Homework 1

Due: September 17th, 5pm (late submission until September 20nd, 5pm -- no submission possible afterwards)

Note that you'll need to submit a pdf version of your notebook and the ipynb file as well on Gradescope!

25 points total (Problem 1: 8 points, Problem 2: 6 points, Coding Assignment: 11 points)

Name: [Yawen Tan]

Link to the github repo: [https://github.com/IsabellaTan/Brown-DATA2060-HW1.git]

Written Assignment

Introduction: Solidifying Background

The purpose of this portion is to fortify your background in probability and statistics, linear algebra, and algorithmic analysis. The topics explored here will be used many times throughout this course.

You may be able to find answers to these problems by searching the problem text. Please search instead for the concepts being applied; the goal is not to solve these specific problems, but to be comfortable with the principles that will be applied later in the course.

Problem 1: Bayes' Rule (8 points)

Bayes' Rule, or Bayes' Theorem is an oft-used identity coming from probability theory. If we have two events of interest, *A* and *B*, we might want to ask what the probability of *B* is, given that we know *A* happened.

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Note that this is the same as

$$P(B|A) = \frac{P(A \cap B)}{P(A)}.$$

Later in this course, the parts of this formula may be relabeled:

$$\text{Posterior} = \frac{\text{Likelihood} * \text{Prior}}{\text{Evidence}}$$

This rule will be explicitly used in Bayesian algorithms, but it is also a principle that will *implicitly* underlie almost all of our machine learning algorithms. This problem consists of four parts, each worth 2 points (1 point if the answer is correct and an additional point for showing correct work). As a hint, none of the four parts have the same answer.

For the purposes of this question, assume that desserts have equal probability of being a cake or ice cream and uniform probability of being any of the following 7 flavors: chocolate, vanilla, strawberry, coconut, cookies & cream, fudge, and raspberry.

1. Suppose Steve has two desserts. What is the probability that both desserts are cakes?
2. Suppose Paul has two desserts, the first of which is ice cream. What is the probability that both desserts are ice cream?
3. Suppose Chace has two desserts and at least one is a cake. What is the probability that both desserts are cakes?
4. Suppose Andrew has two desserts and at least one is chocolate flavored ice cream. What is the probability that both desserts are ice cream?

Solution:

1.

Since probabilities for having desserts and having desserts are equal, so that:

$$P(cake) = \frac{1}{2}$$

Since having both cake are independent, so that:

$$P(cake \cap cake) = P(cake) * P(cake) = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$$

Thus, the probability that both desserts are cakes is 1/4 2.

Since the first dessert is ice cream, then this question is about conditional probability. And we want to know P(ice cream | ice cream).

Based on the formula above, we know:

$$P(ice\ cream|ice\ cream) = \frac{P(ice\ cream \cap ice\ cream)}{P(ice\ cream)}.$$

Since from question 1, we know

$$P(cake \cap cake) = 1/4$$

And since desserts have equal probability of being a cake or ice cream, then

$$P(ice\ cream) = \frac{1}{2}$$

$$P(ice\ cream \cap ice\ cream) = \frac{1}{4}$$

So we plug them into equation and get:

$$P(ice\ cream|ice\ cream) = \frac{P(ice\ cream \cap ice\ cream)}{P(ice\ cream)} = \frac{1/4}{1/2} = \frac{1}{2}$$

Thus, when the first of which is ice cream, the probability that both desserts are ice cream is 1/2.

3.

Since at least one dessert is cake, then this question is about conditional probability. And we want to know P(both two cake | at least one cake)

Based on the formula above, we know:

$$P(both\ two\ cake|at\ least\ one\ cake)$$
$$= \frac{P(at\ least\ one\ cake|both\ two\ cake)P(both\ two\ cake)}{P(at\ least\ one\ cake)}$$

Since P(at least one cake|both two cake) = 1, and based on question 1, P(both two cake) = 1/4, then we need to find P(at least one cake).

Since P(at least one cake) = 1 - P(both two ice cream), then P(at least one cake) = 1 - 1/4 = 3/4.

So we plug P(at least one cake) and P(both two cake) and P(at least one cake|both two cake) into equation and get:

$$P(both\ two\ cake|at\ least\ one\ cake)$$
$$= \frac{P(at\ least\ one\ cake|both\ two\ cake)P(both\ two\ cake)}{P(at\ least\ one\ cake)}$$
$$= \frac{1 * 1/4}{3/4} = \frac{1}{3}$$

Thus, when at least one is a cake, the probability that both desserts are cakes is 1/3.

4.

Since at least one dessert is chocolate ice cream, then this question is about conditional probability. And we want to know P(both two ice cream | at least one chocolate ice cream)

Based on the formula above, we know:

$$P(both\ two\ ice\ cream|at\ least\ one\ chocolate\ ice\ cream)$$
$$= \frac{P(both\ two\ ice\ cream \cap at\ least\ one\ chocolate\ ice\ cream)}{P(at\ least\ one\ chocolate\ ice\ cream)}$$

Firstly, we calculate P(at least one chocolate ice cream).

Since the type of the desserts and the flavors of the desserts are independent and there are 7 flavors desserts, so that:

$$P(chocolate\ ice\ cream) = P(ice\ cream) * P(chocolate\ flavor) = 1/2 * 1/7 =$$

So P(not chocolate ice cream) = 1 - P(chocolate ice cream) = 1 - 1/14 = 13/14, then P(two desserts are not chocolate ice cream) = P(not chocolate ice cream) * P(not chocolate ice cream) = 13/14 * 13/14 = 169/196.

Then P(at least one chocolate ice cream) = 1- P(two desserts are not chocolate ice cream) = 1 - 169/196 = 27/196.

Secondly, we calculate P(both two ice cream and at least one chocolate ice cream).

Based on the Inclusion-Exclusion Principle, P(both two ice cream and at least one chocolate ice cream) = P(first ice cream is chocolate and second ice cream is random) + P(first ice cream is random and second ice cream is chocolate) - P(both ice cream are chocolate).

Then since P (chocolate ice cream) = 1/14, so P(both two ice cream and at least one chocolate ice cream) = 1/14 * 1/2 + 1/2 * 1/14 - 1/14*1/14 = 13/196

So we plug P(both two ice cream and at least one chocolate ice cream) and P(at least one chocolate ice cream) into equation and get:

$$P(both\ two\ ice\ cream|at\ least\ one\ chocolate\ ice\ cream)$$
$$= \frac{P(both\ two\ ice\ cream \cap at\ least\ one\ chocolate\ ice\ cream)}{P(at\ least\ one\ chocolate\ ice\ cream)}$$
$$= \frac{13/196}{27/196} = \frac{13}{27}$$

Thus, when at least one is chocolate flavored ice cream, the probability that both desserts are ice cream is 13/27.

Problem 2: Linear algebra review (6 points)

1. Let A = $\begin{bmatrix} 4 & -5 & 0 & 1 \\ 2 & 8 & 0 & 0 \\ -1 & 5 & 3 & 2 \end{bmatrix}$ and B = $\begin{bmatrix} 3 & -1 & 2 \\ 4 & 2 & 0 \end{bmatrix}$. Calculate the products AB, A^TB, BA and B^TA! Solve this problem in markdown without code.

2. Describe what a determinant is, why it is important, and how to calculate it for a 3×3 matrix!

3. Let $A =\in \mathbb{R}^{n,n}$ be a matrix. A nonzero vector u is an eigenvector of A with a corresponding eigenvalue λ if $Au = \lambda u$. We call the eigenvector together with its eigenvalue an eigenpair. Find matrix A with eigenpairs ([1,0],2) and ([1,1],3). Solve this problem in markdown without code.

Solution:

1.

Firstly, we calculate the product AB.

Since A is a 3*4 matrix and B is 2 * 3 matrix， then the number of column of A doesn't match the number of row of B, so AB cannot be calculated.

Secondly, we calcuate the product A^TB.

Since A^T is a 4*3 matrix and B is 2 * 3 matrix, then the number of column of A^T doesn't match the number of row of B, so A^TB cannot be calculated.

Thirdly, we calculate the product BA.

Since B is 2 * 3 matrix and A is a 3 * 4 matrix, then BA is a 2 * 4 matrix.

BA = $\begin{bmatrix} 3*4+(-1)*2+2*(-1) & 3*(-5)+(-1)*8+2*5 \\ 4*4+2*2+0*(-1) & 4*(-5)+2*8+0*5 \end{bmatrix}$ = $\begin{bmatrix} 8 & -13 & 6 & 7 \\ 20 & -4 & 0 & 4 \end{bmatrix}$

Lastly, we calcuate the product B^TA.

Since B^T is a 3 * 2 matrix and A is 3 * 4 matrix, then the number of column of B^T doesn't match the number of row of A,, so B^TA cannot be calculated.

2. The determinant of a matrix is defined as the determinant of its induced transformation. That means, the determinant is multiplicative with respect to matrix multiplication (because it's multiplicative with respect to function composition).

Determinant is important because we can use it to figure out if the matrix is invertible(if det!=0, then invertible), and find the eigenvalue/eigenvector, and use Cramer's rule to solve function.

For a 3*3 matrix P where P = $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

det(P) = a(ei−fh)−b(di−fg)+c(dh−eg)

3.

Since the eigenvetors u are 2-dimension, then we assume A is 2*2 matrix.

Let A = $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Let eigenvector u_1 = [1,0] with corresponding eigenvalue λ_1 = 2.

Let eigenvector u_2 = [1,1] with corresponding eigenvalue λ_2 = 3.

Since u is an eigenvector of A with a corresponding eigenvalue λ if $Au = \lambda u$, then $Au_1 = \lambda_1 u_1$ and $Au_2 = \lambda_2 u_2$.

We plug u_1, u_2, λ_1, λ_2 into equations and get:

$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ c \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$

$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} a+b \\ c+d \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$

So, we get a = 2, c=0, b=1, d=3.

Thus, A = $\begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$.

Coding Assignment

Numpy and Matplotlib

Introduction

Please follow the instructions in this google doc **before** you start the coding assignment. You will be asked to create a DATA2060 conda environment. We recommend that you use conda but if you are more familiar with other package managers (like docker, homebrew, poetry), feel free to use those. However, please note that the TAs might not be able to help if you do not use conda. The most important thing is to install the packages with their versions as shown in the data2060.yml file of the course's github repository. **Two points will be deducted for each failed test, and you'll receive -14 points if you do not run the cell below.**

The purpose of this section is to introduce you to some tools that you will find useful and/or necessary in order to complete future homeworks. By the end of

this assignment, you will have used numpy to perform efficient computations, loaded standard datasets using sklearn, and used matplotlib to visualize several performance metrics you will be using this semester. This homework will also get you familiar with the hand-in process for jupyter notebooks.

In [1]:
```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12.11 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.11"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.11"):
    print(FAIL, "Python version 3.12.11 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'matplotlib': "3.10.5", 'numpy': "2.3.2",'sklearn': "1.7.1",
                'pandas': "2.3.2", 'pytest': "8.4.1", 'torch':"2.7.1"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.11

[ OK ] matplotlib version 3.10.5 is installed.
[ OK ] numpy version 2.3.2 is installed.
[ OK ] sklearn version 1.7.1 is installed.
[ OK ] pandas version 2.3.2 is installed.
[ OK ] pytest version 8.4.1 is installed.
[ OK ] torch version 2.7.1 is installed.
```

In [1]:
```
# import packages here
import numpy as np  # Used to perform efficient (and convenient) array and matri
from sklearn import datasets  # Used to load standard datasets.
from matplotlib import pyplot as plt  # Used to create plots.
import math  # Used for trigonometric functions, log, pow, etc.
```

Part 1: Matplotlib (4 points)

This question contains functions you need to fill out, `graph_iris_data` and `graph_series_data`. Note that the `plt.show()` call should be the last call, so add all of your graph customization below the TODO, but above the `show()` call!

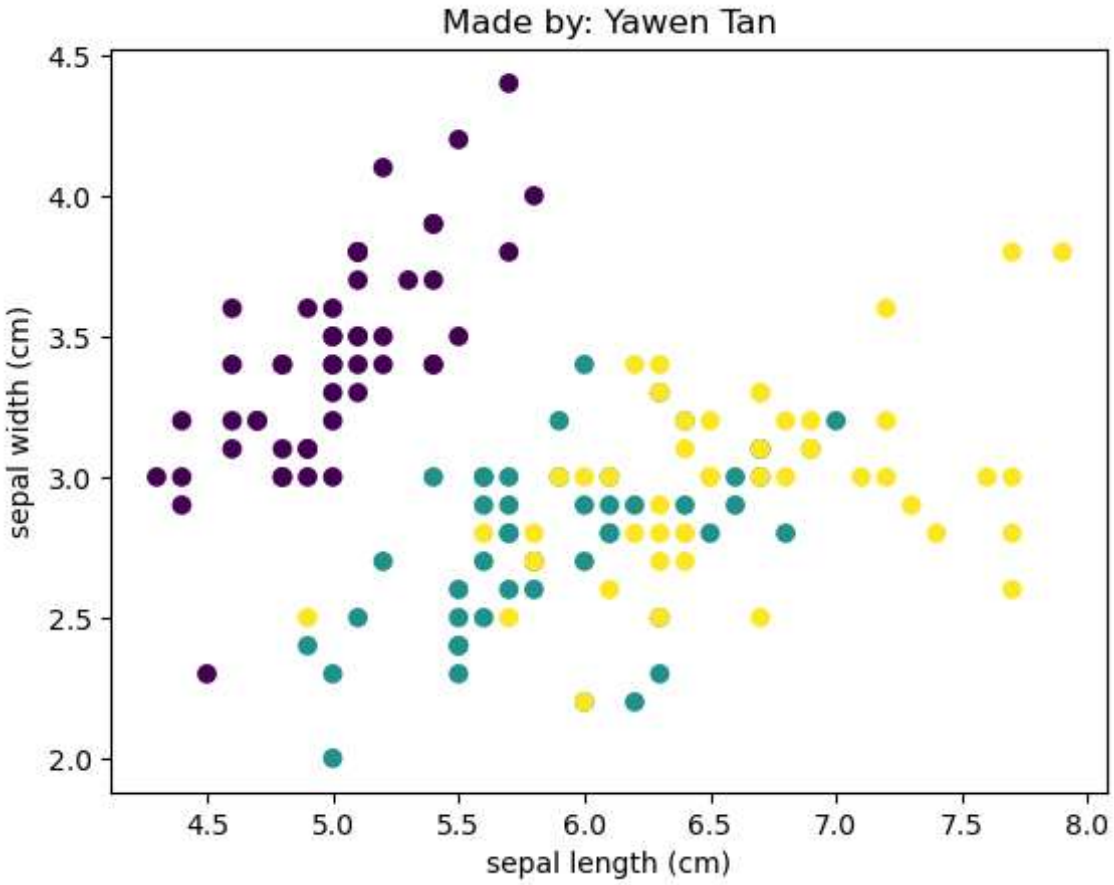
1. call `plt.scatter(x, y, c=None)`, giving it the following arguments:
   - the `x` argument will be `xs`
   - the `y` argument will be `ys`
   - in order to give the plotted points color, we will specify the optional c argument. Thus, we will pass a third argument, `c = iris.target`
2. Look into the matplotlib.pyplot documentation and learn how to add titles and axis labels to plots. Add a title to the plot of the form "Made by: [your name]".
3. Examine the column names of the iris data to find appropriate x and y labels for the plot, and use pyplot commands to label the axes of the two iris plots.

In [ ]:
```
# load the data
iris = datasets.load_iris()
data = iris.data
xs = data[:, 0]
ys = data[:, 1]

# [TODO] plot iris data here

# Create a scatter for the iris data
plt.scatter(xs, ys, c=iris.target)

# Add a title to the plot
plt.title("Made by: Yawen Tan")

# Examine the column names of the iris data to find appropriate x and y labels f
print(iris.feature_names)

# Add x and y labels to the plot
plt.xlabel(iris.feature_names[0])
```

```
plt.ylabel(iris.feature_names[1])

plt.show()
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (c
m)']
```

Out[ ]: Text(0, 0.5, 'sepal width (cm)')

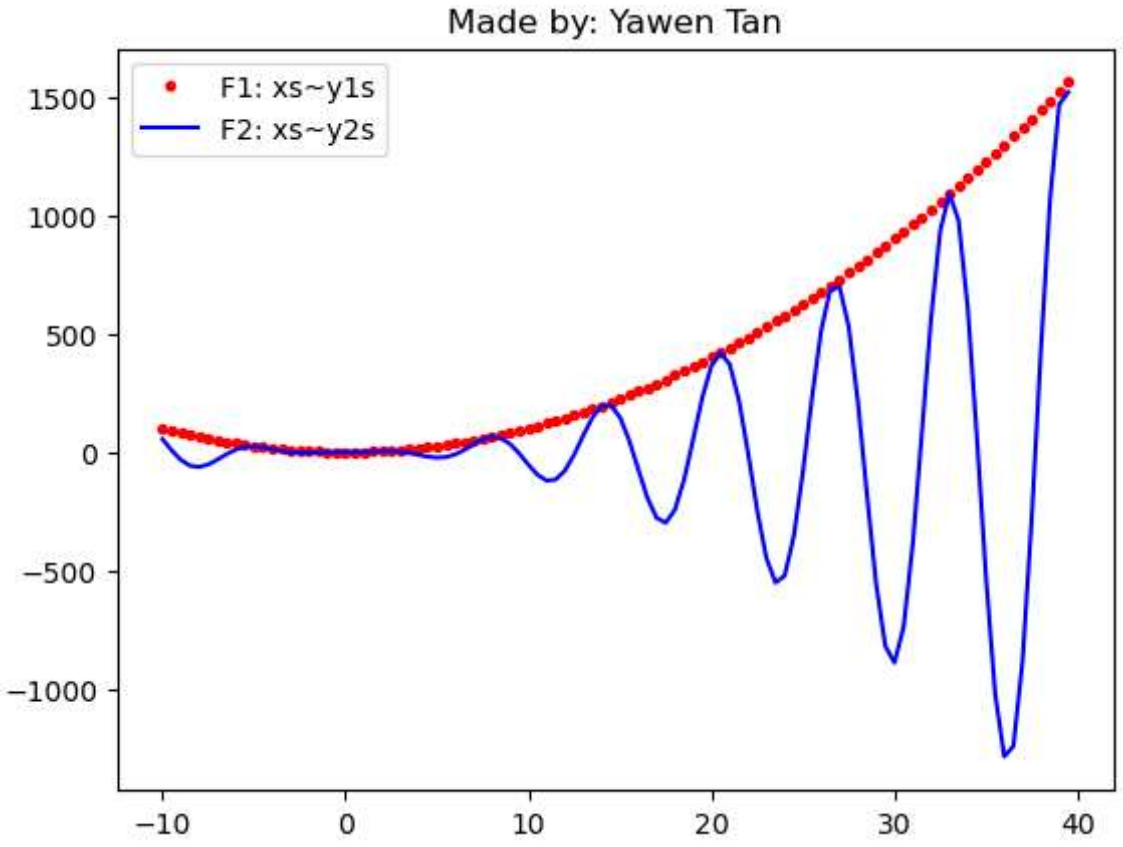


We will use a similar process for `graph_series_data`.

1. call `plt.plot(x, y, format)`, giving it the following arguments:
   - the `x` argument will be `xs`
   - the `y` argument will be `y1s`
   - the `format` argument will be `'.r'`
2. call `plt.plot(x, y, format)` again, giving it the following arguments:
   - the `x` argument will be `xs`
   - the `y` argument will be `y2s`
   - the `format` argument will be `'-b'`
3. Add a title to the plot of the form "Made by: [your name]".
4. Use pyplot commands to add a legend to the series data plot, where each series is labeled with its function. Look here to start!

In [ ]:
```
xs = np.arange(100) * .5 - 10  # Creates a list of 100 values in intervals of .5
# Numpy makes it very easy (and efficient) to do elementwise operations on large
y1s = [x*x for x in xs]  # If you are unfamiliar, this is called Python's "List
y2s = [math.sin(x)*x*x for x in xs]

# https://matplotlib.org/users/pyplot_tutorial.html is a good starting point for
# [TODO] plot series data here

# Create a plot with two series on it
plt.plot(xs, y1s, '.r', label='F1: xs~y1s')
plt.plot(xs, y2s, '-b',label='F2: xs~y2s')

# Add legend and title to the plot
plt.legend()
plt.title("Made by: Yawen Tan")

plt.show()
```

Out[ ]:
```
Text(0.5, 1.0, 'Made by: Yawen Tan')
```



Part 2: Numpy (7 points)

Answer the following questions using numpy functions. Note that when importing numpy, it is often abbreviated to np (i.e., `import numpy as np`), so when calling numpy functions, you can use np.[function]. Further, note that you may NOT use the np array constructor to solve these questions (i.e., `np.array(...)`). Some functions you may want to consider to approach this problem are `np.arange`, `np.zeros`, `np.ones`, `np.eye`, `np.sum`, `np.hstack`, `np.vstack`, `np.transpose`, `np.matmul`, `np.inner`, `np.where` and `np.dot`.

- Using numpy, how would you create a 1D array containing the values 2 through 6 inclusive?

In [ ]:
```
# Solution:
# Create a 1D numpy array with values ranging from 2 to 6 (inclusive)
array_1d = np.arange(2, 7)

print(array_1d)
```

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

- Using numpy, how would you create a 4x4 matrix where all the values are 1?

In [ ]:
```
# Solution:

# Create a 4x4 numpy matrix filled with ones
matrix_1 = np.ones((4, 4),dtype=int)

print(matrix_1)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

- Using numpy, how would you create a 6x6 identity matrix?

In [ ]:
```
# Solution:

# Create a 6x6 identity matrix
identity_matrix = np.eye(6,dtype=int)

print(identity_matrix)
```

```
[[1 0 0 0 0 0]
 [0 1 0 0 0 0]
 [0 0 1 0 0 0]
 [0 0 0 1 0 0]
 [0 0 0 0 1 0]
 [0 0 0 0 0 1]]
```

- Using numpy, how would you sum the values of each column of matrix A?

In [ ]:
```
# Solution:

# Create a matrix A
A = np.array([[1, 2],
              [3, 4]])
# Compute the sum of each column in matrix A
column_sums = np.sum(A, axis=0)

print(column_sums)
```

```
[4 6]
```

- Using numpy, given the matrices `A` and `B`, how would you find the matrix `C`, where `C` = $A^T B$?

In [9]:
```
# Solution:
# Create matrix A and B
A = np.array([[1, 2],
              [3, 4]])
B = np.array([[1, 1],
              [0, 1]])
# Calculate the matrix product of the transpose of A and B
C = np.matmul(A.T, B)

print(C)
```

```
[[1 4]
 [2 6]]
```

- Using numpy, using either np.vstack or np.hstack, how would you create a 4x2 matrix where all the values in the first column are 0's and all the values in the second column are 1's?

In [ ]:
```
# Solution:

# Create 0 column and 1 column
col_0 = np.zeros((4, 1),dtype=int)
col_1 = np.ones((4, 1),dtype=int)

# Combine the two columns side by side to form a 4x2 matrix
matrix = np.hstack((col_0, col_1))

print(matrix)
```

```
[[0 1]
 [0 1]
 [0 1]
 [0 1]]
```

- Given a matrix of floats A, using numpy, how would you return an array of the same shape, where all values > 3.0 are set to 1, and the rest to 0?

In [ ]:
```
# Solution:

# Create matrix of floats A
A = np.array([[4.1, 2.5],
              [3.6, 1.2]])
# Create matrix M where each element is 1 if the corresponding element in A is g
M = np.where(A > 3.0, 1, 0)

print(M)
```

```
[[1 0]
 [1 0]]
```