

Machine Learning (ICP # 3)

Ramya Sadhineni

700757305

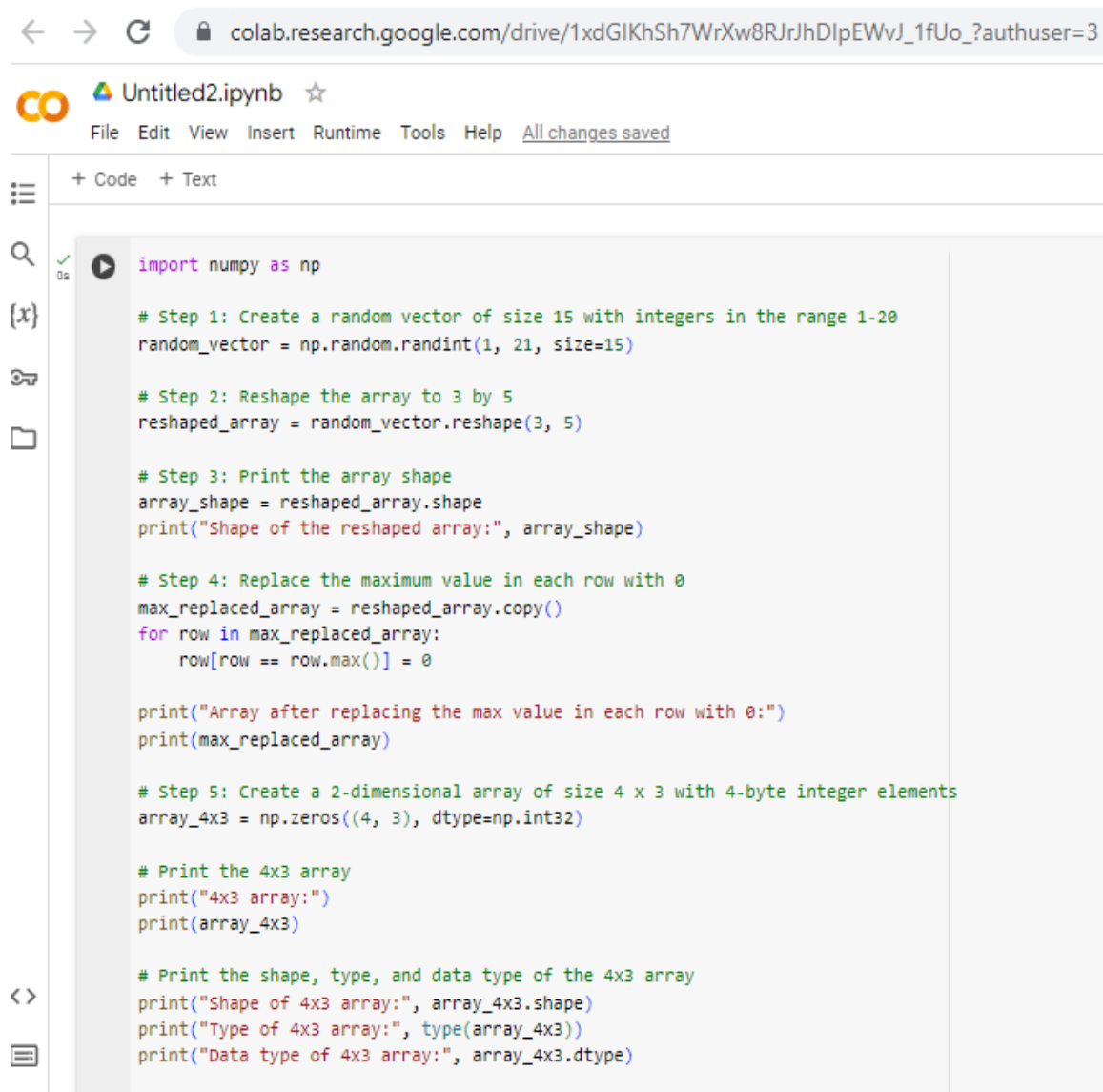
GitHub link: <https://github.com/RamyaSadhineni/Repo>
CRN:30562

a. Using NumPy create random vector of size 15 having only Integers in the range 1-20.

1. Reshape the array to 3 by 5
2. Print array shape.
3. Replace the max in each row by 0

Create a 2-dimensional array of size 4 x 3 (composed of 4-byte integer elements), also print the shape, type and data type of the array.

Code:



```
colab.research.google.com/drive/1xdGKhSh7WrXw8RJrJhDlpEWvJ_1fUo_?authuser=3

Untitled2.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

import numpy as np

# Step 1: Create a random vector of size 15 with integers in the range 1-20
random_vector = np.random.randint(1, 21, size=15)

# Step 2: Reshape the array to 3 by 5
reshaped_array = random_vector.reshape(3, 5)

# Step 3: Print the array shape
array_shape = reshaped_array.shape
print("Shape of the reshaped array:", array_shape)

# Step 4: Replace the maximum value in each row with 0
max_replaced_array = reshaped_array.copy()
for row in max_replaced_array:
    row[row == row.max()] = 0

print("Array after replacing the max value in each row with 0:")
print(max_replaced_array)

# Step 5: Create a 2-dimensional array of size 4 x 3 with 4-byte integer elements
array_4x3 = np.zeros((4, 3), dtype=np.int32)

# Print the 4x3 array
print("4x3 array:")
print(array_4x3)

# Print the shape, type, and data type of the 4x3 array
print("Shape of 4x3 array:", array_4x3.shape)
print("Type of 4x3 array:", type(array_4x3))
print("Data type of 4x3 array:", array_4x3.dtype)
```

Machine Learning (ICP # 3)

Ramya Sadhineni

700757305

GitHub link: <https://github.com/RamyaSadhineni/Repo>

CRN:30562

Explanation:

The Python code mentioned above manipulates arrays in multiple ways using NumPy. First, a random vector consisting of 15 integers between 1 and 20 is created. Next, a 3x5 array is created by reshaping this vector. The modified array's shape is printed. The maximum value in each row of the reshaped array is then replaced by 0 by the code. Furthermore, a 4x3 array that is entirely composed of zeros is made, indicating that every element is a 4-byte integer (int32). The 4x3 array, together with its form, type, and data type, are finally printed by the code. This series of steps illustrates basic array functions in NumPy, including reshaping, element-wise manipulation, and accessing array properties.

Output:

```
➦ Shape of the reshaped array: (3, 5)
Array after replacing the max value in each row with 0:
[[10  7  4  5  0]
 [ 1 12  0  7  4]
 [11  5  4  6  0]]
4x3 array:
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
Shape of 4x3 array: (4, 3)
Type of 4x3 array: <class 'numpy.ndarray'>
Data type of 4x3 array: int32
```

Machine Learning (ICP # 3)

Ramya Sathineni

700757305

GitHub link: <https://github.com/RamyaSathineni/Repo>

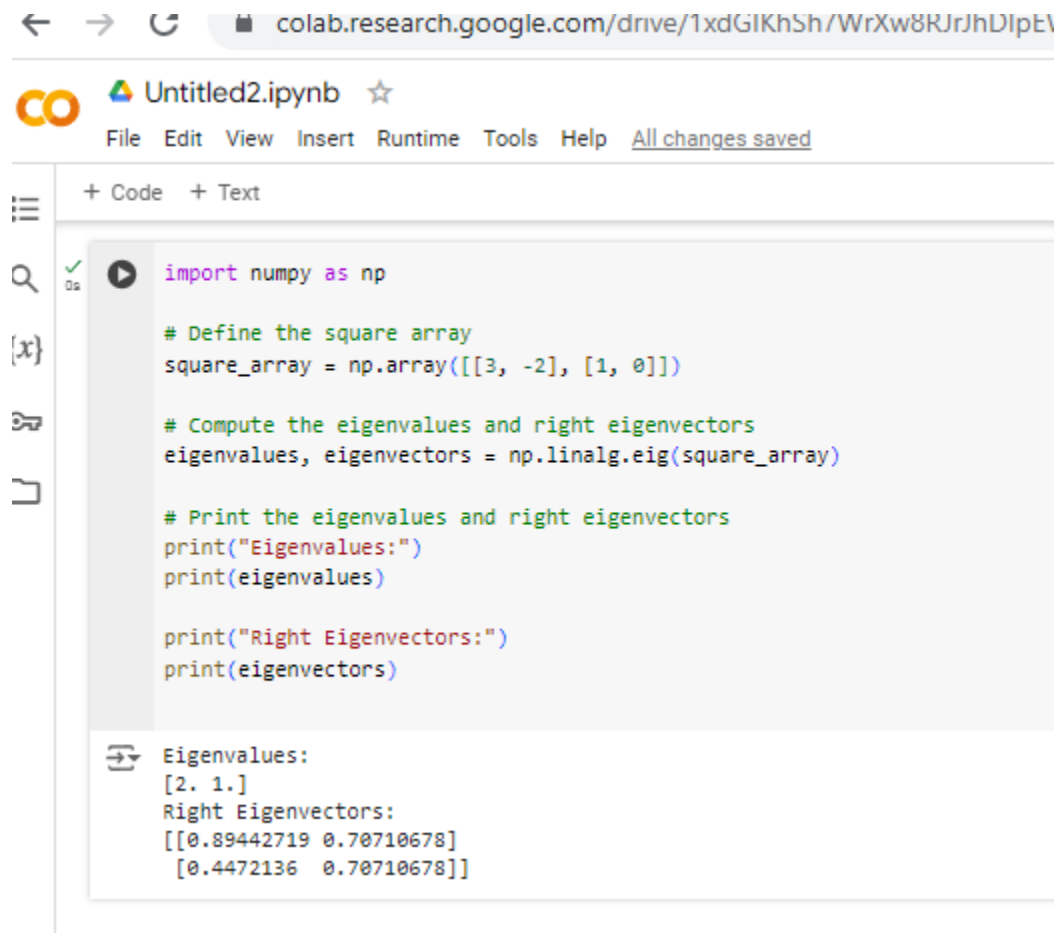
CRN:30562

b. Write a program to compute the eigenvalues and right eigenvectors of a given square array given below:

```
[[ 3 -2]
```

```
[ 1  0]]
```

Code and Output:



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1xdGKhSh/WrXw8RJrJhDlpeV`. The notebook is titled "Untitled2.ipynb" and has a star icon. The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", "Help", and "All changes saved". The left sidebar contains icons for file explorer, search, and other notebook functions. The main code cell is titled "+ Code" and contains the following Python code:

```
import numpy as np

# Define the square array
square_array = np.array([[3, -2], [1, 0]])

# Compute the eigenvalues and right eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(square_array)

# Print the eigenvalues and right eigenvectors
print("Eigenvalues:")
print(eigenvalues)

print("Right Eigenvectors:")
print(eigenvectors)
```

The output of the code is displayed below the code cell:

```
Eigenvalues:
[2.  1.]
Right Eigenvectors:
[[0.89442719  0.70710678]
 [0.4472136   0.70710678]]
```

Explanation :

The provided Python code takes a given 2x2 matrix and uses NumPy to compute its eigenvalues and right eigenvectors. It defines the matrix first. It calculates the matrix's eigenvalues and matching right eigenvectors using the `np.linalg.eig` function. After that, the console prints the results. This illustrates how to find eigenvalues and eigenvectors using NumPy for linear algebraic operations.

Machine Learning (ICP # 3)

Ramya Sadhineni

700757305

GitHub link: <https://github.com/RamyaSadhineni/Repo>

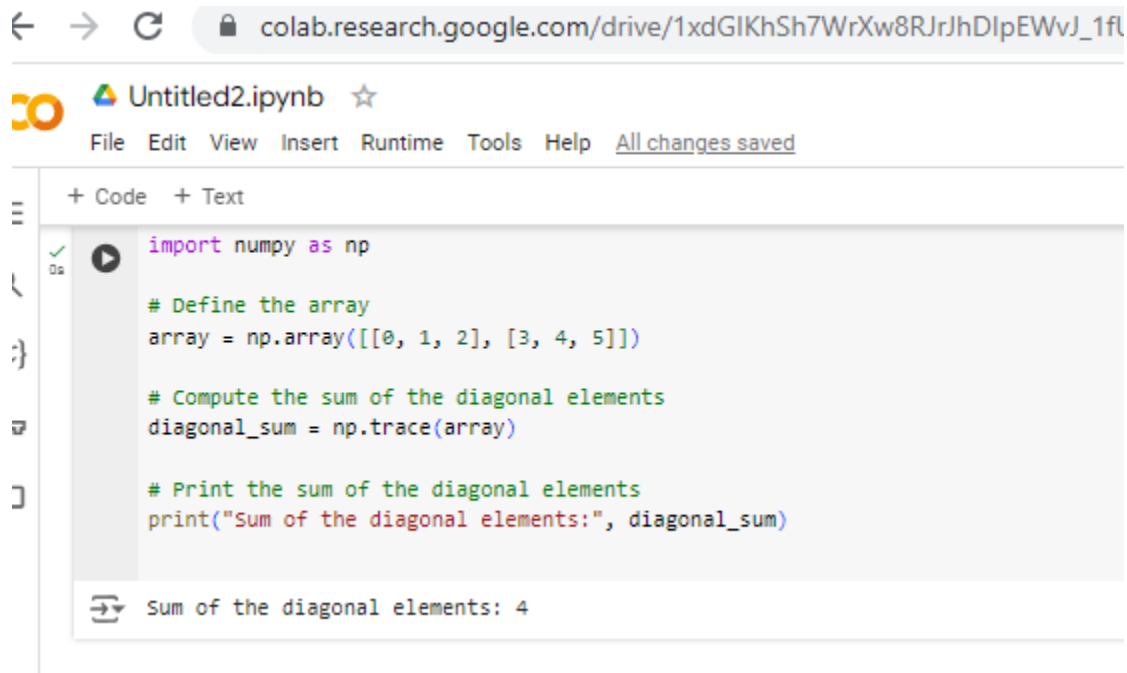
CRN:30562

c. Compute the sum of the diagonal element of a given array.

```
[[0 1 2]
```

```
[3 4 5]]
```

Code and Output:



The screenshot shows a Google Colab notebook interface. At the top, the URL bar displays `colab.research.google.com/drive/1xdGIKhSh7WrXw8RJrJhDIpEWvJ_1fL`. Below the URL bar, the notebook is titled "Untitled2.ipynb". The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", "Help", and "All changes saved". The left sidebar shows a file explorer with a folder icon and a list of files. The main area contains a code cell with the following Python code:

```
import numpy as np

# Define the array
array = np.array([[0, 1, 2], [3, 4, 5]])

# Compute the sum of the diagonal elements
diagonal_sum = np.trace(array)

# Print the sum of the diagonal elements
print("Sum of the diagonal elements:", diagonal_sum)
```

Below the code cell, the output is displayed: "Sum of the diagonal elements: 4".

Explanation :

The offered Python function takes a supplied 2x3 array and returns its diagonal element sum. The NumPy library, which is necessary for numerical operations on arrays, is imported first. After that, the array is defined. It uses the `{np.trace}` function, which adds the items directly on the array's major diagonal (in this case, elements 0 and 4) to get the sum of the diagonal elements. The computed sum is finally printed to the console. This illustrates how to use NumPy to efficiently execute matrix computations.

Machine Learning (ICP # 3)

Ramya Sadhineni

700757305

GitHub link: <https://github.com/RamyaSadhineni/Repo>

CRN:30562

d. Write a NumPy program to create a new shape to an array without changing its data.

Reshape 3x2:

```
[[1 2]
```

```
[3 4]
```

```
[5 6]]
```

Reshape 2x3:

```
[[1 2 3]
```

```
[4 5 6]]
```

Code and output:

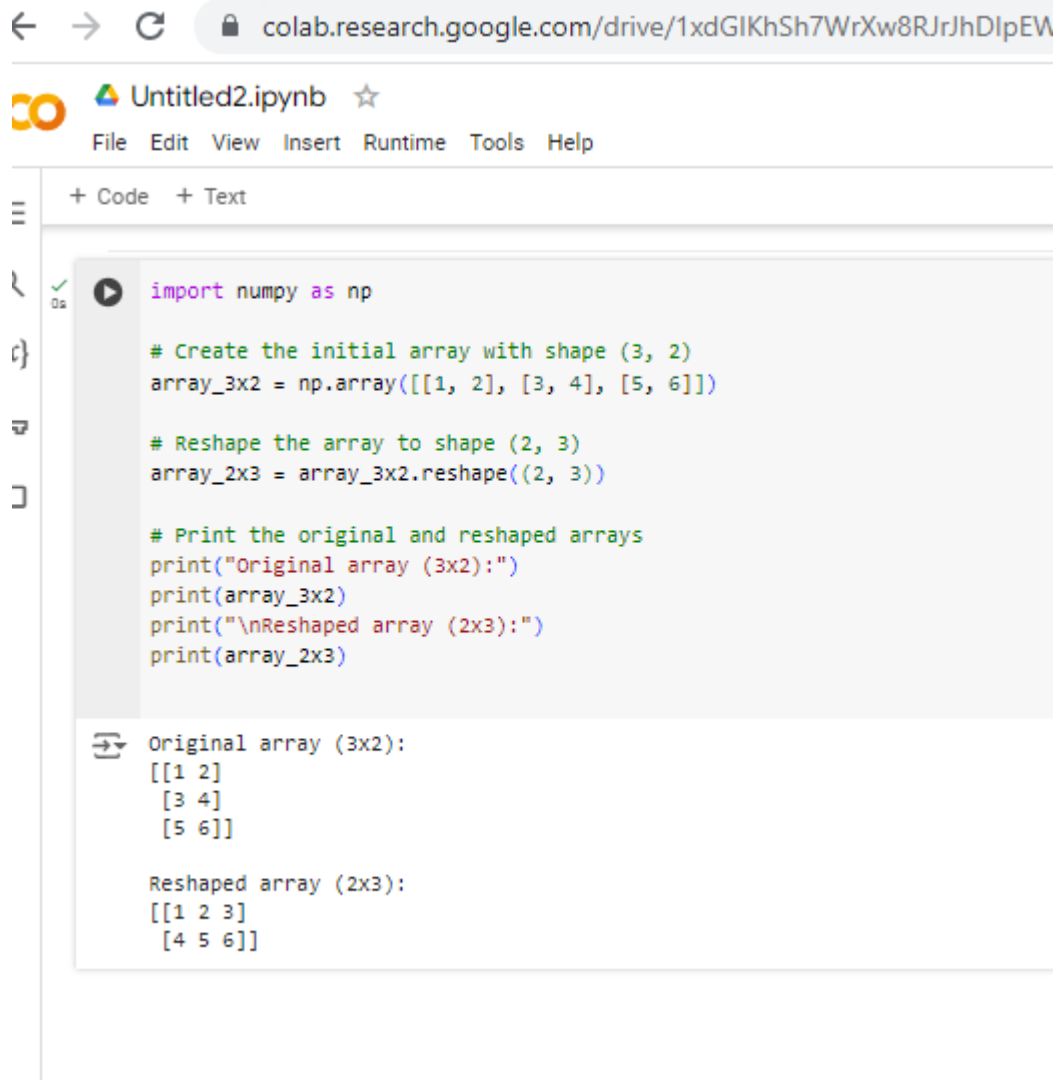
Machine Learning (ICP # 3)

Ramya Sathineni

700757305

GitHub link: <https://github.com/RamyaSathineni/Repo>

CRN:30562



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1xdGIKhSh7WrXw8RJrJhDlpEV`. The notebook is titled "Untitled2.ipynb". The code editor contains the following Python code:

```
import numpy as np

# Create the initial array with shape (3, 2)
array_3x2 = np.array([[1, 2], [3, 4], [5, 6]])

# Reshape the array to shape (2, 3)
array_2x3 = array_3x2.reshape((2, 3))

# Print the original and reshaped arrays
print("Original array (3x2):")
print(array_3x2)
print("\nReshaped array (2x3):")
print(array_2x3)
```

The output of the code is displayed below the code cell:

```
Original array (3x2):
[[1 2]
 [3 4]
 [5 6]]

Reshaped array (2x3):
[[1 2 3]
 [4 5 6]]
```

Explanation:

This sample of code shows how to use NumPy to convert a 3x2 array into a 2x3 array without altering the contents. Using `np.array`, a 3x2 array is first created, and then the `reshape` method is used to reshape it. After reshaping, the array `[[1, 2], [3, 4], [5, 6]]` becomes `[[1, 2, 3], [4, 5, 6]]`. The array's dimensions can be changed while maintaining the elemental order by using the `reshape` function.