`Programování` 1 pro matematiky

13. cvičení, 06-01-2025

Obsah:

Farní oznamy

1. **Materiály k přednáškám** najdete v GitHub repozitáři https://github.com/PKvasnick/Programovani-1. Najdete tam také kód ke cvičením a pdf soubory textů cvičením.
2. **Domácí úkoly** - Pokud jste za domácí úkoly získali **138 bodů a vic**, zapíšu vám v průběhu následujících dnů zápočet. Pokud vám chybí body k zápočtu, dobrý čas s tím něco dělat je teď.
3. **Kde se nacházíme** Končíme.
4. **Poznámka ke dnešnímu cvičení** Pro demonstraci používáme Jupyter notebook. Google Colab notebook vám poslouží stejně dobře. Hlavní důvod jsou grafy, vše ostatní vám poběží i v textovém prostředí a také ve VSCode nebo PyCharmu, musíte si ale sami doinstalovat potřebné moduly:

Domácí úkoly

Vracím se jenom k předchozímu domácímu úkolu o maticové třídě.

Maticová třída

Tento domácí úkol měl sloužit k procvičení vytváření tříd a jejich metod, a řešení mohlo ve velice zjednodušené verzi vypadat nějak takto:

```
class Matrix:

    def __init__(self, ll): # ll je seznam seznamů
        self.matrix = ll

    def __repr__(self):
        return '\n'.join(' '.join(str(val) for val in row) for row in self.matrix)

    def vals(self):
        return self.matrix

    def dims(self):
        return len(self.matrix), len(self.matrix[0])
```

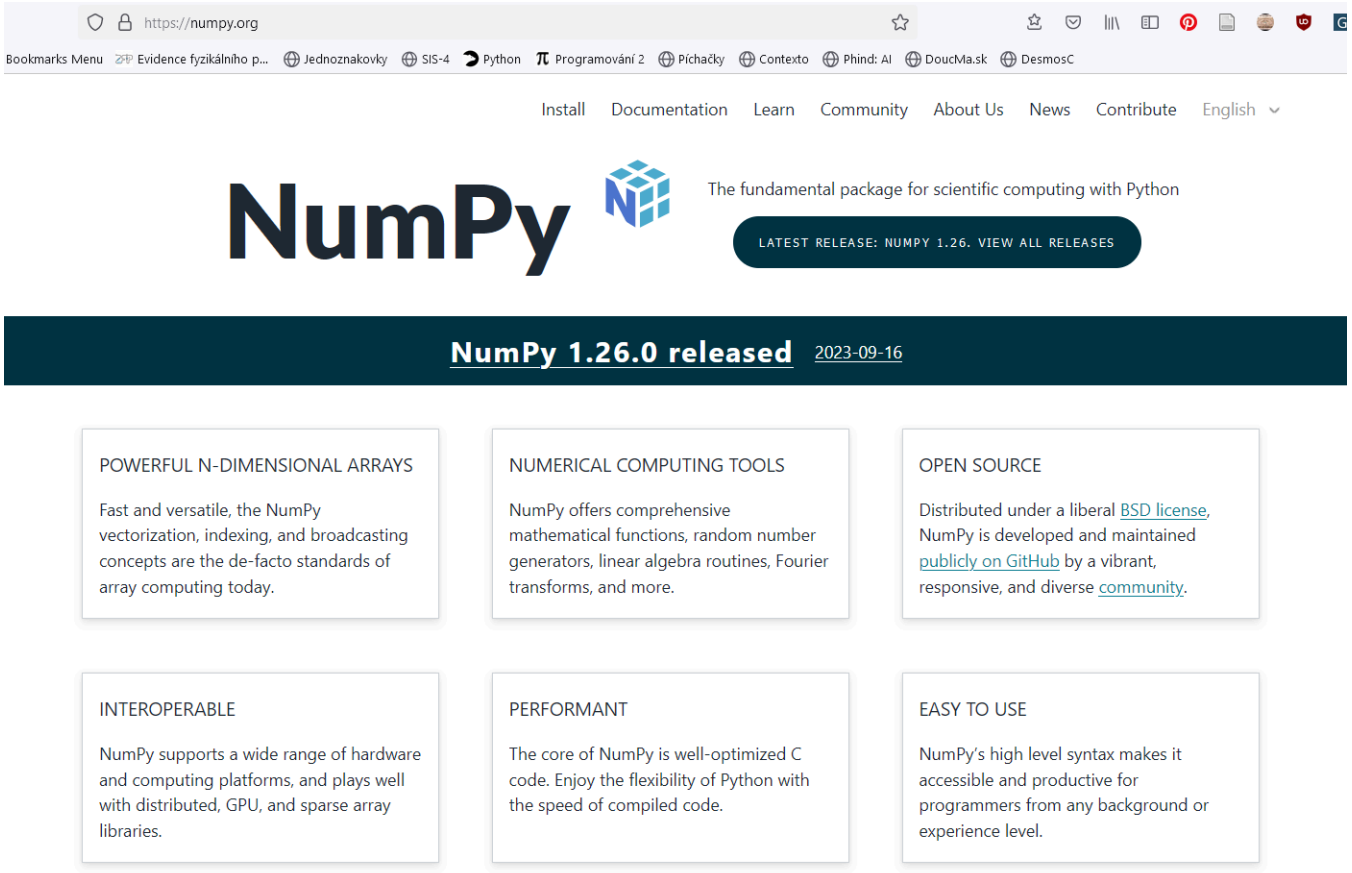
```
14
15      def __add__(self, other):
16          if self.dims() != other.dims():
17              raise ValueError("Sečítat lze pouze matice stejných rozměrů")
18          result = []
19          for i, row in enumerate(self.matrix):
20              result.append([x + y for x, y in zip(row, other.matrix[i])])
21          return Matrix(result)
22
23      def __sub__(self, other):
24         if self.dims() != other.dims():
25              raise ValueError("Odečítat lze pouze matice stejných rozměrů")
26         result = []
27         for i, row in enumerate(self.matrix):
28              result.append([x - y for x, y in zip(row, other.matrix[i])])
29         return Matrix(result)
30
31      def __mul__(self, other):
32          if isinstance(other, Matrix):
33              if self.dims()[1] != other.dims()[0]:
34                  raise ValueError("Násobit lze pouze matice kompatibilních rozměrů")
35              result = [[0] * len(other.matrix[0]) for _ in range(len(self.matrix))]
36              for i in range(len(self.matrix)):
37                  for j in range(len(other.matrix[0])):
38                      for k in range(len(other.matrix)):
39                          result[i][j] += self.matrix[i][k] * other.matrix[k][j]
40              return Matrix(result)
41          else:
42              result = [[x * other for x in row] for row in self.matrix]
43              return Matrix(result)
44
45      def is_symmetric(self):
46          if len(self.matrix) != len(self.matrix[0]):
47              return False
48          for i in range(len(self.matrix)):
49              for j in range(i + 1, len(self.matrix[0])):
50                  if self.matrix[i][j] != self.matrix[j][i]:
51                      return False
52          return True
53
54
55  def zero_matrix(r, c):
56      return Matrix([[0] * c for _ in range(r)])
57
58
59  def identity_matrix(n):
60      return Matrix([[1 if i == j else 0 for j in range(n)] for i in range(n)])
```

Základní moduly pro zpracování dat v Pythonu

Dnes se seznámíme se třemi Pythonskými balíčky, které tvoří zákklad ekosystému pro technické počítání v Pythonu:

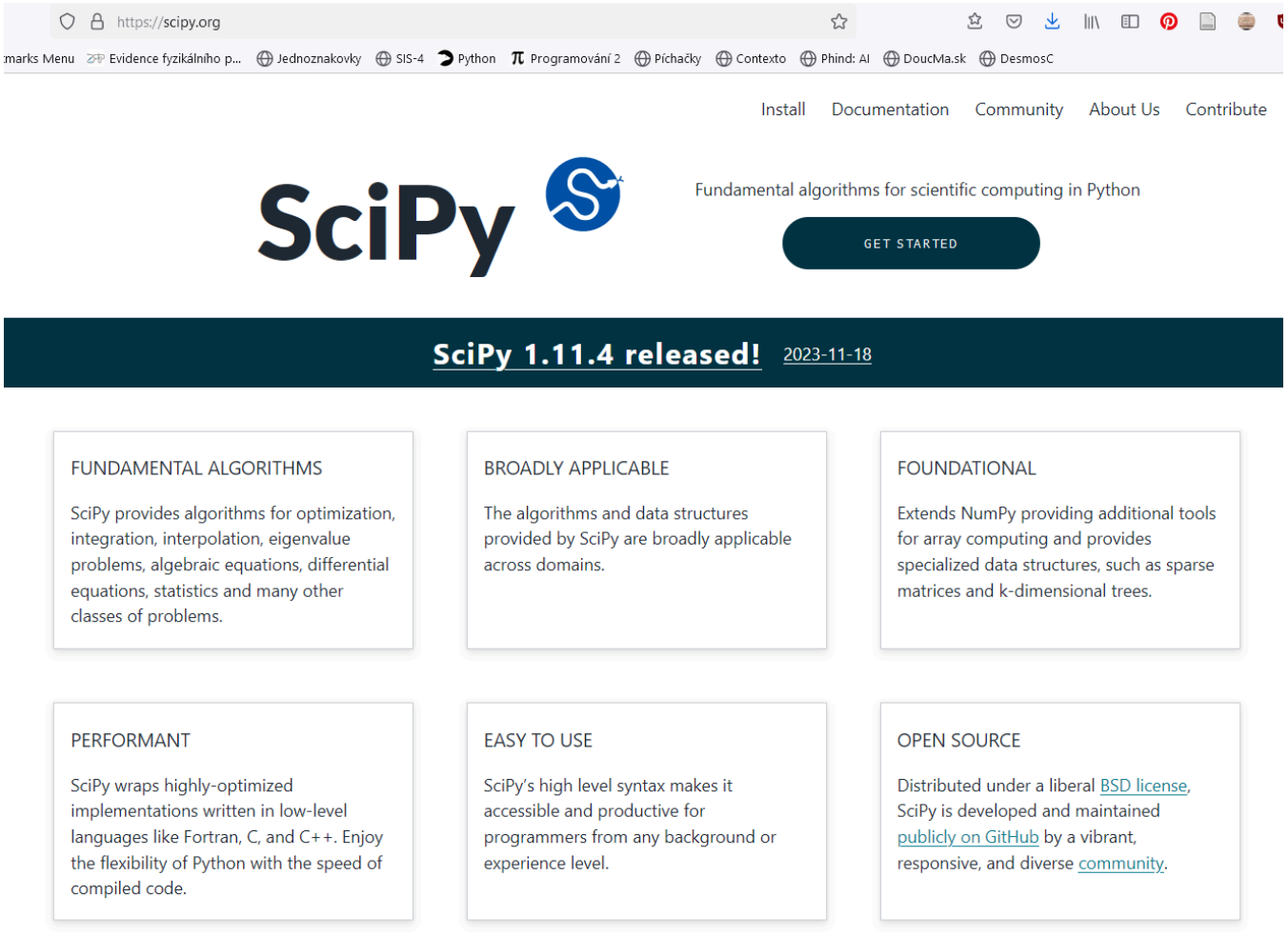
- **numpy** je základní modul, který je "dependencí" pro ostatní moduly. Podporuje vícerozměrná pole, algebru nad nimi, lineární algebru, speciální funkce, optimalizaci, náhodné generátory, podporu hardwaru ,(GPU) a ještě mnoho jiných věcí.

Dolumentace: www.numpy.org



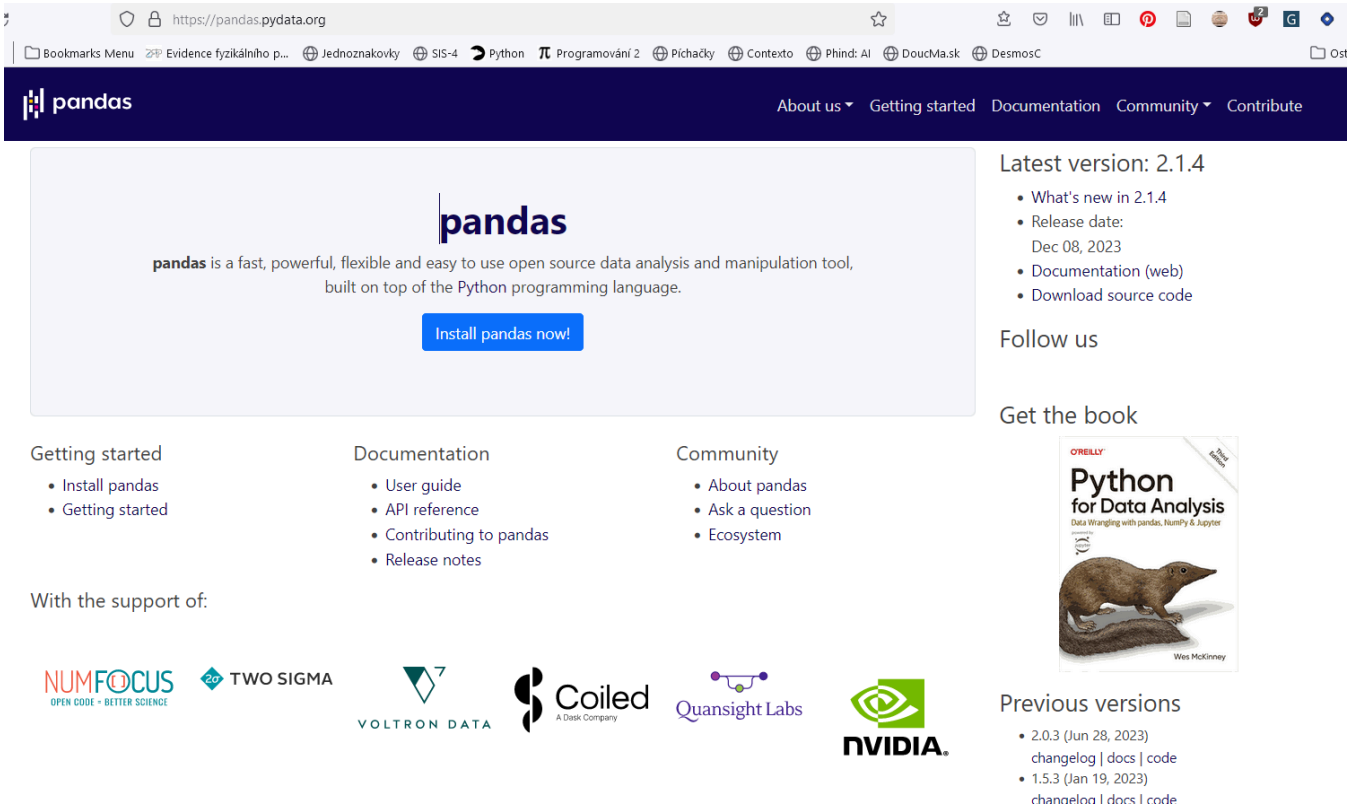
Numpy zabezpečuje také integraci C/C++ a Fortranského kódu s Pythonem.

Úzce navázaný na modul numpy je modul **SciPy**. Obsahuje řadu základních algoritmů, rozšiřujících (nebo duplikujících) `numpy` .



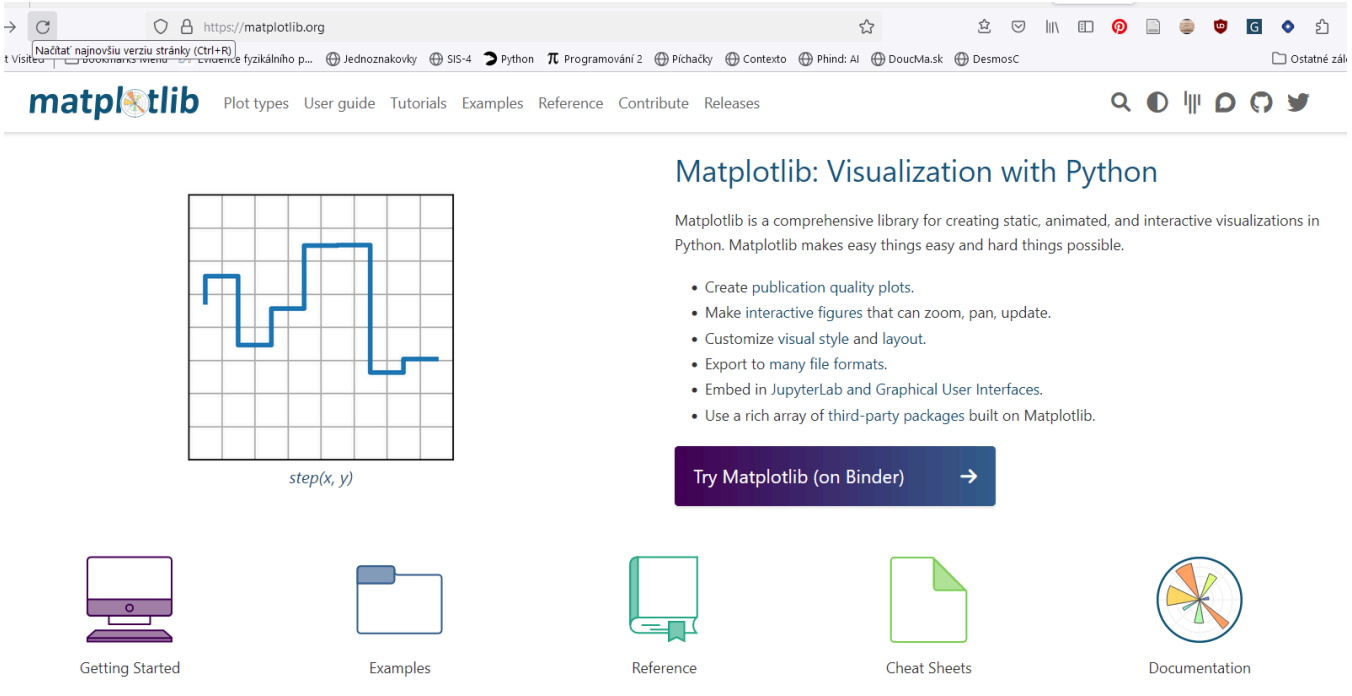
- **Pandas** je modul, definující datové tabulky a operace s nimi. Datové tabulky jsou specifické datové objekty pro zpracování dat a blíž než k maticím mají k Excelovským listům

pandas.pydata.org



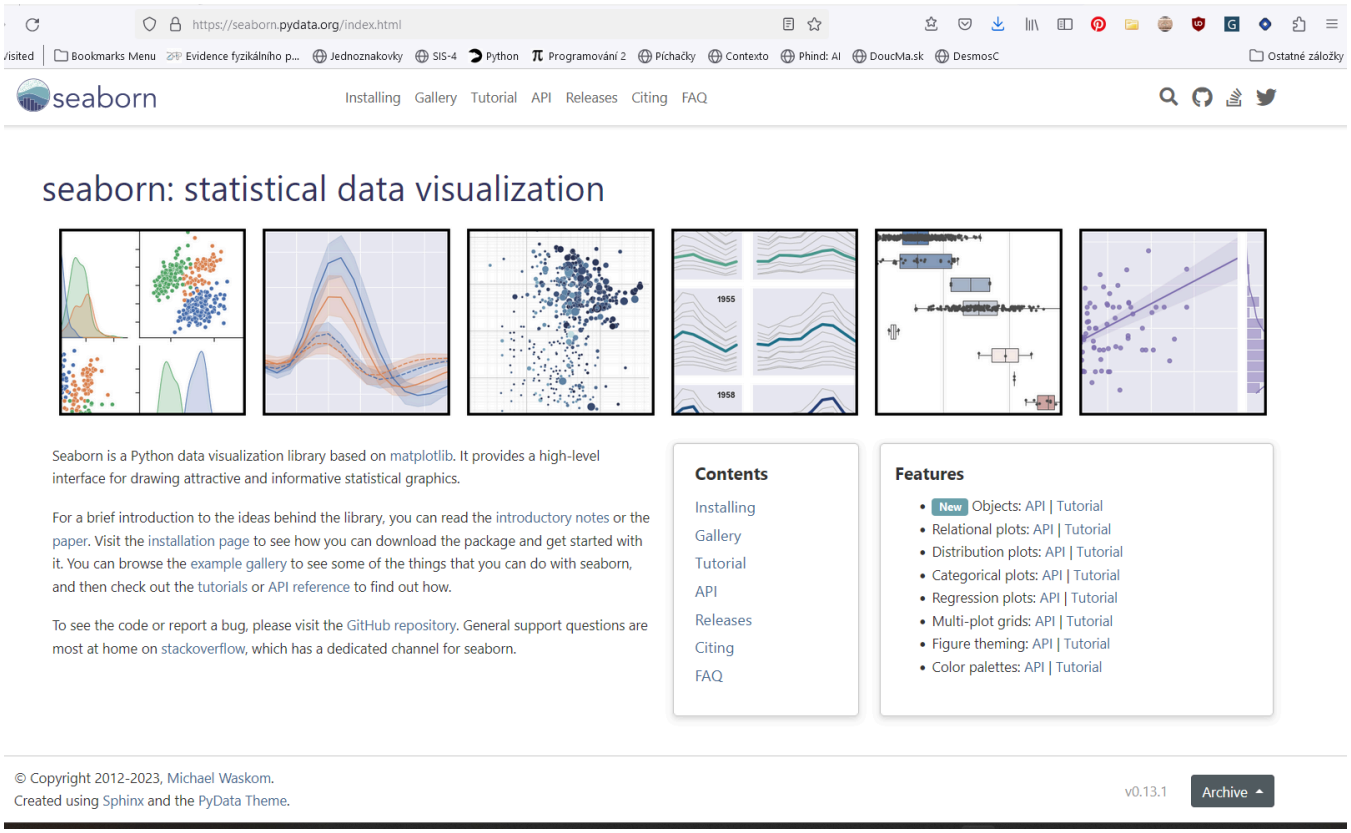
- **matplotlib** je základní knihovna pro vytváření grafů. Má API pro různé jazyky, ale nejširší použití má právě v Pythonu. Podporuje širokou škálu vyjádřovacích možností, typů grafů atd. Je také základnou pro další grafické knihovny.

matplotlib.org



Tento modul nemá vyhrazenou speciální část v následujícím výkladu - je to tím, že ho budeme používat pro grafické znázornění výsledků pro ostatní dva moduly.

- **seaborn** je nadstavbou matplotlib a obsahuje bohatou galerii grafů pro explorativní analýzu datových tabulek.



Instalace

Základní možnost je instalovat moduly pomocí pip (`pip install numpy pandas matplotlib`). Instalace chvíli trvá, moduly jsou veliké a mají další závislosti.

Lepší je proto instalovat nějakou distribuci, která obsahuje tyto moduly a jejich dependence, například `anaconda`. Google Colab notebooky běží na cloudu a instance mají tyto tři (a další) moduly instalované.

---

numpy

```
1 import numpy as np
```

Toto je standardní způsob importu modulu *numpy* a i když není povinný, je rozumné ho používat. Stejně tak máme standardní způsoby importu dalších modulů:

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
```

numpy array

numpy podporuje vícerozměrná pole a operace s nimi.

numpy array lze vytvořit více způsoby:

```
# Vytváříme numpy pole:

# 1. Konverzí seznamu a úpravou tvaru
a = np.arange(12).reshape(3,4)
print("a", a)

a [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

# 2. Zadáním dimenze: prázdné pole
b = np.ndarray((3,4))
print("b", b)

b [[0.0e+000 4.9e-324 9.9e-324 1.5e-323]
 [2.0e-323 2.5e-323 3.0e-323 3.5e-323]
 [4.0e-323 4.4e-323 4.9e-323 5.4e-323]]

# 3. Specializovaným konstruktorem
c = np.zeros((3,4))
print("c",c)

c [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

d = np.ones((3,4))
print("d", d)
d [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Vytváření sekvencí:

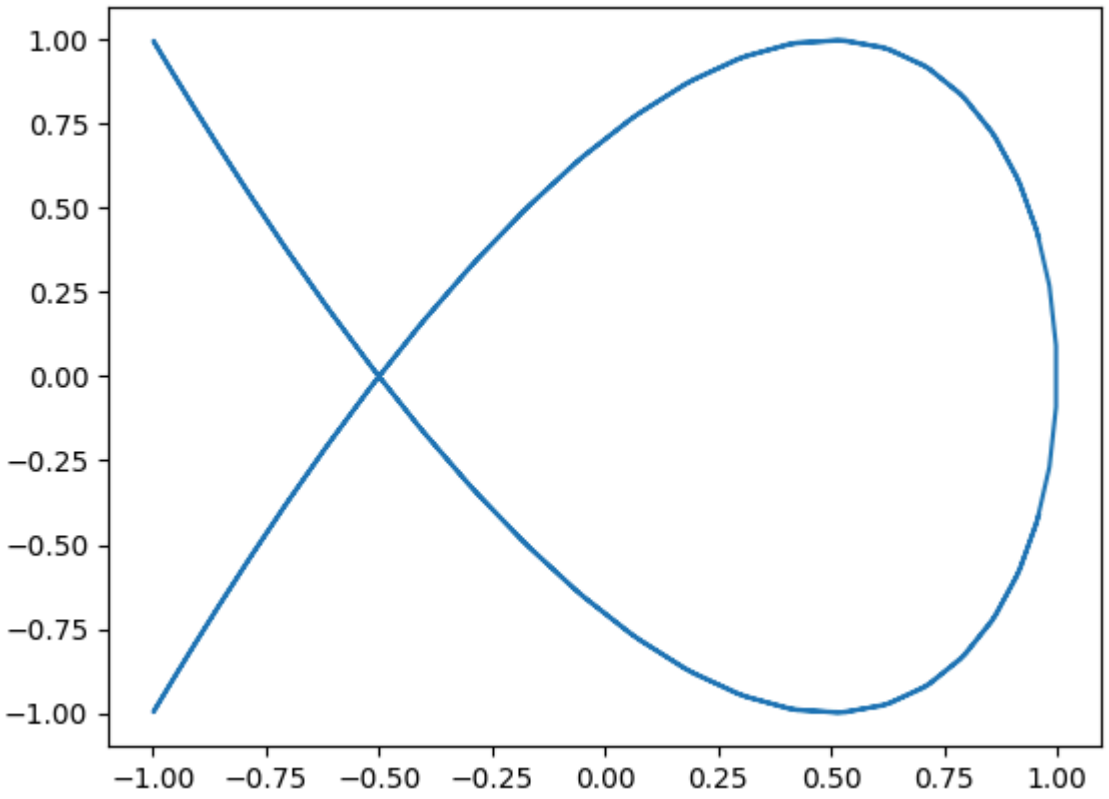
`np.arange` - jako range, ale vytváří np.array

`np.linspace` - pravidelné číselné řady:

```
print(np.linspace(0, 10, 20, endpoint = True))
print(np.linspace(0, 10, 20, endpoint = False))
---
[ 0.          0.52631579  1.05263158  1.57894737  2.10526316  2.63157895
  3.15789474  3.68421053  4.21052632  4.73684211  5.26315789  5.78947368
  6.31578947  6.84210526  7.36842105  7.89473684  8.42105263  8.94736842
  9.47368421 10.        ]
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5 8.  8.5
 9.  9.5]
```

Kreslíme

```
import matplotlib.pyplot as plt
x = np.linspace(-3,3, 100, endpoint = True)
plt.plot(np.cos(2*x), np.sin(3*x))
```



`np.sin, np.cos` jsou vektorizované verze `math.sin, math.cos` a numpy má takovýchto vektorizovaných verzí od běžných funkcí mnoho a umožňuje vytvářet vlastní.

operace s poli

```
print(a)
arev = np.arange(11,-1,-1).reshape(3,4)
print(arev)

print(a+arev)
print(a*arev)
print(a / arev)
---
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[[11 10  9  8]
 [ 7  6  5  4]
 [ 3  2  1  0]]
[[11 11 11 11]
 [11 11 11 11]
 [11 11 11 11]]
[[ 0 10 18 24]
```

```
 [28 30 30 28]
 [24 18 10  0]]
[[ 0.          0.1         0.22222222  0.375     ]
 [ 0.57142857  0.83333333  1.2         1.75      ]
 [ 2.66666667  4.5        10.                inf]]

<ipython-input-20-1a385f80933a>:7: RuntimeWarning: divide by zero encountered in divide
  print(a / arev)

```

Všechny operace jsou mezi odpovídajícími prvky polí, tedy ne matickvé operace. Ty si musíme explicitně vyžádat.

Co když nesouhlasí rozměry?

```
v = np.arange(1,4).reshape(3,1)
print(v)
print(a*v)
print(v*a)
---
[[1]
 [2]
 [3]]
[[ 0  1  2  3]
 [ 8 10 12 14]
 [24 27 30 33]]
[[ 0  1  2  3]
 [ 8 10 12 14]
 [24 27 30 33]]
```

Chybějící data se inteligentně doplní (pravidla jsou velice komplexní) - *broadcasting*

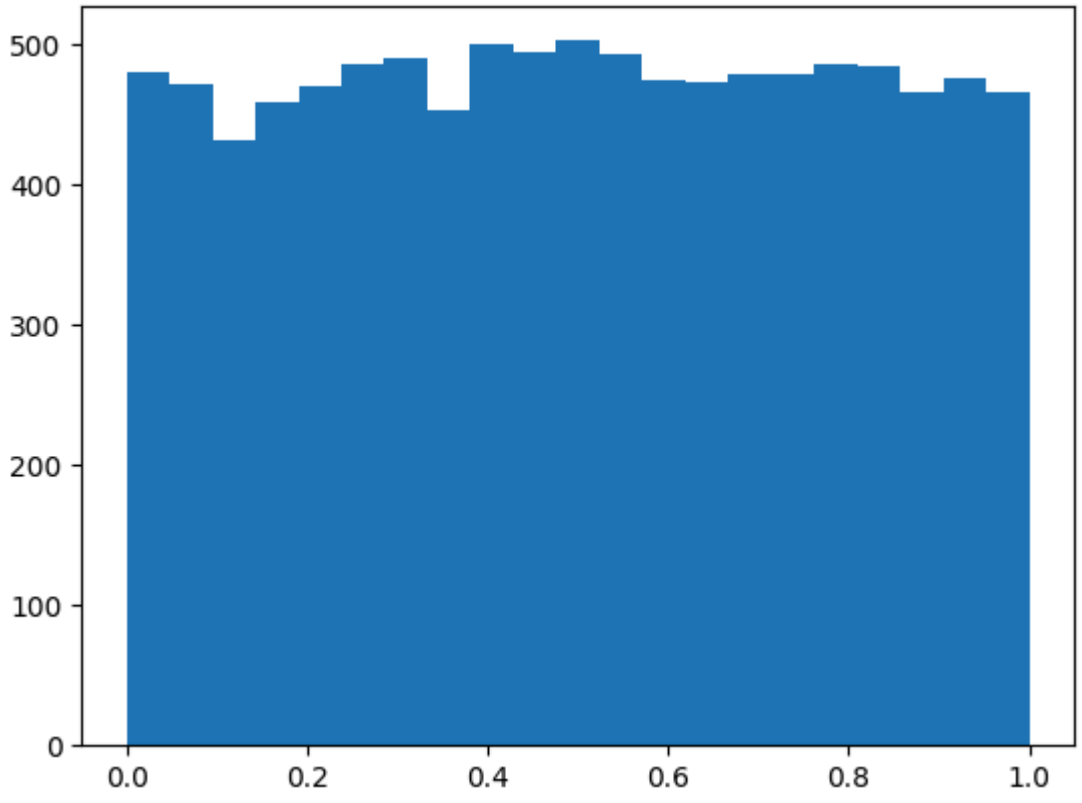
Iterace přes pole:

```
for i in a:
  print(i)

for i in np.nditer(a):
  print(i)
---
[0 1 2 3]
[4 5 6 7]
[ 8  9 10 11]
0
1
2
3
4
5
6
7

```

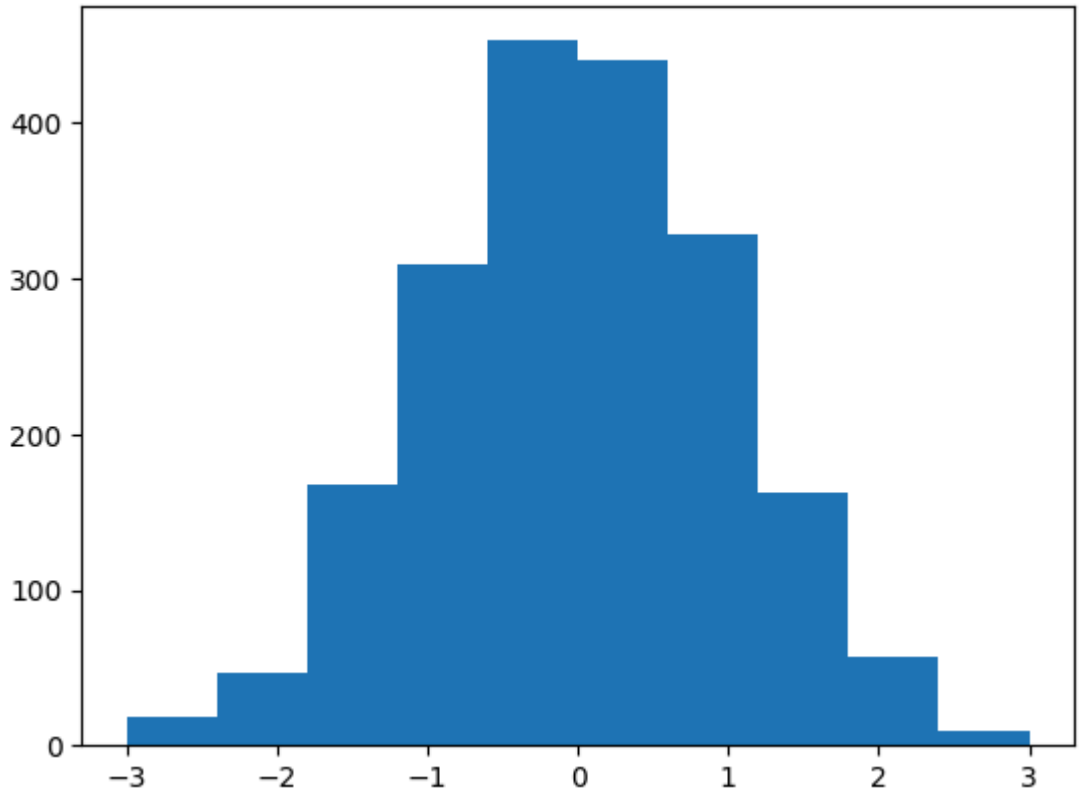
```
8
9
10
11
```

Náhodné generátory v numpy

```
# Náhodný generátor
rng = np.random.default_rng()
print(rng.random(10))
plt.hist(rng.random(10000), bins=21)
---
[0.11413139 0.91544601 0.31044065 0.96895694 0.64600164 0.08063942
 0.65753678 0.94372576 0.96397951 0.88515821]

(array([479., 471., 431., 458., 470., 485., 489., 453., 499., 494., 502.,
        493., 474., 472., 478., 478., 485., 484., 465., 475., 465.]),
 array([3.97768540e-05, 4.76501326e-02, 9.52604884e-02, 1.42870844e-01,
        1.90481200e-01, 2.38091556e-01, 2.85701912e-01, 3.33312267e-01,
        3.80922623e-01, 4.28532979e-01, 4.76143335e-01, 5.23753691e-01,
        5.71364046e-01, 6.18974402e-01, 6.66584758e-01, 7.14195114e-01,
        7.61805469e-01, 8.09415825e-01, 8.57026181e-01, 9.04636537e-01,
        9.52246893e-01, 9.99857248e-01]),
 <BarContainer object of 21 artists>)
```



```
plt.hist(rng.standard_normal(2000), bins=np.linspace(-3,3,11,endpoint=True))
---
(array([ 18.,  47., 168., 309., 453., 441., 329., 162.,  57.,   9.]),
 array([-3. , -2.4, -1.8, -1.2, -0.6,  0. ,  0.6,  1.2,  1.8,  2.4,  3. ]),
 <BarContainer object of 10 artists>)
```



Lineární algebra

```
a = np.arange(10).reshape(5,2)
print(a)
b = np.arange(6).reshape(2,3)
print(b)
print(np.matmul(a, b))
---
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]

[[0 1 2]
 [3 4 5]]

[[ 3  4  5]
 [ 9 14 19]
 [15 24 33]
 [21 34 47]
 [27 44 61]]
```

```
c = np.array([1,1,0,0,-1,1,0,0,1]).reshape(3,3)
print(c)
print(np.linalg.inv(c))
print(np.matmul(c, np.linalg.inv(c)))
---
[[ 1  1  0]
 [ 0 -1  1]
 [ 0  0  1]]
[[ 1.  1. -1.]
 [-0. -1.  1.]
 [ 0.  0.  1.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

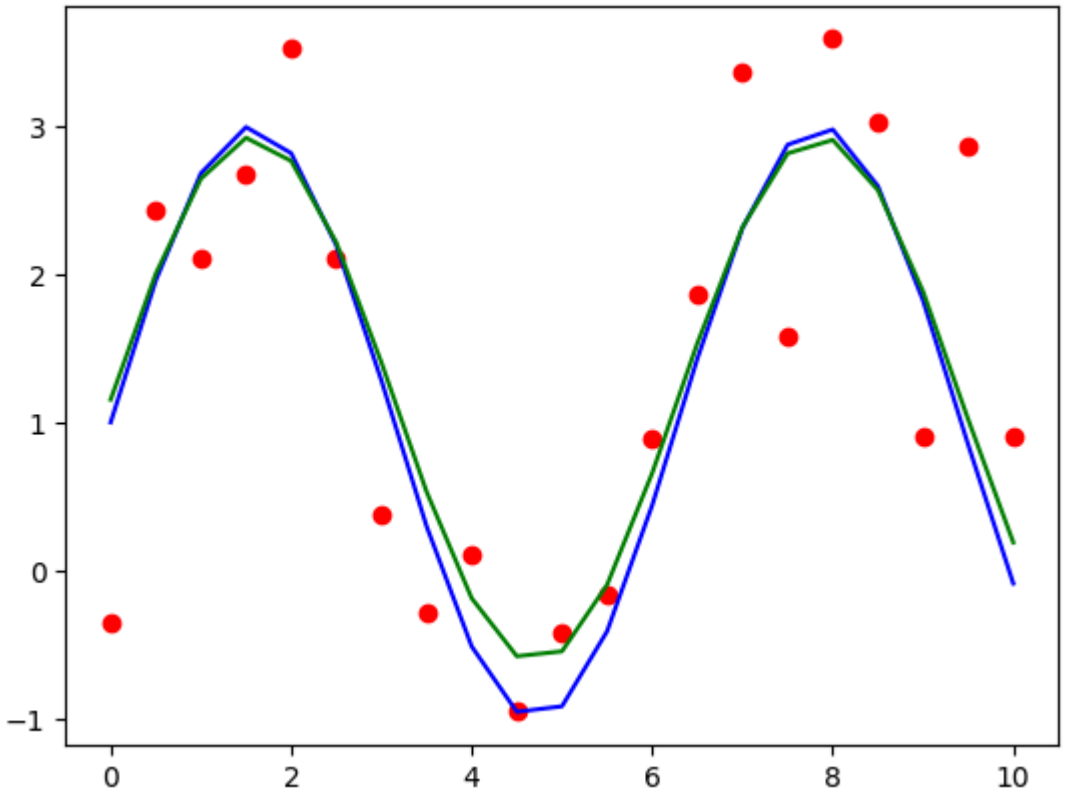
```
np.linalg.eig(c)
---
(array([ 1., -1.,  1.]),
 array([[ 1.00000000e+00, -4.47213595e-01, -1.00000000e+00],
        [ 0.00000000e+00,  8.94427191e-01,  2.22044605e-16],
        [ 0.00000000e+00,  0.00000000e+00,  4.44089210e-16]]))
```

```
print(c)
print(np.linalg.svd(c))
---
[[ 1  1  0]
 [ 0 -1  1]
 [ 0  0  1]]
(array([[ 0.59100905, -0.73697623,  0.32798528],
        [-0.73697623, -0.32798528,  0.59100905],
        [-0.32798528, -0.59100905, -0.73697623]]),
 array([1.80193774, 1.2469796 , 0.44504187]),
 array([[ 0.32798528,  0.73697623, -0.59100905],
        [-0.59100905, -0.32798528, -0.73697623],
        [ 0.73697623, -0.59100905, -0.32798528]]))
```

Lineární regrese

```
x = np.linspace(start=0,stop=10, num=21, endpoint = True)
y0 = 2.0*np.sin(x) + 1.0
y =  y0 + np.random.standard_normal(21)
plt.plot(x,y0, "b")
plt.scatter(x,y, color = "r")
F = np.ndarray((21,2))
F[:,0] = np.sin(x)
F[:,1] = np.repeat(1,21)
print(F)
rhs = y.reshape(21,1)
coeffs, sumsq, cond, sing = np.linalg.lstsq(F,y)
y_fit = np.matmul(F, coeffs)
plt.plot(x, y_fit, "g")
```

```
---
[[ 0.          1.        ]
 [ 0.47942554  1.        ]
 [ 0.84147098  1.        ]
 [ 0.99749499  1.        ]
 [ 0.90929743  1.        ]
 [ 0.59847214  1.        ]
 [ 0.14112001  1.        ]
 [-0.35078323  1.        ]
 [-0.7568025   1.        ]
 [-0.97753012  1.        ]
 [-0.95892427  1.        ]
 [-0.70554033  1.        ]
 [-0.2794155   1.        ]
 [ 0.21511999  1.        ]
 [ 0.6569866   1.        ]
 [ 0.93799998  1.        ]
 [ 0.98935825  1.        ]
 [ 0.79848711  1.        ]
 [ 0.41211849  1.        ]
 [-0.07515112  1.        ]
 [-0.54402111  1.        ]]

<ipython-input-69-72e06b7502cb>:11: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  coeffs, sumsq, cond, sing = np.linalg.lstsq(F,y)

[<matplotlib.lines.Line2D at 0x7880ef1d02b0>]
```



pandas

Modul `pandas` podporuje datové tabulky (*DataFrame*) a operace nad nimi.

1. Přístup

Datové tabulky jsou myšleny jako read-only, tedy se nepředpokládá, že budete chtít měnit hodnoty položek. Můžete ale různě přeskupovat data a přidávat nové sloupce či souhrny. Základní mód zpracování:

$$\text{group} \rightarrow \text{apply} \rightarrow \text{combine}$$

2. Struktura

Datové tabulky jsou uchovávány po sloupcích. Proto přidání sloupce je jednoduché, ale přidání řádku je velice časově náročné.

Pro práci s tabulkami se musíte naučit základní filozofii a idiomy, které se poněkud liší od práce se seznamem seznamů v Pythonu.

Data Wrangling

with pandas Cheat Sheet
http://pandas.pydata.org

Pandas API Reference Pandas User Guide

Tidy Data – A foundation for wrangling in pandas

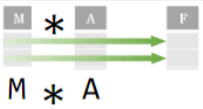
In a tidy data set:

Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4, 5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
N	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

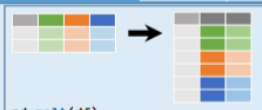
```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```
Create DataFrame with a MultiIndex

Method Chaining

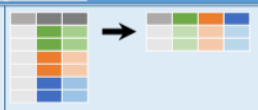
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
    )
```

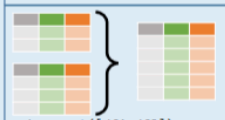
Reshaping Data – Change layout, sorting, reindexing, renaming



`pd.melt(df)`
Gather columns into rows.



`df.pivot(columns='var', values='val')`
Spread rows into columns.



`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg', ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length', 'Height'])`
Drop columns from DataFrame

Subset Observations - rows

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)` Randomly select n rows.

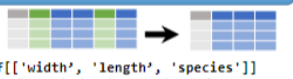
`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

Subset Variables - columns



`df[['width', 'length', 'species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use `df.loc[]` and `df.iloc[]` to select only rows, only columns or both.
Use `df.at[]` and `df.iat[]` to access a single value by row and column.
First index selects rows, second index columns.

`df.iloc[10:20]`
Select rows 10-20.

`df.iloc[:, [1, 2, 5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[:, 'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.loc[df['a'] > 10, ['a', 'c']]`
Select rows meeting logical condition, and only the specific columns .

`df.iat[1, 2]` Access single value by index

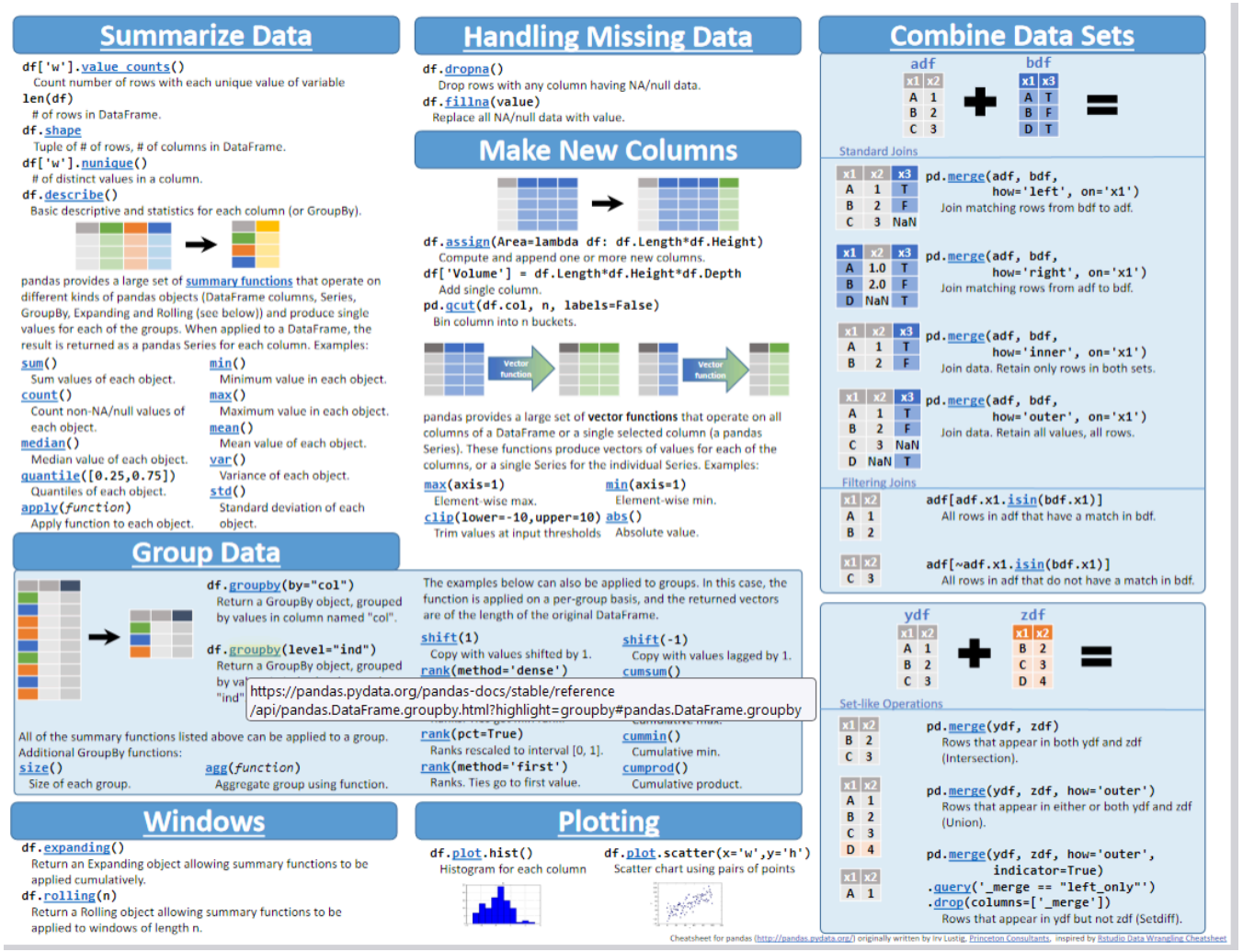
`df.at[4, 'A']` Access single value by label

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples

`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'




```python
import pandas as pd

data = pd.DataFrame({
    "sex": np.random.choice(["M","F"], 100),
    "group": np.random.choice(["A", "B", "C", "D"], 100),
    "age": np.random.uniform(low = 18, high=90, size = 100)
}, index = np.arange(100)
)
data["weight"] = data.apply(lambda row: 80 if row.sex == "M" else 60, axis = 1) + 10*np.random.standard_normal(100)
print(data)
```

```
---
   sex group        age     weight
0    M     A  27.189528  80.531588
1    M     B  82.141697  91.257771
2    F     B  50.923255  56.787014
3    F     D  53.439978  62.726728
4    M     D  36.728620  94.885223
..  ..   ...        ...        ...
95   F     C  80.382851  57.107162
96   M     D  23.650393  75.769245
97   F     C  30.025270  61.050458
98   M     C  42.891945  70.084252
99   M     C  20.146695  93.561390
```

```
[100 rows x 4 columns]
```

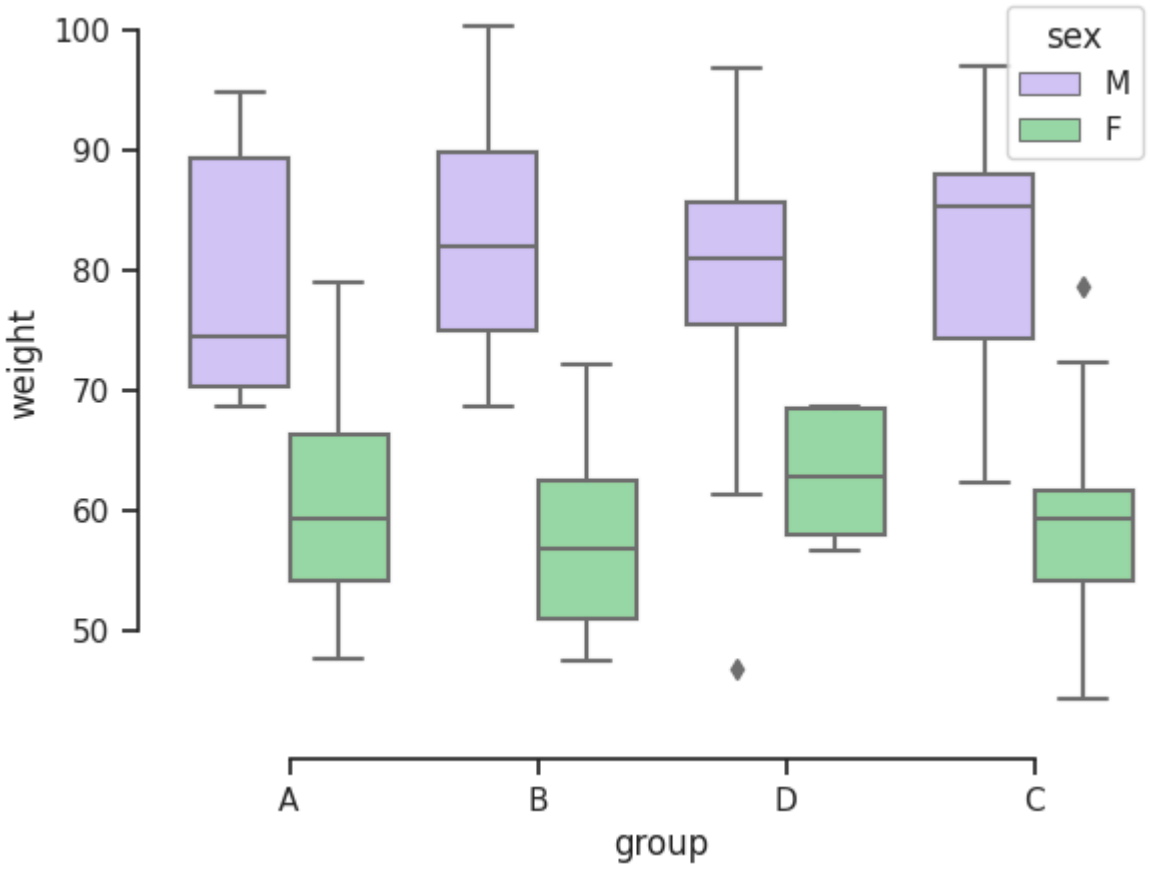
```
data.pivot_table(values="weight", index="group", columns = "sex", aggfunc= (len, np.mean, np.std))
```

	len		mean		std	
sex	F	M	F	M	F	M
group						
A	15	10	60.004561	78.863990	8.459412	10.462918
B	15	13	57.395915	82.179252	7.277524	9.864408
C	14	9	59.403209	82.063862	8.818018	11.242939
D	7	17	62.933266	79.355646	5.549616	12.609093

Kreslení pomocí modulu seaborn

Modul `seaborn` obsahuje bohatou galerii grafů pro zkoumání vztahů dat v tabulkách pandas.

```
import seaborn as sns
sns.set_theme(style="ticks", palette="pastel")

# Draw a nested boxplot
sns.boxplot(x="group", y="weight",
            hue="sex", palette=["m", "g"],
            data=data)
sns.despine(offset=10, trim=True)
```



Závěr

Tento přehledy byl nutně velice stručný a je to spíš přehled toho, co je k dispozici než instruktáž jak to používat.