"From bits to information"

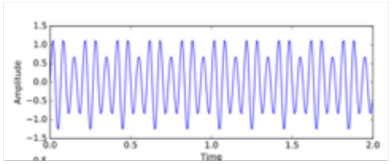
Data to Information: NumPy and Pandas

Outline

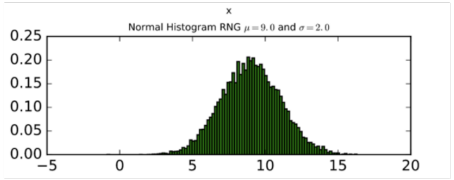
- Data to Information.
- Arrays and Matrices.
- Python: Arrays, Lists and Dictionaries.
- NumPy.
- Pandas.
- Tuples

Data to Information

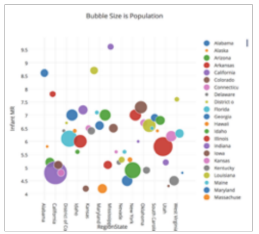
Pie chart



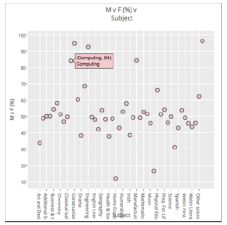
Time based



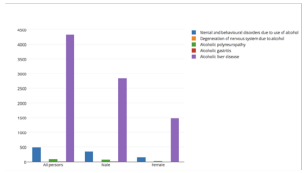
Histogram



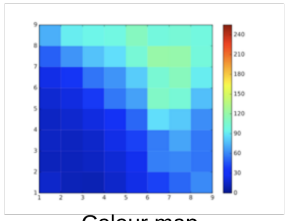
Bubble chart



Scatter graph



Bar chart



Colour map

Charting Tools

Table, Scatter, Bar, Histogram, Colour Map

Data

Integers
Floating point values
Strings
Audio
Video
Images

Hypothesis

Analytic Tools

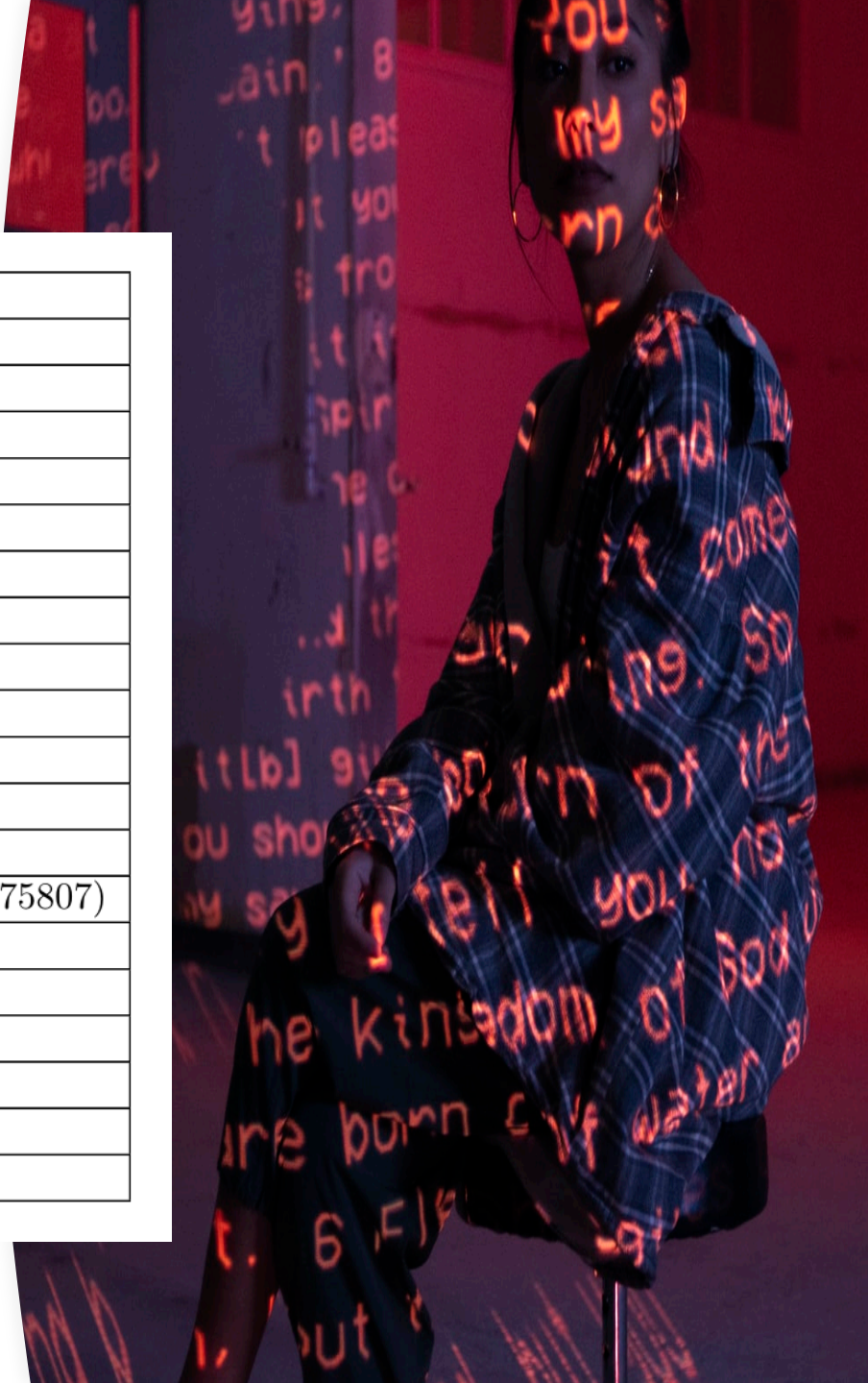
Correlation, Trends, Models

Information

Alerting
Reporting
Auditing
Compliance
Engagement
Scientific Discovery
Influencing

Data formats

Numpy type	C type	Description
np.bool	bool	Boolean (True or False)
np.byte	signed char	8-bit integer value
np.ubyte	unsigned char	8-bit unsigned integer value
np.short	short	16-bit signed integer value
np.ushort	unsigned short	16-bit unsigned integer value
np.intc	int	32-bit signed integer value
np.uintc	unsigned int	32-bit unsigned integer value
np.int_	long	64-bit signed integer value
np.uint	unsigned long	32-bit unsigned integer value
np.int8	int8_t	Byte (-128 to 127)
np.int16	int16_t	Integer (-32768 to 32767)
np.int32	int32_t	Integer (-2147483648 to 2147483647)
np.int64	int64_t	Integer (-9223372036854775808 to 9223372036854775807)
np.uint8	uint8_t	Unsigned integer (0 to 255)
np.uint16	uint16_t	Unsigned integer (0 to 65535)
np.uint32	uint32_t	Unsigned integer (0 to 4294967295)
np.uint64	uint64_t	Unsigned integer (0 to 18446744073709551615)
np.float32	float	32-bit floating point value
np.float64	double	64-bit floating point value.



cyber & data

"From bits to information"

Lists and Tuples

[] ()

Tuples

```
attacks=['DDoS','Malware','Insider','Data Loss']
print (attacks[1])
attacks[1]='Credential theft'
print (attacks[1])
```

Code

```
Malware
Credential theft
```

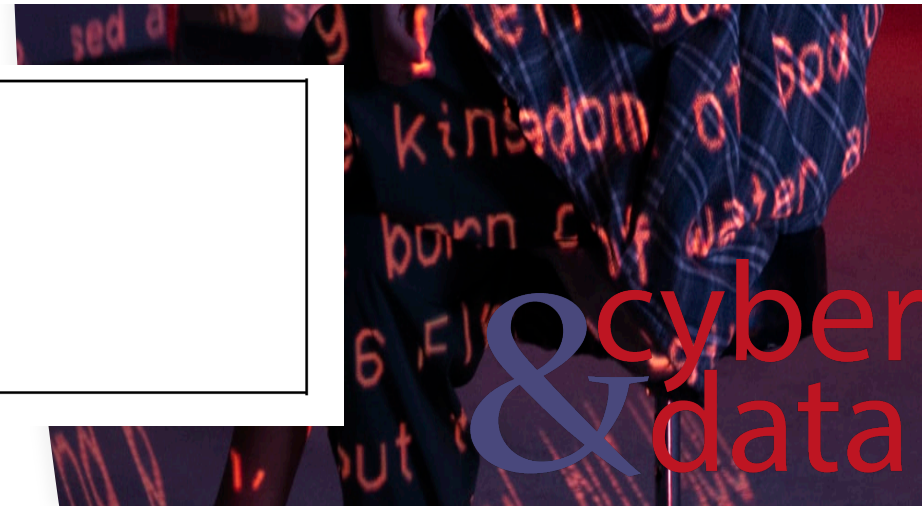
```
attacks=('DDoS','Malware','Insider','Data Loss')
print (attacks[1])
attacks[1]='Credential theft'
print (attacks[1])
```

Code

```
Malware
Traceback (most recent call last):
  File "main.py", line X, in <module>
    attacks[1]='Credential theft'
TypeError: 'tuple' object does not support item assignment
```

```
attacks=('DDoS','Malware','Insider','Data Loss')
temp = list(attacks)
temp[1] = 'Credential theft'
attacks = tuple(temp)
print (attacks)
```

Code



Arrays

$$A_{m,n} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \end{bmatrix}$$

$$A_{m,n} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdot & \end{bmatrix}$$

$$A_{1,3} = \begin{bmatrix} 5 & 7 & 8 \end{bmatrix}$$

$$A_{3,1} = \begin{bmatrix} 5 \\ 7 \\ 8 \end{bmatrix}$$

$$A_{3,3} = \begin{bmatrix} 6 & 2 & 3 \\ 1 & 3 & 5 \\ 5 & 3 & 8 \end{bmatrix}$$

Arrays

$$B = \begin{bmatrix} 5 & 7 & 8 \end{bmatrix}$$

$$C = \begin{bmatrix} 6 & 2 & 3 \\ 1 & 3 & 5 \\ 5 & 3 & 8 \end{bmatrix}$$

$$A = B \times C = \begin{bmatrix} 5 & 7 & 8 \end{bmatrix} \times \begin{bmatrix} 6 & 2 & 3 \\ 1 & 3 & 5 \\ 5 & 3 & 8 \end{bmatrix} = \begin{bmatrix} 5 \times 6 & 7 \times 2 & 8 \times 3 \\ 5 \times 1 & 7 \times 3 & 8 \times 5 \\ 5 \times 5 & 7 \times 3 & 8 \times 8 \end{bmatrix} = \begin{bmatrix} 30 & 14 & 24 \\ 5 & 21 & 40 \\ 25 & 21 & 64 \end{bmatrix}$$

$$A = B \cdot C = \begin{bmatrix} 5 & 7 & 8 \end{bmatrix} \cdot \begin{bmatrix} 6 & 2 & 3 \\ 1 & 3 & 5 \\ 5 & 3 & 8 \end{bmatrix} =$$

$$\begin{bmatrix} 5 \times 6 + 7 \times 2 + 8 \times 3 & 5 \times 1 + 7 \times 3 + 8 \times 5 & 5 \times 5 + 7 \times 3 + 8 \times 8 \end{bmatrix} = \begin{bmatrix} 77 & 55 & 114 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}^T = \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

&cyber data

Lists

Square brackets for a list

List – Can contains strings and numeric values

$$Login_{m,n} = \begin{bmatrix} 192.168.0.1 & 15 & 5 \\ 192.168.0.2 & 20 & 3 \\ 192.168.0.15 & 24 & 7 \end{bmatrix}$$

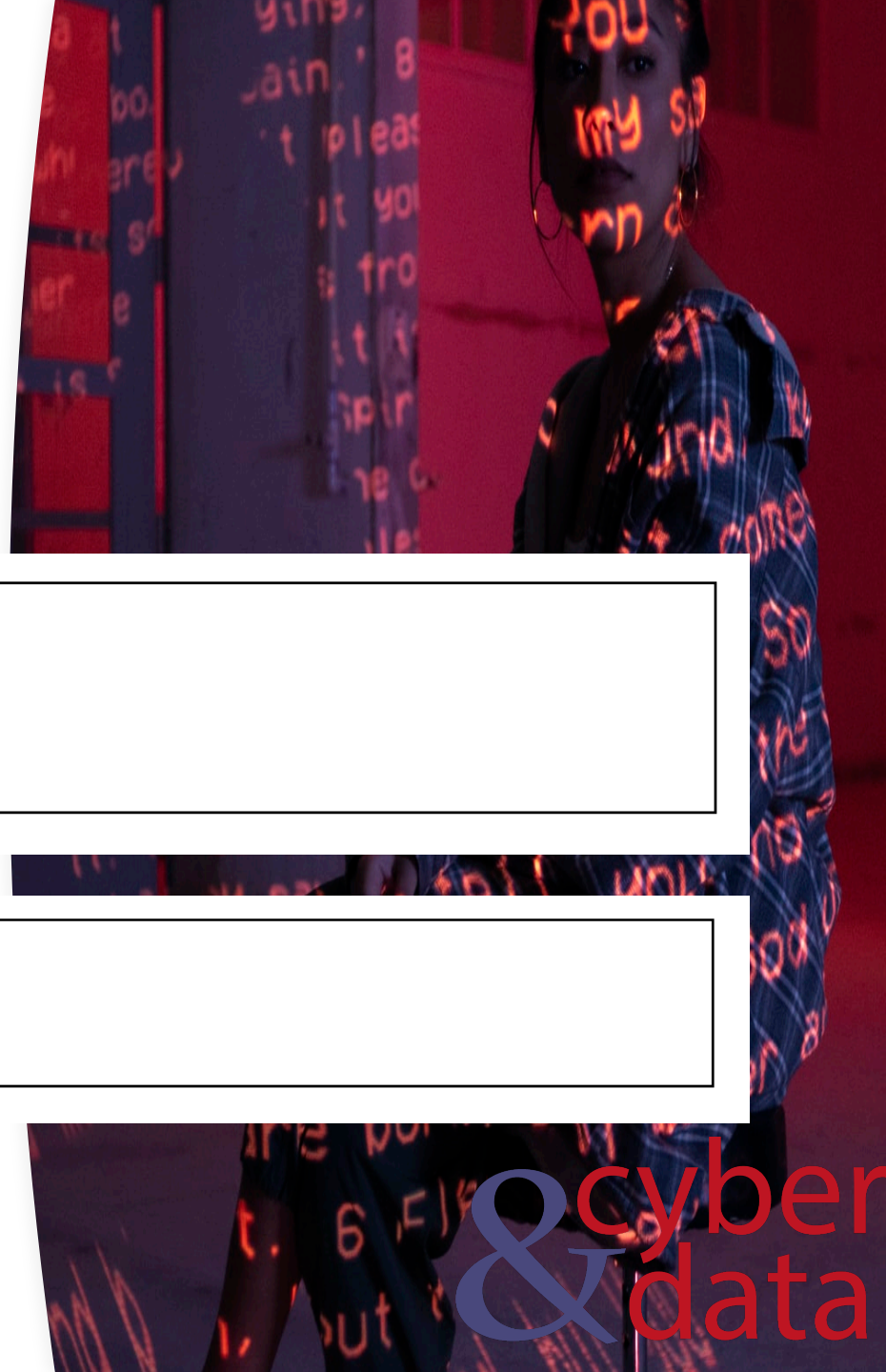
```
login = [10,51,33,44]
print ("Login: ",login)
print ("login[1]: ",login[1])
```

```
login:  [10, 51, 33, 44]
login[1]:  51
```

Code

Tuple: val = (4,5,3)

Immutable – cannot change



Lists

$$Login_{m,n} = \begin{bmatrix} 192.168.0.1 & 15 & 5 \\ 192.168.0.2 & 20 & 3 \\ 192.168.0.15 & 24 & 7 \end{bmatrix}$$

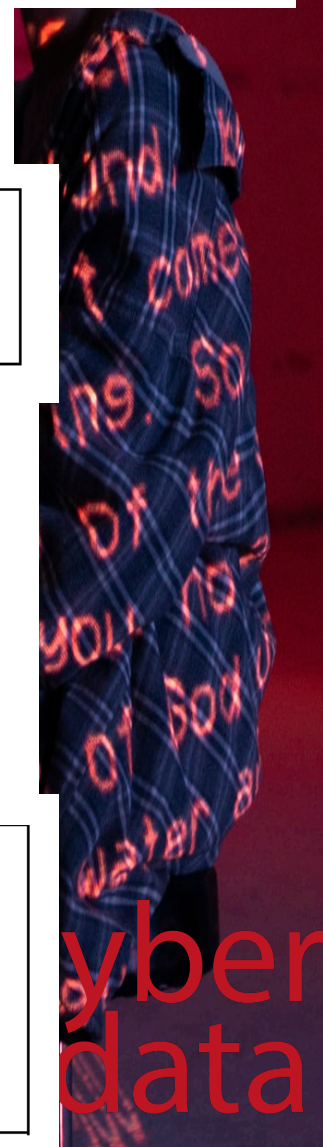
```
login = [10,51,33,44]
print ("Login: ",login)
print ("login[1]: ",login[1])
```

```
login: [10, 51, 33, 44]
login[1]: 51
```

```
login = [[10,51,33,44], [6,3,5,11]]
print ("login: ",login)
print ("login [0]: ",login[0])
print ("login [1]: ",login[1])
print ("login [1][1]: ",login[1][1])
```

Code

```
login: [[10, 51, 33, 44], [6, 3, 5, 11]]
login [0]: [10, 51, 33, 44]
login [1]: [6, 3, 5, 11]
login [1][1]: 3
```



Dictionaries

Dictionaries – accessed by key word

```
login = ['192.168.0.1',15 , 5]
print ("login: ",login)
print ("login [0]: ",login[0])
login.append(10)
print ("login: ",login)
```

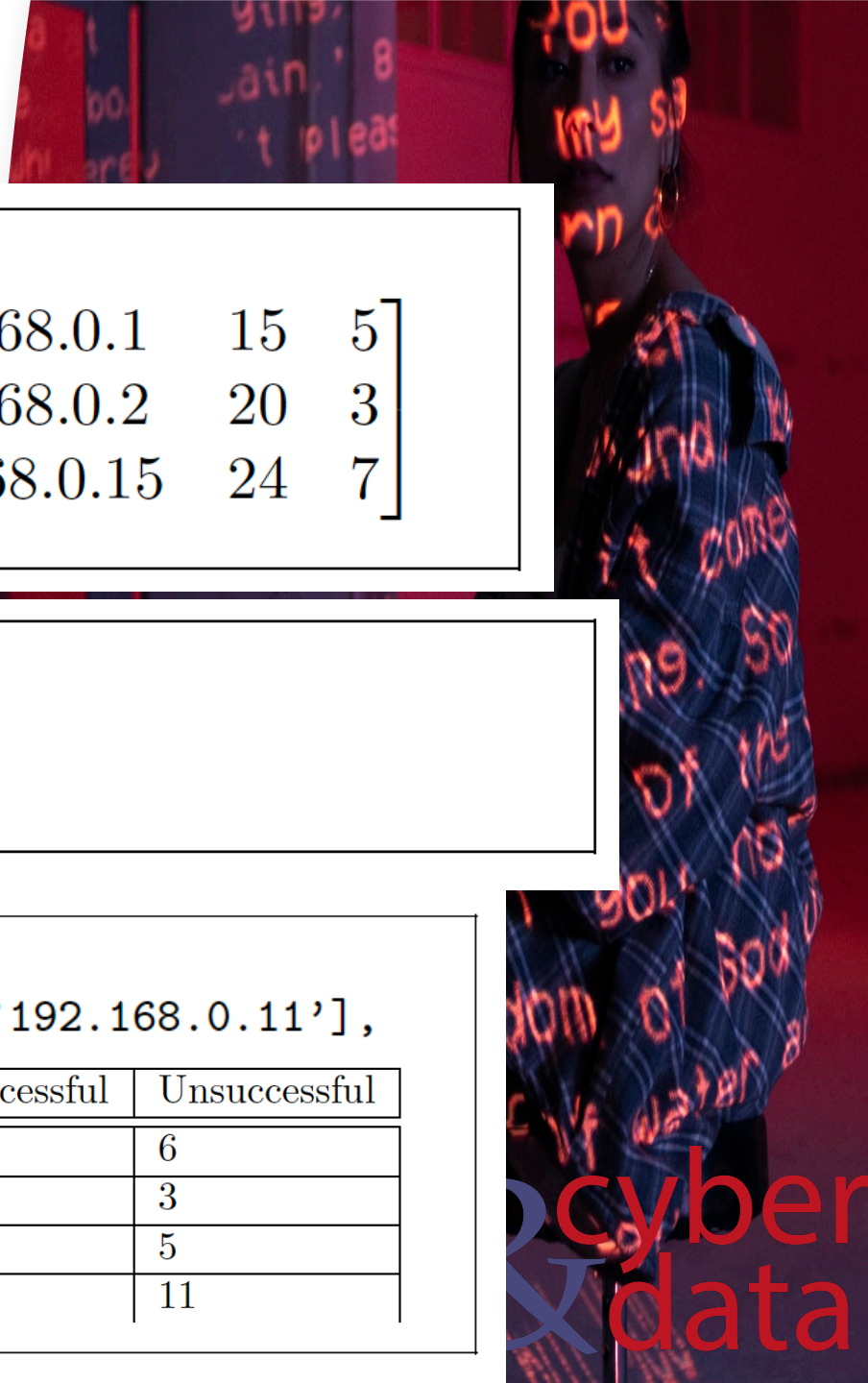
Code

$$Login_{m,n} = \begin{bmatrix} 192.168.0.1 & 15 & 5 \\ 192.168.0.2 & 20 & 3 \\ 192.168.0.15 & 24 & 7 \end{bmatrix}$$

```
login: ['192.168.0.1', 15, 5]
login [0]: 192.168.0.1
login: ['192.168.0.1', 15, 5, 10]
```

```
login = {
 'IP': ['192.168.0.1', '192.168.0.2', '192.168.0.3', '192.168.0.11'],
 'Successful': [10,51,33,44],
 'Unsuccessful': [6,3,5,11]}

print ("Login: ",login)
print ("Login['IP']: ",login['IP'])
```

IP	Successful	Unsuccessful
192.168.0.1	10	6
192.168.0.2	51	3
192.168.0.3	33	5
192.168.0.11	44	11



Dictionaries

```
login = ['192.168.0.1',15 , 5]
print ("login: ",login)
print ("login [0]: ",login[0])
login.append(10)
print ("login: ",login)
```

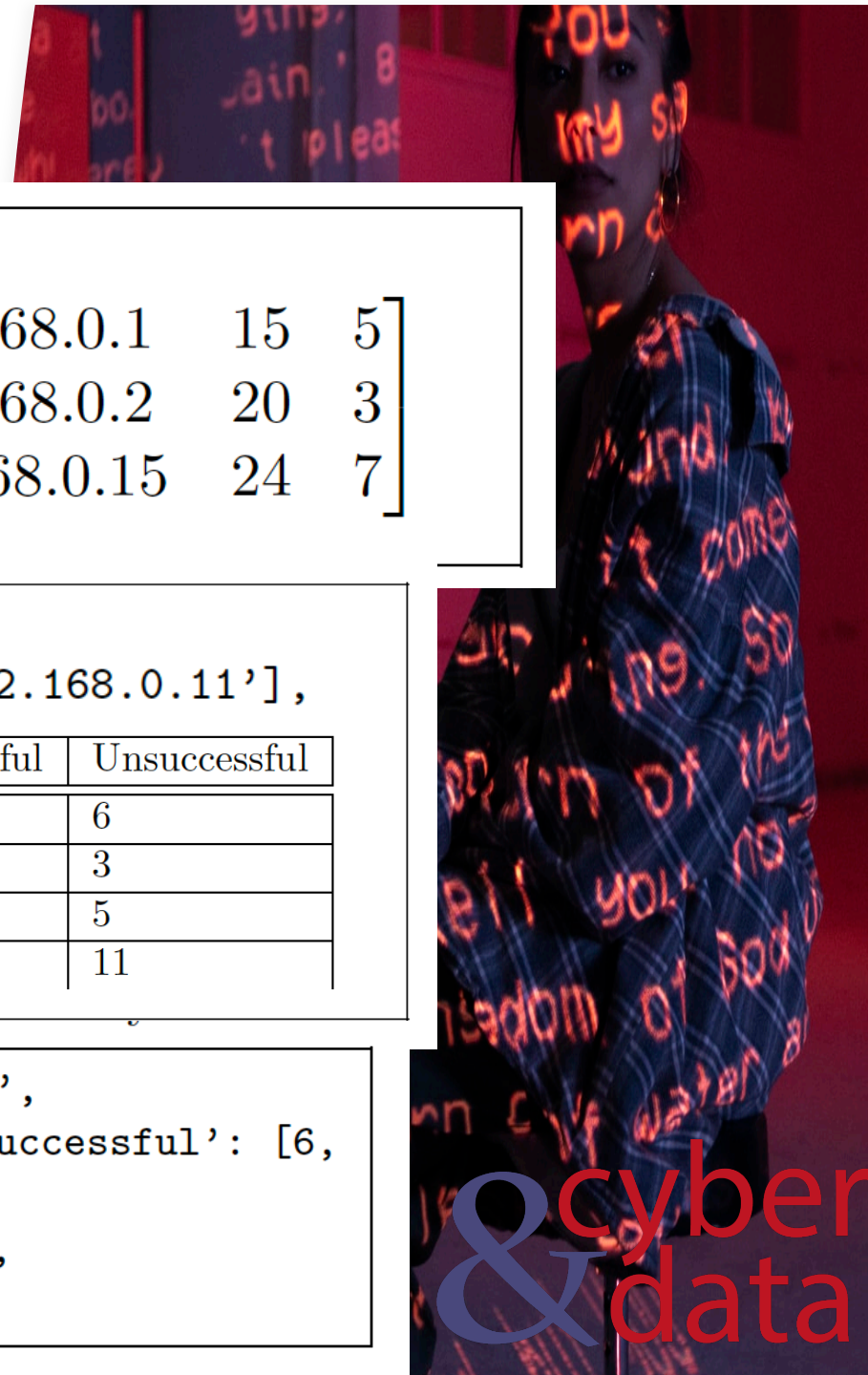
Code

$$Login_{m,n} = \begin{bmatrix} 192.168.0.1 & 15 & 5 \\ 192.168.0.2 & 20 & 3 \\ 192.168.0.15 & 24 & 7 \end{bmatrix}$$

```
login = {
 'IP': ['192.168.0.1', '192.168.0.2', '192.168.0.3', '192.168.0.11'],
 'Successful': [10,51,33,44],
 'Unsuccessful': [6,3,5,11]}

print ("Login: ",login)
print ("Login['IP']: ",login['IP'])
```

IP	Successful	Unsuccessful
192.168.0.1	10	6
192.168.0.2	51	3
192.168.0.3	33	5
192.168.0.11	44	11

```
Login: {'IP': ['192.168.0.1', '192.168.0.2', '192.168.0.3',
    '192.168.0.11'], 'Successful': [10, 51, 33, 44], 'Unsuccessful': [6,
     3, 5, 11]}
Login['IP']: ['192.168.0.1', '192.168.0.2', '192.168.0.3',
    '192.168.0.11']
```

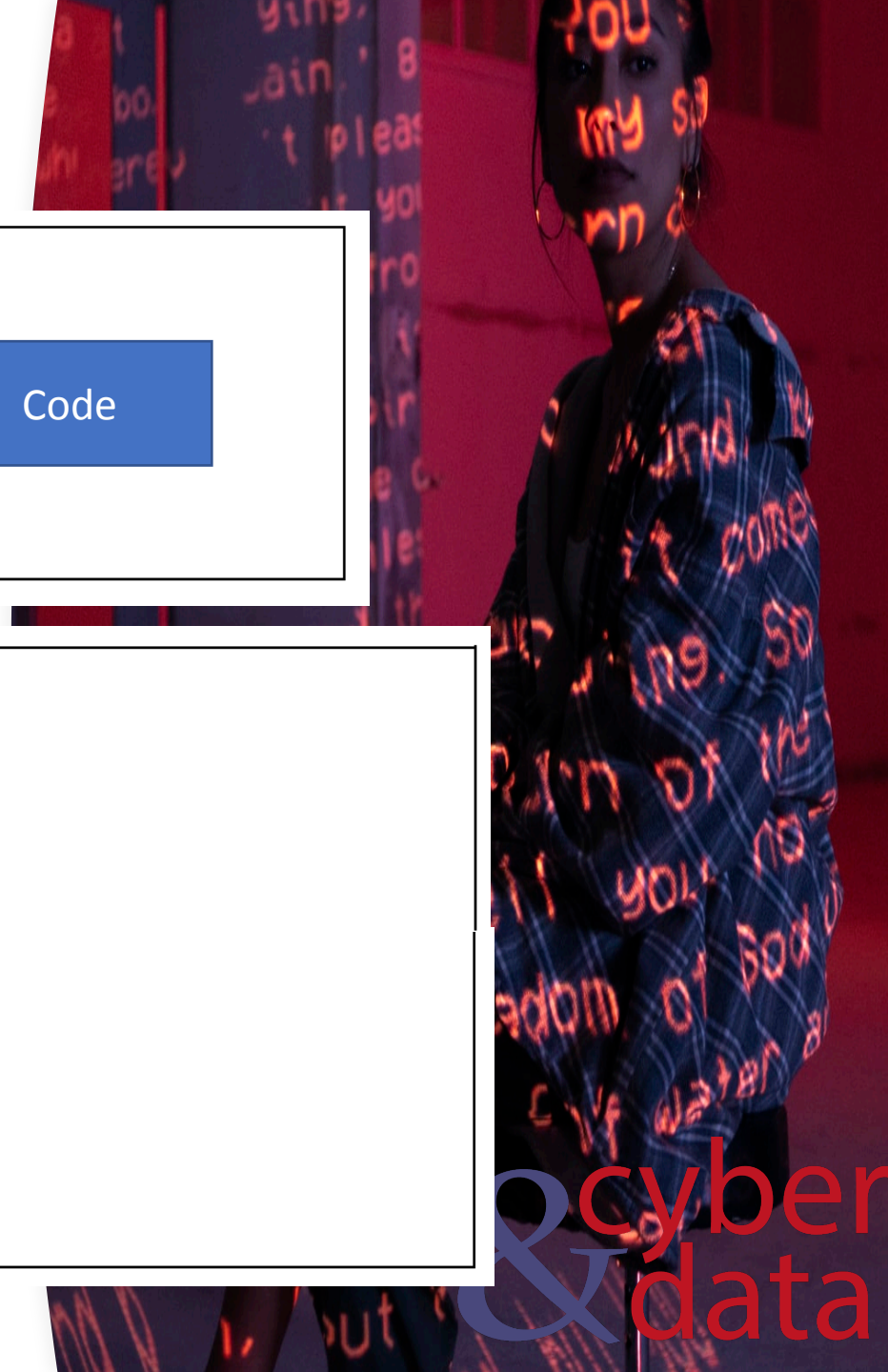


Arrays, Lists and Dictionaries: Using NumPy

```
login = np.array([[10,51,33,44],[6,3,5,11]])
print ("login: ",login)
login = np.append(login,[[1,13,13,41],[3,10,11,40]])
print ("Added row for login: ",login)
login = login.transpose()
print ("Transpose login: ",login)
```

Code

```
login:  [[10 51 33 44]
 [ 6  3  5 11]]
Added row for login:
[[10 51 33 44]
 [ 6  3  5 11]
 [ 1 13 13 41]
 [ 3 10 11 40]]
Transpose login:  [[10  6  1  3]
 [51  3 13 10]
 [33  5 13 11]
 [44 11 41 40]]
```

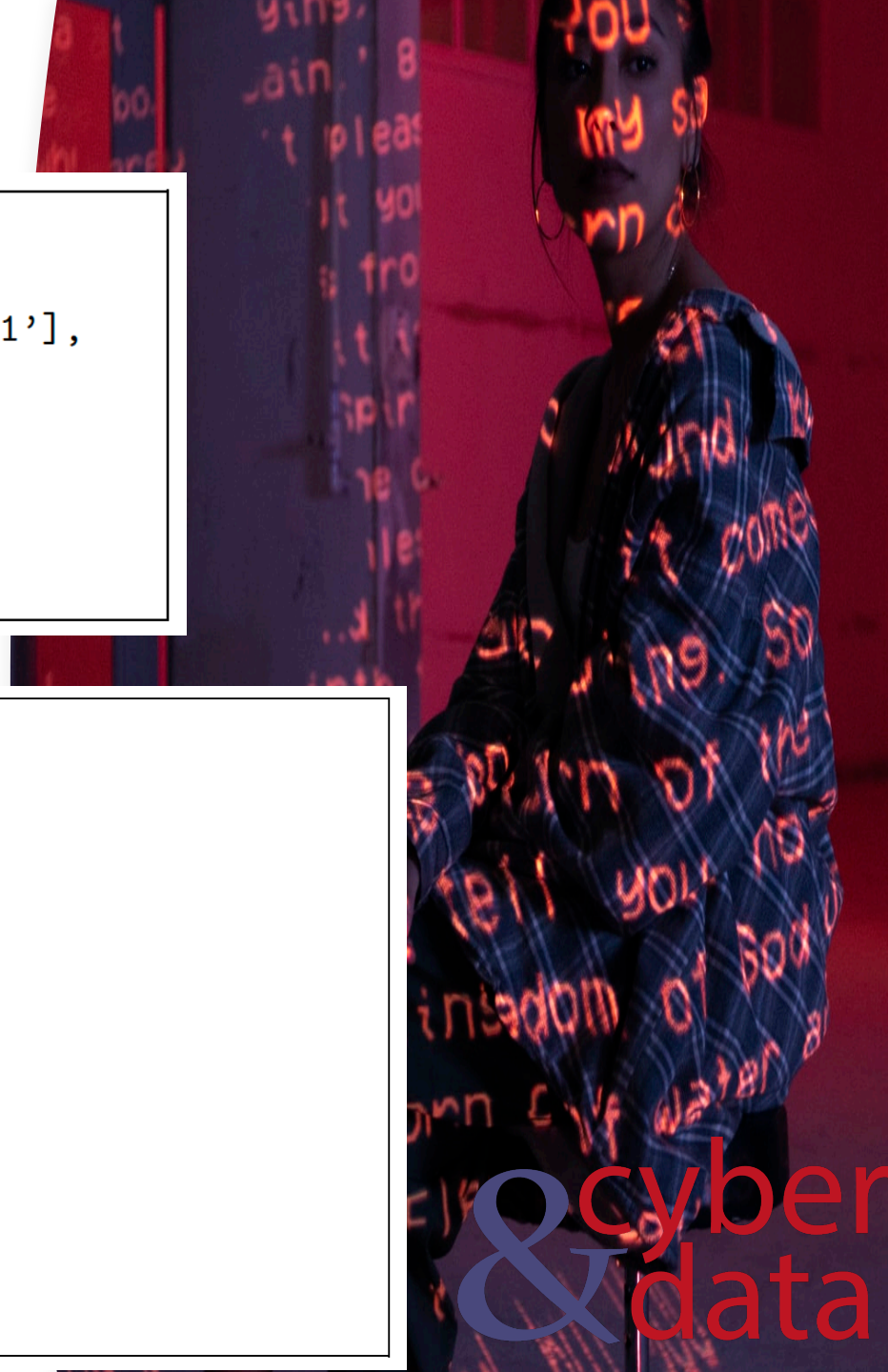


Arrays, Lists and Dictionaries: Using a Dataframe

```
import pandas as pd
login = {
  'IP': ['192.168.0.1', '192.168.0.2', '192.168.0.3', '192.168.0.11'],
  'Successful': [10,51,33,44],
  'Unsuccessful': [6,3,5,11]}
df = pd.DataFrame(login)
print ("Login:\n",df)
print ("Login['IP']:\n",df['IP'])
```

Code

```
Login:
             IP  Successful  Unsuccessful
0   192.168.0.1          10             6
1   192.168.0.2          51             3
2   192.168.0.3          33             5
3  192.168.0.11          44            11
Login['IP']:
 0     192.168.0.1
1     192.168.0.2
2     192.168.0.3
3    192.168.0.11
Name: IP, dtype: object
After insert:
             IP  Successful  Unsuccessful Owner
0   192.168.0.1          10             6  carol
1   192.168.0.2          51             3    bob
2   192.168.0.3          33             5  alice
3  192.168.0.11          44            11    eve
```



cyber & data

"From bits to information"

NumPy

NumPy

```
 import numpy as np
a = np.array([5, 19, 32,20,10])
print (a)
print (np.arange( 10, 30, 3 )) # start, end, increment
print (np.linspace(1,10,num=10)) # Return evenly spaced numbers over a
    specified interval.
```

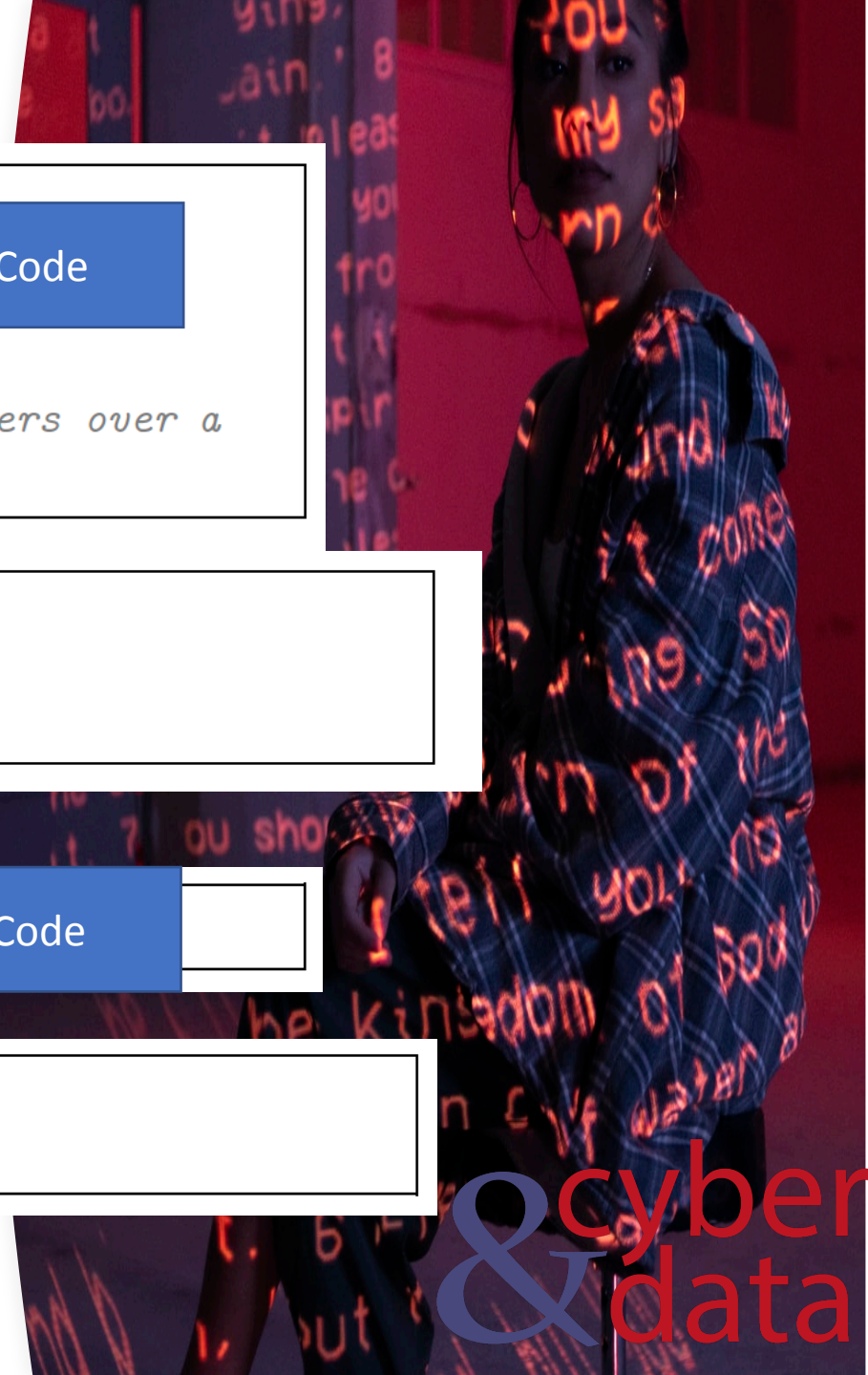
Code

```
[ 5 19 32 20 10]
[10 13 16 19 22 25 28]
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

```
print (np.random.rand(2,3))
```

Code

```
[[0.92000513 0.72482518 0.65812391]
 [0.28474997 0.79968384 0.11472591]]
```



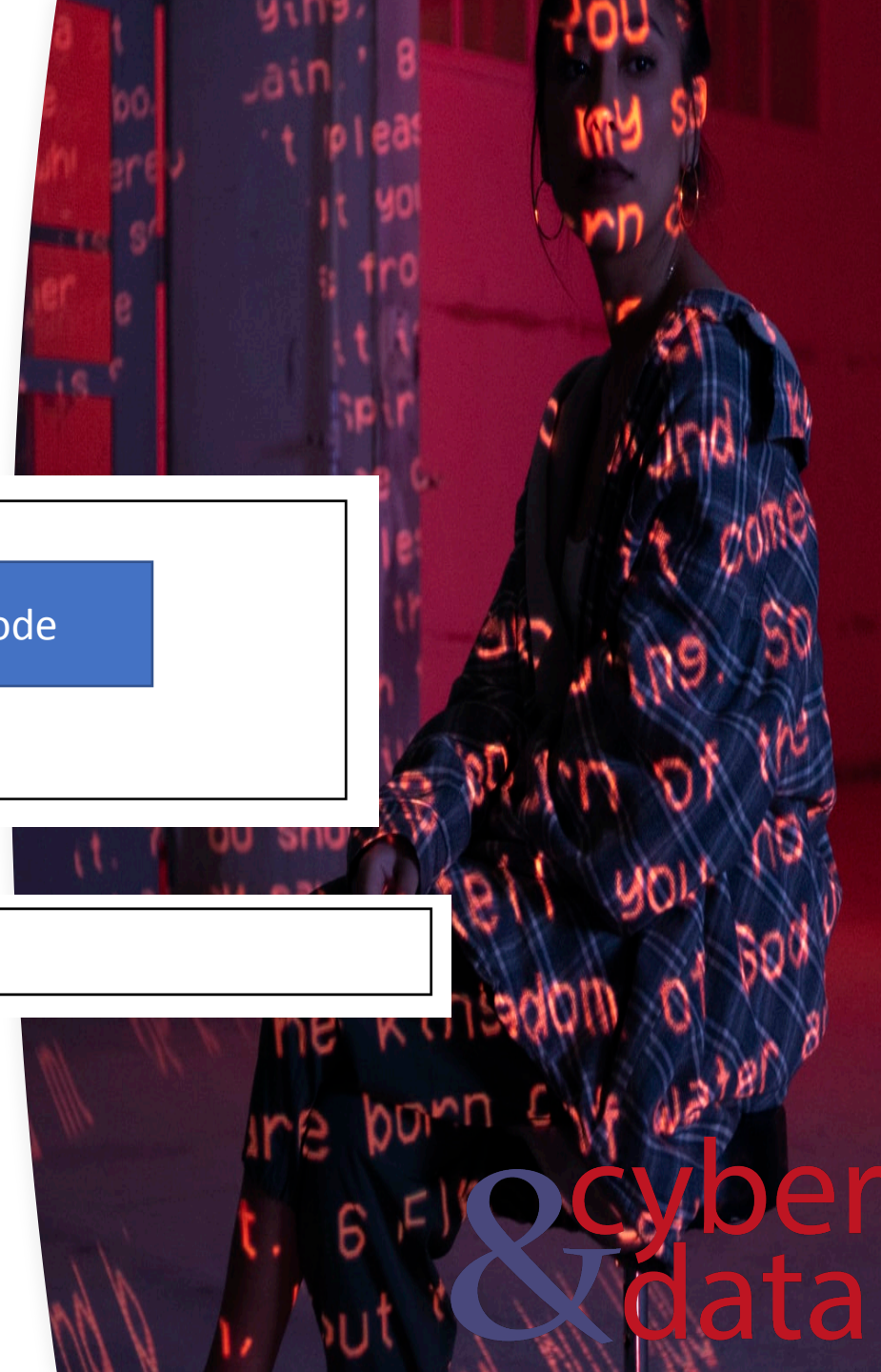
NumPy: Linear Equation Solving

$$5x + 3y = 18$$
$$8x - 2y = 22$$

```
import numpy as np
coeffs = np.array([[5, 3], [8, -2]])
depvars = np.array([18, 22])
solution = np.linalg.solve(coeffs, depvars)
print (solution)
```

Code

```
[3. 1.]
```



NumPy Array Operations

```
a = np.array([5, 9, 7, 1, 8, 3, 2, 6, 10, 9, 4, 2])
a[1:3] = 0
print (a)
```

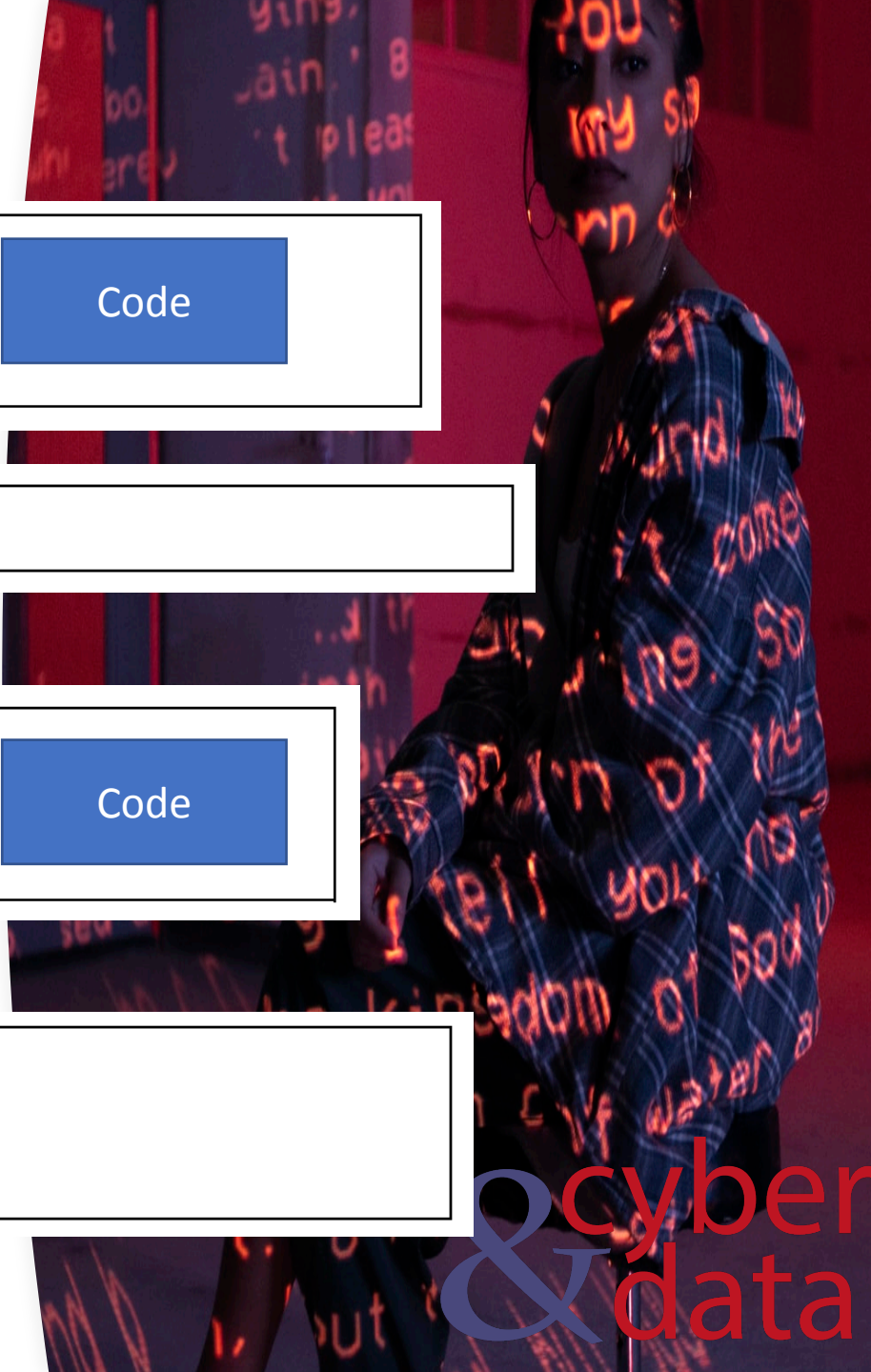
Code

```
[ 5  0  0  1  8  3  2  6 10  9  4  2]
```

```
a = np.array([5, 9, 7, 1, 8, 3, 2, 6, 10, 9, 4, 2])
a = a.reshape(3, 4)
print (a)
```

Code

```
[[ 5  9  7  1]
 [ 8  3  2  6]
 [10  9  4  2]]
```



NumPy Maths Operations

```
a = np.array([5, 9, 7, 1, 8, 3, 2, 6, 10, 9, 4, 2])
val1 = np.sum(a)
val2 = np.average(a)
val3 = np.std(a)
val4 = np.log(10)
val5 = np.sin(np.pi/4)
val6 = np.sort(a)
val7 = np.ceil(9.5)
val8 = np.floor(9.5)
print ("Sum: ",val1)
print ("Average: ",val2)
print ("Std Dev: ",val3)
print ("Log(10)=",val4)
print ("Sine(10)=",val5)
print ("Sort: ",val6)
print ("Ceil(9.5)=",val7)
print ("Floor(9.5)=",val8)
```

Code

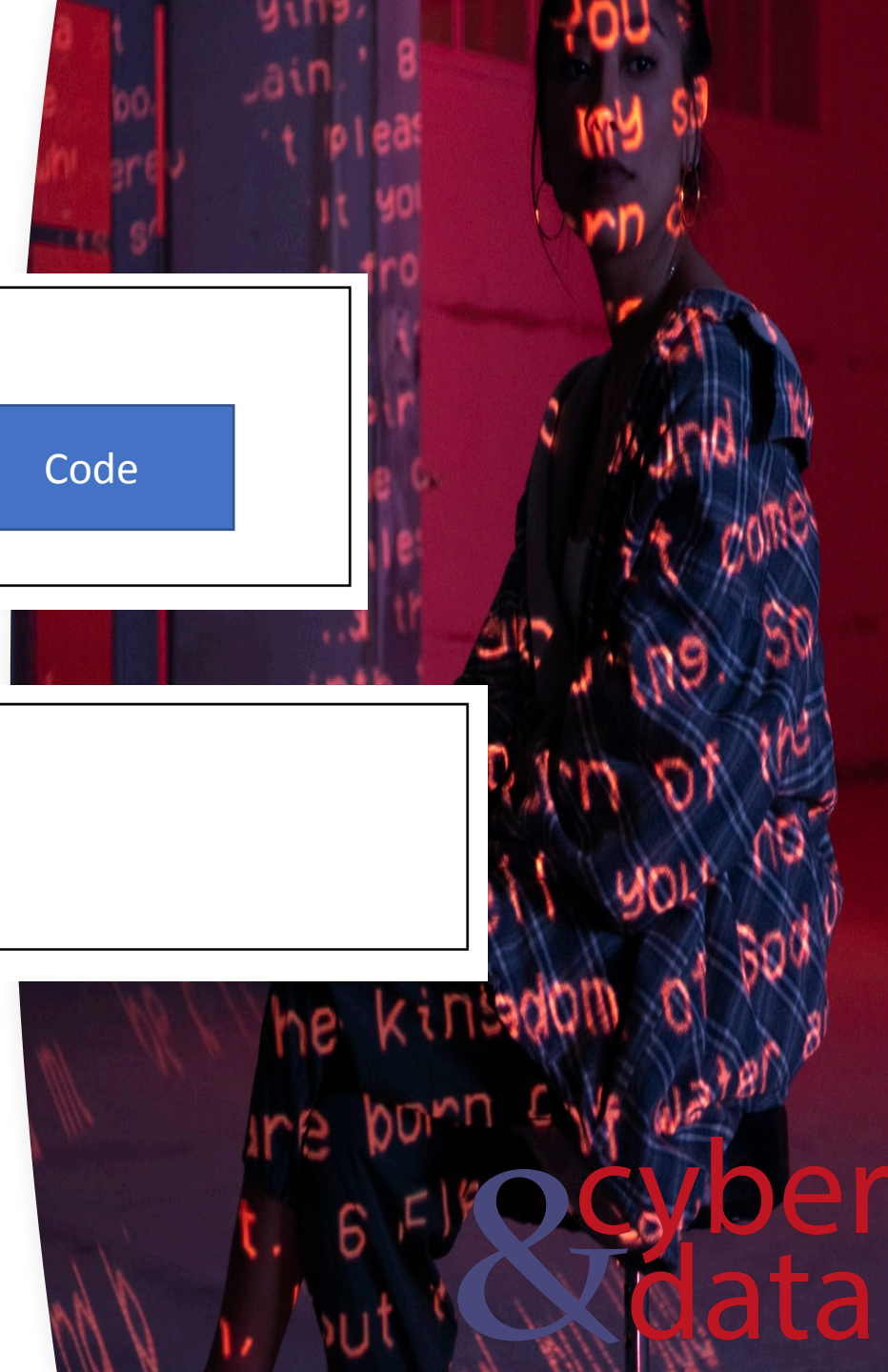

```
Sum:  66
Average: 5.5
Std Dev: 2.9860788111948193
Log(10)= 2.302585092994046
Sine(10)= 0.7071067811865475
Sort:  [ 1  2  2  3  4  5  6  7  8  9  9 10]
Ceil(9.5)= 10.0
Floor(9.5)= 9.0
```

NumPy Array Operations

```
a = np.array([5, 9, 7, 1, 8, 3, 2, 6, 10, 9, 4, 2])
print ("2*a=",2*a)
print ("3+a=",3+a)
print ("a-4=",a-4)
print ("a/2=",a/2)
```

Code

```
2*a= [10 18 14 2 16 6  4 12 20 18 8  4]
3+a= [ 8 12 10 4 11 6  5  9 13 12 7  5]
a-4= [ 1 5  3 -3 4 -1 -2 2  6  5  0 -2]
a/2= [2.5 4.5 3.5 0.5 4. 1.5 1. 3. 5. 4.5 2. 1. ]
```



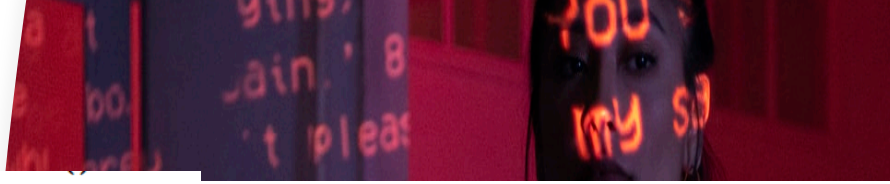
NumPy: Matrix Operations

```
import numpy as np

g=[5,7,8]
h=[[6,2,3],[1,3,5],[5,3,8]]

def multi(m, g):
    en = np.multiply(m, g)
    return en

def dot(m, g):
    en = np.dot(m, g)
    return en
print ("Input:\n",g)
print ("Input:\n",h)

print ("-----------")

res=multi(g,h)
print ("Multiply:")
print (res)

res=dot(g,h)
print ("Dot:")
```

```
Input:
 [5, 7, 8]
Input:
 [[6, 2, 3], [1, 3, 5], [5, 3, 8]]
-----------
Multiply:
[[30 14 24]
 [ 5 21 40]
 [25 21 64]]
Dot:
[ 77  55 114]
```

Code

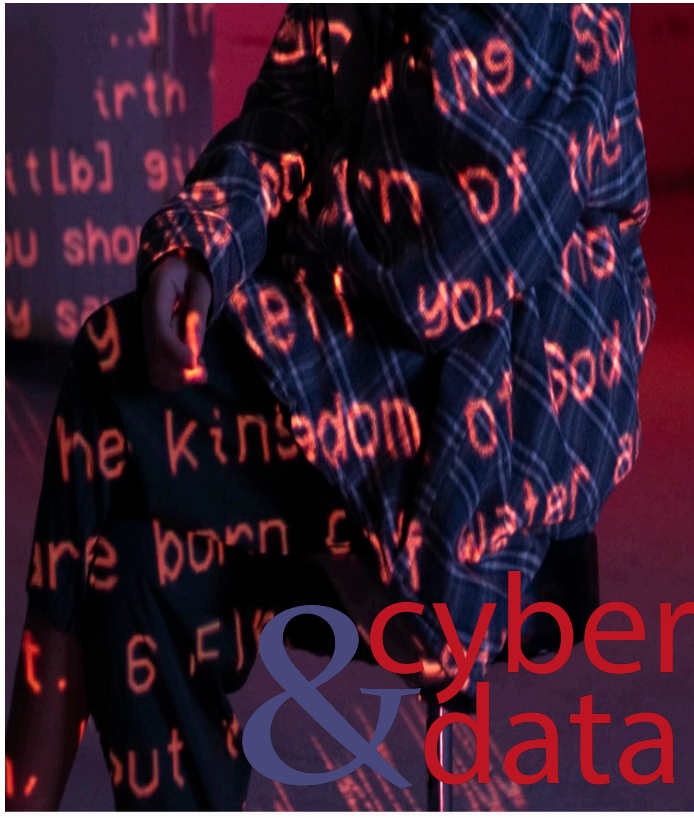
NumPy: Bitwise



```
import numpy as np
a = 11
b = 30

print("binary representation of a:",bin(a))
print("binary representation of b:",bin(b))

print("Bitwise a:\t\t\t\t",np.binary_repr(a,width=8 ))
print("Bitwise b:\t\t\t\t",np.binary_repr(b,width=8 ))
print("Bitwise Invert of a:\t",np.binary_repr(np.invert(a),width=8 ))
print("Bitwise Invert of b:\t",np.binary_repr(np.invert(b),width=8 ))

print("Bitwise AND of a and b:\t",np.binary_repr(np.bitwise_and(a,b),
    width=8 ) )
print("Bitwise OR of a and b:\t",np.binary_repr(np.bitwise_or(a,b),width
    =8 ))
print("Bitwise XOR of a and b:\t",np.binary_repr(np.bitwise_xor(a,b),
    width=8 ))
print("Bitwise left shift of a by 2:\t",np.binary_repr(np.left_shift(a,2)
    ,width=8 ))
print("Bitwise right shift of a by 2:\t",np.binary_repr(np.right_shift(a
    ,2),width=8 ))
```

Code

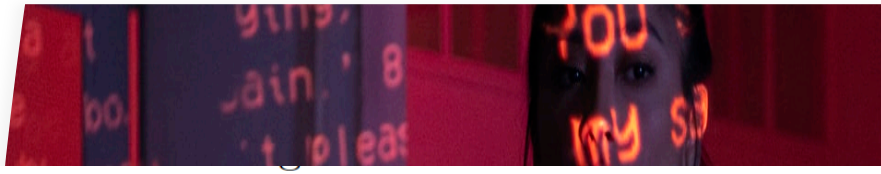
```
binary representation of a: 0b1011
binary representation of b: 0b11110
Bitwise-and of a and b: 0b1010
Bitwise-and of a and b: 0b11111
Bitwise-and of a and b: 0b10101
Bitwise-and of a and b: 00001010
Bitwise-and of a and b: 00011111
Bitwise-and of a and b: 00010101
```



"From bits to information"

Pandas

Pandas

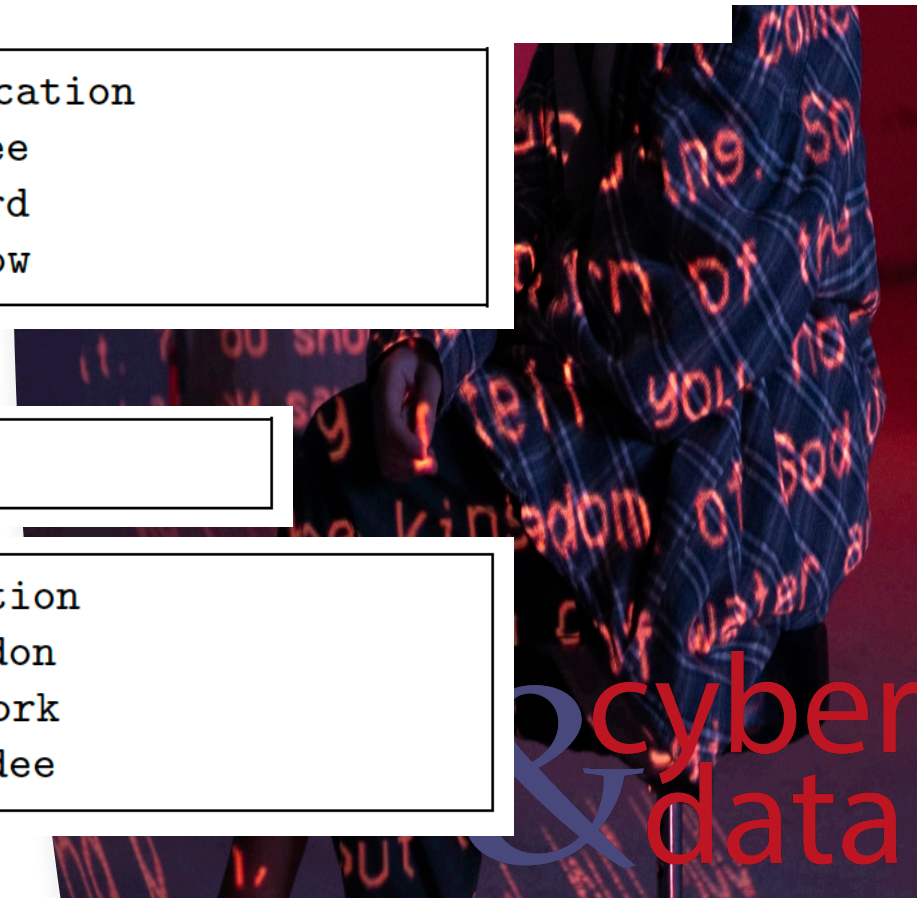


Code

```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
IP Address        Good login Bad login CPU Average Location
2  192.168.0.10         10         21        44.3   Dundee
3  192.168.0.11         15         16         4.5   Oxford
4  192.168.0.12         17          1         2.3  Glasgow
```




```
print (ver.tail(3))
```

```
     IP Address Good login Bad login CPU Average Location
0   192.168.0.1         50         20        33.2   London
1   192.168.0.3         17          5        10.1 New York
2  192.168.0.10         10         21        44.3   Dundee
```

Pandas: Data Types

```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
print (ver.dtypes)
```

```
IP Address     object
Good login      int64
Bad login       int64
CPU Average   float64
Location       object
```

Pandas: Len() and Columns

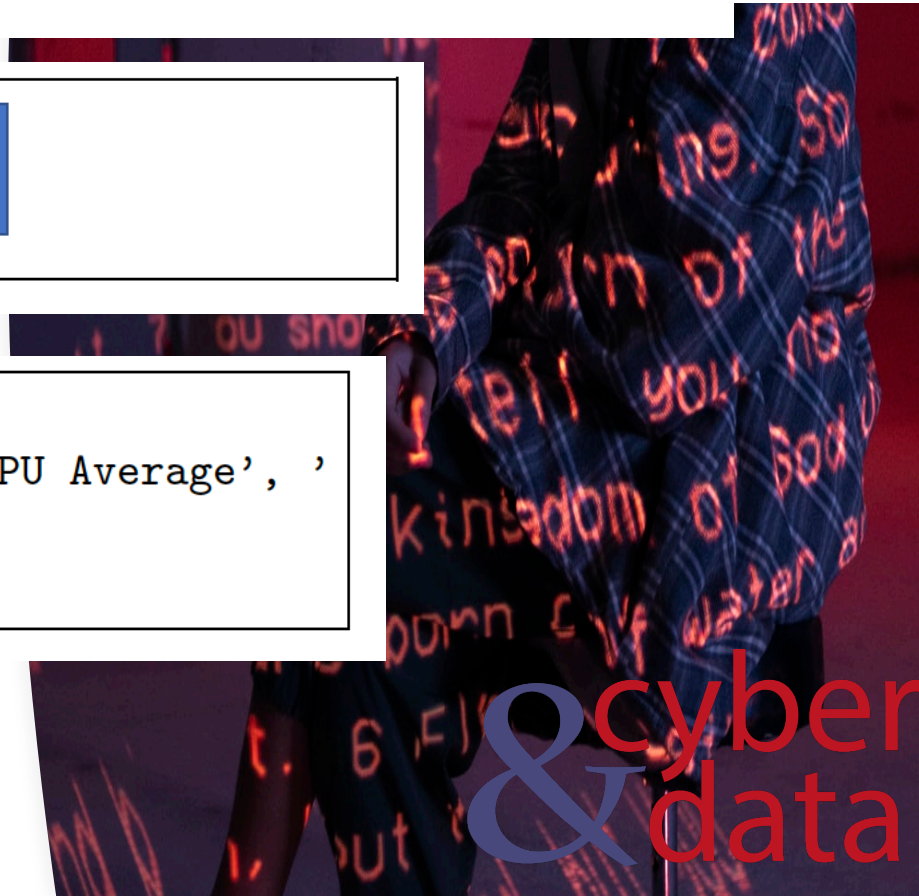
```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
print ("Length of dataframe: ",len(ver))
print ("Columns: ",ver.columns)
print (ver.dtypes)
```

Code

```
Length of dataframe: 5
Columns: Index(['IP Address', 'Good login', 'Bad login', 'CPU Average', '
    Location'], dtype=
'object')
```



Pandas: Columns

```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
print (ver['Good login'])
```

Code

```
0    50
1    17
2    10
3    15
4    17
Name: Good login, dtype: int64
```

Pandas: Describe()

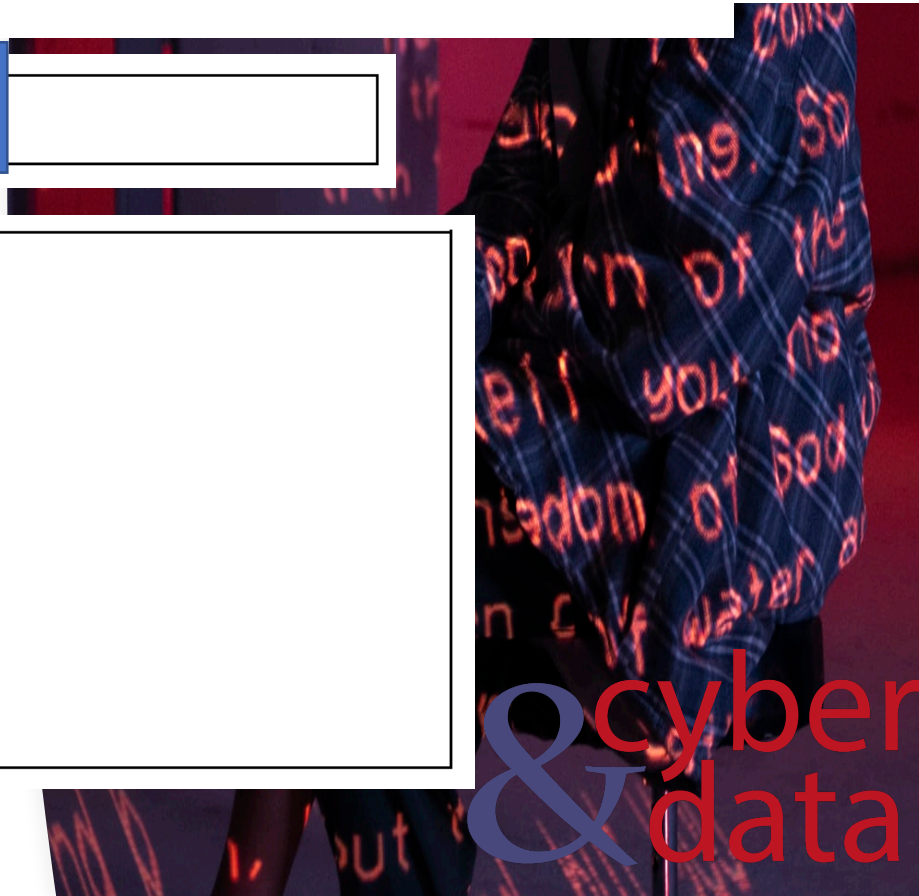
```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
print (ver.describe())
```

Code

```
       Good login  Bad login  CPU Average
count     5.00000   5.000000     5.000000
mean     21.80000  12.600000    18.880000
std      16.02186   9.071935    18.775037
min      10.00000   1.000000     2.300000
25%      15.00000   5.000000     4.500000
50%      17.00000  16.000000    10.100000
75%      17.00000  20.000000    33.200000
max      50.00000  21.000000    44.300000
```



Pandas: Sorting

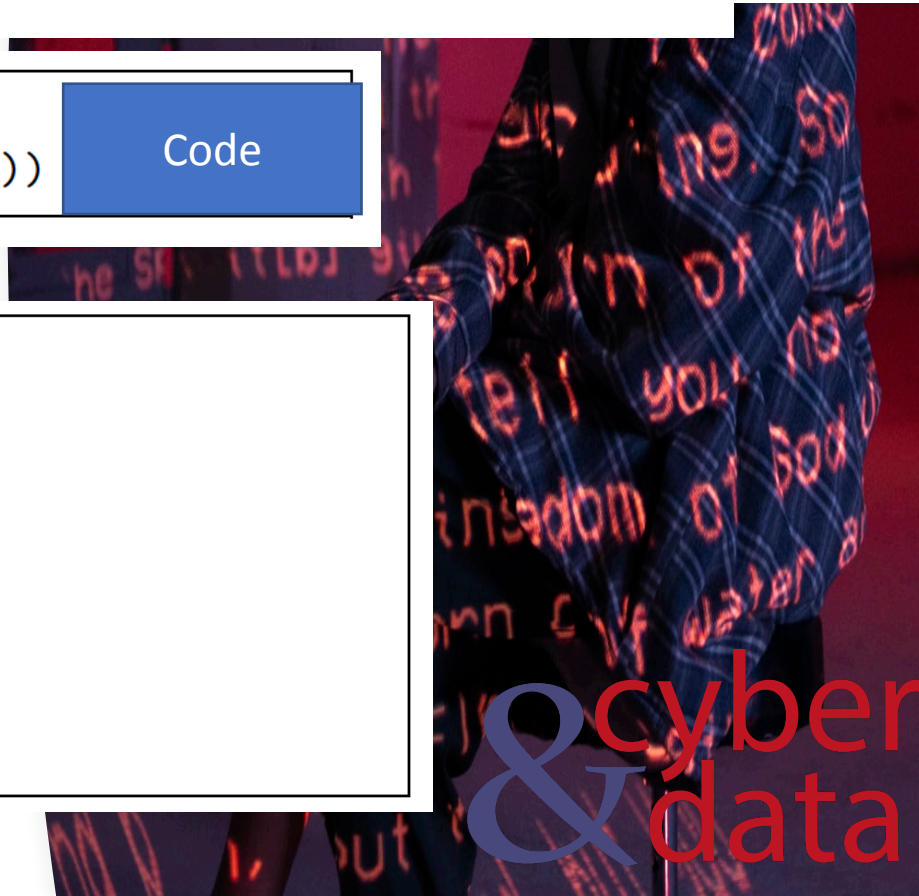
```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
print (ver.sort_values(['Location']).head(3))
print (ver.sort_values(['Location'],ascending=False).head(3))
```

Code

```
     IP Address Good login Bad login CPU Average Location
2  192.168.0.10         10         21        44.3   Dundee
4  192.168.0.12         17          1         2.3  Glasgow
0   192.168.0.1         50         20        33.2   London
     IP Address Good login Bad login CPU Average Location
3  192.168.0.11         15         16         4.5    Oxford
1   192.168.0.3         17          5        10.1  New York
0   192.168.0.1         50         20        33.2    London
```



Pandas: Correlate()

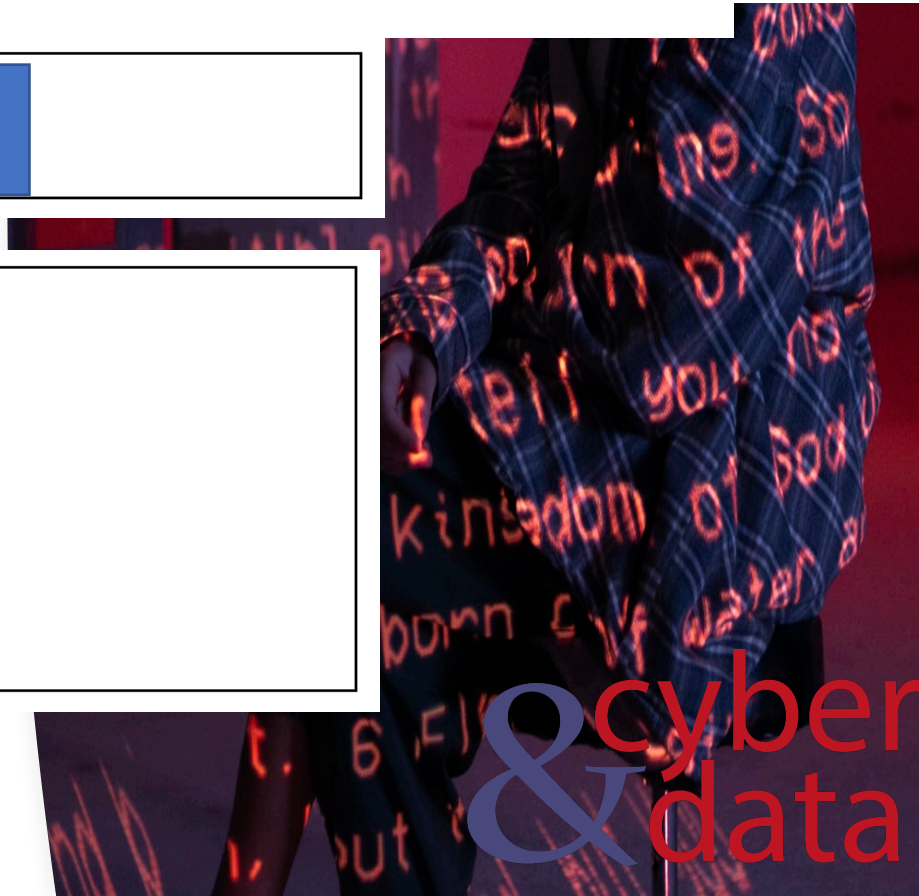
```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
print (ver.corr())
print (ver[['Good login','Bad login']].corr())
```

Code

```
             Good login  Bad login  CPU Average
Good login     1.000000   0.307191     0.268757
Bad login      0.307191   1.000000     0.777421
CPU Average    0.268757   0.777421     1.000000
            Good login  Bad login
Good login    1.000000   0.307191
Bad login     0.307191   1.000000
```



Pandas: Filtering

```
import pandas as pd
ver=pd.read_csv("test.csv"
print (ver.head(3))
```

```
IP Address,Good login,Bad login,CPU Average,Location
192.168.0.1,50,20,33.2,London
192.168.0.3,17,5,10.1,New York
192.168.0.10,10,21,44.3,Dundee
192.168.0.11,15,16,4.5,Oxford
192.168.0.12,17,1,2.3,Glasgow
```

```
ver2=ver['Good login']> 12
print (ver2)
ver2=ver[ver['Good login']> 12]
print (ver2)
```

Code

```
0     True
1     True
2    False
3     True
4     True
Name: Good login, dtype: bool
     IP Address Good login Bad login CPU Average Location
0   192.168.0.1         50         20        33.2   London
1   192.168.0.3         17          5        10.1 New York
3  192.168.0.11         15         16         4.5   Oxford
4  192.168.0.12         17          1         2.3  Glasgow
```

cyber & data

"From bits to information"

Tuples

"From bits to information"

Data to Information: NumPy and Pandas