # ·#  #
#    ############          ##     ·  #
#                                  x #
######################################
```

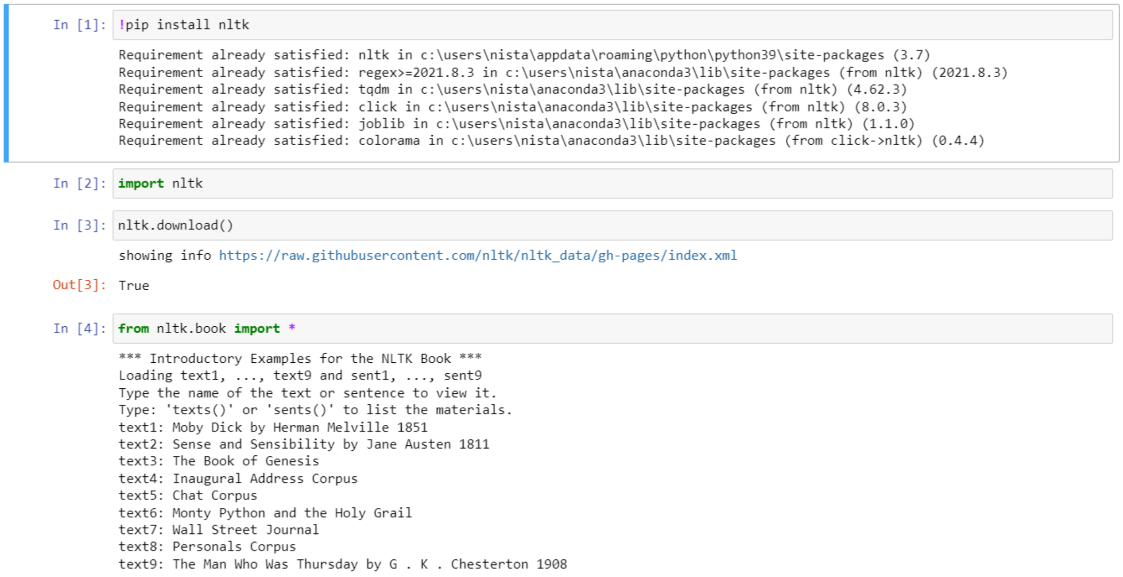
I researched the suggested search algorithms and methodology and built a scenario in which each one might be used, along with a list of benefits and drawbacks for each.

Week 4

In this week we learned about the natural language processing, scope of natural language processing, example of the applications and discusse the complex approaches.

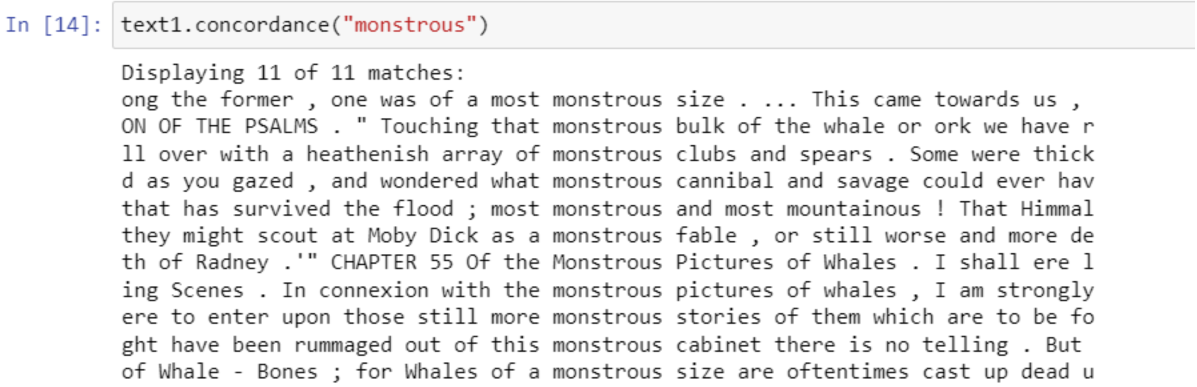
To complete the weekly task,

First, I used the following command to import all of the packages and libraries from nltk.book.

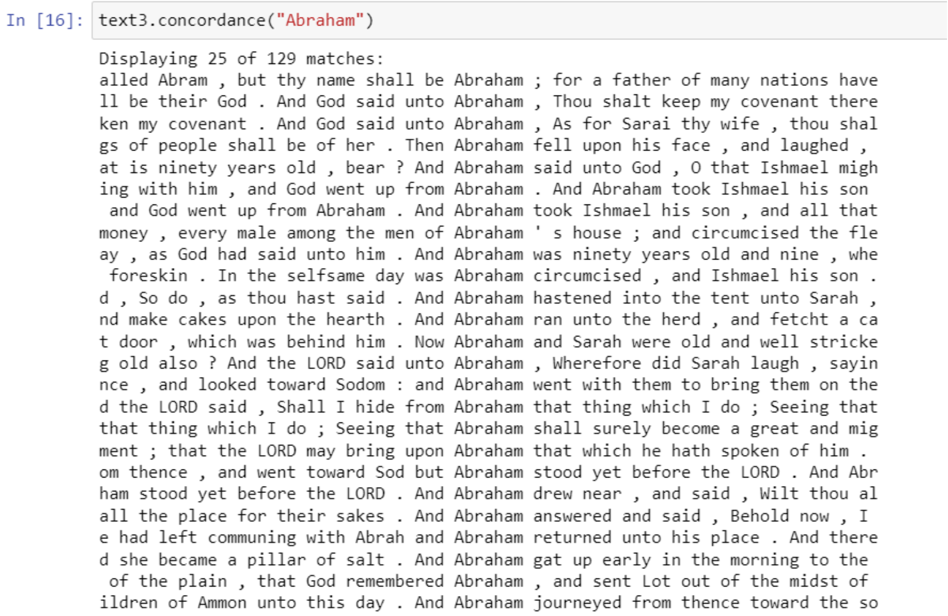


```
In [1]: !pip install nltk

        Requirement already satisfied: nltk in c:\users\nista\appdata\roaming\python\python39\site-packages (3.7)
        Requirement already satisfied: regex>=2021.8.3 in c:\users\nista\anaconda3\lib\site-packages (from nltk) (2021.8.3)
        Requirement already satisfied: tqdm in c:\users\nista\anaconda3\lib\site-packages (from nltk) (4.62.3)
        Requirement already satisfied: click in c:\users\nista\anaconda3\lib\site-packages (from nltk) (8.0.3)
        Requirement already satisfied: joblib in c:\users\nista\anaconda3\lib\site-packages (from nltk) (1.1.0)
        Requirement already satisfied: colorama in c:\users\nista\anaconda3\lib\site-packages (from click->nltk) (0.4.4)

In [2]: import nltk

In [3]: nltk.download()

        showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

Out[3]: True

In [4]: from nltk.book import *

        *** Introductory Examples for the NLTK Book ***
        Loading text1, ..., text9 and sent1, ..., sent9
        Type the name of the text or sentence to view it.
        Type: 'texts()' or 'sents()' to list the materials.
        text1: Moby Dick by Herman Melville 1851
        text2: Sense and Sensibility by Jane Austen 1811
        text3: The Book of Genesis
        text4: Inaugural Address Corpus
        text5: Chat Corpus
        text6: Monty Python and the Holy Grail
        text7: Wall Street Journal
        text8: Personals Corpus
        text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

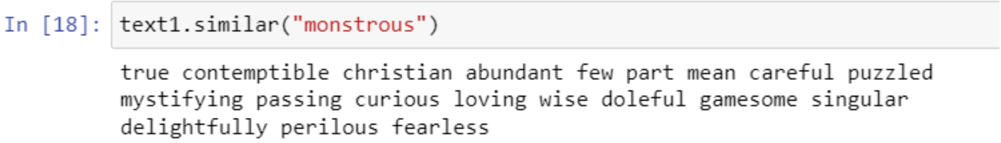
By applying the codes below, I was able to find other terms that were used in the same or comparable context.



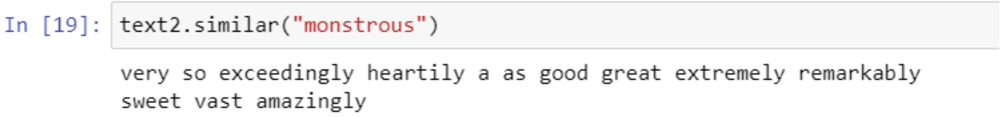
```
In [14]: text1.concordance("monstrous")

         Displaying 11 of 11 matches:
         ong the former , one was of a most monstrous size . ... This came towards us ,
         ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
         ll over with a heathenish array of monstrous clubs and spears . Some were thick
         d as you gazed , and wondered what monstrous cannibal and savage could ever hav
         that has survived the flood ; most monstrous and most mountainous ! That Himmal
         they might scout at Moby Dick as a monstrous fable , or still worse and more de
         th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
         ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
         ere to enter upon those still more monstrous stories of them which are to be fo
         ght have been rummaged out of this monstrous cabinet there is no telling . But
         of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
```



```
In [16]: text3.concordance("Abraham")

         Displaying 25 of 129 matches:
         alled Abram , but thy name shall be Abraham ; for a father of many nations have
         ll be their God . And God said unto Abraham , Thou shalt keep my covenant there
         ken my covenant . And God said unto Abraham , As for Sarai thy wife , thou shal
         gs of people shall be of her . Then Abraham fell upon his face , and laughed ,
         at is ninety years old , bear ? And Abraham said unto God , O that Ishmael migh
         ing with him , and God went up from Abraham . And Abraham took Ishmael his son
          and God went up from Abraham . And Abraham took Ishmael his son , and all that
         money , every male among the men of Abraham ' s house ; and circumcised the fle
         ay , as God had said unto him . And Abraham was ninety years old and nine , whe
          foreskin . In the selfsame day was Abraham circumcised , and Ishmael his son .
         d , So do , as thou hast said . And Abraham hastened into the tent unto Sarah ,
         nd make cakes upon the hearth . And Abraham ran unto the herd , and fetcht a ca
         t door , which was behind him . Now Abraham and Sarah were old and well stricke
         g old also ? And the LORD said unto Abraham , Wherefore did Sarah laugh , sayin
         nce , and looked toward Sodom : and Abraham went with them to bring them on the
         d the LORD said , Shall I hide from Abraham that thing which I do ; Seeing that
         that thing which I do ; Seeing that Abraham shall surely become a great and mig
         ment ; that the LORD may bring upon Abraham that which he hath spoken of him .
         om thence , and went toward Sod but Abraham stood yet before the LORD . And Abr
         ham stood yet before the LORD . And Abraham drew near , and said , Wilt thou al
         all the place for their sakes . And Abraham answered and said , Behold now , I
         e had left communing with Abrah and Abraham returned unto his place . And there
         d she became a pillar of salt . And Abraham gat up early in the morning to the
          of the plain , that God remembered Abraham , and sent Lot out of the midst of
         ildren of Ammon unto this day . And Abraham journeyed from thence toward the so
```

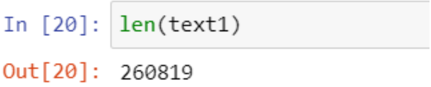


```
In [18]: text1.similar("monstrous")

         true contemptible christian abundant few part mean careful puzzled
         mystifying passing curious loving wise doleful gamesome singular
         delightfully perilous fearless
```

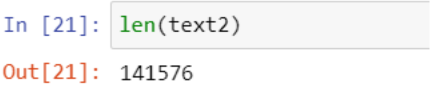


```
In [19]: text2.similar("monstrous")

         very so exceedingly heartily a as good great extremely remarkably
         sweet vast amazingly
```

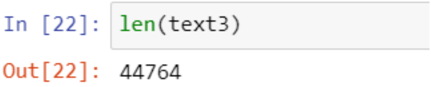
Calculating the length of the texts



```
In [20]: len(text1)

Out[20]: 260819
```

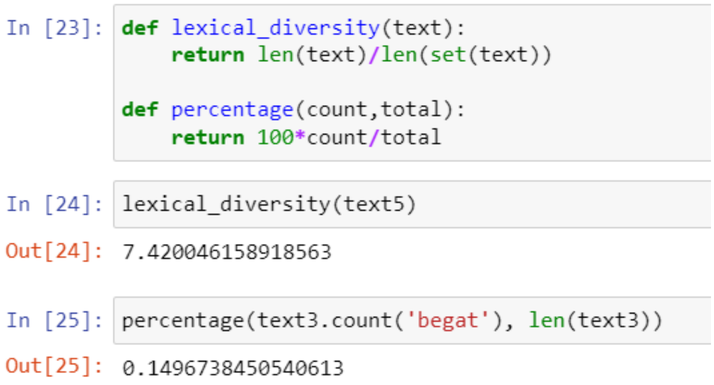


```
In [21]: len(text2)

Out[21]: 141576
```

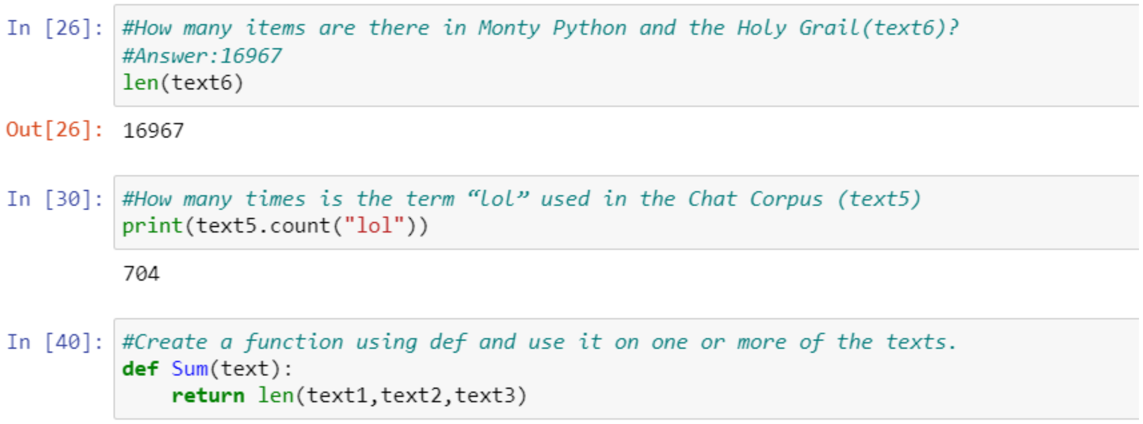


```
In [22]: len(text3)

Out[22]: 44764
```

I was able to utilize def to create a function that I could apply to one or more of the texts.



```
In [23]: def lexical_diversity(text):
             return len(text)/len(set(text))

         def percentage(count,total):
             return 100*count/total

In [24]: lexical_diversity(text5)

Out[24]: 7.420046158918563

In [25]: percentage(text3.count('begat'), len(text3))

Out[25]: 0.1496738450540613
```

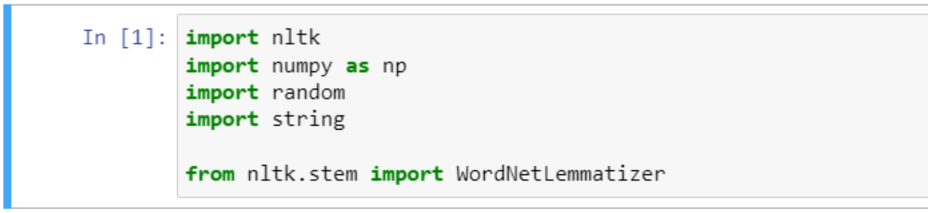


```
In [26]: #How many items are there in Monty Python and the Holy Grail(text6)?
         #Answer:16967
         len(text6)

Out[26]: 16967

In [30]: #How many times is the term “lol” used in the Chat Corpus (text5)
         print(text5.count("lol"))

         704

In [40]: #Create a function using def and use it on one or more of the texts.
         def Sum(text):
             return len(text1,text2,text3)
```

Week 5

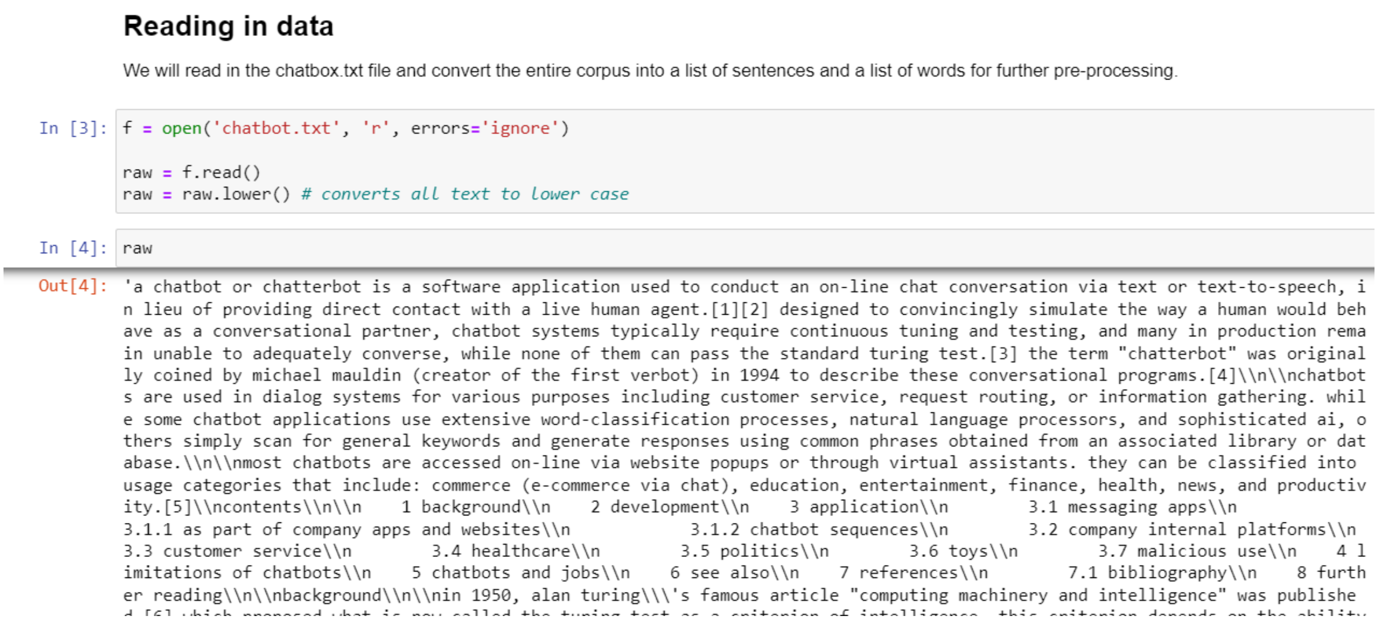
On the fifth week, I was assigned the task of building a chatbot from the scratch level in Python.

To develop a chatbot, I first needed to import the following packages.



```
In [1]: import nltk
        import numpy as np
        import random
        import string

        from nltk.stem import WordNetLemmatizer
```

Reading data from text file



Reading in data

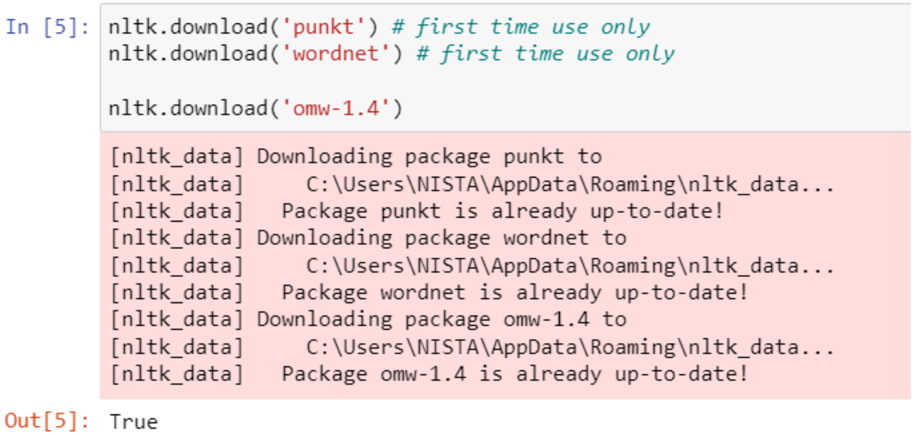
We will read in the chatbox.txt file and convert the entire corpus into a list of sentences and a list of words for further pre-processing.

```
In [3]: f = open('chatbot.txt', 'r', errors='ignore')

        raw = f.read()
        raw = raw.lower() # converts all text to lower case
```

```
In [4]: raw
```

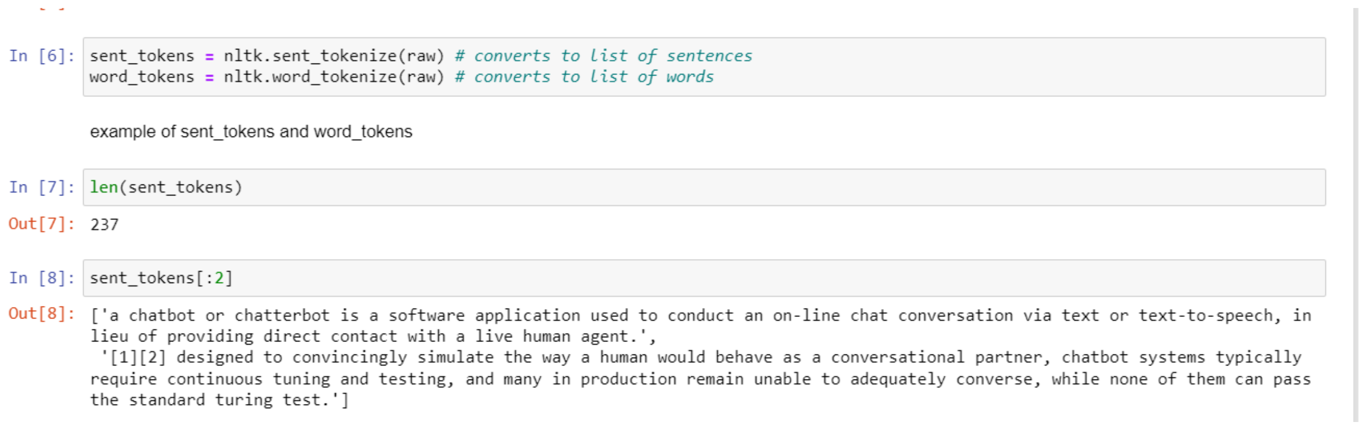
```
Out[4]: 'a chatbot or chatterbot is a software application used to conduct an on-line chat conversation via text or text-to-speech, i
        n lieu of providing direct contact with a live human agent.[1][2] designed to convincingly simulate the way a human would beh
        ave as a conversational partner, chatbot systems typically require continuous tuning and testing, and many in production rema
        in unable to adequately converse, while none of them can pass the standard turing test.[3] the term "chatterbot" was original
        ly coined by michael mauldin (creator of the first verbot) in 1994 to describe these conversational programs.[4]\\n\\nchatbot
        s are used in dialog systems for various purposes including customer service, request routing, or information gathering. whil
        e some chatbot applications use extensive word-classification processes, natural language processors, and sophisticated ai, o
        thers simply scan for general keywords and generate responses using common phrases obtained from an associated library or dat
        abase.\\n\\nmost chatbots are accessed on-line via website popups or through virtual assistants. they can be classified into
        usage categories that include: commerce (e-commerce via chat), education, entertainment, finance, health, news, and productiv
        ity.[5]\\ncontents\\n\\n    1 background\\n    2 development\\n    3 application\\n        3.1 messaging apps\\n
        3.1.1 as part of company apps and websites\\n            3.1.2 chatbot sequences\\n        3.2 company internal platforms\\n
        3.3 customer service\\n        3.4 healthcare\\n        3.5 politics\\n        3.6 toys\\n        3.7 malicious use\\n    4 l
        imitations of chatbots\\n    5 chatbots and jobs\\n    6 see also\\n    7 references\\n        7.1 bibliography\\n    8 furth
        er reading\\n\\nbackground\\n\\nin 1950, alan turing\\\'s famous article "computing machinery and intelligence" was publishe
        d [6] which proposed what is now called the turing test as a criterion of intelligence. this criterion depends on the ability
```

Download ‘punket’ and ‘wordnet’



```
In [5]: nltk.download('punkt') # first time use only
        nltk.download('wordnet') # first time use only

        nltk.download('omw-1.4')

        [nltk_data] Downloading package punkt to
        [nltk_data]     C:\Users\NISTA\AppData\Roaming\nltk_data...
        [nltk_data]   Package punkt is already up-to-date!
        [nltk_data] Downloading package wordnet to
        [nltk_data]     C:\Users\NISTA\AppData\Roaming\nltk_data...
        [nltk_data]   Package wordnet is already up-to-date!
        [nltk_data] Downloading package omw-1.4 to
        [nltk_data]     C:\Users\NISTA\AppData\Roaming\nltk_data...
        [nltk_data]   Package omw-1.4 is already up-to-date!

Out[5]: True
```

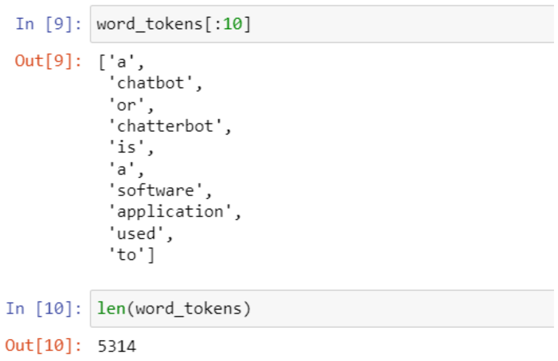
Setting up tokens



```
In [6]: sent_tokens = nltk.sent_tokenize(raw) # converts to list of sentences
        word_tokens = nltk.word_tokenize(raw) # converts to list of words
```

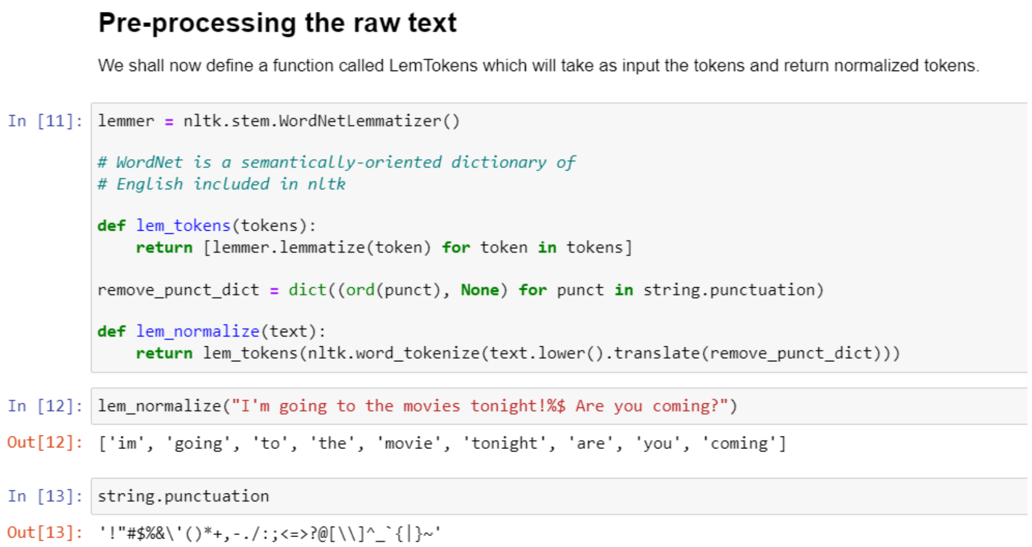
example of sent_tokens and word_tokens

```
In [7]: len(sent_tokens)

Out[7]: 237

In [8]: sent_tokens[:2]

Out[8]: ['a chatbot or chatterbot is a software application used to conduct an on-line chat conversation via text or text-to-speech, in
        lieu of providing direct contact with a live human agent.',
         '[1][2] designed to convincingly simulate the way a human would behave as a conversational partner, chatbot systems typically
        require continuous tuning and testing, and many in production remain unable to adequately converse, while none of them can pass
        the standard turing test.']
```



```
In [9]: word_tokens[:10]

Out[9]: ['a',
         'chatbot',
         'or',
         'chatterbot',
         'is',
         'a',
         'software',
         'application',
         'used',
         'to']

In [10]: len(word_tokens)

Out[10]: 5314
```

Raw data processing:

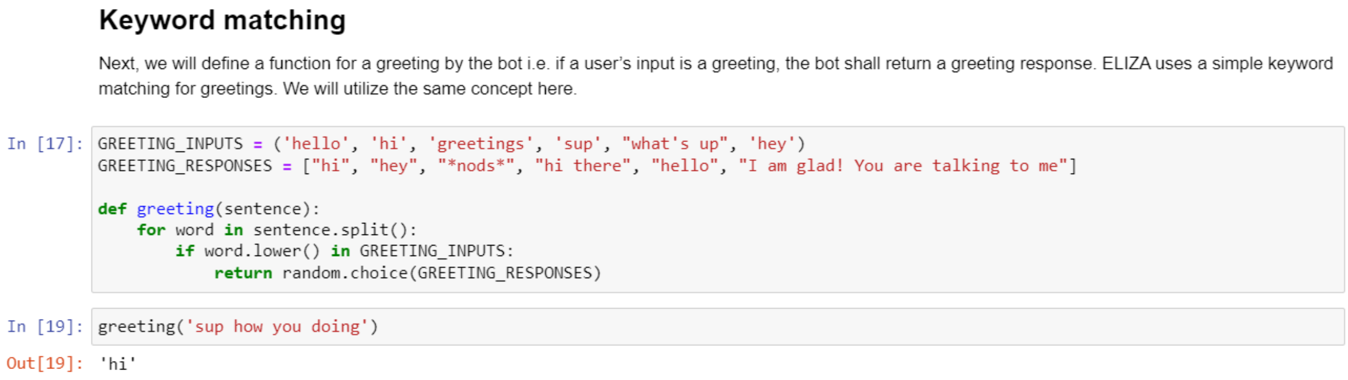


Pre-processing the raw text

We shall now define a function called LemTokens which will take as input the tokens and return normalized tokens.

```
In [11]: lemmer = nltk.stem.WordNetLemmatizer()

         # WordNet is a semantically-oriented dictionary of
         # English included in nltk

         def lem_tokens(tokens):
             return [lemmer.lemmatize(token) for token in tokens]

         remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)

         def lem_normalize(text):
             return lem_tokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))

In [12]: lem_normalize("I'm going to the movies tonight!%$ Are you coming?")

Out[12]: ['im', 'going', 'to', 'the', 'movie', 'tonight', 'are', 'you', 'coming']

In [13]: string.punctuation

Out[13]: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Creating the keyword matching:



Keyword matching

Next, we will define a function for a greeting by the bot i.e. if a user’s input is a greeting, the bot shall return a greeting response. ELIZA uses a simple keyword matching for greetings. We will utilize the same concept here.

In [17]:

```
GREETING_INPUTS = ('hello', 'hi', 'greetings', 'sup', "what's up", 'hey')
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are talking to me"]

def greeting(sentence):
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
```

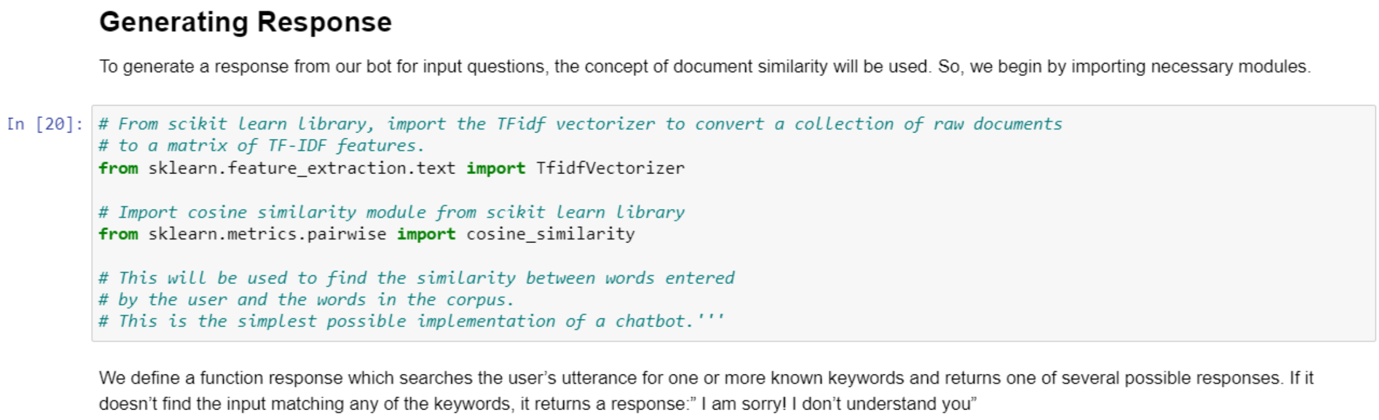
In [19]:

```
greeting('sup how you doing')
```

Out[19]:

```
'hi'
```

Generating the response:



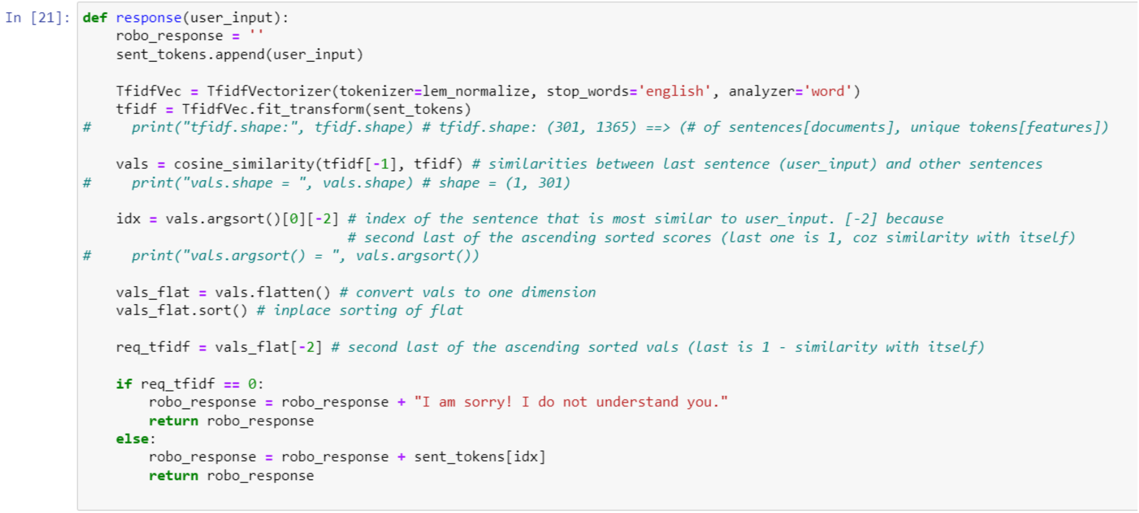
Generating Response

To generate a response from our bot for input questions, the concept of document similarity will be used. So, we begin by importing necessary modules.

In [20]:

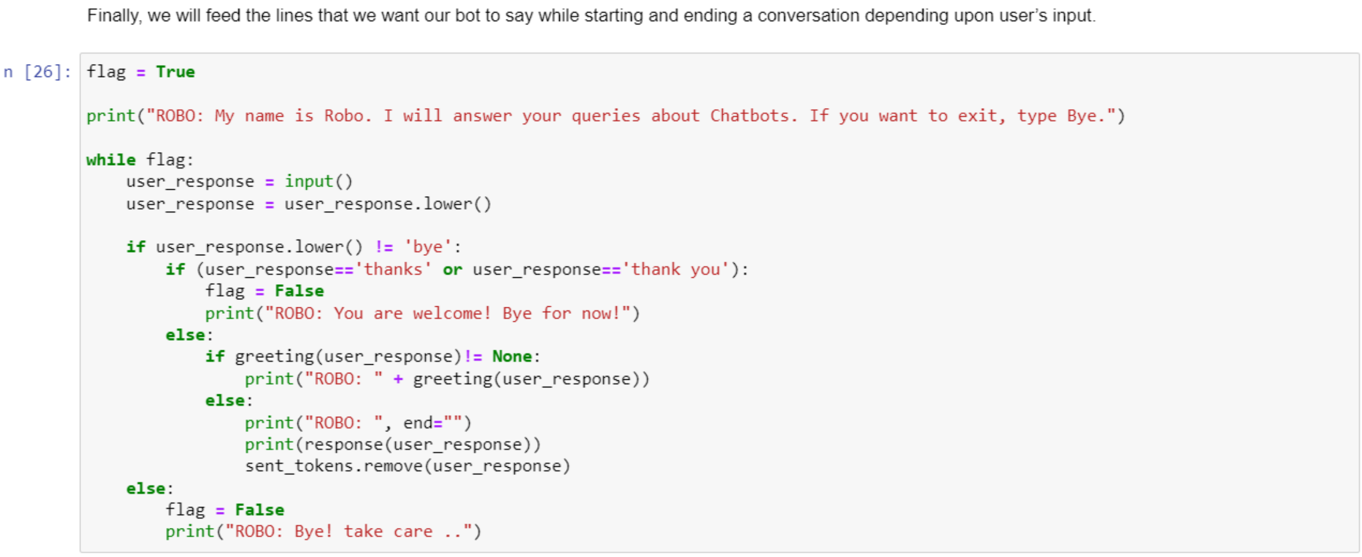
```
# From scikit learn library, import the TFidf vectorizer to convert a collection of raw documents
# to a matrix of TF-IDF features.
from sklearn.feature_extraction.text import TfidfVectorizer

# Import cosine similarity module from scikit learn library
from sklearn.metrics.pairwise import cosine_similarity

# This will be used to find the similarity between words entered
# by the user and the words in the corpus.
# This is the simplest possible implementation of a chatbot.'''
```

We define a function response which searches the user’s utterance for one or more known keywords and returns one of several possible responses. If it doesn’t find the input matching any of the keywords, it returns a response:” I am sorry! I don’t understand you”



```
In [21]: def response(user_input):
             robo_response = ''
             sent_tokens.append(user_input)

             TfidfVec = TfidfVectorizer(tokenizer=lem_normalize, stop_words='english', analyzer='word')
             tfidf = TfidfVec.fit_transform(sent_tokens)
         #     print("tfidf.shape:", tfidf.shape) # tfidf.shape: (301, 1365) ==> (# of sentences[documents], unique tokens[features])

             vals = cosine_similarity(tfidf[-1], tfidf) # similarities between last sentence (user_input) and other sentences
         #     print("vals.shape = ", vals.shape) # shape = (1, 301)

             idx = vals.argsort()[0][-2] # index of the sentence that is most similar to user_input. [-2] because
                                         # second last of the ascending sorted scores (last one is 1, coz similarity with itself)
         #     print("vals.argsort() = ", vals.argsort())

             vals_flat = vals.flatten() # convert vals to one dimension
             vals_flat.sort() # inplace sorting of flat

             req_tfidf = vals_flat[-2] # second last of the ascending sorted vals (last is 1 - similarity with itself)

             if req_tfidf == 0:
                 robo_response = robo_response + "I am sorry! I do not understand you."
                 return robo_response
             else:
                 robo_response = robo_response + sent_tokens[idx]
                 return robo_response
```

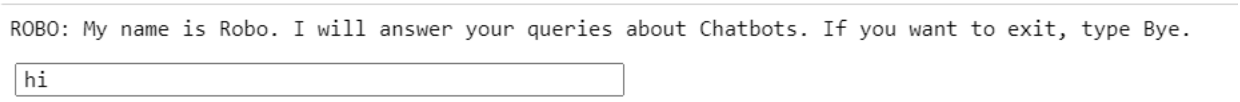
Lastly robo is ready:



Finally, we will feed the lines that we want our bot to say while starting and ending a conversation depending upon user’s input.

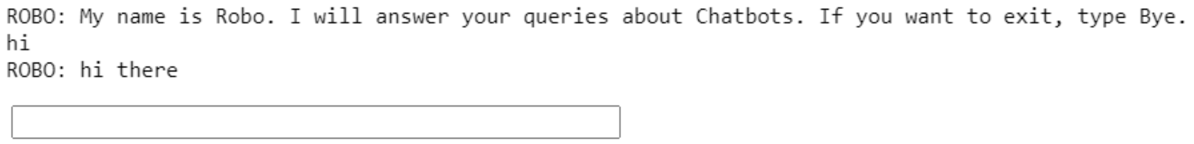
```
n [26]: flag = True

        print("ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye.")

        while flag:
            user_response = input()
            user_response = user_response.lower()

            if user_response.lower() != 'bye':
                if (user_response=='thanks' or user_response=='thank you'):
                    flag = False
                    print("ROBO: You are welcome! Bye for now!")
                else:
                    if greeting(user_response)!= None:
                        print("ROBO: " + greeting(user_response))
                    else:
                        print("ROBO: ", end="")
                        print(response(user_response))
                        sent_tokens.remove(user_response)
            else:
                flag = False
                print("ROBO: Bye! take care ..")
```

Output:



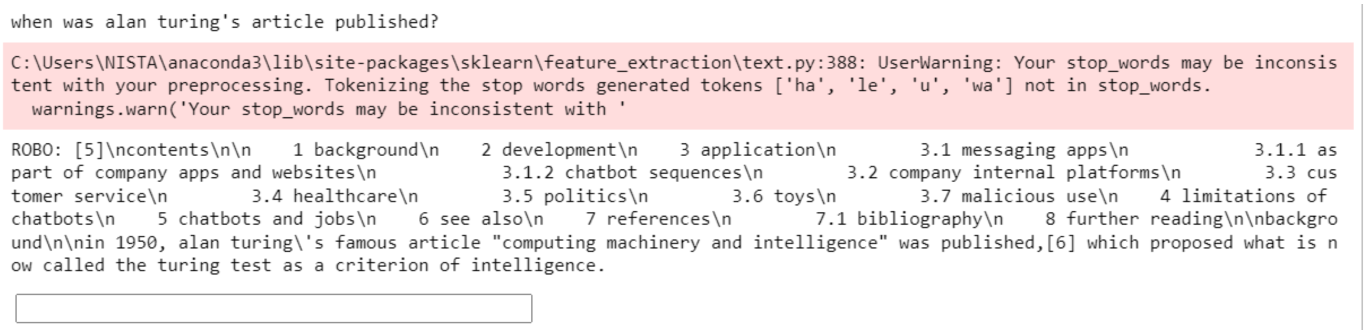
```
ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye.
hi
```

I type hi:



```
ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye.
hi
ROBO: hi there
```

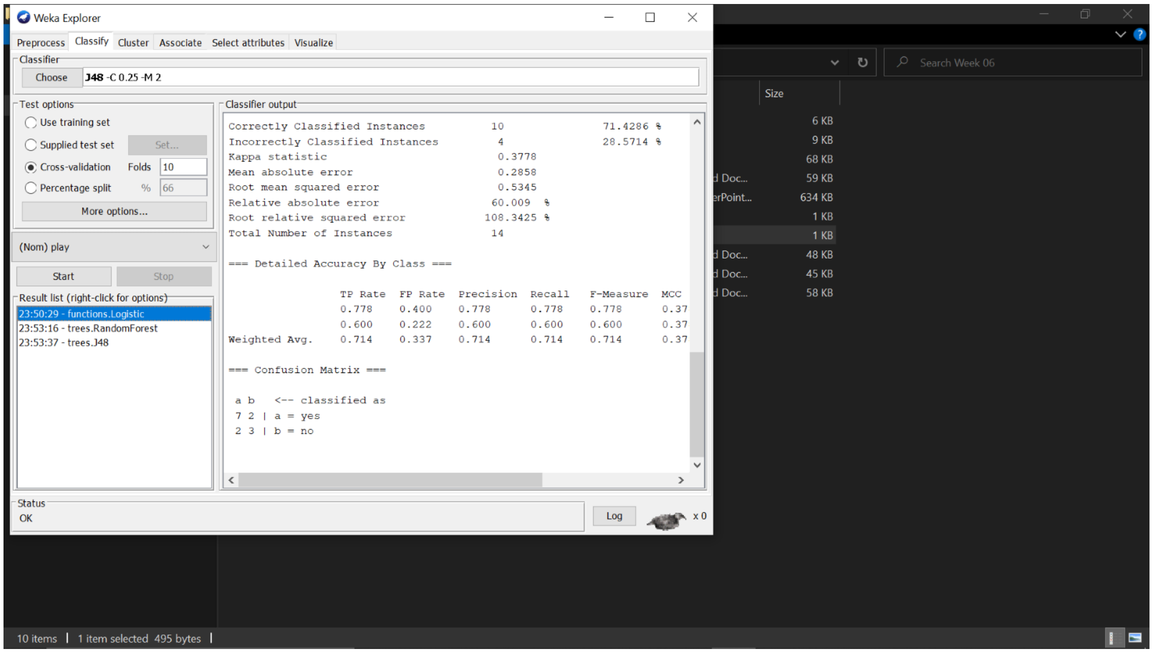
This is how robo response:



```
when was alan turing's article published?
C:\Users\NISTA\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words may be inconsis
tent with your preprocessing. Tokenizing the stop words generated tokens ['ha', 'le', 'u', 'wa'] not in stop_words.
  warnings.warn('Your stop_words may be inconsistent with '
ROBO: [5]\ncontents\n\n    1 background\n    2 development\n    3 application\n        3.1 messaging apps\n            3.1.1 as
part of company apps and websites\n            3.1.2 chatbot sequences\n        3.2 company internal platforms\n        3.3 cus
tomer service\n        3.4 healthcare\n        3.5 politics\n        3.6 toys\n        3.7 malicious use\n    4 limitations of
chatbots\n    5 chatbots and jobs\n    6 see also\n    7 references\n        7.1 bibliography\n    8 further reading\n\nbackgro
und\n\nin 1950, alan turing\'s famous article "computing machinery and intelligence" was published,[6] which proposed what is n
ow called the turing test as a criterion of intelligence.
```

Week 6

In sixth week, I learnt how to use WEKA with the goal of predicting whether the weather would be suitable for football tomorrow using a sample of 14 weather samples based on the forecast. Temperature, humidity, wind, and when the game was played were all factors to consider. To compute probabilities, I utilized the Loestic method.



Classifier: J48 -C 0.25 -M 2

Test options:
- ○ Use training set
- ○ Supplied test set
- ● Cross-validation Folds 10
- ○ Percentage split % 66

(Nom) play

Result list:
- 23:50:29 - functions.Logistic
- 23:53:16 - trees.RandomForest
- 23:53:37 - trees.J48

```
Correctly Classified Instances          10               71.4286 %
Incorrectly Classified Instances         4               28.5714 %
Kappa statistic                          0.3778
Mean absolute error                      0.2858
Root mean squared error                  0.5345
Relative absolute error                 60.009  %
Root relative squared error            108.3425 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
                 0.778    0.400    0.778      0.778    0.778      0.37
                 0.600    0.222    0.600      0.600    0.600      0.37
Weighted Avg.    0.714    0.337    0.714      0.714    0.714      0.37

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 2 3 | b = no
```

Status: OK

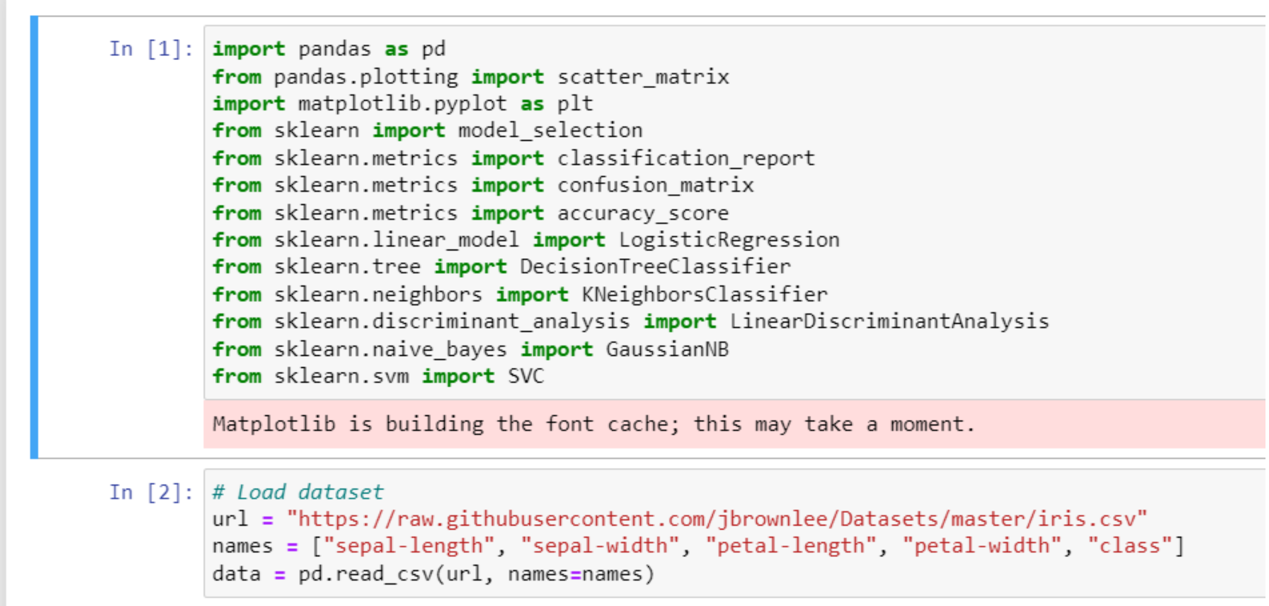
10 items | 1 item selected 495 bytes |

Week 7

In seventh week, we had a break.

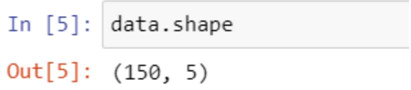
Week 8

I was given the responsibility of downloading and displaying the dataset from Kaggle.com.



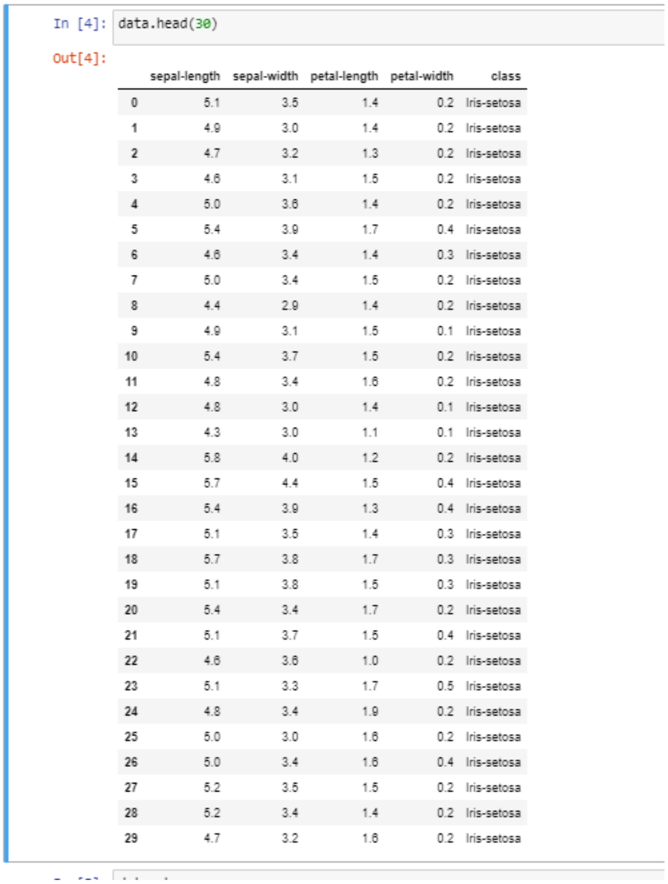
```
In [1]: import pandas as pd
        from pandas.plotting import scatter_matrix
        import matplotlib.pyplot as plt
        from sklearn import model_selection
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
        from sklearn.naive_bayes import GaussianNB
        from sklearn.svm import SVC

        Matplotlib is building the font cache; this may take a moment.

In [2]: # Load dataset
        url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
        names = ["sepal-length", "sepal-width", "petal-length", "petal-width", "class"]
        data = pd.read_csv(url, names=names)
```

Shape of the data set



```
In [5]: data.shape

Out[5]: (150, 5)
```

First **30 rows** of the dataset

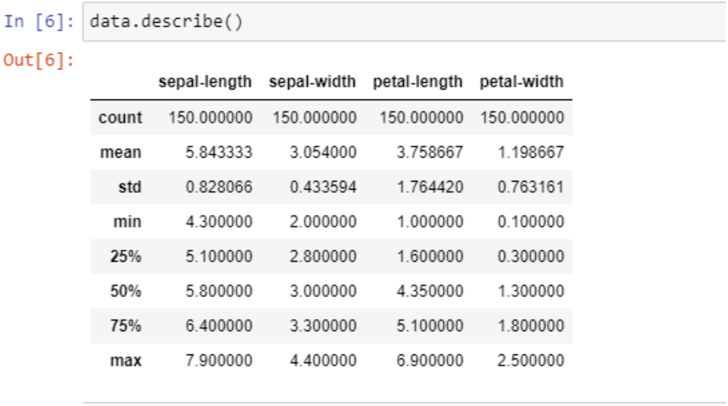


In [4]:
```
data.head(30)
```

Out[4]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa
20	5.4	3.4	1.7	0.2	Iris-setosa
21	5.1	3.7	1.5	0.4	Iris-setosa
22	4.6	3.6	1.0	0.2	Iris-setosa
23	5.1	3.3	1.7	0.5	Iris-setosa
24	4.8	3.4	1.9	0.2	Iris-setosa
25	5.0	3.0	1.6	0.2	Iris-setosa
26	5.0	3.4	1.6	0.4	Iris-setosa
27	5.2	3.5	1.5	0.2	Iris-setosa
28	5.2	3.4	1.4	0.2	Iris-setosa
29	4.7	3.2	1.6	0.2	Iris-setosa

Description of the dataset.

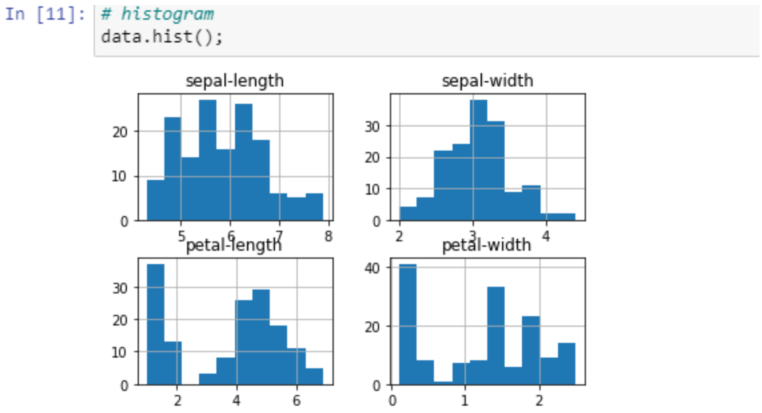


In [6]:
```
data.describe()
```

Out[6]:

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Creating a **histogram** for elements of the dataset.



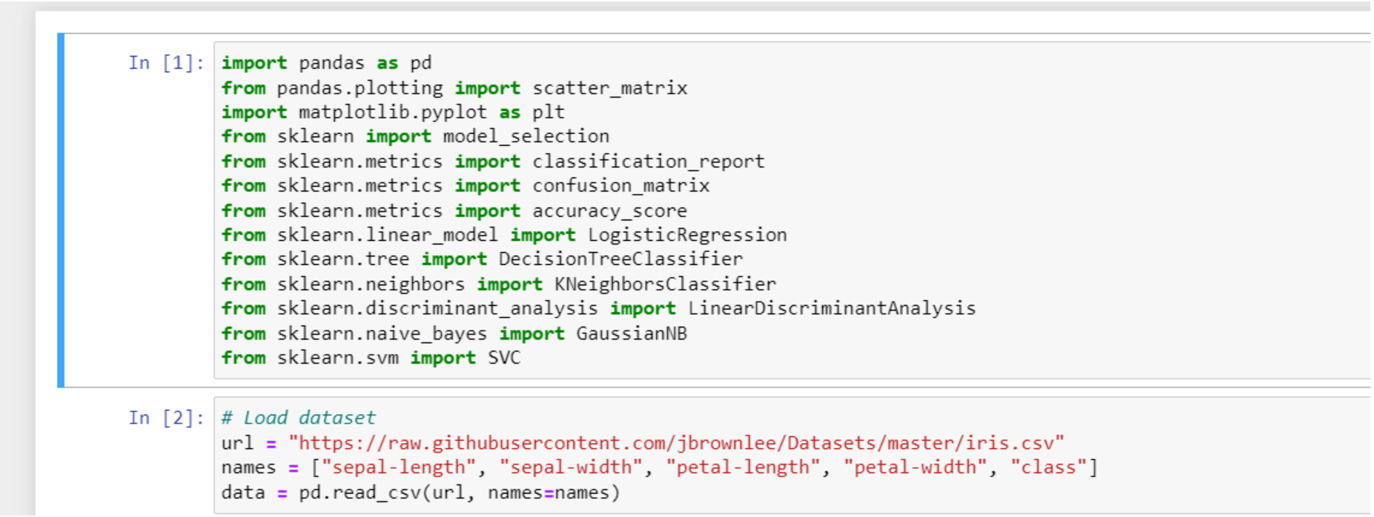
```
In [11]: # histogram
         data.hist();
```

I was asked to perform research on machine learning clustering methods and write a list of the greatest applications for each of the techniques that I discovered, which I subsequently submitted to my e-portfolio.

Week 9

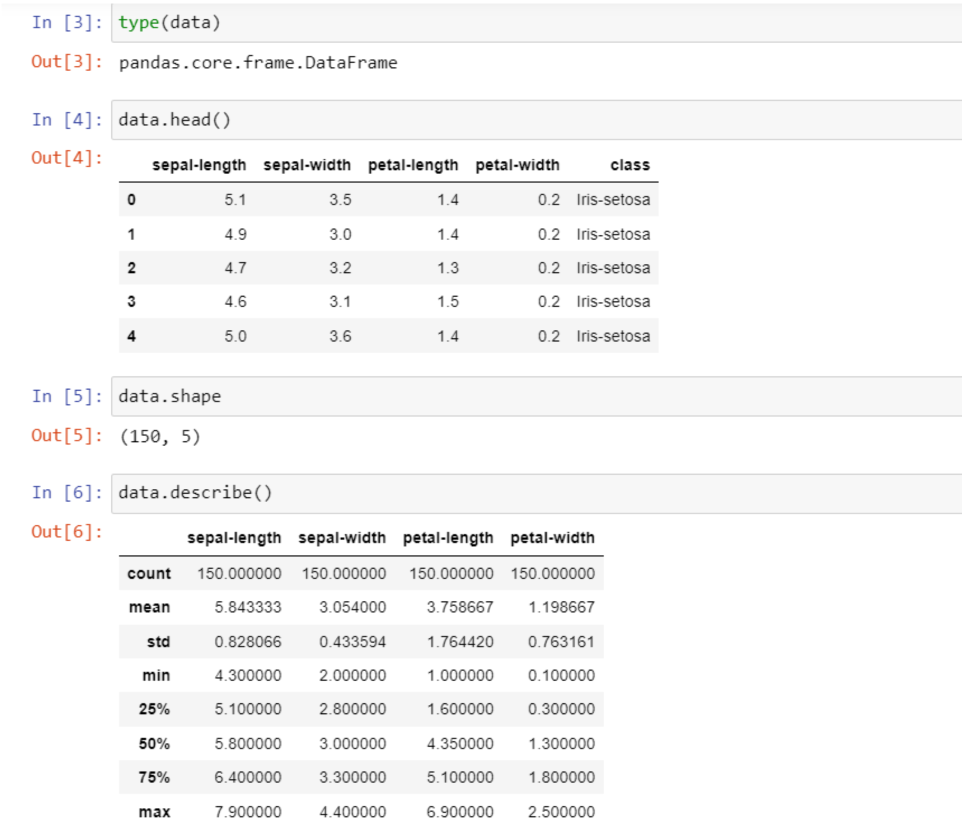
I was asked to create my own algorithm and data visualization, and then submit the code to my e-portfolio. The code is shown in the screenshots below.

Importing necessary libraries and datasets into Jupyter Notebook:



```
In [1]: import pandas as pd
        from pandas.plotting import scatter_matrix
        import matplotlib.pyplot as plt
        from sklearn import model_selection
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
        from sklearn.naive_bayes import GaussianNB
        from sklearn.svm import SVC

In [2]: # Load dataset
        url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
        names = ["sepal-length", "sepal-width", "petal-length", "petal-width", "class"]
        data = pd.read_csv(url, names=names)
```

Information of the data was looked and described for further analysis. Looking for shape.



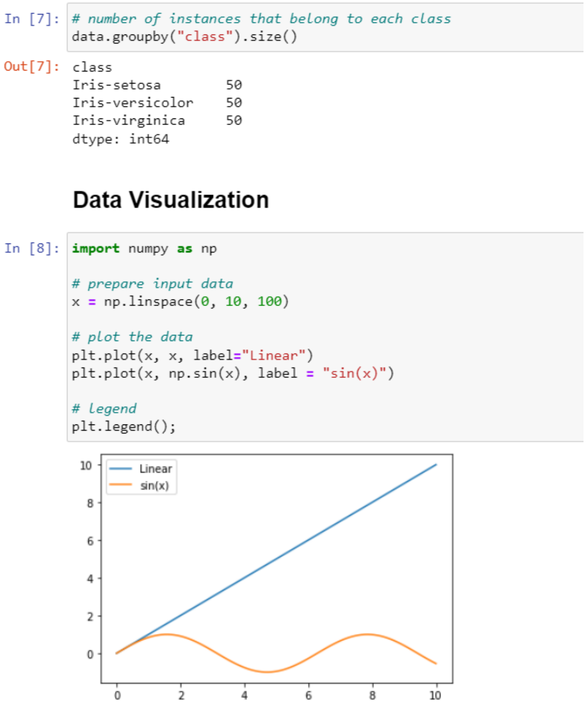
```
In [3]: type(data)

Out[3]: pandas.core.frame.DataFrame

In [4]: data.head()

Out[4]:
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [5]: data.shape

Out[5]: (150, 5)

In [6]: data.describe()

Out[6]:
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Creating the data visualization:



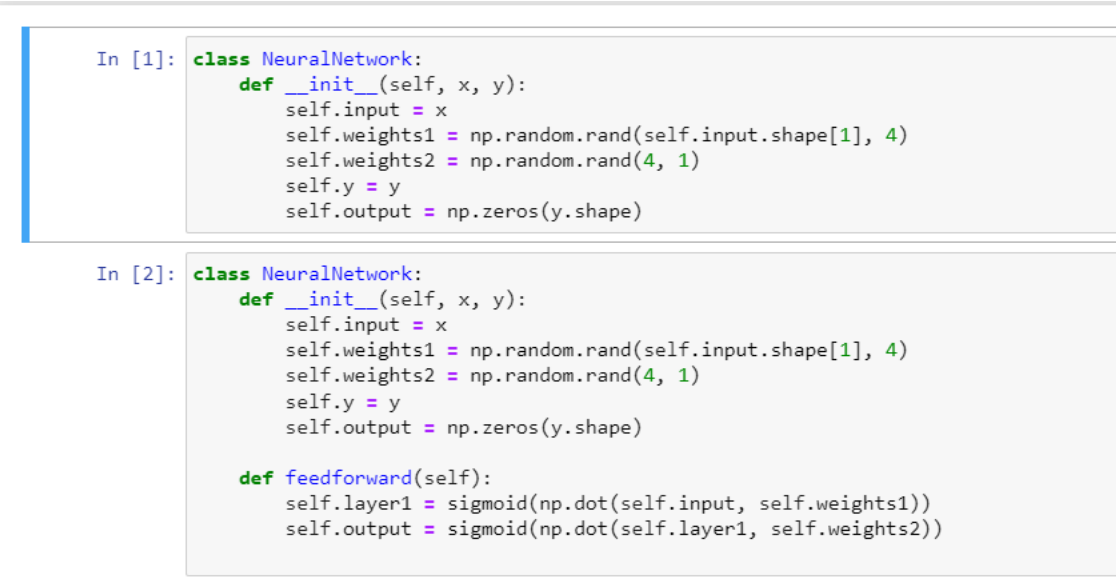
```
In [7]: # number of instances that belong to each class
        data.groupby("class").size()

Out[7]: class
        Iris-setosa        50
        Iris-versicolor    50
        Iris-virginica     50
        dtype: int64
```

Data Visualization

```
In [8]: import numpy as np

        # prepare input data
        x = np.linspace(0, 10, 100)

        # plot the data
        plt.plot(x, x, label="Linear")
        plt.plot(x, np.sin(x), label = "sin(x)")

        # Legend
        plt.legend();
```

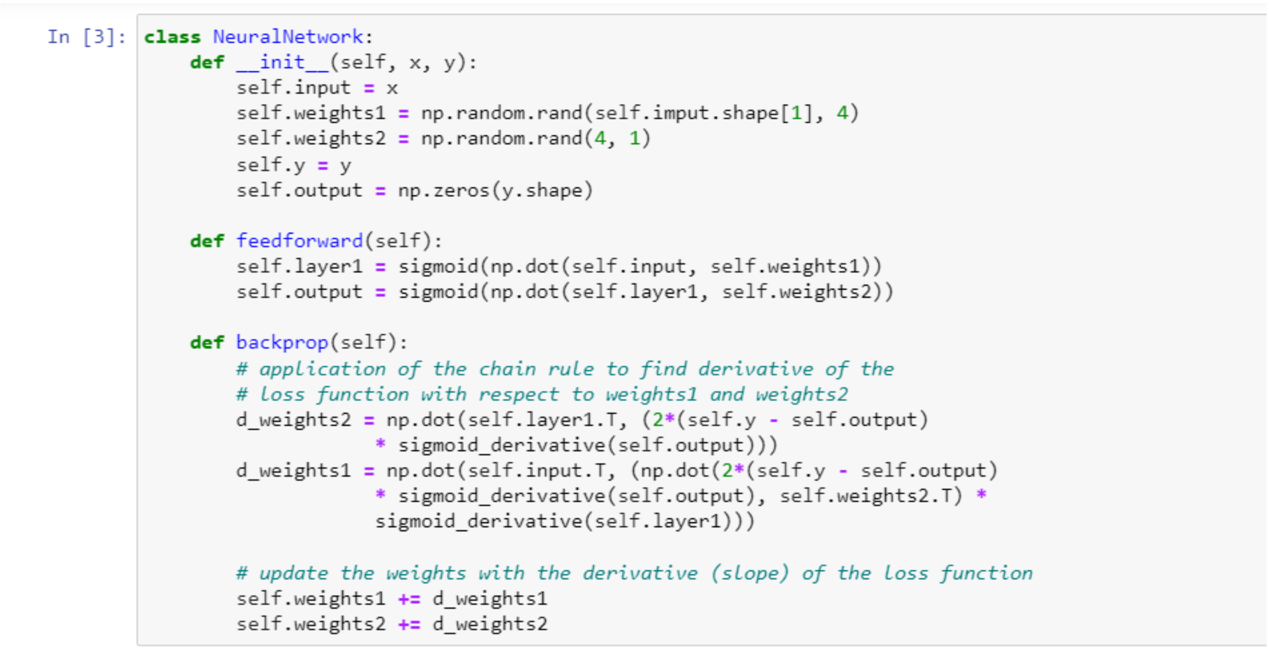
Week 10

We created our own neural network this week:



```
In [1]: class NeuralNetwork:
            def __init__(self, x, y):
                self.input = x
                self.weights1 = np.random.rand(self.input.shape[1], 4)
                self.weights2 = np.random.rand(4, 1)
                self.y = y
                self.output = np.zeros(y.shape)
```

```
In [2]: class NeuralNetwork:
            def __init__(self, x, y):
                self.input = x
                self.weights1 = np.random.rand(self.input.shape[1], 4)
                self.weights2 = np.random.rand(4, 1)
                self.y = y
                self.output = np.zeros(y.shape)

            def feedforward(self):
                self.layer1 = sigmoid(np.dot(self.input, self.weights1))
                self.output = sigmoid(np.dot(self.layer1, self.weights2))
```



```
In [3]: class NeuralNetwork:
            def __init__(self, x, y):
                self.input = x
                self.weights1 = np.random.rand(self.imput.shape[1], 4)
                self.weights2 = np.random.rand(4, 1)
                self.y = y
                self.output = np.zeros(y.shape)

            def feedforward(self):
                self.layer1 = sigmoid(np.dot(self.input, self.weights1))
                self.output = sigmoid(np.dot(self.layer1, self.weights2))

            def backprop(self):
                # application of the chain rule to find derivative of the
                # loss function with respect to weights1 and weights2
                d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output)
                            * sigmoid_derivative(self.output)))
                d_weights1 = np.dot(self.input.T, (np.dot(2*(self.y - self.output)
                            * sigmoid_derivative(self.output), self.weights2.T) *
                            sigmoid_derivative(self.layer1)))

                # update the weights with the derivative (slope) of the loss function
                self.weights1 += d_weights1
                self.weights2 += d_weights2
```

In [4]:

```
import numpy as np
```

In [5]:

```
# Each row is a training example, each column is a feature  [X1, X2, X3]
X = np.array(([0,0,1],[0,1,1],[1,0,1],[1,1,1]), dtype=float)
y = np.array(([0],[1],[1],[0]), dtype=float)
```

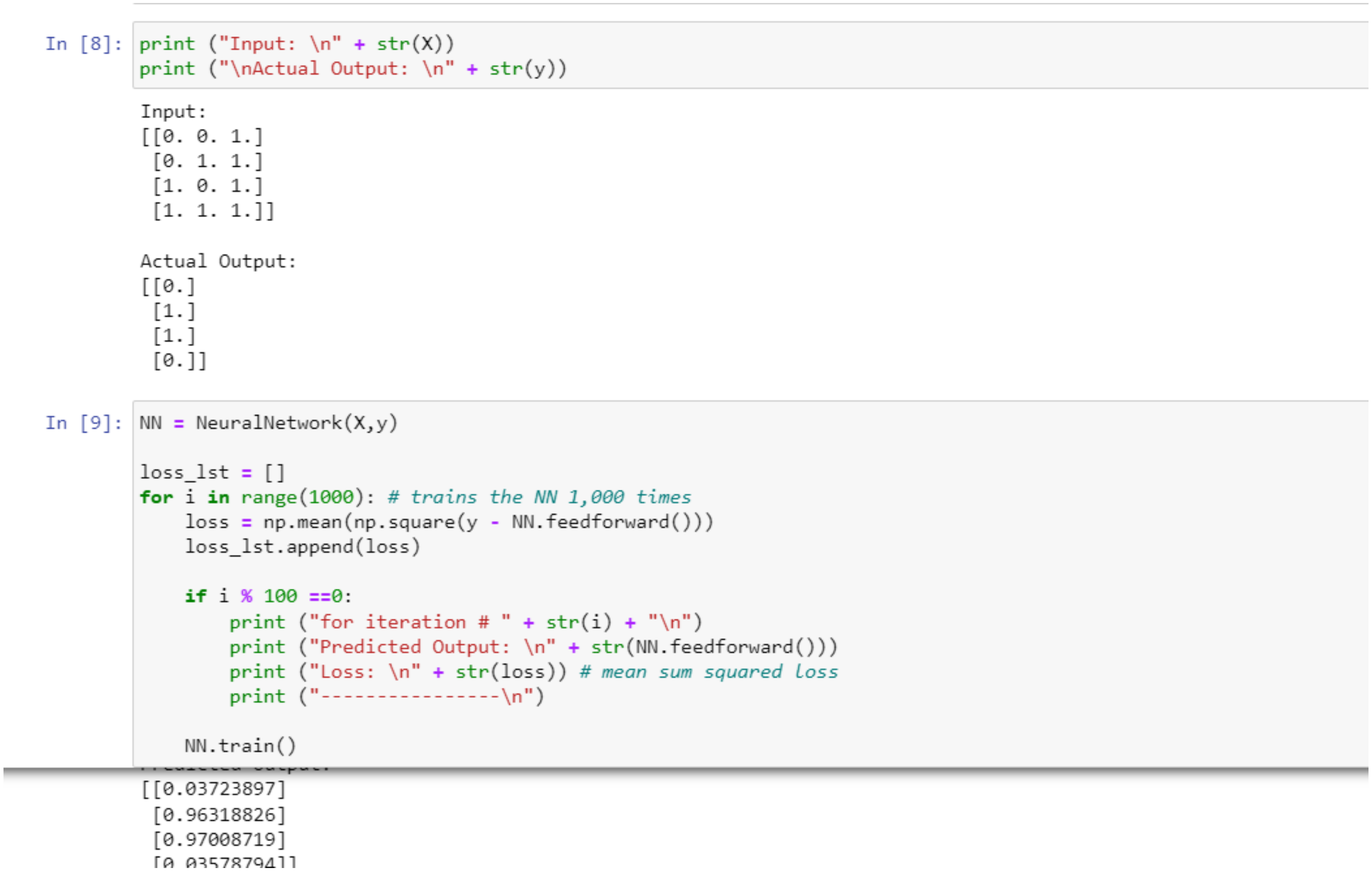
In [6]:

```
print(X)
print('Shape of X =', X.shape)
print(10*'-')
print(y)
print('Shape of y =', y.shape)
```

```
[[0. 0. 1.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 1.]]
Shape of X = (4, 3)
----------
[[0.]
 [1.]
 [1.]
 [0.]]
Shape of y = (4, 1)
```

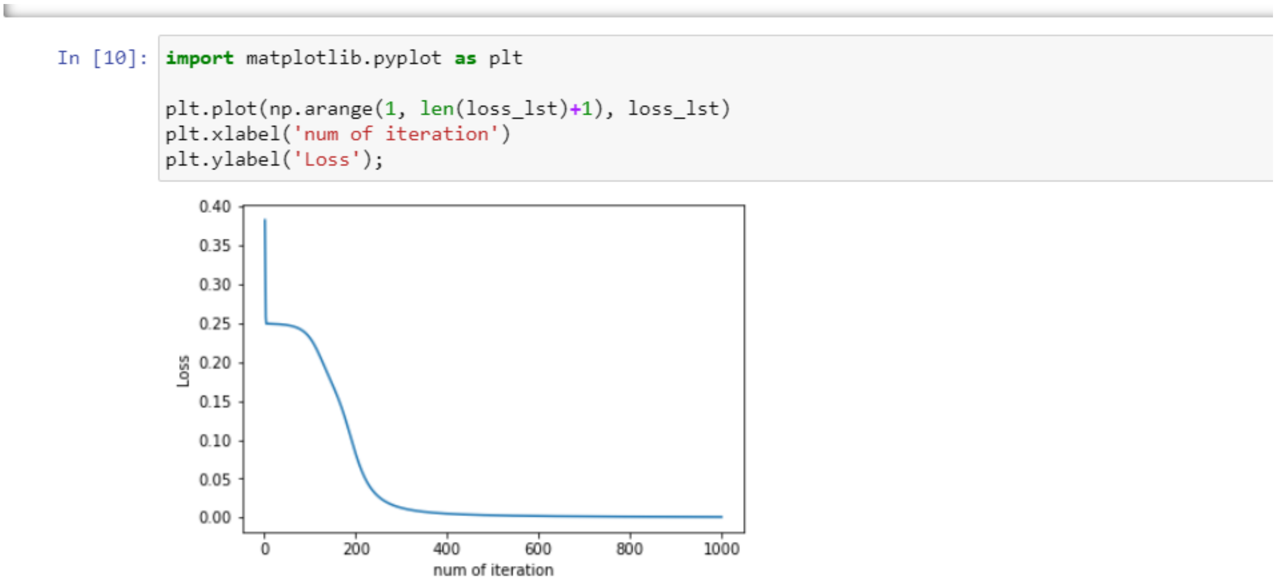
In [7]:

```
# Activation function
def sigmoid(t):
    return 1/(1+np.exp(-t))

# Derivative of sigmoid
def sigmoid_derivative(p):
    return p * (1 - p)

# Class definition
class NeuralNetwork:
    def __init__(self, x,y):
        self.input = x
        self.weights1= np.random.rand(self.input.shape[1],4) # considering we have 4 nodes in the hidden layer
        self.weights2 = np.random.rand(4,1)
        self.y = y
        self.output = np. zeros(y.shape)

    def feedforward(self):
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        return self.layer2

    def backprop(self):
        d_weights2 = np.dot(self.layer1.T, 2*(self.y -self.output)*sigmoid_derivative(self.output))
        d_weights1 = np.dot(self.input.T, np.dot(2*(self.y - self.output)*sigmoid_derivative(self.output),
                                                  self.weights2.T)*sigmoid_derivative(self.layer1))

        self.weights1 += d_weights1
        self.weights2 += d_weights2

    def train(self):
        self.output = self.feedforward()
        self.backprop()
```



```
In [8]: print ("Input: \n" + str(X))
        print ("\nActual Output: \n" + str(y))

        Input:
        [[0. 0. 1.]
         [0. 1. 1.]
         [1. 0. 1.]
         [1. 1. 1.]]

        Actual Output:
        [[0.]
         [1.]
         [1.]
         [0.]]
```

```
In [9]: NN = NeuralNetwork(X,y)

        loss_lst = []
        for i in range(1000): # trains the NN 1,000 times
            loss = np.mean(np.square(y - NN.feedforward()))
            loss_lst.append(loss)

            if i % 100 ==0:
                print ("for iteration # " + str(i) + "\n")
                print ("Predicted Output: \n" + str(NN.feedforward()))
                print ("Loss: \n" + str(loss)) # mean sum squared loss
                print ("----------------\n")

            NN.train()
        Predicted Output:
        [[0.03723897]
         [0.96318826]
         [0.97008719]
         [0.03578794]]
```

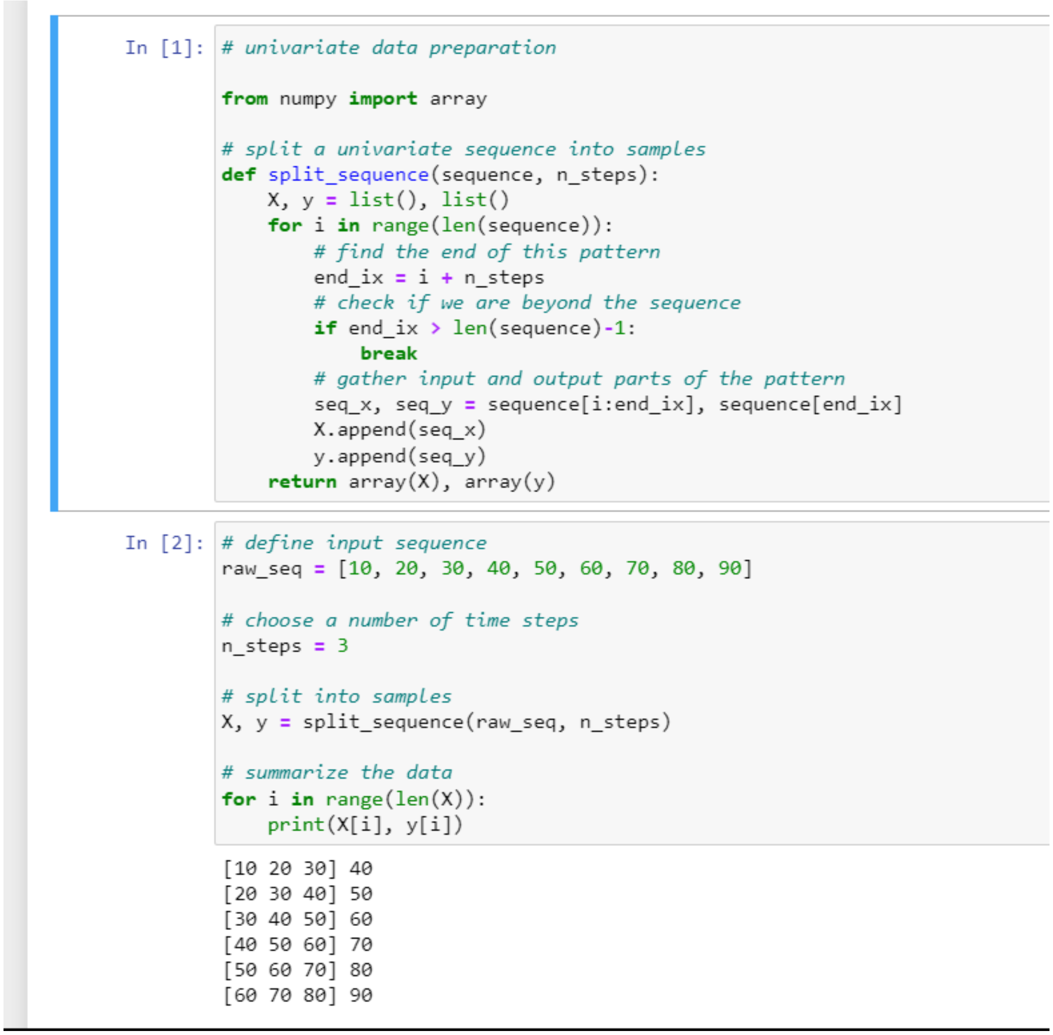


```
In [10]: import matplotlib.pyplot as plt

         plt.plot(np.arange(1, len(loss_lst)+1), loss_lst)
         plt.xlabel('num of iteration')
         plt.ylabel('Loss');
```

In this way, I completed building my own neural network and completed the task of this week.

Week 11

In this week, we are asked to complete the code of Convolutional Neural Network.



In [1]:

```
# univariate data preparation

from numpy import array

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

In [2]:

```
# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# choose a number of time steps
n_steps = 3

# split into samples
X, y = split_sequence(raw_seq, n_steps)

# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

```
[10 20 30] 40
[20 30 40] 50
[30 40 50] 60
[40 50 60] 70
[50 60 70] 80
[60 70 80] 90
```

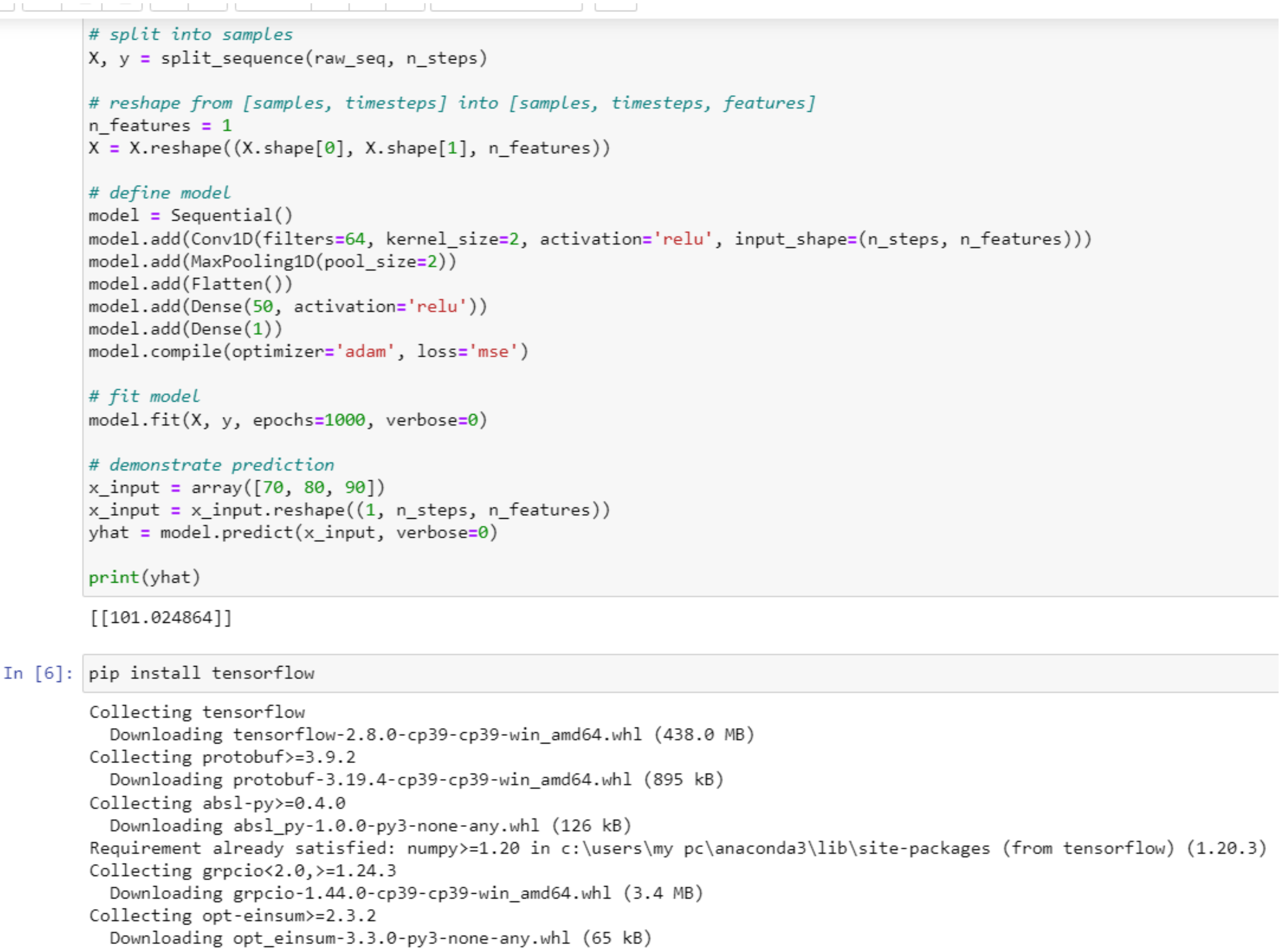
In [4]:

```
!pip install keras
```

```
Collecting keras
  Downloading keras-2.8.0-py2.py3-none-any.whl (1.4 MB)
Installing collected packages: keras
Successfully installed keras-2.8.0
```

In [7]:

```
# univariate cnn example
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# choose a number of time steps
n_steps = 3
```



```
# split into samples
X, y = split_sequence(raw_seq, n_steps)

# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))

# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# fit model
model.fit(X, y, epochs=1000, verbose=0)

# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)

print(yhat)
```

```
[[101.024864]]
```

In [6]:

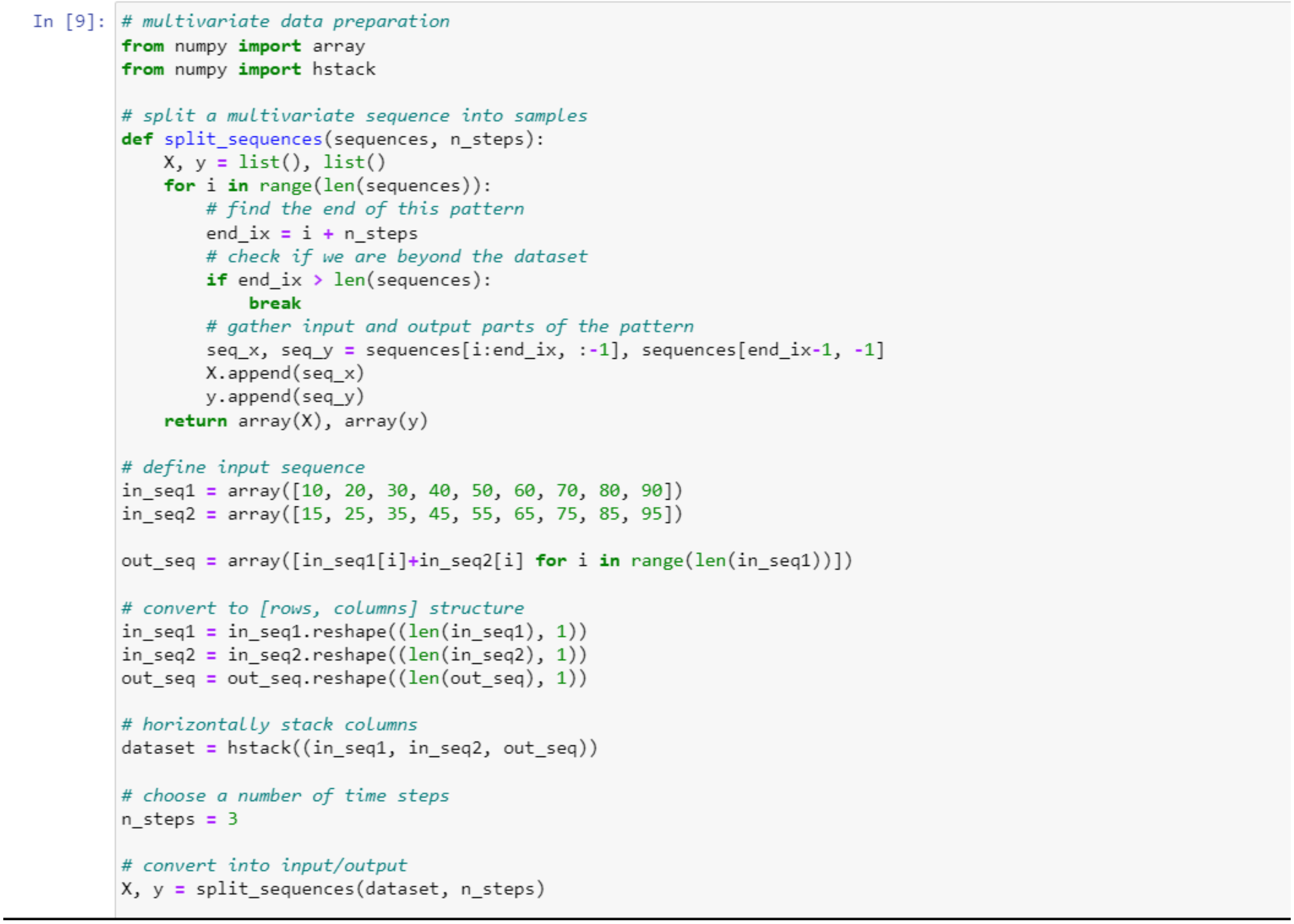
```
pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.8.0-cp39-cp39-win_amd64.whl (438.0 MB)
Collecting protobuf>=3.9.2
  Downloading protobuf-3.19.4-cp39-cp39-win_amd64.whl (895 kB)
Collecting absl-py>=0.4.0
  Downloading absl_py-1.0.0-py3-none-any.whl (126 kB)
Requirement already satisfied: numpy>=1.20 in c:\users\my pc\anaconda3\lib\site-packages (from tensorflow) (1.20.3)
Collecting grpcio<2.0,>=1.24.3
  Downloading grpcio-1.44.0-cp39-cp39-win_amd64.whl (3.4 MB)
Collecting opt-einsum>=2.3.2
  Downloading opt_einsum-3.3.0-py3-none-any.whl (65 kB)
```

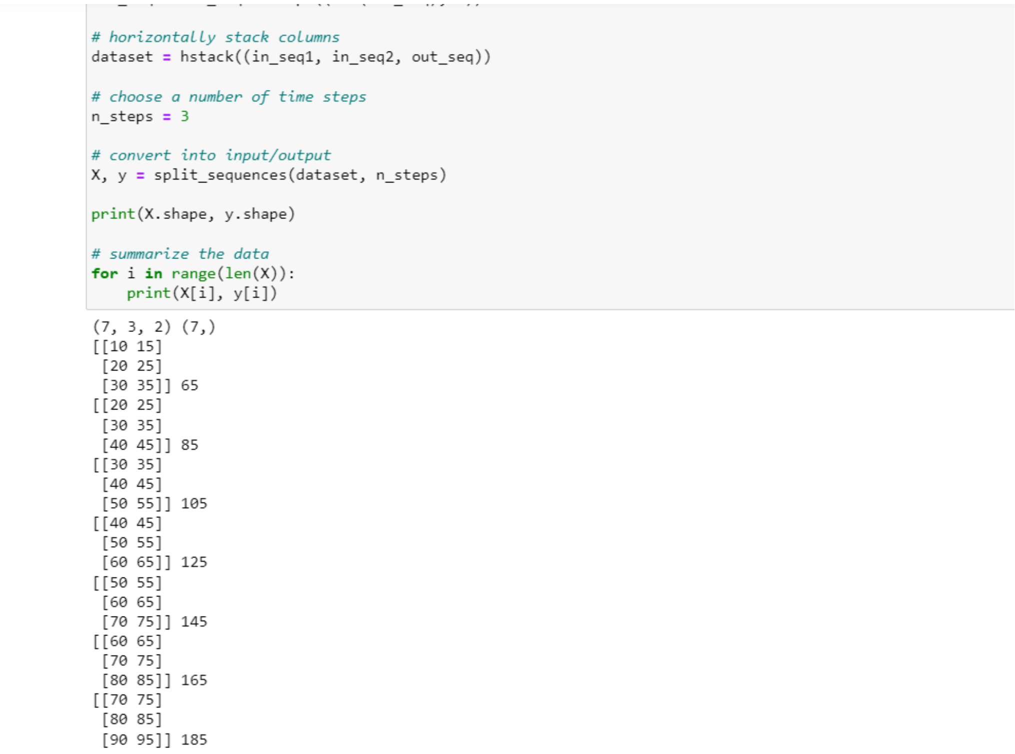
In [8]:

```
# multivariate data preparation
from numpy import array
from numpy import hstack

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])

out_seq = array([in_seq1[i] + in_seq2[i] for i in range(len(in_seq1))])

# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))

# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
print(dataset)  # Running the example prints the dataset
                # with one row per time step and one
                # column for each of the two input and
                # one output parallel time series.
```

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
```



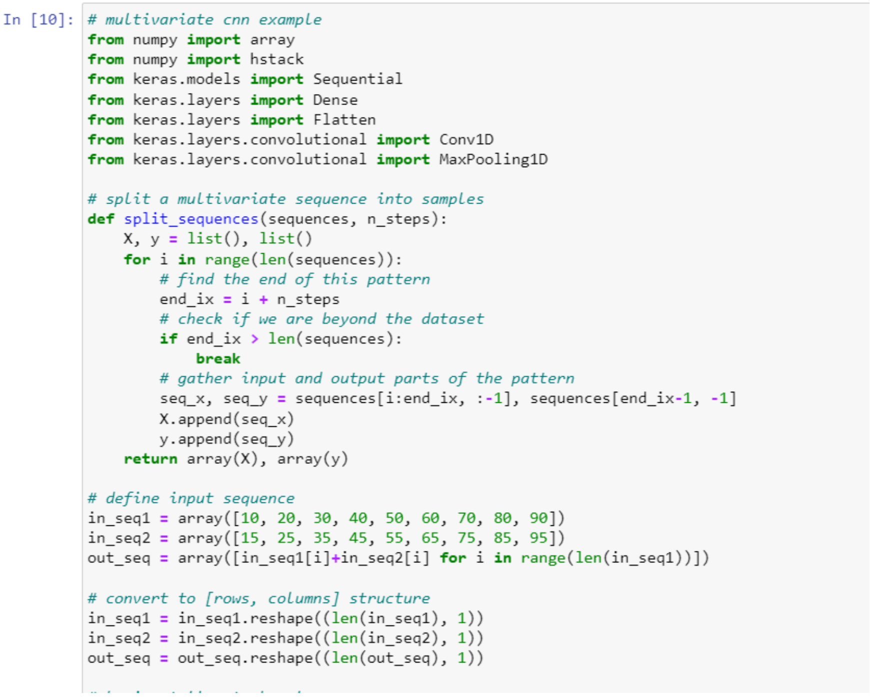
```
In [9]: # multivariate data preparation
        from numpy import array
        from numpy import hstack

        # split a multivariate sequence into samples
        def split_sequences(sequences, n_steps):
            X, y = list(), list()
            for i in range(len(sequences)):
                # find the end of this pattern
                end_ix = i + n_steps
                # check if we are beyond the dataset
                if end_ix > len(sequences):
                    break
                # gather input and output parts of the pattern
                seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
                X.append(seq_x)
                y.append(seq_y)
            return array(X), array(y)

        # define input sequence
        in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
        in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])

        out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])

        # convert to [rows, columns] structure
        in_seq1 = in_seq1.reshape((len(in_seq1), 1))
        in_seq2 = in_seq2.reshape((len(in_seq2), 1))
        out_seq = out_seq.reshape((len(out_seq), 1))

        # horizontally stack columns
        dataset = hstack((in_seq1, in_seq2, out_seq))

        # choose a number of time steps
        n_steps = 3

        # convert into input/output
        X, y = split_sequences(dataset, n_steps)
```



```
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))

# choose a number of time steps
n_steps = 3

# convert into input/output
X, y = split_sequences(dataset, n_steps)
```

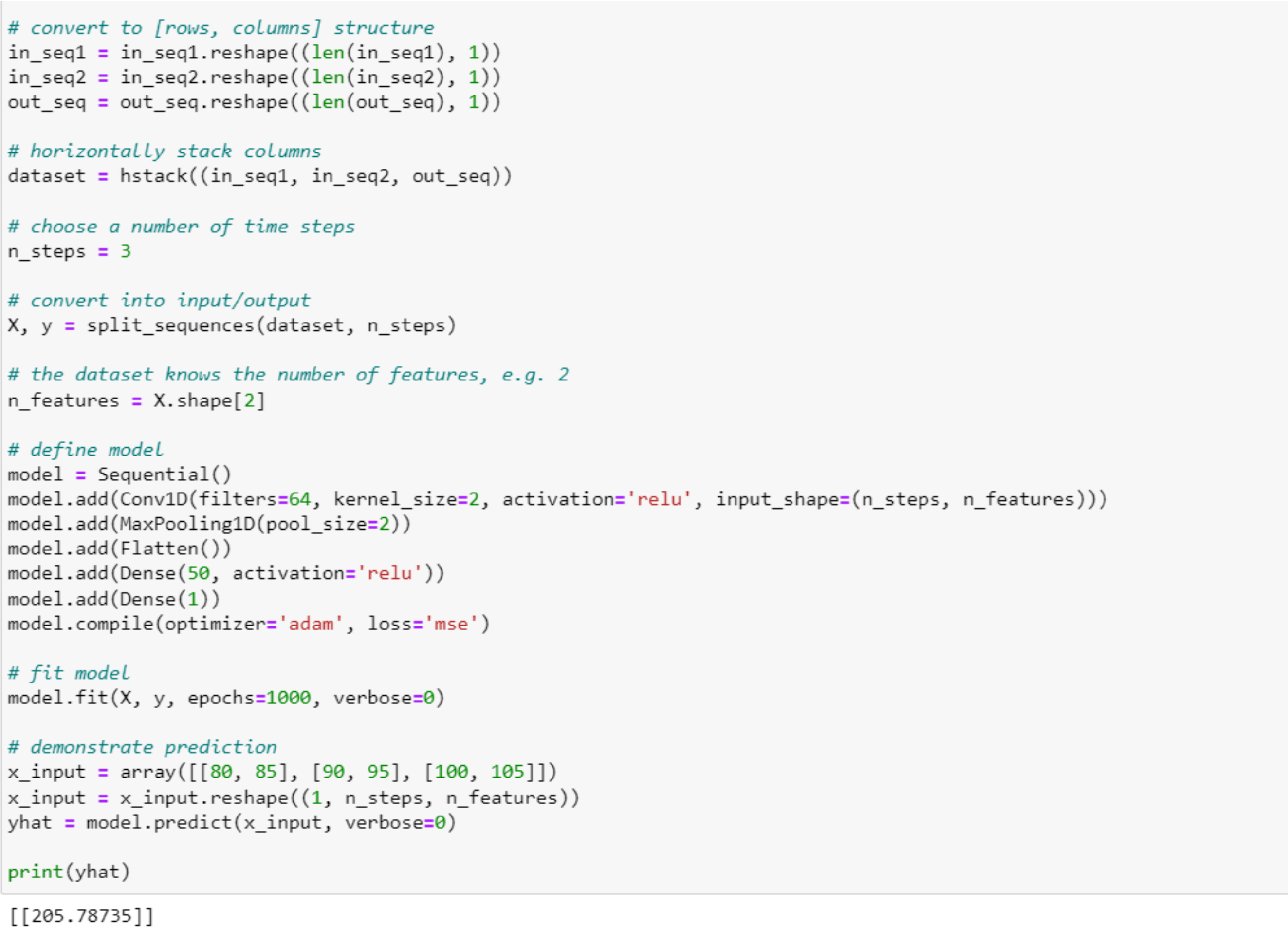
```
print(X.shape, y.shape)

# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

```
(7, 3, 2) (7,)
[[10 15]
 [20 25]
 [30 35]] 65
[[20 25]
 [30 35]
 [40 45]] 85
[[30 35]
 [40 45]
 [50 55]] 105
[[40 45]
 [50 55]
 [60 65]] 125
[[50 55]
 [60 65]
 [70 75]] 145
[[60 65]
 [70 75]
 [80 85]] 165
[[70 75]
 [80 85]
 [90 95]] 185
```



In [10]:

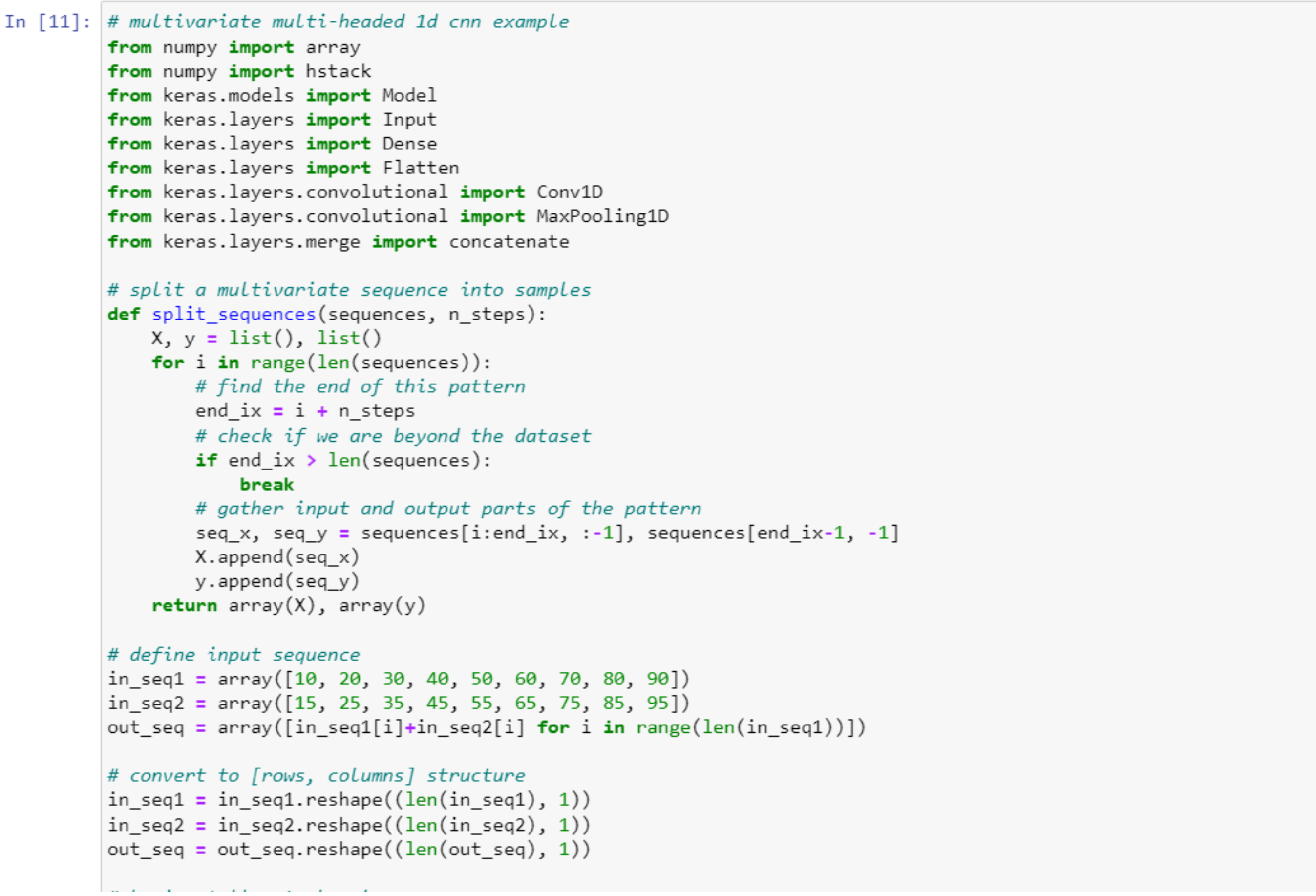
```
# multivariate cnn example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])

# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
```



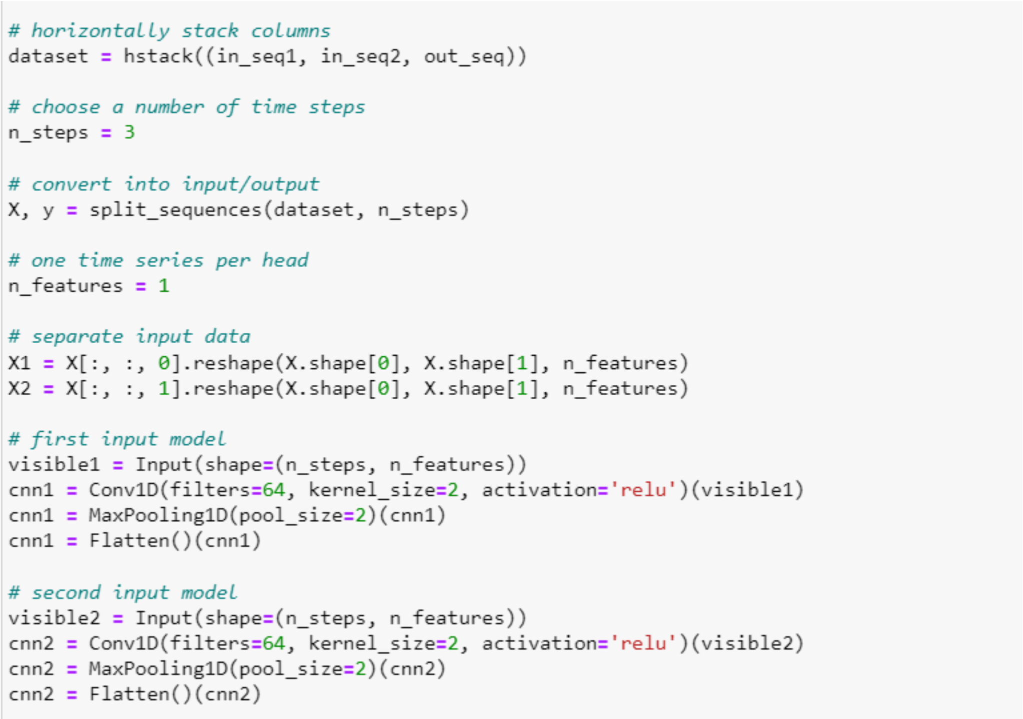
```
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
```

```
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))

# choose a number of time steps
n_steps = 3

# convert into input/output
X, y = split_sequences(dataset, n_steps)

# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]

# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# fit model
model.fit(X, y, epochs=1000, verbose=0)

# demonstrate prediction
x_input = array([[80, 85], [90, 95], [100, 105]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)

print(yhat)
```

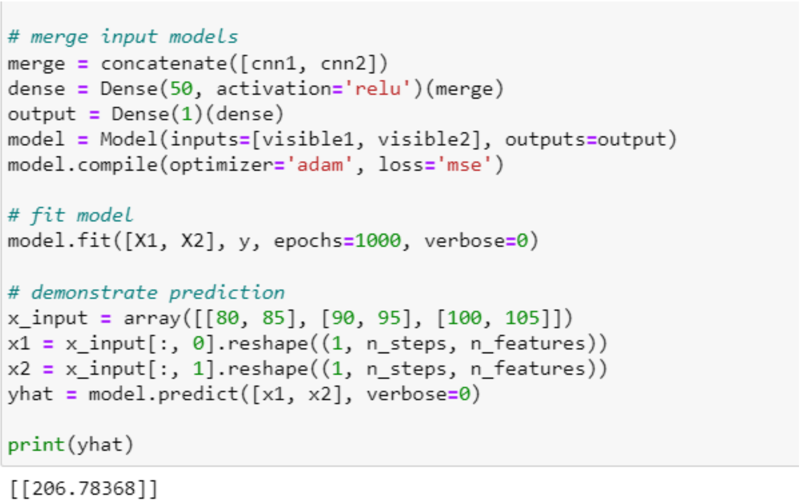
```
[[205.78735]]
```



```
In [11]: # multivariate multi-headed 1d cnn example
         from numpy import array
         from numpy import hstack
         from keras.models import Model
         from keras.layers import Input
         from keras.layers import Dense
         from keras.layers import Flatten
         from keras.layers.convolutional import Conv1D
         from keras.layers.convolutional import MaxPooling1D
         from keras.layers.merge import concatenate

         # split a multivariate sequence into samples
         def split_sequences(sequences, n_steps):
             X, y = list(), list()
             for i in range(len(sequences)):
                 # find the end of this pattern
                 end_ix = i + n_steps
                 # check if we are beyond the dataset
                 if end_ix > len(sequences):
                     break
                 # gather input and output parts of the pattern
                 seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
                 X.append(seq_x)
                 y.append(seq_y)
             return array(X), array(y)

         # define input sequence
         in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
         in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
         out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])

         # convert to [rows, columns] structure
         in_seq1 = in_seq1.reshape((len(in_seq1), 1))
         in_seq2 = in_seq2.reshape((len(in_seq2), 1))
         out_seq = out_seq.reshape((len(out_seq), 1))
```



```
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))

# choose a number of time steps
n_steps = 3

# convert into input/output
X, y = split_sequences(dataset, n_steps)

# one time series per head
n_features = 1

# separate input data
X1 = X[:, :, 0].reshape(X.shape[0], X.shape[1], n_features)
X2 = X[:, :, 1].reshape(X.shape[0], X.shape[1], n_features)

# first input model
visible1 = Input(shape=(n_steps, n_features))
cnn1 = Conv1D(filters=64, kernel_size=2, activation='relu')(visible1)
cnn1 = MaxPooling1D(pool_size=2)(cnn1)
cnn1 = Flatten()(cnn1)

# second input model
visible2 = Input(shape=(n_steps, n_features))
cnn2 = Conv1D(filters=64, kernel_size=2, activation='relu')(visible2)
cnn2 = MaxPooling1D(pool_size=2)(cnn2)
cnn2 = Flatten()(cnn2)
```



```
# merge input models
merge = concatenate([cnn1, cnn2])
dense = Dense(50, activation='relu')(merge)
output = Dense(1)(dense)
model = Model(inputs=[visible1, visible2], outputs=output)
model.compile(optimizer='adam', loss='mse')

# fit model
model.fit([X1, X2], y, epochs=1000, verbose=0)

# demonstrate prediction
x_input = array([[80, 85], [90, 95], [100, 105]])
x1 = x_input[:, 0].reshape((1, n_steps, n_features))
x2 = x_input[:, 1].reshape((1, n_steps, n_features))
yhat = model.predict([x1, x2], verbose=0)

print(yhat)
```

```
[[206.78368]]
```

This is how I finish the lab code.

In week 11, I watched a video of an AI robo doing surgery in a Turing test. Here, I discovered that machines can emulate human behavior as well.

Below is a link to my e-portfolio.

https://canvas.sunderland.ac.uk/eportfolios/7730?verifier=IRv5vhOHhyZX8ykWMfJ1ctLnaY62WawtFOExxNAe