1DA2223 - Python programming and data analysis

Exercise 4 - Self done tasks [complete 2022L] *Ostatnia modyfikacja: R. Szmurło 22.03.2022 05:53*

NumPy Exercises

Now that we've learned about NumPy let's test your knowledge. We'll start off with a few simple tasks, and then you'll be asked some more complicated questions.

Import NumPy as np

Create an array of 10 zeros

```
array([ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Create an array of 10 ones

```
array([ 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Create an array of 10 fives

```
array([ 5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

Create an array of integers from 10 to 50

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50])
```

Create an array of all the even integers from 10 to 50

```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46,
48, 50])
```

Create a 3x3 matrix with values ranging from 0 to 8

```
array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
```

Create a 3x3 identity matrix

```
array([[ 1., 0., 0.], [ 0., 1., 0.], [ 0., 0., 1.]])
```

Use NumPy to generate a random number between 0 and 1

```
array([ 0.42829726])
```

Use NumPy to generate an array of 25 random numbers sampled from a standard normal distribution

```
array([ 1.32031013,  1.6798602 , -0.42985892, -1.53116655,  0.85753232,
        0.87339938,  0.35668636, -1.47491157,  0.15349697,  0.99530727,
       -0.94865451, -1.69174783,  1.57525349, -0.70615234,  0.10991879,
       -0.49478947,  1.08279872,  0.76488333, -2.3039931 ,  0.35401124,
       -0.45454399, -0.64754649, -0.29391671,  0.02339861,  0.38272124])
```

Create the following matrix:

```
array([[ 0.01,  0.02,  0.03,  0.04,  0.05,  0.06,  0.07,  0.08,  0.09,  0.1 ],
       [ 0.11,  0.12,  0.13,  0.14,  0.15,  0.16,  0.17,  0.18,  0.19,  0.2 ],
       [ 0.21,  0.22,  0.23,  0.24,  0.25,  0.26,  0.27,  0.28,  0.29,  0.3 ],
       [ 0.31,  0.32,  0.33,  0.34,  0.35,  0.36,  0.37,  0.38,  0.39,  0.4 ],
       [ 0.41,  0.42,  0.43,  0.44,  0.45,  0.46,  0.47,  0.48,  0.49,  0.5 ],
       [ 0.51,  0.52,  0.53,  0.54,  0.55,  0.56,  0.57,  0.58,  0.59,  0.6 ],
       [ 0.61,  0.62,  0.63,  0.64,  0.65,  0.66,  0.67,  0.68,  0.69,  0.7 ],
       [ 0.71,  0.72,  0.73,  0.74,  0.75,  0.76,  0.77,  0.78,  0.79,  0.8 ],
       [ 0.81,  0.82,  0.83,  0.84,  0.85,  0.86,  0.87,  0.88,  0.89,  0.9 ],
       [ 0.91,  0.92,  0.93,  0.94,  0.95,  0.96,  0.97,  0.98,  0.99,  1.  ]])
```

Create an array of 20 linearly spaced points between 0 and 1:

```
array([ 0.        ,  0.05263158,  0.10526316,  0.15789474,  0.21052632,
        0.26315789,  0.31578947,  0.36842105,  0.42105263,  0.47368421,
        0.52631579,  0.57894737,  0.63157895,  0.68421053,  0.73684211,
        0.78947368,  0.84210526,  0.89473684,  0.94736842,  1.        ])
```

Numpy Indexing and Selection

Now you will be given a few matrices, and be asked to replicate the resulting matrix outputs:

```
mat = np.arange(1,26).reshape(5,5)
mat
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

WRITE CODE HERE THAT REPRODUCES THE OUTPUT OF THE CELL BELOW BE CAREFUL NOT TO RUN THE CELL BELOW, OTHERWISE YOU WON'T BE ABLE TO SEE THE OUTPUT ANY MORE

a)

```
array([[12, 13, 14, 15],
       [17, 18, 19, 20],
       [22, 23, 24, 25]])
```

b)

```
20
```

c)

```
array([[ 2],
       [ 7],
       [12]])
```

d)

```
array([21, 22, 23, 24, 25])
```

e)

```
array([[16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

Now do the following

Get the sum of all the values in mat

```
325
```

Get the standard deviation of the values in mat

```
7.2111025509279782
```

Get the sum of all the columns in mat

```
array([55, 60, 65, 70, 75])
```

Find median values in all columns

```
[11. 12. 13. 14. 15.]
```

Find average values in all columns

'[11. 12. 13. 14. 15.]'

Find median values in all rows

```
[ 3. 8. 13. 18. 23.]
```

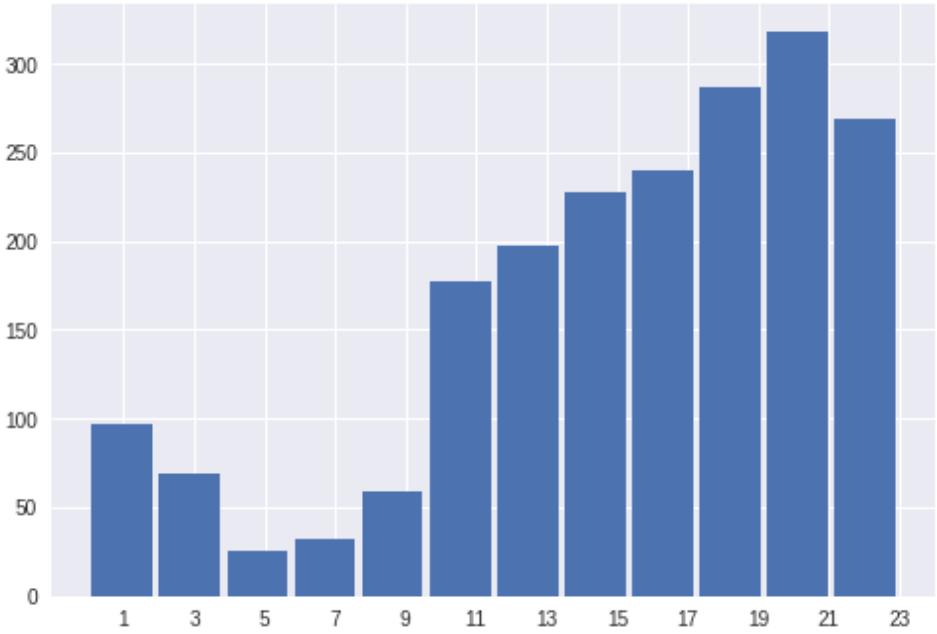
Find average values in all rows

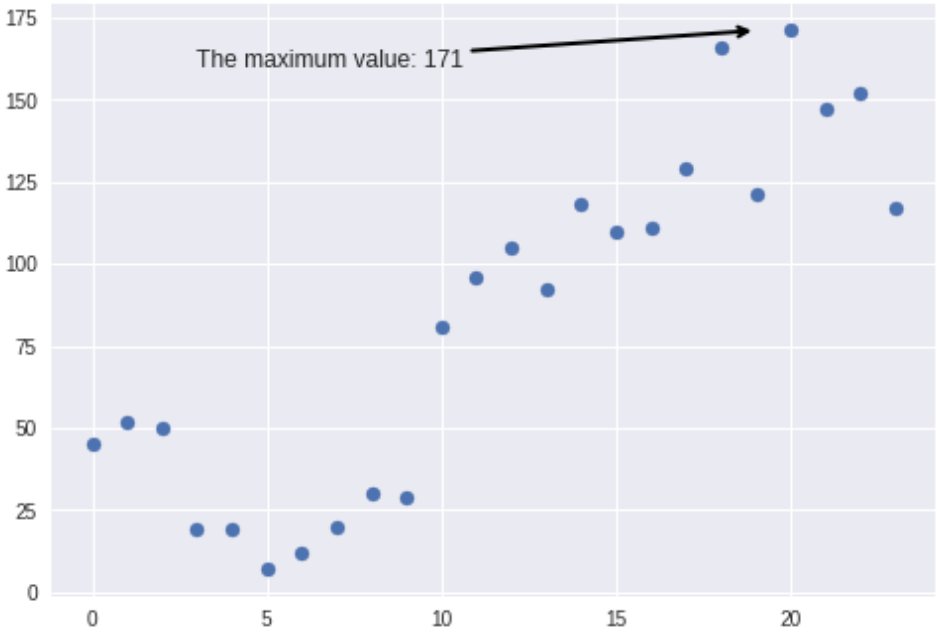
```
[ 3. 8. 13. 18. 23.]
```

Matplotlib exercise

Read the data attached in ISOD about the crimes in Los Angeles from 2010. And generate a histogram analyzing the number of crimes committed along the day. Show the hour of day distribution of crimes using two kinds of plots: histiogram and a scatter plot. (Caution! For scatter plot you will have to calculate number of crime occurrences in each hour.)

Generate the plot similar to:





Useful functions and modules:

- `import csv`

```
import csv
with open('file') as f:
  reader = csv.reader(f, delimiter=',')
  for row in reader:
    print(row)
```

- Assuming that `occurances_hours` is a list with an hour for each occurrence `ax.hist(occurances_hours, rwidth=0.9, bins=12)` , eg.:

```
occurances_hours = [23,24,23,23,1,6,1,...]
```

- you may need to explicitely set xticks with:

```
ax.set_xticks(...)
```

- use collections for counting objects

```
import collections
cnt = collections.Counter(occurances_hours)
```

- use `ax.annotate(...)` to draw a pointer to maximum value

```
ax.annotate(f'The maximum value: {max(y)}', xytext=(3,160),xy=(19, 171),
            arrowprops=dict(arrowstyle="->",lw=2))
```

Implement a Gaussian elimination algorithm with a function taking two arguments: A - the matrix, b - right hand side vector

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

Inside `gaussian` function you should create a copy of original matrix A, and create an augmented matrix of the form:

$$\mathbf{A_{aug}} = [\mathbf{Ab}]$$

and then perform the Gaussian elimination.

```
def gaussian(A,b):
  ...
  return A_aug
```

Implement a backward substitution and find a solution

```
A = np.arange(1, 17, dtype=np.float64).reshape(4,4)
A[1,2] = 88
A[1,3] = -3
A[2,3] = -3
print(f'A = {A}')

x = np.ones(A.shape[0])
```

```
print(f'Original x = {x}')
b = A  @ x.T
print(f'Right hand side for testing: b = {b}')


Ae = gaussian(A, b)
print(f'Check if A was unchanged ')
print(f'Eliminated augmented matrix:\n {Ae}')
print(f'Eliminated augmented matrix A part: {Ae[:,:-1]}')
print(f'Eliminated augmented matrix b part: {Ae[:,Ae.shape[1]-1]}')

# Find solution
x = back(Ae[:,:-1],Ae[:,Ae.shape[1]-1])
print(f'Solution: {x}')
```

Output:

```
A = [[ 1.  2.  3.  4.]
 [ 5.  6. 88. -3.]
 [ 9. 10. 11. -3.]
 [13. 14. 15. 16.]]
Original x = [1. 1. 1. 1.]
Right hand side for testing: b = [10. 96. 27. 58.]
Check if A was unchanged
Eliminated augmented matrix:
 [[   1.     2.     3.     4.    10. ]
 [   0.    -4.    73.   -23.    46. ]
 [   0.     0.  -162.     7.  -155. ]
 [   0.     0.     0.    22.5   22.5]]
Eliminated augmented matrix A part: [[   1.     2.     3.     4. ]
 [   0.    -4.    73.   -23. ]
 [   0.     0.  -162.     7. ]
 [   0.     0.     0.    22.5]]
Eliminated augmented matrix b part: [  10.    46.  -155.    22.5]
Solution: [1. 1. 1. 1.]
```

Great Job!