NumPy and Pandas

NumPy

- NUMPY JEST TO BLIBLITOEKA Z ALGEBRY LINIOWEJ DLA PYTHONA, POWOD DLA KTOREGO JEST TAKA WAZNA, TO DLATEGO, ZE JEST UZYWANA W PRAWIE WSZYSTKICH BIBLIOTEKACH DLA DATA SCIENCE W ECOSYSTEM PYTHONA

- **NUMPY JEST ROWNIEZ NIESAMOWICIE SZYBKI, DZIEKI TEMU, ZE ODWOLUJE SIE DO BIBLIOTEK C**

INSTALLATION:

- conda install numpy
- pip install numpy

USING NumPy:

- import numpy as np

Po zainstalowaniu importuje tak jak zwyklad biblioteke
Numy posiada wiele wbudowanych funkcji i mozliwosci. Zajmiemy się tylko częścią z nich, tymi najważniejszymi aspektami jak : wektory, tablice, macierze, generacja liczb.

NumPy Arrays:

Głównie będziemy działali na arrays. Szczególnie na jedno wymiarowych array czyli vectors oraz na dwu wymiarowych czyli matrices, przy czym matrice moze miec 1 kolumne lub 1 wiersz.

Więc przejdźmy do tworzenia NumPy Arrays

- tworzenie z Python List

```
In [19]: my_list = [1,2,3]
         my_list

Out[19]: [1, 2, 3]

In [16]: np.array(my_list)

Out[16]: array([1, 2, 3])

In [20]: my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
         my_matrix

Out[20]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

In [21]: np.array(my_matrix)

Out[21]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

- **metody wbudowane**

arange :

Rownomiernie rozlozone wartosci w danym przedziale

```
In [22]: np.arange(0,10)
Out[22]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [23]: np.arange(0,11,2)
Out[23]: array([ 0,  2,  4,  6,  8, 10])
```

zeros and ones :

Generujemy arrays zer lub jedynek

```
In [24]: np.zeros(3)
Out[24]: array([ 0.,  0.,  0.])

In [26]: np.zeros((5,5))
Out[26]: array([[ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.]])

In [27]: np.ones(3)
Out[27]: array([ 1.,  1.,  1.])

In [28]: np.ones((3,3))
Out[28]: array([[ 1.,  1.,  1.],
                [ 1.,  1.,  1.],
                [ 1.,  1.,  1.]])
```

linspace :

Rownomiernie roozlozona okreslona liczba wartosci w danym przedziale

```
In [29]: np.linspace(0,10,3)
Out[29]: array([  0.,   5.,  10.])

In [31]: np.linspace(0,10,50)
Out[31]: array([  0.        ,   0.20408163,   0.4
         22449 ,
                  0.81632653,   1.02040816,   1.2
```

eye:

Macierz jednostkowa

```
In [37]: np.eye(4)

Out[37]: array([[ 1.,  0.,  0.,  0.],
                [ 0.,  1.,  0.,  0.],
                [ 0.,  0.,  1.,  0.],
                [ 0.,  0.,  0.,  1.]])
```

- Tworzenie arrays z liczb losowych

rand:

Tworzy array z losowych wartosci pomiedzy [0,1] o okreslonym ksztalcie

```
In [47]: np.random.rand(2)

Out[47]: array([ 0.11570539,  0.35279769])

In [46]: np.random.rand(5,5)

Out[46]: array([[ 0.66660768,  0.87589888,  0.12421056,  0.65074
         126,  0.60260888],
                [ 0.70027668,  0.85572434,  0.8464595 ,  0.27354
```

randn:

Tak jak rand, ale wartosci sa z rozkladu normalnego, a nie jednolite jak w przypadku rand

```
In [48]: np.random.randn(2)

Out[48]: array([-0.27954018,  0.90078368])

In [45]: np.random.randn(5,5)

Out[45]: array([[ 0.70154515,  0.22441999,  1.33563186,  0.82872
         577, -0.28247509],
                [ 0.64489788,  0.61815094, -0.81693168, -0.30102
         424, -0.290305741,
```

x = np.random.randn(100)
x.sort()

import matplotlib.pyplot as plt
from scipy.stats import norm
std = np.std(x)
mean = np.mean(x)
plt.plot(norm.pdf(x,mean,std))

```
import scipy.stats as stats
import pylab as pl
fit = stats.norm.pdf(x, np.mean(x), np.std(x))  #this is a fitting indeed
pl.plot(x,fit,'-o')
pl.hist(x,normed=True)      #use this to draw histogram of your data
pl.show()
```

randint:

Zwraca losowy integer z przedzialu [low,high)

```
In [50]: np.random.randint(1,100)
Out[50]: 44

In [51]: np.random.randint(1,100,10)
Out[51]: array([13, 64, 27, 63, 46, 68, 92, 10, 58, 24])
```

Metody i Atrybuty of Array

- **tworzymy**
  - **arr = np.arrange(16)**
  - **ranarr = np.random.randint(0,20,10)**

RESHAPE :

Zwraca array w nowym formacie

arr.reshape(4,4)

MAX, MIN, ARGMAX, ARGMIN:

Max I min zwracaja najwieksza/najmniejsza wartosc z Array
ARGMAX/ARGMIN zwracaja indexy gdzie znajduja sie max i min

```
index_x = np.unravel_index(np.argmax(matrix_arr),matrix_arr.shape)
index_x
```

SHAPE :

Jest to atrybut numpy array

-vector **arr.shape** **-> (16,)**

-macierz **Operacja arr.reshape(1,16).shape -> (1,16)**

DTYPE:

Typ danych w array

arr.dtype **-> dtype('int64')**

INDEXOWANIE I WYBIERANIE

import numpy as np
arr = np.arange(0,11)

- wybor poprzez nawiasy :
  arr[index]
  arr[index_inclusive : index_exclusive]

- nadawanie wartosci
  arr[index1 : index 2] = 4
    wszystkie wartosci [index1,index2) zamienia sie w 4
  slice_of_array = arr[i1: i2]
    kawalek array arr od [i1,i2)
  arr[:] = 3
    odwolujem sie do wszystkich wartosci w array

  slice_of_array odwoluje sie do prawdziwych wartosci arr, NIE JEST TO KOPIA, Gdy wykonamy zmiane na slice_of_array to zmieni sie oryginal

  arr_copy = are.copy()
    To jest kopia

  arr_2d[1]
    zwroci caly wiersz pod indeksem 1
  arr_2d[1][2] lub arr_2d[1,2]
    zwroci wartosc z wiersza indeks 1 i kolumny indeks 2

- **fancy indexing**

```
In [21]: #Set up matrix
         arr2d = np.zeros((10,10))

In [22]: #Length of array
         arr_length = arr2d.shape[1]

In [23]: #Set up array

         for i in range(arr_length):
             arr2d[i] = i

         arr2d

Out[23]: array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0
         .],
                [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1
         .],
                [ 2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2
         .],
                [ 3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3
         .],
```

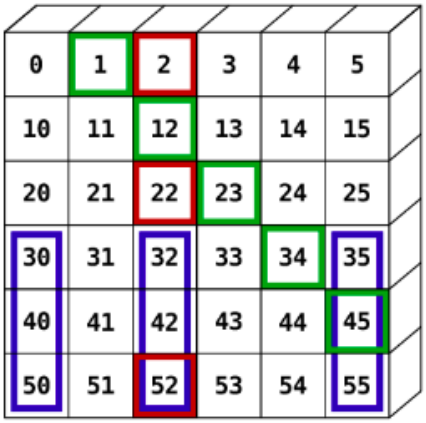
```
In [24]: arr2d[[2,4,6,8]]
Out[24]: array([[ 2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.],
                [ 4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.],
                [ 6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.],
                [ 8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.]])
```

```
In [25]: #Allows in any order
```

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])

>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



- selection

```
In [28]: arr = np.arange(1,11)
         arr

Out[28]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [30]: arr > 4

Out[30]: array([False, False, False, False,  True,  True,  True,
         True,  True,  True], dtype=bool)
```

```
In [31]: bool_arr = arr>4
```

```
In [32]: bool_arr

Out[32]: array([False, False, False, False,  True,  True,  True,
         True,  True,  True], dtype=bool)
```

```
In [33]: arr[bool_arr]

Out[33]: array([ 5,  6,  7,  8,  9, 10])
```

```
In [34]: arr[arr>2]

Out[34]: array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [37]: x = 2
         arr[arr>x]

Out[37]: array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

ARYTMETYKA

arr = np.arange(0,10)
arr + arr -> dodajemy indeksami
1/arr
arr3**

Uniwersalne funkcje na Arrays

Np.sqrt(arr)

Np.exp(arr)

Np.max(arr)

np.sin(arr)

Np.log(arr)

EXERCISES NUMPY ->

an array of 10 zeros ->
// 10 ones ->
// 10 fives ->

an array of integers from 10 to 80 ->

all odd numbers from 10 to 60 ->

4x4 matrix values in range (0,15) ->

identity matrix 4x4 ->

losowa liczba z przedzialu (0 , 1) ->

10 losowych w rozkaldzie normalnym ->

macierz 10x10 z wartosciami od 0.01 do 1. ->

array od 0 do 1 z 20 przedzialami ->

Tworzymy macierz
matrix = np.arange(1,26).reshape(5,5)

wyswietlic macierz od 3 wiersza i 2 kolumny ->

srodkowa wartosc w macierzy ->

druga kolumna do 3 wiersza ->

4 wiersz ->

sumowanie 3 kolumny ->

odchylenie standardowe ->

suma kolumn ->

Pandas

Pandas to taka mocna wersja Exela z wieloma udogodnieniami i mozliwosciami.

Series

Jest to nowy typ danych series pandas data type i jest bardzo podobny do array numpy, tak naprawde zostal on zbudowany na nim)
Series moze przechowywac dowolny Python Object, oraz mozna wyciagac dowolna oś zaetykietowana, a nie poprzez indeksowanie
Dobra sprawdzmy to i sie wyjasni

```
import numpy as np
import pandas as pd
```

labels = ['a','b','c']
my_list = [10,20,30]
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}

Tworzenie z listy -> **pd.Series (data = my_list , index = labels)** // optymalnie bez data i index, oraz nie trzeba dodawac index, data najwazniejsze :P

Tworzenie z numpy array -> **pd.Series(data = arr , index = labels)**

Tworzenie ze slownika -> **pd.Series(d)**

Pandas Series moze przechowywac przerozne typy obiektow, nawet funkcje

pd.Series(data = labels)

Pd.Series([sum,print,len])

Uzywanie indeksow w pandas, poprzez podawanie nazw lub numerow mozemy szybko dotrzec do informacji, prawie jak w tablicach hashowanych i slownikach

ser1 = pd.Series([1,2,3,4],index = ['apple', 'pear','plum', 'banana'])
ser2 = pd.Series([1,2,5,4],index = ['apple', 'pear','orange', 'banana'])

ser1['apple'] -> 1
Ser1 + ser2 -> sume tam gdzie w obu byly wartosci

DataFrames

Glowny kon napedowy tej biblioteki, zainspirowane z R, jest to wiele Series objects upchane razem i wspoldzielom te same indeksy

```
Import numpy as np
Import pandas as pd
From numpy.random import rands
Np.random.seed(101)
Df = pd.DataFrame(randn(5,4), index ='A B C D E'.split(), columns = 'W X Y Z'.split())

Df [ 'W' ]
Df [[ 'W' , 'Z' ]]  list of column names

Type ( df[ 'W' ] )
df[ ' new ' ] = df [ 'W' ] + df [ ' Y ' ]  // dodawanie kolumny

Df.drop('new' , axis = 1)       // usuwanie columny
```

Jest teraz wyswietlimy df to wartosc nie zostala tak naprawde usunieta
Nalezy dodac inplace

```
Df.drop('new' , axis = 1, inplace = True)

df.drop('E', axis = 0)          // usuwanie wierszy
```

INDEXING

Wybieranie wierszy poprzez :

nazwe	**df.loc['A']**
index	**df.iloc[2]**

```
df.loc['B' , 'Y']

df.loc[ [ 'A' , 'B' ] , [ 'W' , 'Y'] ]
```

Wybieranie poprzez warunki logiczne

Df > 0 -> zwraca matrix True and Falses

df[df>0] -> tylko wartosci wieksze od 0

Df [df['W'] >0] wyswietli wiersze, gdzie w kolumnie W wartosc byla > 0

Df [df['W'] > 0] ['Y' , 'X']

Laczenie warunkow poprzez & lub |

Df [(df['W'] > 0) & (df['Y'] > 1)]

```
# Reset to default 0,1...n index
df.reset_index()
```

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

new_column = ‘AA BB CC DD EE’.split() // tworzenie nowej kolumny

df[‘NewColumn’] = new_column

Df.set_index(‘NewColumn’) // nowe indeksy z jednej z kolumn

// pamietac trzeba o inplace = True !!!!

MULTIINDEXED DATAFRAME

```
# Index Levels
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)

kierunki =
['Data_Science','Data_Science','Data_Science','Infa','Infa','Infa','IBM','IBM','IBM','EleTele','E
leTele','EleTele','AiR','AiR','AiR']
grupy = [1,2,3,1,2,3,1,2,3,1,2,3]
multi_index = list(zip(kierunki,grupy))
multi_index = pd.MultiIndex.from_tuples(multi_index)
df = pd.DataFrame(np.random.randn(12,2),index = multi_index, columns =
['student','srednia'] )

df.index.names = [‘kierunek','ID']



df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
df

Df.loc[‘G1’]

df.loc[‘G1’].loc[1]

df.index.names

df.index.names = [‘Group’,’Num’]
```

df.xs(‘G1’) // specified index

df.xs([‘G1’, 1])

df.xs(1 , level = ‘Num’)

___ from pandas library

```
>>> d = {'num_legs': [4, 4, 2, 2],
...      'num_wings': [0, 0, 2, 2],
...      'class': ['mammal', 'mammal', 'mammal', 'bird'],
...      'animal': ['cat', 'dog', 'bat', 'penguin'],
...      'locomotion': ['walks', 'walks', 'flies', 'walks']}
>>> df = pd.DataFrame(data=d)
>>> df = df.set_index(['class', 'animal', 'locomotion'])
>>> df
```

```
>>> df.xs(‘mammal')    //Get values at specified index
>>> df.xs(('mammal', 'dog'))    //Get values at several
indexes
>>> df.xs('cat', level=1)   //specified index and level
>>> df.xs(('bird', 'walks'),    //several indexes and levels
...       level=[0, 'locomotion'])
>>> df.xs('num_wings', axis=1)   //specified column and axi
```

Grupownaie

```
import pandas as pd
# Create dataframe
data = {'Uniwerek':['PG','PG','UG','UG','GUMED','GUMED'],
     'Student':['Tomek','Hania','Jan','Patrycja','Marek','Ola'],
     ‘DlugECTS’:[2,4,16,8,5,3]}

df = pd.DataFrame(data)

po_uniwerku = df.groupby(‘Uniwerek’) //sortowanie po kolumnie uniwerek
po_uniwerku.mean() // srednia
po_uniwerku.std()    // odchylenie standardowe
po_uniwerku.min()    // najnizsze wartosci
po_uniwerku.max()   // najwyzsze wartsoci
po_uniwerku.count()// zliczenie ilosci danych
po_uniwerku.describe()  // wszystko
po_uniwerku.describe().transpose()
po_uniwerku.describe().transpose()[ ‘PG’ ]
```

Łączenie, łączenie oraz łączenie :P

Merging, Joining, and Concatenating

łączenie :P

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])

df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                    index=[8, 9, 10, 11])
```

Concatenation - czyli dolaczanie, koniec do konca

```
pd.concat( [df1, df2, df3] )
```

Mona to tez zrobic wzgledem osi poziomej

```
pd.concat( [df1, df2, df3]  , axis = 1)
```

```
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
```

Merging - scalanie

```
pd.merge(left,right,how='inner',on='key')
```

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
'key2': ['K0', 'K1', 'K0', 'K1'],
'A': ['A0', 'A1', 'A2', 'A3'],
'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
'key2': ['K0', 'K0', 'K0', 'K0'],
'C': ['C0', 'C1', 'C2', 'C3'],
'D': ['D0', 'D1', 'D2', 'D3']})

```
pd.merge(left, right, on=['key1', 'key2'])
```

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2

```
pd.merge(left, right, how='outer', on=['key1', 'key2'])
```

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	C1	D1
3	A2	B2	K1	K0	C2	D2
4	A3	B3	K2	K1	NaN	NaN
5	NaN	NaN	K2	K0	C3	D3

```
pd.merge(left, right, how='right', on=['key1', 'key2'])
```

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2
3	NaN	NaN	K2	K0	C3	D3

Joining - dolaczanie

```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                      index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                      'D': ['D0', 'D2', 'D3']},
                       index=['K0', 'K2', 'K3'])
```

```
left.join(right)
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

```
left.join(right, how='outer')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

Operations

```
import pandas as pd
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','xyz']})
df.head()
```

// info about unique values

```
df['col2'].unique()     // unikalne
df['col2'].nunique()    // liczba unikalnych
df['col2'].value_counts()     // liczenie ilosci zmienych przypisanych do unikalnych komorek
```

// Selecting Data

```
Newer = df[ (df['col1'] > 2) & (df['col2'] == 444) ]
```

// Applying Functions

```
Def times2(x) :
        return x*2

df['col1'].apply(times2)       // times 2
df['col3'].apply(len)  // dlugosc
df['col1'].sum()       // suma
```

```
Del df['col1']          // permamentne usuniecie
```

```
df.columns    //info about columns
Df.index      // info about indexes
```

// sorting

```
df.sort_values(by='col2' , inplace = True)  //       sortowanie
df.isnull() // checking True / False if value is 0
df.dropna()    // usuwanie wierszy z NaN
```

Missing Data Problem

```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})

df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[np.nan,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

Jest uzyjemy komendy df.dropna() wyrzuci nam wszystkie NaN

df.dropna(axis = 1) -> odrzuci kolumny z NaN

Df.dropna(thresh = 2) -> z tolerancja

df.fillna(value = 'FILL VALUE'). -> uzupelnienie wartosci NaN

df['col1'].fillna(value=df['col1'].mean()) -> uzupelnienie wartoscia srednia

df.loc[df['col2'].isna(),'col2'] = 3. -> Wykorzystanie isna()

```
data = {'A':['foo','foo','foo','bar','bar','bar'],
    'B':['one','one','two','two','one','one'],
     'C':['x','y','x','y','x','y'],
     'D':[1,3,2,5,4,1]}

df = pd.DataFrame(data)

df.pivot_table(values = 'D', index = ['A','B'], columns=['C'])
```

Data Input and Output

```
dane = np.arange(16).reshape(4,4)

df = pd.DataFrame(index = ['0','1', '2', '3'],
                  columns = 'a b c d'.split(),
                  data = dane)
df
```

CSV

```
df.to_csv('example',index=False)

Df = pd.read_csv('example')
```

EXCEL

df.to_excel('Excel_Sample.xlsx',sheet_name='Sheet1')

pd.read_excel('Excel_Sample.xlsx',sheetname='Sheet1')

HTML

```
df = pd.read_html('http://www.fdic.gov/bank/individual/failed/banklist.html')
```

```
df[0]
```

```
table results of premier league
```

```
df = pd.read_html('https://www.premierleague.com/matchweek/3291/table')
```

Ecommerce Purchases Exercise -

Fake dane z transakcji z Amazon.

Wszystkie rozwiazania, a przynajmniej wiekszosc to jedna linijka
Nie rozpisujcie sie za bardzo ;)

Import pandas as pd
ecom = pd.read_csv('Ecommerce Purchases')

// check head

ecom.head()

// how many data ?

// srednia Purchase Price

// najwyzsza i najnizsza cena

// ile osob uzywa angielskiej wersji przegladarki

// ile osob jest prawnikami

// Ile osob kupowalo popoludniu a ile rano

// 5 najpopularniejszych zawodow

// Dane dotyczace osoby ktorej transakcja pochodzila z Lot = 90WT

// adres email wszystkich Sprzedawcow ksiazek

// ile osob ma karty kredytowe z american express I zrobilo zakupy powyzej 90 dolcow

// ile osob ma karty kredytowe ktore przedawniaja sie 2023

// Zliczyc dostawcow mailowych