Python
LSH and Matplotlib

Περικλής Ανδρίτσος

A short Matplotlib presentatic..

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

It creates 2D plots from data that is stored in arrays

```
#Install using pip
pip install matplotlib
```

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

Link: https://matplotlib.org/

Source: https://www.bu.edu/tech/files/2017/09/Python-for-Data-Analysis.pptx
https://www.kaggle.com/learn/pandas

A short Matplotlib presentatic..

Matplotlib is a library in Python and it is a numerical – mathematical extension for the NumPy library.

Pyplot is a state-based interface to a **Matplotlib** module which provides a MATLAB-like interface. There are various plots that can be used in Pyplot are Line Plot, Contour, Histogram, Scatter, 3D Plot, etc.

Link: https://matplotlib.org/

Source: https://www.bu.edu/tech/files/2017/09/Python-for-Data-Analysis.pptx
https://www.kaggle.com/learn/pandas

Types of (Main) Plots

- **Bar**: Creates a bar plot
- **Barh**: Creates a horizontal bar plot
- **Boxplot**: Creates a bar and whisker plot
- **Hist**: Creates a histogram
- **Pie**: Creates a pie chart
- **Plot**: Creates lines across the axes
- **Scatter**: Creates a scatter plot

Functions across the axes (x and y)

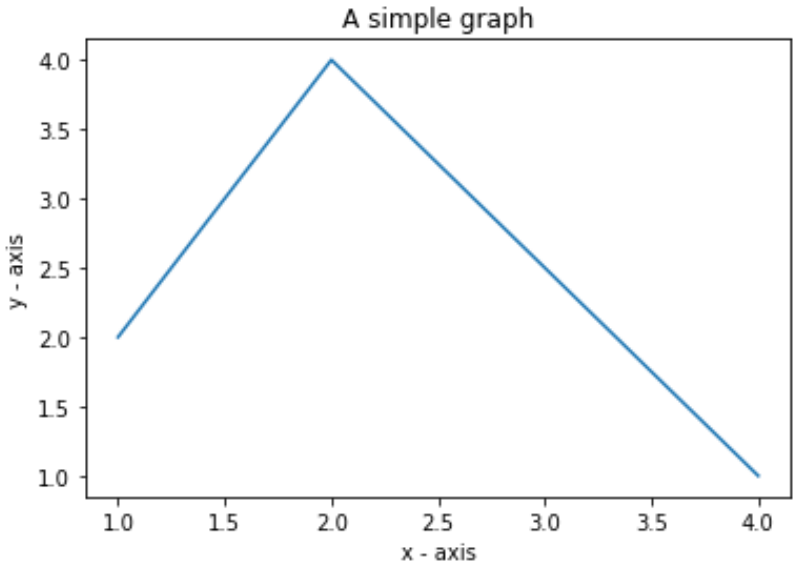
- **Axes**: Adds axes to the plot
- **Text**: Adds text to the plot
- **Title**: Adds a title on the plot
- **Xlabel**: Sets the x axis label
- **Ylabel**: Sets the y axis label
- **Xlim**: Sets the limit of the x axis
- **Xticks**: Sets the x-limits of the tick locations and labels
- **Ylim**: Sets the limit of the y axis
- **Yticks**: Sets the x-limits of the tick locations and labels

Functions on figures

- **Figtext**: Adds text on the figure
- **Figure**: Creates a new figure
- **Show**: Shows a figure
- **Savefig**: Saves the current figure
- **Close**: Closes the current figure

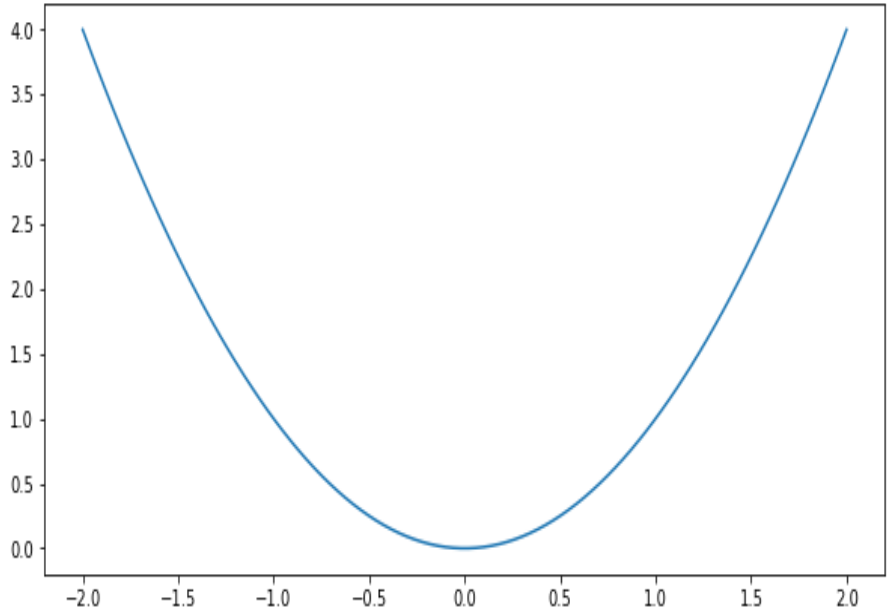
Draw a simple graph

```
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,4]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('Two lines')

# function to show the plot
plt.show()
```



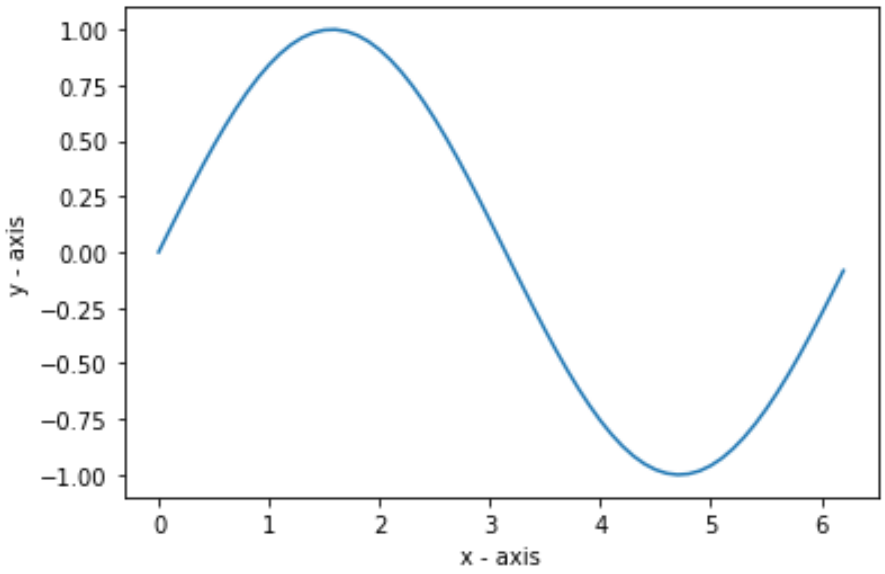
Draw a parabola

```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np

# Creating vectors X and Y
x = np.linspace(-2, 2, 100)
y = x ** 2

fig = plt.figure(figsize = (10, 5))
# Create the plot
plt.plot(x, y)

# Show the plot
plt.show()
```



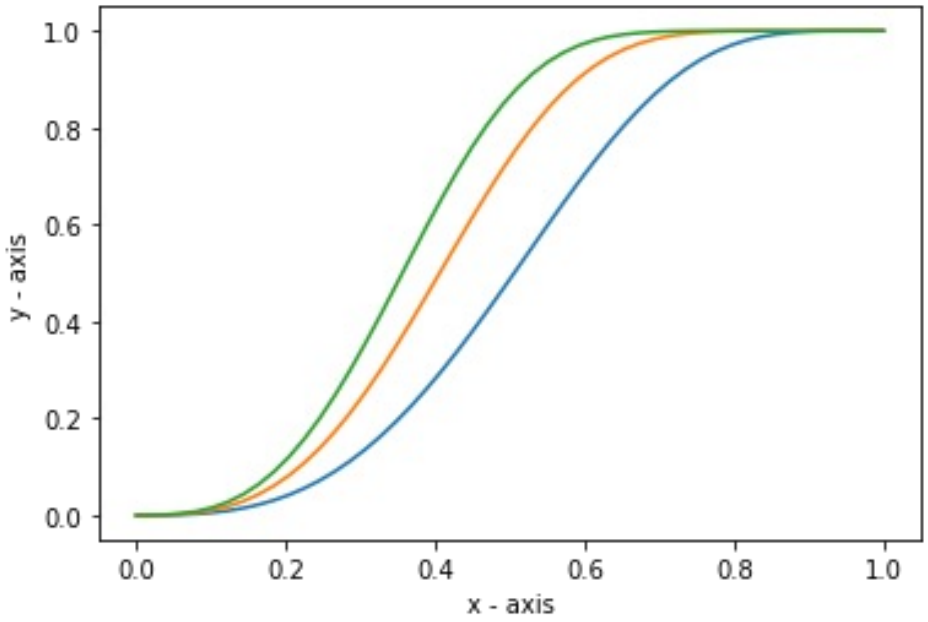
Draw a sine graph

```
# importing the required modules
import matplotlib.pyplot as plt
import numpy as np

# setting the x - coordinates
x = np.arange(0, 2*(np.pi), 0.1)
# setting the corresponding y - coordinates
y = np.sin(x)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# plotting the points
plt.plot(x, y)

# function to show the plot
plt.show()
```



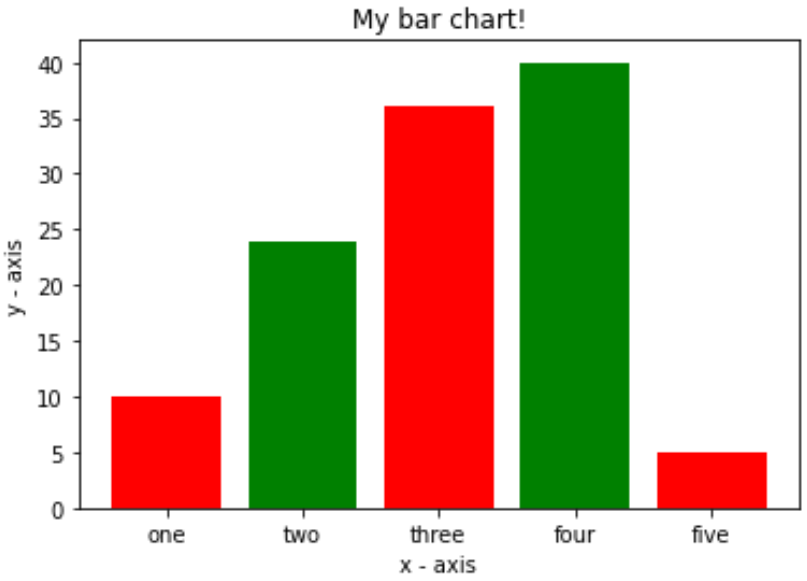
Draw the graph of $P = 1 - (1 - s^r)^b$

```
import matplotlib.pyplot as plt
import numpy as np
b=5
r=3
s = np.linspace(0,1)
f1 = 1-(1-s**r)**b

b=10
f2 = 1-(1-s**r)**b

b=15
f3 = 1-(1-s**r)**b

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

plt.plot(s, f1, s, f2, s, f3)
plt.show()
```



Draw a bar chart

```
import matplotlib.pyplot as plt

# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]

# heights of bars
height = [10, 24, 36, 40, 5]

# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
        width = 0.8, color = ['red', 'green'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')

# function to show the plot
plt.show()
```

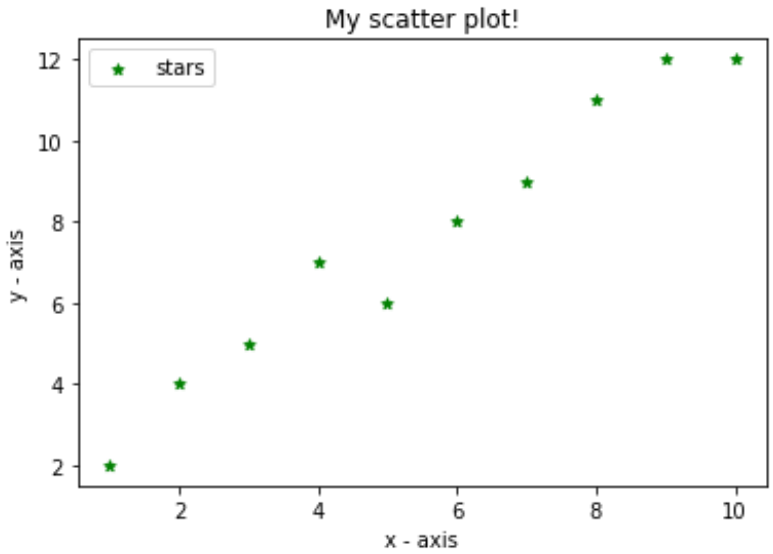


Draw a pie chart

```
import matplotlib.pyplot as plt

# defining labels
activities = ['eat', 'sleep', 'work', 'play']

# portion covered by each label
slices = [3, 7, 8, 6]

# color for each label
colors = ['r', 'y', 'g', 'b']

# plotting the pie chart
plt.pie(slices, labels = activities, colors=colors,
        startangle=90, shadow = True, explode = (0, 0, 0.1, 0),
        radius = 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()

# showing the plot
plt.show()
```

Draw a scatter plot

```
# x-axis values
x = [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = [2,4,5,7,6,8,9,11,12,12]

# plotting points as a scatter plot
plt.scatter(x, y, label= "stars", color= "green",
      marker= "*", s=30)

# x-axis label
plt.xlabel('x - axis')
# frequency label
plt.ylabel('y - axis')
# plot title
plt.title('My scatter plot!')
# showing legend
plt.legend()

# function to show the plot
plt.show()
```



Implementation **of minhashing in Python**

```
# Python numerical library
import numpy as np
import pandas as pd

# Regular expressions library
import re
```

https://www.pinecone.io/learn/locality-sensitive-hashing/

Sample data

```
example_data = {'text': ['machine learning is the future', 'our future cannot be read by a machine', ]}
data = pd.DataFrame(data=example_data)
data
```

	text
0	machine learning is the future
1	our future cannot be read by a machine

Creating shingles of words (1)

```
#Preprocess will split a string of text into individual tokens/shingles based on whitespace.
def preprocess(text):
    text = re.sub(r'[^\w\s]','',text)
    tokens = text.lower()
    tokens = tokens.split()
    return tokens
```

```
preprocess(data.text[0])
```

```
['machine', 'learning', 'is', 'the', 'future']
```

```
preprocess(data.text[1])
```

```
['our', 'future', 'cannot', 'be', 'read', 'by', 'a', 'machine']
```

Creating shingles of words (2)

```
# shingles of length 2
k = 2
kshingles = list()
# add the two pieces of text into a list
docs = []
docs.append(data.text[0])
docs.append(data.text[1])

# create the 2-shingles
for doc in docs:
  split_doc = doc.split(" ")
  temp = set()
  for word in split_doc:
    word_len = len(word)
    for i in range(word_len - k + 1):
      word_slice = word[i:i + k]
      temp.add(word_slice)
  kshingles.append(temp)

print(kshingles)
```

Creating the shingles-by-docs matrix M

```
# Initialize a 2D matrix M of shingles-by-documents size
union_of_shingles = kshingles[0] | kshingles[1]

# Matrix M is filled with zeroes
M = np.zeros((len(union_of_shingles),len(docs)))

# assign a 1 in each cell where a shingle appear in a doc (column)
idx_s = -1
for s in union_of_shingles:
  idx_s += 1
  for d in range(len(docs)):
   if s in kshingles[d]:
     M[idx_s][d] = 1

print(M)
```

Creating the minhash permutations

```
#create N=3 permutations
import random
num_perms = 3
perms = [[]] * num_perms

for i in range(num_perms):
  perms[i] = list(range(0, len(union_of_shingles)))
  random.shuffle(perms[i])

print(perms)
```

Creating the minhash signatures

```
#create the signatures based on the permutations
sigs = np.zeros((num_perms,len(docs)))

for i in range(num_perms):
  for d in range(len(docs)):
    flags = np.zeros((len(docs)))
    for u in range(len(union_of_shingles)):
      # find the index of u inside the permutation (u starts from 1....)
      idx = perms[i].index(u)
      if flags[d] == 0:
        if M[idx][d] == 1:
          sigs[i][d] = u
          flags[d]=1

print(sigs)
```

Exact Jaccard similarity based on matrix M

- Based on the definition in the book, Section 3.3.3

```
# Function of the exact jaccard coefficient based on the original matrix of shingles-by-documents
def exact_jaccard(col1,col2):
  # Type x agreements (1-1 agreements)
  x = 0
  # Type y disagreements (0-1 or 1-0 disageeements)
  y = 0
  for element in range(len(col1)):
    if ( (col1[element]==1) and (col2[element]==1) ):
      x+=1
    if ( ((col1[element]==1) and (col2[element]==0)) or ((col1[element]==0) and (col2[element]==1))):
      y+=1
  # exact jaccard similarity is equal to x/(x+y)
  similarity = x / float(x+y)
  return similarity
```

Approximate Jaccard based on signatures

```
def approx_jaccard(col1, col2):
  # Find the agreements (column elements are the same)
  agree = 0
  # Find the disagreements (column elements are not the same)
  disagree = 0
  for element in range(len(col1)):
    if ( (col1[element]==col2[element]) ):
      agree+=1
    else:
      disagree+=1

  similarity = agree / float(agree+disagree)
  return similarity
```

Calling the Jaccard functions

```
exact_jaccard(M[:,0],M[:,1])
```

```
0.413793103448275 86
```

```
approx_jaccard(sigs[:,0],sigs[:,1])
```

```
0.6666666666666666
```

Python help() Method

The Python help() function invokes the interactive built-in help system. If the argument is a string, then the string is treated as the name of a module, function, class, keyword, or documentation topic, and a help page is printed on the console. If the argument is any other kind of object, a help page on the object is displayed

Python help() Method

```
>>> help('print')
Help on built-in function print in module builtins:

print(...)
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

Useful Links

https://matplotlib.org/

https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/

https://www.pinecone.io/learn/locality-sensitive-hashing/