

Programování 1 pro matematiky

13. cvičení, 06-01-2025

Obsah:

0. Farní oznamy
1. Úvod: numpy, pandas, matplotlib
2. Numpy
3. Pandas

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 víc**, 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:

```
1 class Matrix:  
2  
3     def __init__(self, ll): # ll je seznam seznamů  
4         self.matrix = ll  
5  
6     def __repr__(self):  
7         return '\n'.join(' '.join(str(val) for val in row) for row in self.matrix)  
8  
9     def vals(self):  
10        return self.matrix  
11  
12    def dims(self):  
13        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 základ 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.

Dokumentace: [www.numpy.org](https://numpy.org)

The screenshot shows the official NumPy website at <https://numpy.org>. The header includes a navigation bar with links for Install, Documentation, Learn, Community, About Us, News, Contribute, and English language selection. Below the header is the NumPy logo and the tagline "The fundamental package for scientific computing with Python". A dark banner at the top of the main content area announces "NumPy 1.26.0 released" on "2023-09-16". The main content is organized into six boxes: "POWERFUL N-DIMENSIONAL ARRAYS", "NUMERICAL COMPUTING TOOLS", "OPEN SOURCE", "INTEROPERABLE", "PERFORMANT", and "EASY TO USE". Each box contains a brief description of NumPy's capabilities in that specific area.

POWERFUL N-DIMENSIONAL ARRAYS
Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

NUMERICAL COMPUTING TOOLS
NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

OPEN SOURCE
Distributed under a liberal [BSD license](#), NumPy is developed and maintained [publicly on GitHub](#) by a vibrant, responsive, and diverse [community](#).

INTEROPERABLE
NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

PERFORMANT
The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

EASY TO USE
NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

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`.



Fundamental algorithms for scientific computing in Python

GET STARTED

SciPy 1.11.4 released! 2023-11-18

FUNDAMENTAL ALGORITHMS

SciPy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems.

BROADLY APPLICABLE

The algorithms and data structures provided by SciPy are broadly applicable across domains.

FOUNDATIONAL

Extends NumPy providing additional tools for array computing and provides specialized data structures, such as sparse matrices and k-dimensional trees.

PERFORMANT

SciPy wraps highly-optimized implementations written in low-level languages like Fortran, C, and C++. Enjoy the flexibility of Python with the speed of compiled code.

EASY TO USE

SciPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

OPEN SOURCE

Distributed under a liberal [BSD license](#), SciPy is developed and maintained [publicly on GitHub](#) by a vibrant, responsive, and diverse [community](#).

- **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

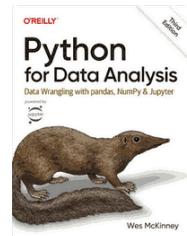
pandas.pydata.org

Latest version: 2.1.4

- What's new in 2.1.4
- Release date:
Dec 08, 2023
- Documentation (web)
- Download source code

Follow us

Get the book



Getting started

- Install pandas
- Getting started

Documentation

- User guide
- API reference
- Contributing to pandas
- Release notes

Community

- About pandas
- Ask a question
- Ecosystem

With the support of:



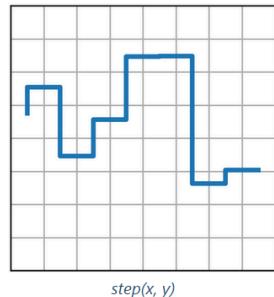
Previous versions

- 2.0.3 (Jun 28, 2023)
[changelog](#) | [docs](#) | [code](#)
- 1.5.3 (Jan 19, 2023)
[changelog](#) | [docs](#) | [code](#)

- **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

Matplotlib: Visualization with Python



Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

[Try Matplotlib \(on Binder\)](#)



Getting Started



Examples



Reference



Cheat Sheets



Documentation

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.

The screenshot shows the official Seaborn website at <https://seaborn.pydata.org/index.html>. The page features a header with a search bar and social media links. Below the header is a section titled "seaborn: statistical data visualization" displaying six different types of plots: density plots, scatter plots, contour plots, and box plots. To the left of the plots is a brief introduction text. To the right are two boxes: "Contents" listing links to "Installing", "Gallery", "Tutorial", "API", "Releases", "Citing", and "FAQ"; and "Features" listing "Relational plots", "Distribution plots", "Categorical plots", "Regression plots", "Multi-plot grids", "Figure theming", and "Color palettes". At the bottom of the page are copyright and version information.

© Copyright 2012-2023, Michael Waskom.

Created using Sphinx and the PyData Theme.

v0.13.1

[Archive](#)

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:

```
1 # Vytváříme numpy pole:  
2  
3 # 1. Konverzí seznamu a úpravou tvaru  
4 a = np.arange(12).reshape(3,4)  
5 print("a", a)  
6  
7 a [[ 0  1  2  3]  
8  [ 4  5  6  7]  
9  [ 8  9 10 11]]  
10  
11 # 2. zadáním dimenze: prázdné pole  
12 b = np.ndarray((3,4))  
13 print("b", b)  
14  
15 b [[0.0e+000 4.9e-324 9.9e-324 1.5e-323]  
16  [2.0e-323 2.5e-323 3.0e-323 3.5e-323]  
17  [4.0e-323 4.4e-323 4.9e-323 5.4e-323]]  
18  
19 # 3. Specializovaným konstruktorem  
20 c = np.zeros((3,4))  
21 print("c",c)  
22  
23 c [[0. 0. 0. 0.]  
24  [0. 0. 0. 0.]  
25  [0. 0. 0. 0.]]  
26  
27 d = np.ones((3,4))  
28 print("d", d)  
29 d [[1. 1. 1. 1.]  
30  [1. 1. 1. 1.]  
31  [1. 1. 1. 1.]]
```

Vytváření sekvencí:

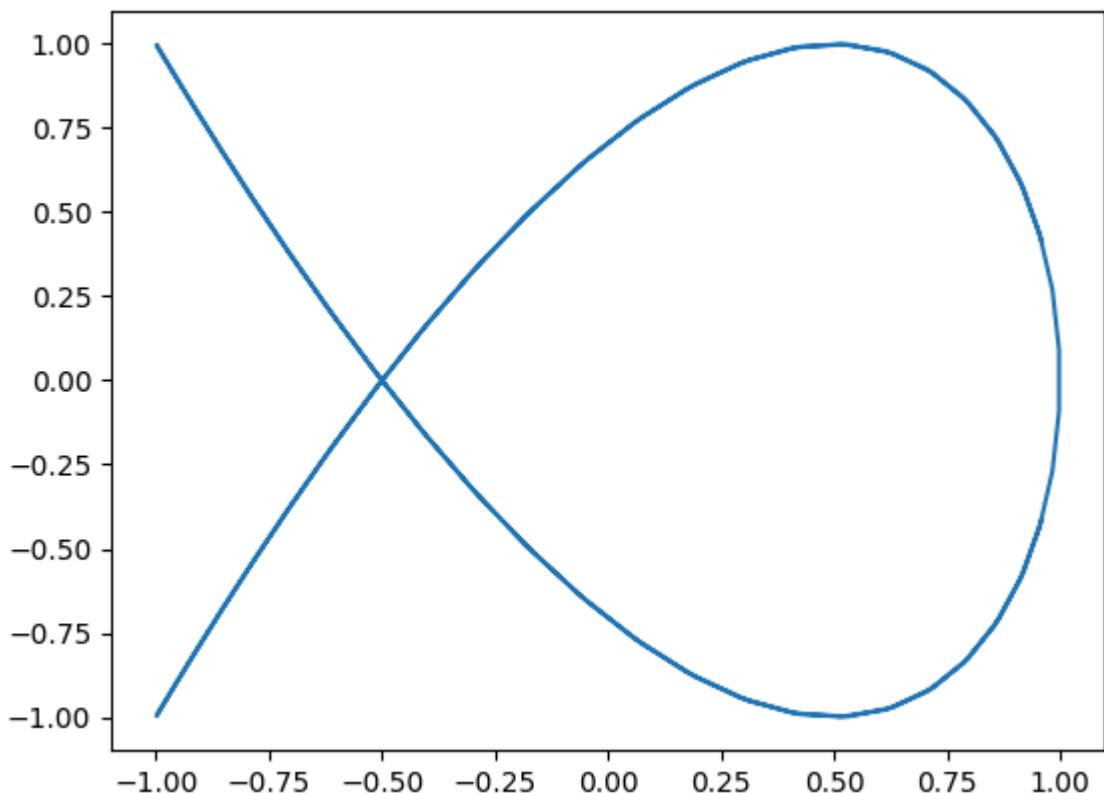
`np.arange` - jako range, ale vytváří np.array

`np.linspace` - pravidelné číselné řady:

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

Kreslíme

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

```
1 print(a)  
2 arev = np.arange(11,-1,-1).reshape(3,4)  
3 print(arev)  
4  
5 print(a+arev)  
6 print(a*arev)  
7 print(a / arev)  
8 ---  
9 [[ 0  1  2  3]  
10 [ 4  5  6  7]  
11 [ 8  9 10 11]]  
12 [[11 10  9  8]  
13 [ 7  6  5  4]  
14 [ 3  2  1  0]]  
15 [[11 11 11 11]  
16 [11 11 11 11]  
17 [11 11 11 11]]  
18 [[ 0 10 18 24]]
```

```

19 [28 30 30 28]
20 [24 18 10  0]]
21 [[ 0.          0.1          0.22222222  0.375      ]
22 [ 0.57142857  0.83333333  1.2          1.75       ]
23 [ 2.66666667  4.5          10.          inf]]
24
25 <ipython-input-20-1a385f80933a>:7: RuntimeWarning: divide by zero encountered in divide
26     print(a / arev)
27

```

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

Co když nesouhlasí rozměry?

```

1 v = np.arange(1,4).reshape(3,1)
2 print(v)
3 print(a*v)
4 print(v*a)
5 ---
6 [[1]
7 [2]
8 [3]]
9 [[ 0  1  2  3]
10 [ 8 10 12 14]
11 [24 27 30 33]]
12 [[ 0  1  2  3]
13 [ 8 10 12 14]
14 [24 27 30 33]]

```

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

Iterace přes pole:

```

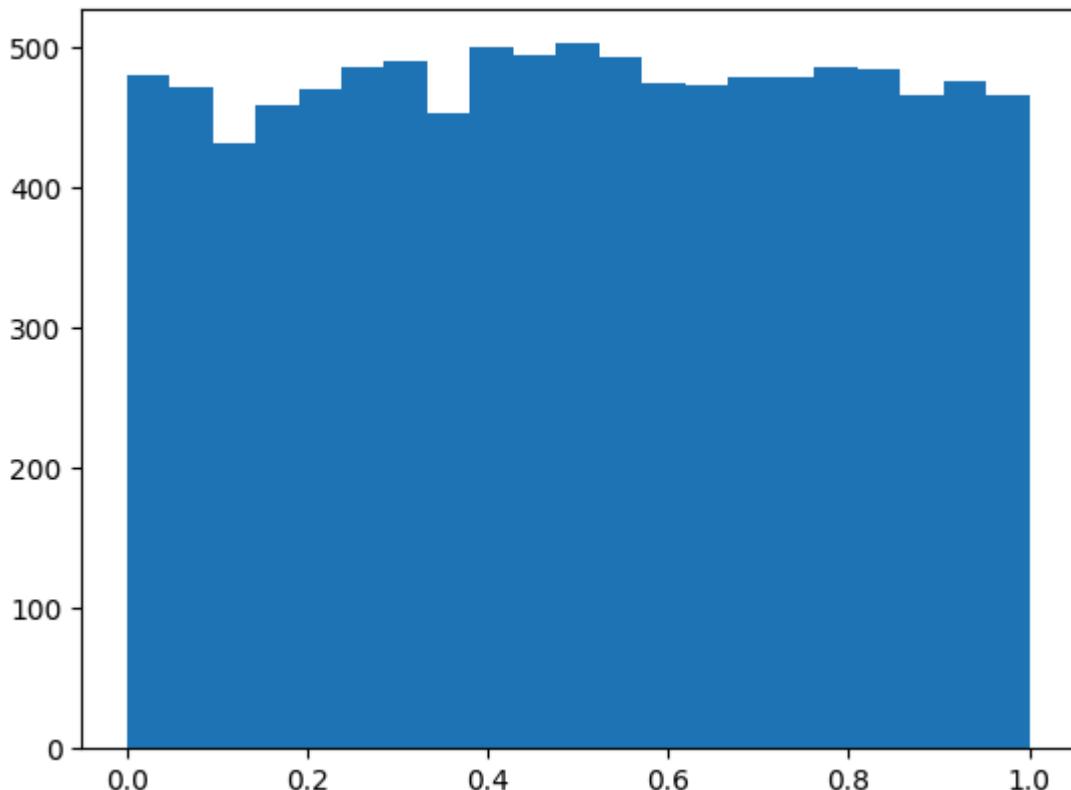
1 for i in a:
2     print(i)
3
4 for i in np.nditer(a):
5     print(i)
6 ---
7 [0 1 2 3]
8 [4 5 6 7]
9 [ 8  9 10 11]
10 0
11 1
12 2
13 3
14 4
15 5
16 6
17 7

```

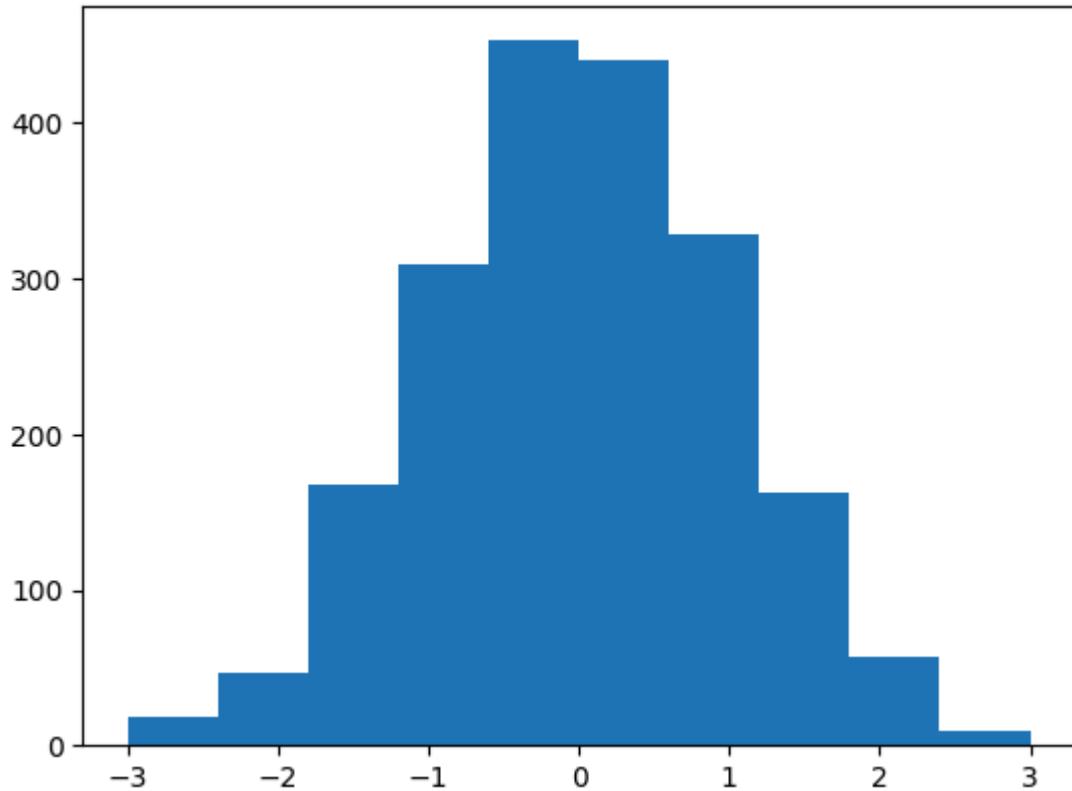
```
18 | 8
19 | 9
20 | 10
21 | 11
```

Náhodné generátory v numpy

```
1 # Náhodný generátor
2 rng = np.random.default_rng()
3 print(rng.random(10))
4 plt.hist(rng.random(10000), bins=21)
5 ---
6 [0.11413139 0.91544601 0.31044065 0.96895694 0.64600164 0.08063942
7 0.65753678 0.94372576 0.96397951 0.88515821]
8
9 (array([479., 471., 431., 458., 470., 485., 489., 453., 499., 494.,
10      502., 493., 474., 472., 478., 478., 485., 484., 465., 475., 465.]),
11 array([3.97768540e-05, 4.76501326e-02, 9.52604884e-02, 1.42870844e-01,
12      1.90481200e-01, 2.38091556e-01, 2.85701912e-01, 3.33312267e-01,
13      3.80922623e-01, 4.28532979e-01, 4.76143335e-01, 5.23753691e-01,
14      5.71364046e-01, 6.18974402e-01, 6.66584758e-01, 7.14195114e-01,
15      7.61805469e-01, 8.09415825e-01, 8.57026181e-01, 9.04636537e-01,
16      9.52246893e-01, 9.99857248e-01]),  
17 <BarContainer object of 21 artists>)
```



```
1 plt.hist(rng.standard_normal(2000), bins=np.linspace(-3,3,11,endpoint=True))
2 ---
3 array([ 18.,  47., 168., 309., 453., 441., 329., 162.,  57.,   9.]),
4 array([-3. , -2.4, -1.8, -1.2, -0.6,  0. ,  0.6,  1.2,  1.8,  2.4,  3. ]),
5 <BarContainer object of 10 artists>
```



Lineární algebra

```
1 a = np.arange(10).reshape(5,2)
2 print(a)
3 b = np.arange(6).reshape(2,3)
4 print(b)
5 print(np.matmul(a, b))
6 ---
7 [[0 1]
8 [2 3]
9 [4 5]
10 [6 7]
11 [8 9]]
12
13 [[0 1 2]
14 [3 4 5]]
15
16 [[ 3  4  5]
17 [ 9 14 19]
18 [15 24 33]
19 [21 34 47]
20 [27 44 61]]
```

```

1 c = np.array([1,1,0,0,-1,1,0,0,1]).reshape(3,3)
2 print(c)
3 print(np.linalg.inv(c))
4 print(np.matmul(c, np.linalg.inv(c)))
5 ---
6 [[ 1  1  0]
7 [ 0 -1  1]
8 [ 0  0  1]]
9 [[ 1.  1. -1.]
10 [-0. -1.  1.]
11 [ 0.  0.  1.]]
12 [[1.  0.  0.]
13 [0.  1.  0.]
14 [0.  0.  1.]]
```

```

1 np.linalg.eig(c)
2 ---
3 (array([[ 1., -1.,  1.]),
4 array([[ 1.0000000e+00, -4.47213595e-01, -1.0000000e+00],
5 [ 0.0000000e+00,  8.94427191e-01,  2.22044605e-16],
6 [ 0.0000000e+00,  0.0000000e+00,  4.44089210e-16]]))
```

```

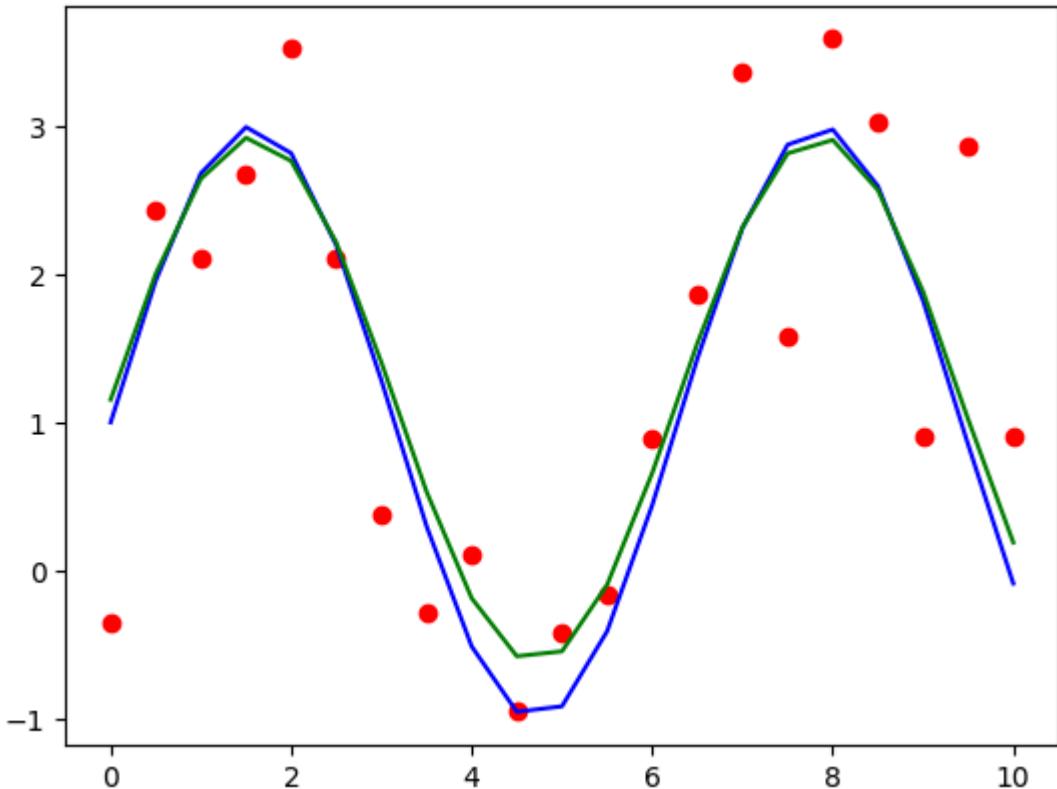
1 print(c)
2 print(np.linalg.svd(c))
3 ---
4 [[ 1  1  0]
5 [ 0 -1  1]
6 [ 0  0  1]]
7 (array([[ 0.59100905, -0.73697623,  0.32798528],
8 [ -0.73697623, -0.32798528,  0.59100905],
9 [ -0.32798528, -0.59100905, -0.73697623]]),
10 array([1.80193774,  1.2469796 ,  0.44504187]),
11 array([[ 0.32798528,  0.73697623, -0.59100905],
12 [ -0.59100905, -0.32798528, -0.73697623],
13 [ 0.73697623, -0.59100905, -0.32798528]]))
```

Lineární regrese

```

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

```
14 ---  
15 [[ 0. 1.  
16 [ 0.47942554 1.  
17 [ 0.84147098 1.  
18 [ 0.99749499 1.  
19 [ 0.90929743 1.  
20 [ 0.59847214 1.  
21 [ 0.14112001 1.  
22 [-0.35078323 1.  
23 [-0.7568025 1.  
24 [-0.97753012 1.  
25 [-0.95892427 1.  
26 [-0.70554033 1.  
27 [-0.2794155 1.  
28 [ 0.21511999 1.  
29 [ 0.6569866 1.  
30 [ 0.93799998 1.  
31 [ 0.98935825 1.  
32 [ 0.79848711 1.  
33 [ 0.41211849 1.  
34 [-0.07515112 1.  
35 [-0.54402111 1. ]]  
36  
37 <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.  
38 To use the future default and silence this warning we advise to pass `rcond=None`, to  
keep using the old, explicitly pass `rcond=-1`.  
39     coeffs, sumsq, cond, sing = np.linalg.lstsq(F,y)  
40  
41 [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:

`group → apply → 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 idiomu, 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](#)

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.
```

N	a	b	c
0	1	4	7
1	2	5	8
2	3	6	9

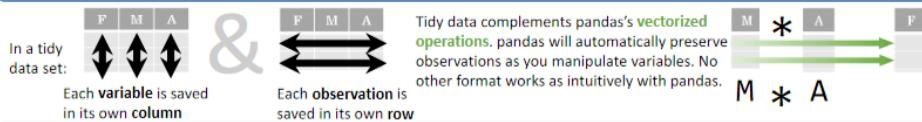
```
df = pd.DataFrame(
    {"a" : [4, 5, 6],
     "b" : [7, 8, 9],
     "c" : [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
Create DataFrame with a MultiIndex
```

Method Chaining

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

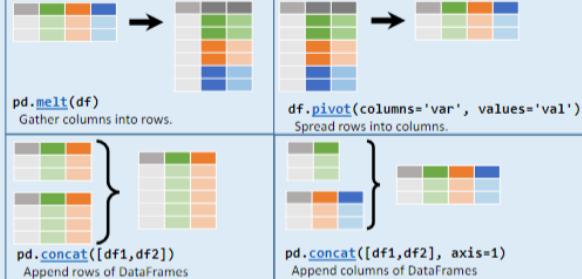
```
df = (pd.melt(df)
      .rename(columns={'variable':'var',
                      'value':'val'})
      .query('val > 200')
     )
```

Tidy Data – A foundation for wrangling in pandas



M * A

Reshaping Data – Change layout, sorting, reindexing, renaming



```
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.loc[:, [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.all(), df.any(), ~df.all(), ~df.any()	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
'^(?i:Species)\$' or '^Species\$'	Matches strings except the string 'Species'

Chartsheet for pandas: <http://pandas.pydata.org/pandas-docs/stable/releasenotes/0.13.0.html#cheatsheets> – originally written by Joe Kington, Cédric Beaumont – inspired by Brett Victor's [MicroCheatsheets](http://www.brettvictor.com/cheatsheets).

Summarize Data

```
df['w'].value_counts()
Count number of rows with each unique value of variable
len(df)
# of rows in DataFrame.
df.shape
Tuple of # of rows, # of columns in DataFrame.
df['w'].unique()
# of distinct values in a column.
df.describe()
Basic descriptive and statistics for each column (or GroupBy).
```

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

- sum()**: Sum values of each object.
- count()**: Count non-NA/null values of each object.
- median()**: Median value of each object.
- quantile([0.25, 0.75])**: Quantiles of each object.
- apply(function)**: Apply function to each object.

Group Data

```
df.groupby(by="col")
Return a GroupBy object, grouped by values in column named "col".
df.groupby(level="ind")
Return a GroupBy object, grouped by var "ind"
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html
```

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

- size()**: Size of each group.
- agg(function)**: Aggregate group using function.

Windows

```
df.expanding()
Return an Expanding object allowing summary functions to be applied cumulatively.
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.
```

Handling Missing Data

```
df.dropna()
Drop rows with any column having NA/null data.
df.fillna(value)
Replace all NA/null data with value.
```

Make New Columns

```
df.assign(Area=lambda df: df.Length*df.Height)
Compute and append one or more new columns.
df['Volume'] = df.Length*df.Height*df.Depth
Add single column.
pd.qcut(df.col, n, labels=False)
Bin column into n buckets.
```

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

max(axis=1)	min(axis=1)
Element-wise max.	Element-wise min.

```
clip(lower=-10,upper=10) abs()
Trim values at input thresholds Absolute value.
```

Combine Data Sets

adf	x1 x2	bdf	x1 x3
A	1	A	T
B	2	B	F
C	3	D	T

Standard Joins

x1 x2 x3	pd.merge(adf, bdf, how='left', on='x1')
A 1 T	Join matching rows from bdf to adf.
B 2 F	
C 3 NaN	

x1 x2 x3	pd.merge(adf, bdf, how='right', on='x1')
A 1.0 T	Join matching rows from adf to bdf.
B 2.0 F	
D NaN T	

x1 x2 x3	pd.merge(adf, bdf, how='inner', on='x1')
A 1 T	Join data. Retain only rows in both sets.
B 2 F	

x1 x2 x3	pd.merge(adf, bdf, how='outer', on='x1')
A 1 T	Join data. Retain all values, all rows.
B 2 F	
C 3 NaN	
D NaN T	

Filtering Joins

x1 x2	adf[adf.x1.isin(bdf.x1)]
A 1	All rows in adf that have a match in bdf.
B 2	

x1 x2	adf[~adf.x1.isin(bdf.x1)]
C 3	All rows in adf that do not have a match in bdf.

ydf	x1 x2	zdf	x1 x2
A 1	1	B 2	2
B 2	2	C 3	3
C 3	3	D 4	4

Set-like Operations

x1 x2	pd.merge(ydf, zdf)
A 1	Rows that appear in both ydf and zdf (Intersection).
B 2	
C 3	

x1 x2	pd.merge(ydf, zdf, how='outer')
A 1	Rows that appear in either or both ydf and zdf (Union).
B 2	
C 3	
D 4	

x1 x2	pd.merge(ydf, zdf, how='outer', indicator=True)
A 1	.query('_merge == "left_only"')
	.drop(columns=['_merge'])

Rows that appear in ydf but not zdf (Setdiff).

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

shift(1)	shift(-1)
Copy with values shifted by 1.	Copy with values lagged by 1.

rank(method='dense')

rank(pct=True)	cummin()
Ranks rescaled to interval [0, 1].	Cumulative min.

rank(method='first')

Ranks. Ties go to first value.	cumprod()
	Cumulative product.

df.plot.hist()

Histogram for each column



Plotting

df.plot.scatter(x='w', y='h')

Scatter chart using pairs of points



Cheatsheet for pandas (<http://pandas.pydata.org/>) originally written by Irv Lustig, Princeton Consultants, inspired by RStudio Data Wrangling CheatSheet

```
1 import pandas as pd
2
3 data = pd.DataFrame({
4     "sex": np.random.choice(["M", "F"], 100),
5     "group": np.random.choice(["A", "B", "C", "D"], 100),
6     "age": np.random.uniform(low = 18, high=90, size = 100)
7 }, index = np.arange(100))
8
9 data["weight"] = data.apply(lambda row: 80 if row.sex == "M" else 60, axis = 1) +
10 *np.random.standard_normal(100)
11 print(data)
12
13   sex group        age      weight
14 0    M     A  27.189528  80.531588
15 1    M     B  82.141697  91.257771
16 2    F     B  50.923255  56.787014
17 3    F     D  53.439978  62.726728
18 4    M     D  36.728620  94.885223
19 ...
20 95   F     C  80.382851  57.107162
21 96   M     D  23.650393  75.769245
22 97   F     C  30.025270  61.050458
23 98   M     C  42.891945  70.084252
24 99   M     C  20.146695  93.561390
```

```
24  
25 [100 rows x 4 columns]  
26
```

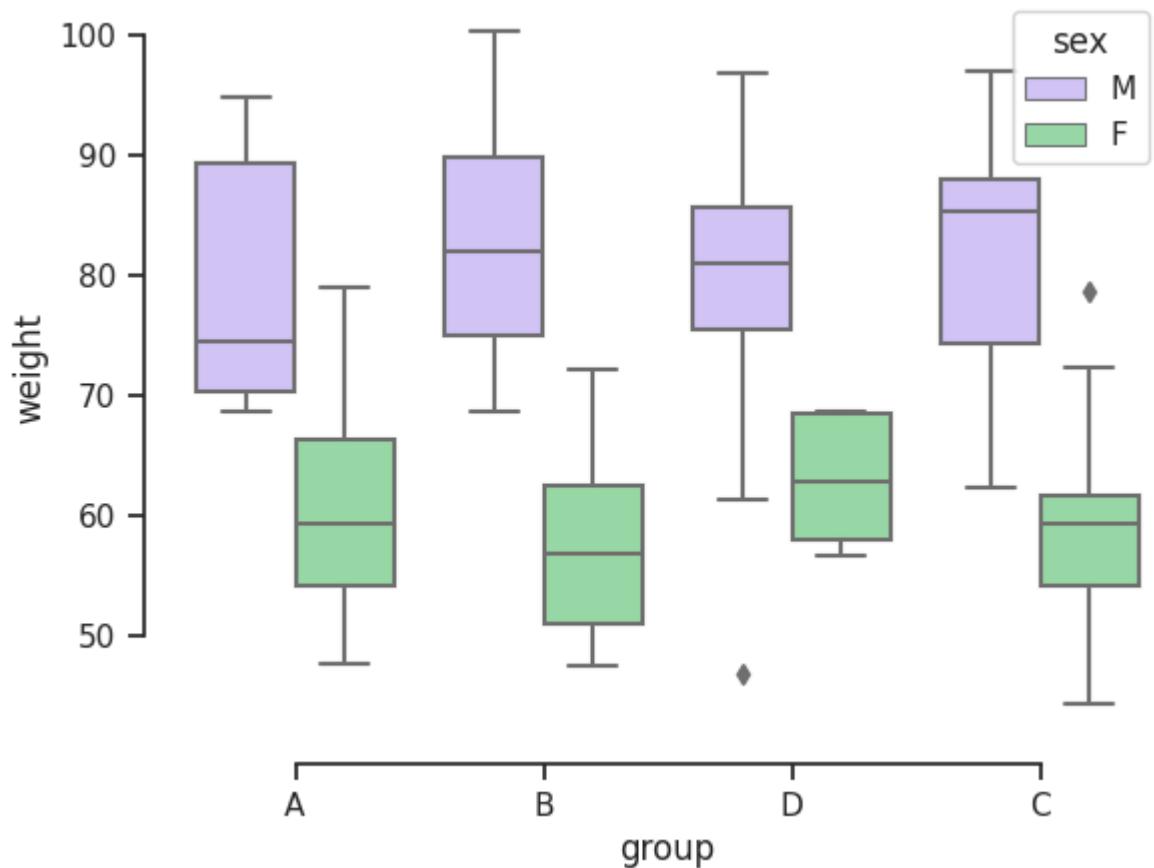
```
1 | data.pivot_table(values="weight", index="group", columns = "sex", aggfunc= (len,  
np.mean, np.std))
```

group	sex	len		mean		std	
		F	M	F	M	F	M
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.

```
1 | import seaborn as sns  
2 | sns.set_theme(style="ticks", palette="pastel")  
3 |  
4 | # Draw a nested boxplot  
5 | sns.boxplot(x="group", y="weight",  
6 |               hue="sex", palette=["m", "g"],  
7 |               data=data)  
8 | 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.