Introduction To Machine Learning and Applications

Assignment 1.
Submitted by,
Md Rifat Ul Karim Shovon
Hr1396

Question 1:

Problem: During physical design, layout verification checks for violations of design rules (such as minimum spacing between components, metal layer widths, etc.) that could lead to manufacturing defects or suboptimal performance. Detecting these violations automatically using machine learning could significantly speed up the design process.

Classify: Basically, this would fall into a problem of supervised classification. The model of machine learning shall be trained on labeled data of known good and bad layouts, with and without rule violations, respectively. Then the model will predict whether new layouts have any violations or classify the type if any.

Unique Challenges to Standard Machine Learning Methods:

<u>High dimensionality:</u> The VLSI layouts are large and complex designs with several design rules that come into consideration together. Standard models such as logistic regression may not be able to handle this complexity without dimensionality reduction or feature selection.

<u>Imbalanced Sparse Data:</u> There may be a case where the violation of rules is sparse compared to valid designs. For instance, logistic regression struggles when there is class imbalance. This could also make the models biased toward the class that contains a majority-normally the valid designs. More powerful techniques, such as SVM or ensemble methods, might do better on this count.

<u>Geometrical nature of data:</u> layout design is intrinsically geometric, which can't be innately handled by typical machine learning models. That might be suitable for a problem that could be handled well by CNNs, considering the grid-like nature of images.

<u>Real-time constraints:</u> The model must operate in an agile design setting. Therefore, used algorithms should be efficient to offer rapid processing for large layout designs. This poses a challenge to some machine learning approaches that may demand very high computational resources.

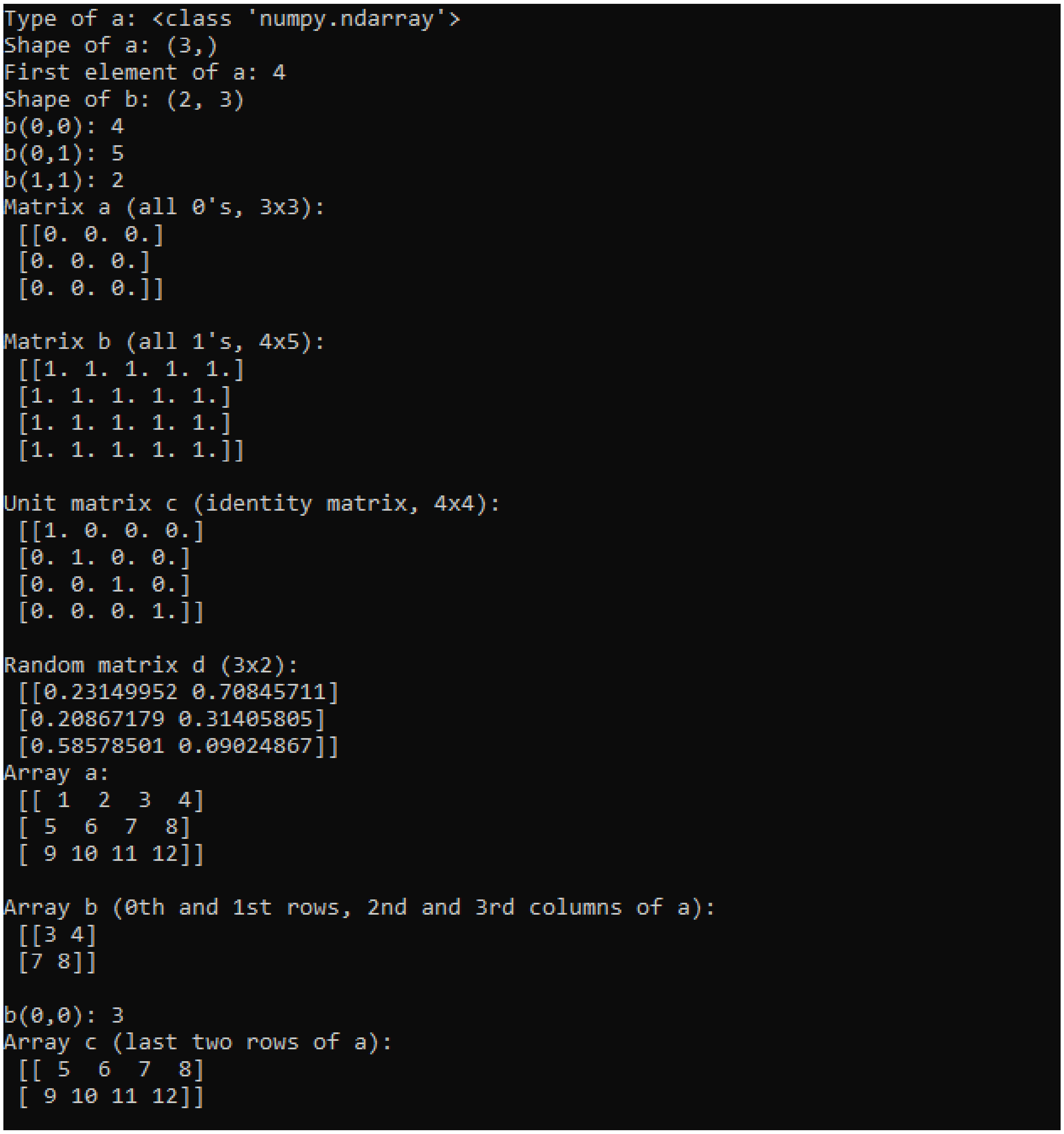
Question2:

```
import numpy as np
### 2
# Create a one-dimensional array a and initialize it as [4, 5, 6]
a = np.array([4, 5, 6])

# (1) Print the type of a
print("Type of a:", type(a))

# (2) Print the shape of a
print("Shape of a:", a.shape)

# (3) Print the first element in a (the value should be 4)
print("First element of a:", a[0])

#### 3
# Create a two-dimensional array b and initialize it as [ [4,5,6], [1,2,3] ]
b = np.array([[4, 5, 6], [1, 2, 3]])

# (1) Print the shape of b
print("Shape of b:", b.shape)

# (2) Print b(0,0), b(0,1), b(1,1) (the values should be 4, 5, 2)
print("b(0,0):", b[0, 0])
print("b(0,1):", b[0, 1])
print("b(1,1):", b[1, 1])
# #### 4.

# (1) Create a matrix a, which is all 0, of size 3x3
a = np.zeros((3, 3))
print("Matrix a (all 0's, 3x3):\n", a)

# (2) Create a matrix b, which is all 1, of size 4x5
b = np.ones((4, 5))
print("\nMatrix b (all 1's, 4x5):\n", b)

# (3) Create a unit matrix c, of size 4x4 (identity matrix)
c = np.eye(4)
print("\nUnit matrix c (identity matrix, 4x4):\n", c)

# (4) Create a random matrix d, of size 3x2
d = np.random.rand(3, 2)
print("\nRandom matrix d (3x2):\n", d)
# #### 5
# Create array a and initialize it as [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

# (1) Print array a
print("Array a:\n", a)

# (2) Put the 0th and 1st rows, 2nd and 3rd columns of array a into array b, then print b
b = a[0:2, 2:4]  # Select 0th and 1st rows, 2nd and 3rd columns
print("\nArray b (0th and 1st rows, 2nd and 3rd columns of a):\n", b)

# (3) Print b(0,0)
print("\nb(0,0):", b[0, 0])

# #### 6.

# Array a from question 5
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

# (1) Put all the elements of the last two rows of array a into a new array c
c = a[-2:, :]  # Select the last two rows of array a
print("Array c (last two rows of a):\n", c)

# (2) Print the last element of the first row in c (using -1 for the last element)
print("\nLast element of the first row in c:", c[0, -1])
###7

# (1) Create an array x
x = np.array([[1, 2], [3, 4]], dtype=np.float64)

# (2) Create an array y
y = np.array([[5, 6], [7, 8]], dtype=np.float64)

# (3) Print x + y and np.add(x, y)
print("x + y:\n", x + y)
print("np.add(x, y):\n", np.add(x, y))

# (4) Print x - y and np.subtract(x, y)
print("x - y:\n", x - y)
print("np.subtract(x, y):\n", np.subtract(x, y))

# (5) Print x * y, np.multiply(x, y) and np.dot(x, y), and compare the results
print("x * y:\n", x * y)
print("np.multiply(x, y):\n", np.multiply(x, y))
print("np.dot(x, y):\n", np.dot(x, y))
```

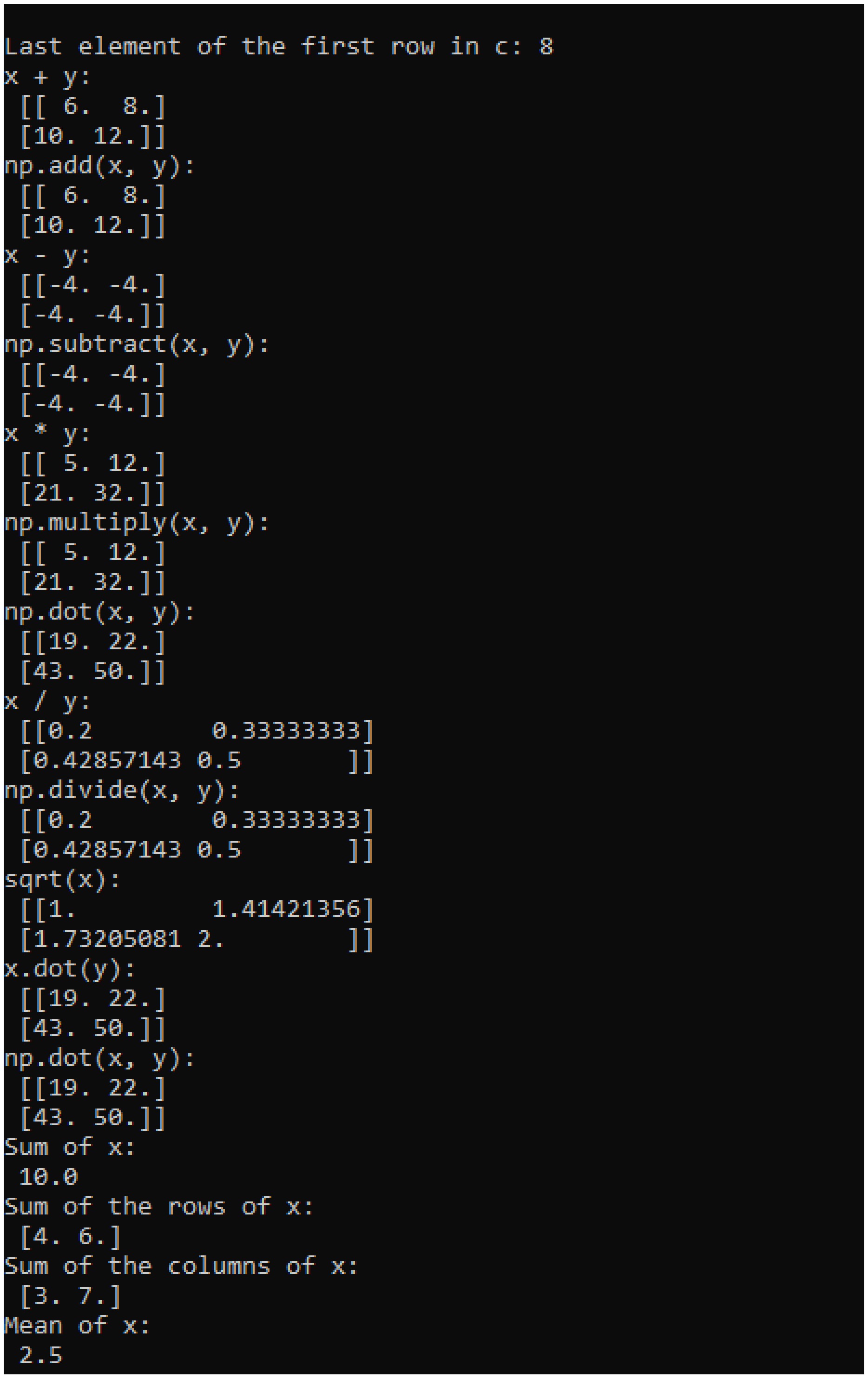
```
# (6) Print x / y and np.divide(x, y)
print("x / y:\n", x / y)
print("np.divide(x, y):\n", np.divide(x, y))

# (7) Print the square root of x
print("sqrt(x):\n", np.sqrt(x))

# (8) Print x.dot(y) and np.dot(x, y)
print("x.dot(y):\n", x.dot(y))
print("np.dot(x, y):\n", np.dot(x, y))

####8
# (1) Print the sum of x
print("Sum of x:\n", np.sum(x))

# (2) Print the sum of the rows of x
print("Sum of the rows of x:\n", np.sum(x, axis=0))

# (3) Print the sum of the columns of x
print("Sum of the columns of x:\n", np.sum(x, axis=1))


###9
# (1) Print the mean of x
print("Mean of x:\n", np.mean(x))

# (2) Print the mean of the rows of x
print("Mean of the rows of x:\n", np.mean(x, axis=0))

# (3) Print the mean of the columns of x
print("Mean of the columns of x:\n", np.mean(x, axis=1))

###10
# Print the matrix transpose of x
print("Transpose of x:\n", x.T)

###11
# (1) Print the index of the max element of x
print("Index of the max element of x:\n", np.argmax(x))

# (2) Print the index of the max element in the rows of x
print("Index of the max element in the rows of x:\n", np.argmax(x, axis=0))

# (3) Print the index of the max element in the columns of x
print("Index of the max element in the columns of x:\n", np.argmax(x, axis=1))
```

```
###12
import matplotlib.pyplot as plt

x = np.arange(0, 100, 0.1)
y = x * x

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y = x * x')
plt.title('Plot of y = x * x')
plt.grid(True)
#plt.show()
plt.savefig("y_x_srq_fig.png")

###13
x = np.arange(0, 3 * np.pi, 0.1)
plt.clf() ## clears the previous plots
# (1) Plot sin(x)
y_sin = np.sin(x)
plt.plot(x, y_sin, label='sin(x)')
plt.xlabel('x')
plt.ylabel('y = sin(x)')
plt.title('Plot of y = sin(x)')
plt.legend()
plt.grid(True)
#plt.show()
plt.savefig("sine_X.png")

# (2) Plot cos(x)
plt.clf() ## clears the previous plots
y_cos = np.cos(x)
plt.plot(x, y_cos, label='cos(x)')
plt.xlabel('x')
plt.ylabel('y = cos(x)')
plt.title('Plot of y = cos(x)')
plt.legend()
plt.grid(True)
#plt.show()
plt.savefig("cos_X.png")
```

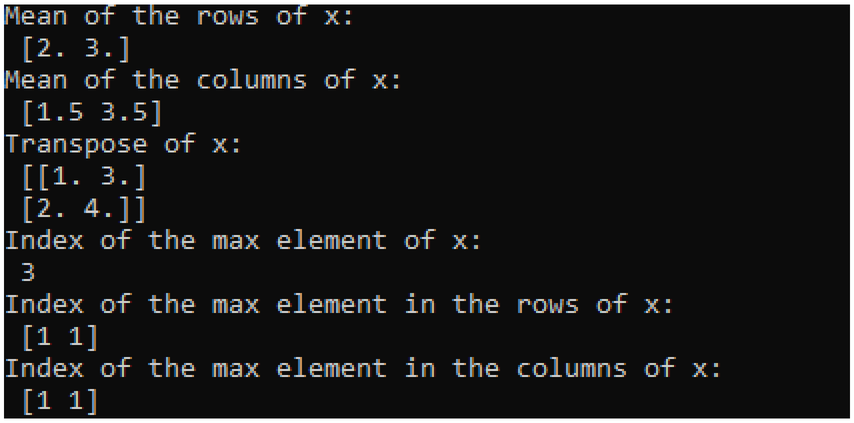
Output from the shell;



```
Type of a: <class 'numpy.ndarray'>
Shape of a: (3,)
First element of a: 4
Shape of b: (2, 3)
b(0,0): 4
b(0,1): 5
b(1,1): 2
Matrix a (all 0's, 3x3):
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

Matrix b (all 1's, 4x5):
 [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]

Unit matrix c (identity matrix, 4x4):
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Random matrix d (3x2):
 [[0.23149952 0.70845711]
 [0.20867179 0.31405805]
 [0.58578501 0.09024867]]
Array a:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

Array b (0th and 1st rows, 2nd and 3rd columns of a):
 [[3 4]
 [7 8]]

b(0,0): 3
Array c (last two rows of a):
 [[ 5  6  7  8]
 [ 9 10 11 12]]
```

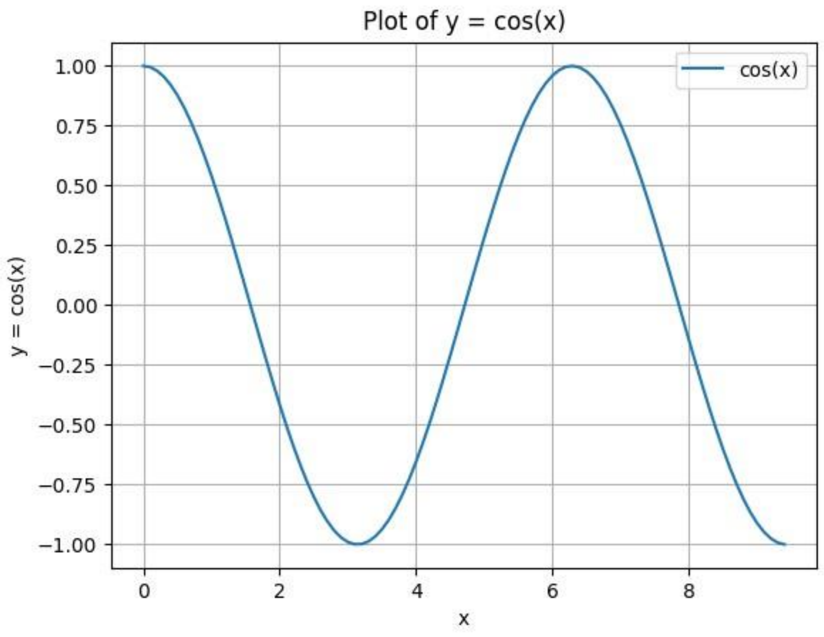


```
Last element of the first row in c: 8
x + y:
 [[ 6.  8.]
 [10. 12.]]
np.add(x, y):
 [[ 6.  8.]
 [10. 12.]]
x - y:
 [[-4. -4.]
 [-4. -4.]]
np.subtract(x, y):
 [[-4. -4.]
 [-4. -4.]]
x * y:
 [[ 5. 12.]
 [21. 32.]]
np.multiply(x, y):
 [[ 5. 12.]
 [21. 32.]]
np.dot(x, y):
 [[19. 22.]
 [43. 50.]]
x / y:
 [[0.2        0.33333333]
 [0.42857143 0.5       ]]
np.divide(x, y):
 [[0.2        0.33333333]
 [0.42857143 0.5       ]]
sqrt(x):
 [[1.         1.41421356]
 [1.73205081 2.        ]]
x.dot(y):
 [[19. 22.]
 [43. 50.]]
np.dot(x, y):
 [[19. 22.]
 [43. 50.]]
Sum of x:
 10.0
Sum of the rows of x:
 [4. 6.]
Sum of the columns of x:
 [3. 7.]
Mean of x:
 2.5
```

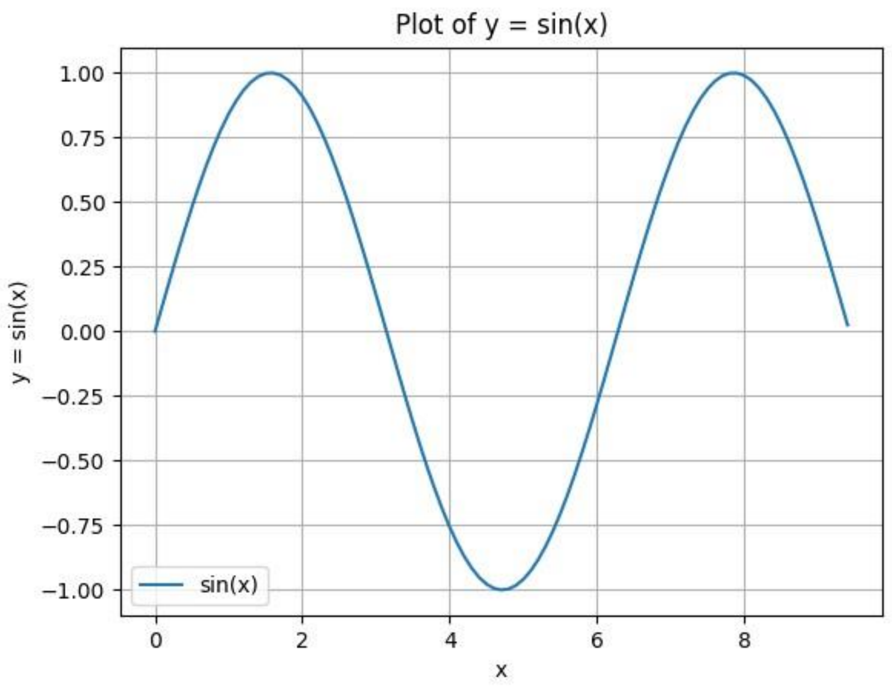


```
Mean of the rows of x:
 [2. 3.]
Mean of the columns of x:
 [1.5 3.5]
Transpose of x:
 [[1. 3.]
 [2. 4.]]
Index of the max element of x:
 3
Index of the max element in the rows of x:
 [1 1]
Index of the max element in the columns of x:
 [1 1]
```

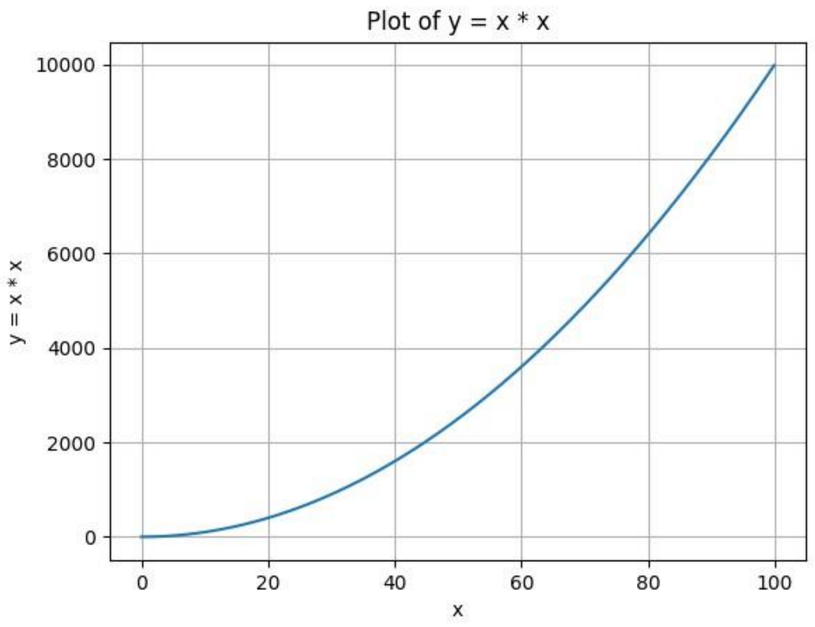
Cos(x) plotting:



Sin(x) plotting:



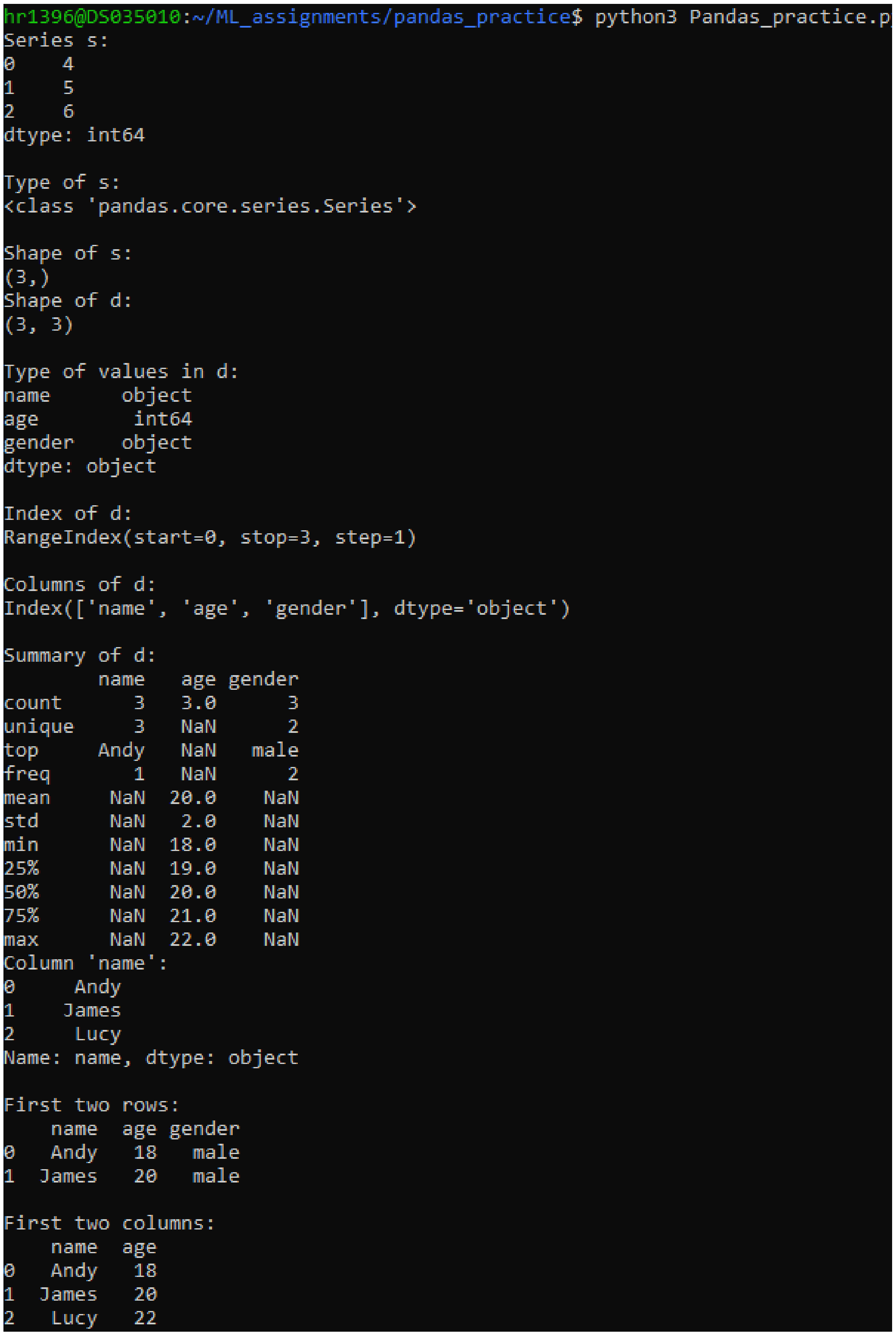
Y=x^2 plotting:



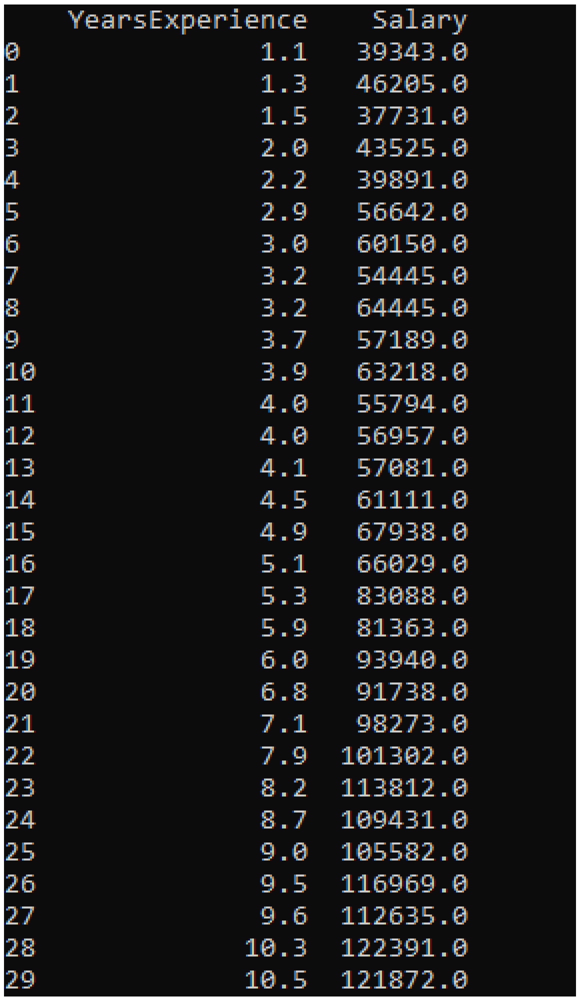
Question3:

```
import matplotlib.pyplot as plt
import pandas as pd
###2
# (1) Create a pandas Series s
s = pd.Series([4, 5, 6])

# (2) Print s
print("Series s:")
print(s)

# (3) Print the type of s
print("\nType of s:")
print(type(s))

# (4) Print the shape of s
print("\nShape of s:")
print(s.shape)
###3

# Create a DataFrame based on the given dictionary
dic = {
    'name': ['Andy', 'James', 'Lucy'],
    'age': [18, 20, 22],
    'gender': ['male', 'male', 'female']
}
d = pd.DataFrame(dic)

# (1) Print the shape of d
print("Shape of d:")
print(d.shape)

# (2) Print the type of values in d
print("\nType of values in d:")
print(d.dtypes)

# (3) Print the index of d
print("\nIndex of d:")
print(d.index)

# (4) Print the columns of d
print("\nColumns of d:")
print(d.columns)

# (5) Print the summary of d using describe()
print("\nSummary of d:")
print(d.describe(include='all'))

###4

# (1) Select and print one column ('name') from DataFrame d
print("Column 'name':")
print(d['name'])

# (2) Select and print the first two rows from DataFrame d
print("\nFirst two rows:")
print(d.head(2))

# (3) Select and print the first two columns from DataFrame d
print("\nFirst two columns:")
print(d.iloc[:, :2])

#####
# Read the CSV file
df = pd.read_csv('salary.csv')

# Print the DataFrame to verify the contents
print(df)

###6
# Plot the 'Salary' column
plt.figure(figsize=(10, 6))  # Optional: Set the figure size
plt.plot(df['Salary'], marker='o', linestyle='-', color='b')  # Plotting the salary data
plt.xlabel('Index')  # Label for the x-axis
plt.ylabel('Salary')  # Label for the y-axis
plt.title('Salary Data')  # Title of the plot
plt.grid(True)  # Show grid
plt.show()  # Display the plot
plt.savefig("Salary_vs_age.png")
```

Output from the shell:



```
hr1396@DS035010:~/ML_assignments/pandas_practice$ python3 Pandas_practice.p
Series s:
0    4
1    5
2    6
dtype: int64

Type of s:
<class 'pandas.core.series.Series'>

Shape of s:
(3,)
Shape of d:
(3, 3)

Type of values in d:
name      object
age        int64
gender    object
dtype: object

Index of d:
RangeIndex(start=0, stop=3, step=1)

Columns of d:
Index(['name', 'age', 'gender'], dtype='object')

Summary of d:
        name   age gender
count      3   3.0      3
unique     3   NaN      2
top     Andy   NaN   male
freq       1   NaN      2
mean     NaN  20.0    NaN
std      NaN   2.0    NaN
min      NaN  18.0    NaN
25%      NaN  19.0    NaN
50%      NaN  20.0    NaN
75%      NaN  21.0    NaN
max      NaN  22.0    NaN
Column 'name':
0     Andy
1    James
2     Lucy
Name: name, dtype: object

First two rows:
    name  age gender
0   Andy   18   male
1  James   20   male

First two columns:
    name  age
0   Andy   18
1  James   20
2   Lucy   22
```



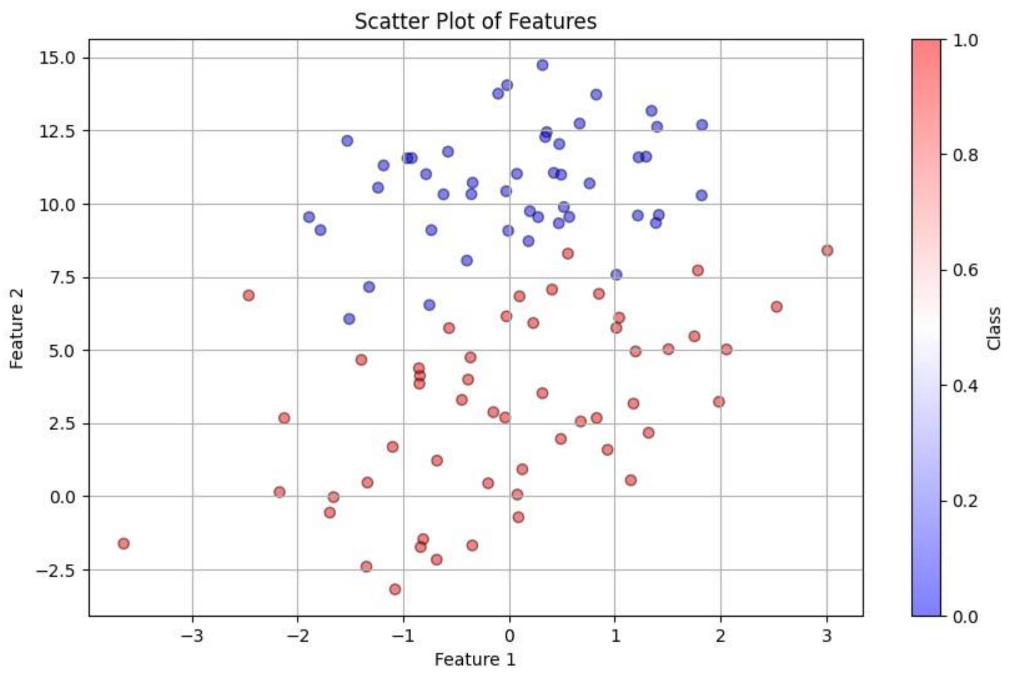
```
    YearsExperience    Salary
0               1.1   39343.0
1               1.3   46205.0
2               1.5   37731.0
3               2.0   43525.0
4               2.2   39891.0
5               2.9   56642.0
6               3.0   60150.0
7               3.2   54445.0
8               3.2   64445.0
9               3.7   57189.0
10              3.9   63218.0
11              4.0   55794.0
12              4.0   56957.0
13              4.1   57081.0
14              4.5   61111.0
15              4.9   67938.0
16              5.1   66029.0
17              5.3   83088.0
18              5.9   81363.0
19              6.0   93940.0
20              6.8   91738.0
21              7.1   98273.0
22              7.9  101302.0
23              8.2  113812.0
24              8.7  109431.0
25              9.0  105582.0
26              9.5  116969.0
27              9.6  112635.0
28             10.3  122391.0
29             10.5  121872.0
```

Question4:

```
import numpy as np
import pandas as pd
from numpy import matrix, zeros
from matplotlib import pyplot as plt

# Load data from file
def loadDataSet():
    X = []
    y = []
    with open('Question4.txt') as f:
        for line in f.readlines():
            nline = line.strip().split()
            X.append([float(nline[0]), float(nline[1])])
            y.append(int(nline[2]))
    return matrix(X).T, matrix(y).T

X, y = loadDataSet()

def sigmoid(t):
    return 1 / (1 + np.exp(-t))

def logistic_regression(X, y, W, b, alpha, iterations):
    m, n = X.shape
    J = zeros((iterations, 1))

    for i in range(iterations):
        # Step 1: Forward propagation
        Z = np.dot(W.T, X) + b
        A = sigmoid(Z)

        # Compute cost function
        #cost = - (1 / m) * np.sum(y * np.log(A) + (1 - y) * np.log(1 - A))
        cost = - (1 / m) * np.sum(np.multiply(y, np.log(A)) + np.multiply(1 - y, np.log(1 - A)))

        J[i] = cost

        # Step 2: Backpropagation
        dZ = A - y
        dW = (1 / m) * np.dot(X, dZ.T)
        db = (1 / m) * np.sum(dZ)

        # Step 3: Gradient descent
        W = W - alpha * dW
        b = b - alpha * db
    return W, b, J

def plotBestFit(X, y, J, W, b):
    # Plot cost function
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(J)
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.title('Cost Function over Iterations')

    # Plot decision boundary
    plt.subplot(1, 2, 2)
    plt.scatter(X[0, y.flatten() == 0], X[1, y.flatten() == 0], color='red', label='Class 0')
    plt.scatter(X[0, y.flatten() == 1], X[1, y.flatten() == 1], color='blue', label='Class 1')

    # Plot decision boundary
    x1_min, x1_max = X[0, :].min(), X[0, :].max()
    x2_min, x2_max = X[1, :].min(), X[1, :].max()
    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max, 100), np.linspace(x2_min, x2_max,
100))
    Z = sigmoid(np.dot(np.c_[xx1.ravel(), xx2.ravel()], W) + b)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap='viridis')

    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Logistic Regression Decision Boundary')
    plt.legend()
    plt.grid(True)
    plt.show()

# Initialize parameters
num = X.shape[0]  # number of features
n = X.shape[1]    # number of samples
W = np.random.randn(num, 1)  # Random initialization of weights
b = np.random.randn()        # Random initialization of bias
alpha = 0.01  # Learning rate
iterations = 1000  # Number of iterations

# Train logistic regression model
W, b, J = logistic_regression(X, y, W, b, alpha, iterations)

# Plot results
plotBestFit(X, y, J, W, b)
```

```
# Experiment with different learning rates
alphas = [0.1, 0.01, 0.001]
plt.figure(figsize=(12, 6))
for alpha in alphas:
    W, b, J = logistic_regression(X, y, W, b, alpha, iterations)
    plt.plot(J, label=f'Alpha = {alpha}')

plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Effect of Learning Rate on Cost Function')
plt.legend()
plt.grid(True)
plt.show()
```

Output:


Scatter Plot of Features
Feature 1
Feature 2
Class