

13. cvičení, 010-01-2024

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 vám chybí body k zápočtu, dobrý čas s tím něco dělat je **ted**.
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.

Úvod

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

Bookmarks Menu Evidence fyzikálního p... Jednoznakovky SIS-4 Python Programování 2 Píchačky Contexto Phind: AI DoucMa.sk DesmosC

Install Documentation Learn Community About Us News Contribute English

NumPy

The fundamental package for scientific computing with Python

LATEST RELEASE: NUMPY 1.26. VIEW ALL RELEASES

NumPy 1.26.0 released 2023-09-16

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`.

https://scipy.org

marks Menu Evidence fyzikálního p... Jednoznakovky SIS-4 Python Programování 2 Píchačky Contexto Phind: AI DoucMa.sk DesmosC

Install Documentation Community About Us Contribute

SciPy

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 blíže než k maticím mají k Excelovským listům

pandas

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

[Install pandas now!](#)

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:

NUMFOCUS, TWO SIGMA, VOLTRON DATA, Coiled, Quansight Labs, NVIDIA.

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

Plot types User guide Tutorials Examples Reference Contribute Releases

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

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.

Instalace

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

Lepší je proto instalovat nějakou distribuci, která obsahuje tyto moduly a jejich dependence, například `anaconda`.

numpy

```
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:

```
import pandas as pd
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:

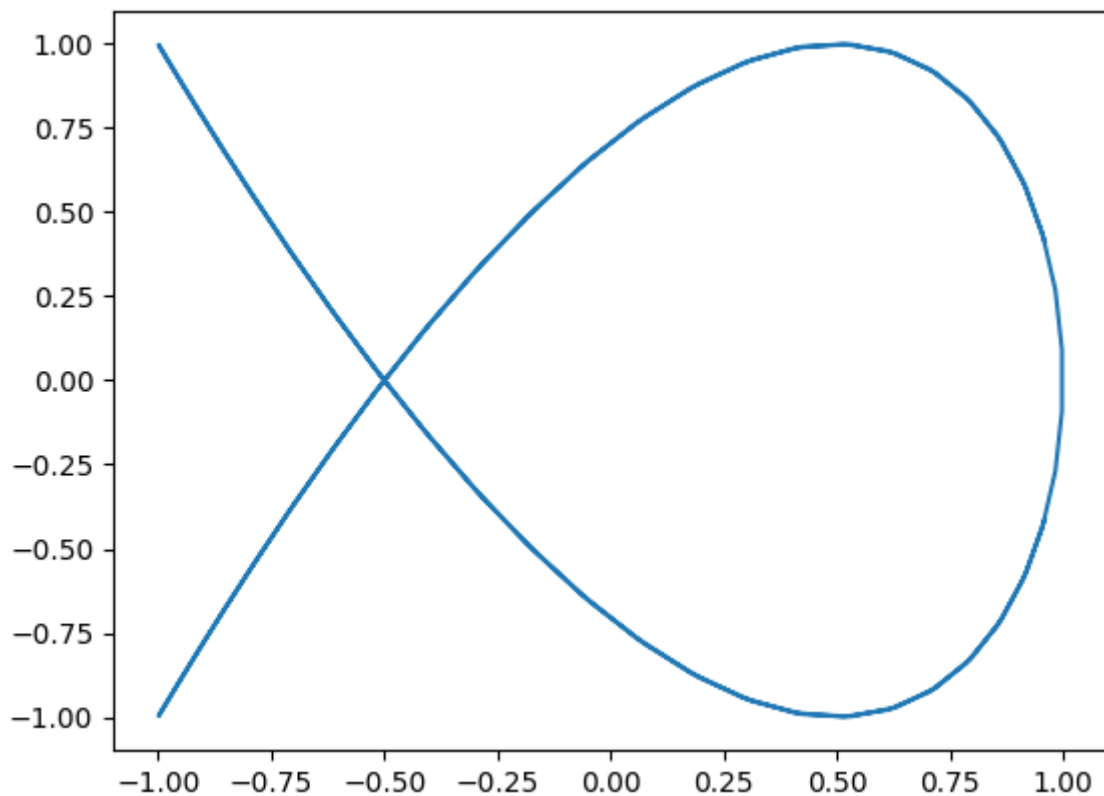
`np.arange` - jako range, ale vytváří `np.array`

`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 matickové 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]]
```

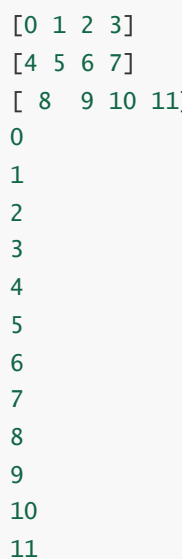
Chybějící data se inteligentně doplní (pravidla jsou velice komplexní) - *broadcasting*

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
```

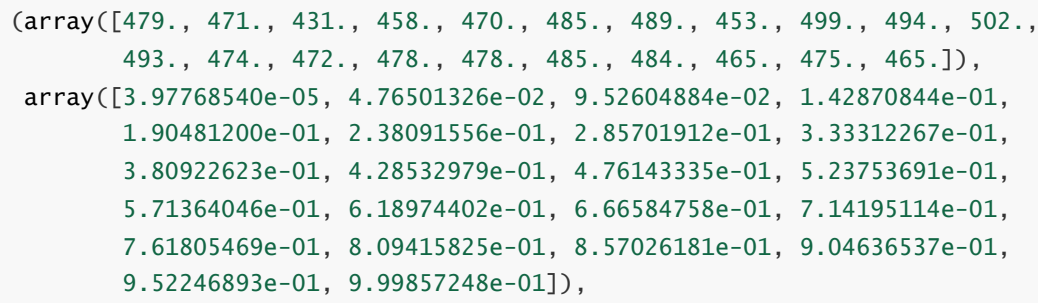
Náhodné generátory v numpy

```
# Náhodný generátor
rng = np.random.default_rng()
print(rng.random(10))
plt.hist(rng.random(10000), bins=21)

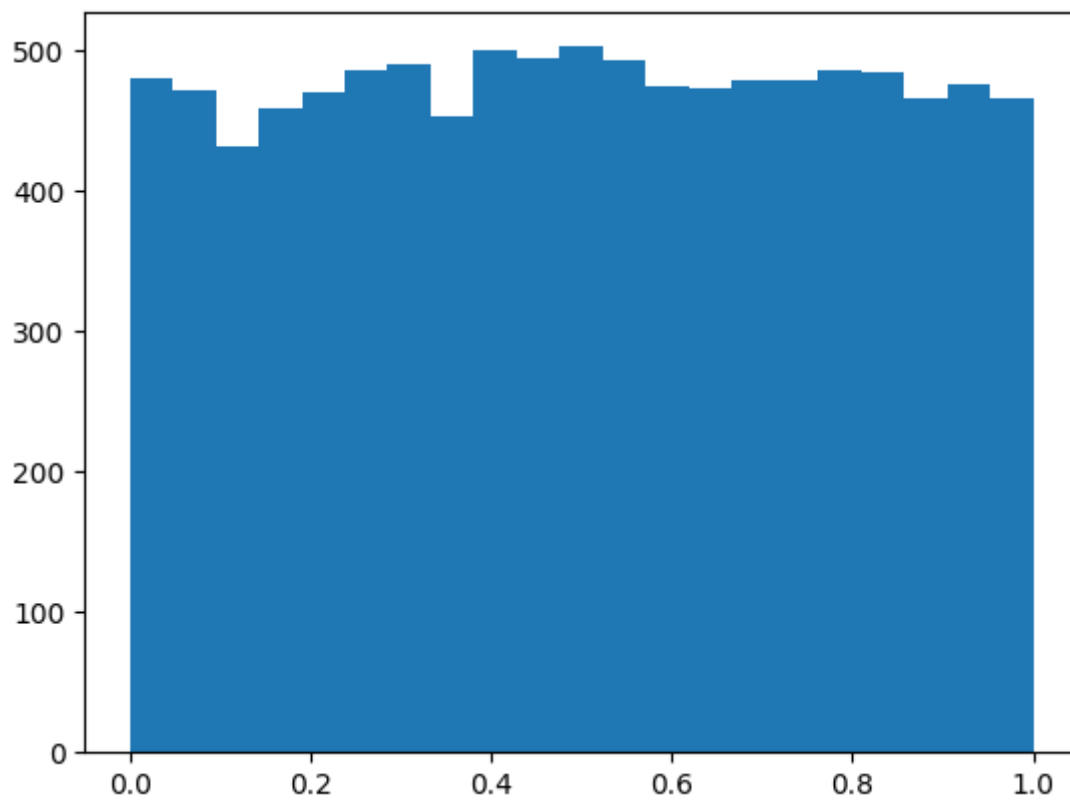
---
```



```
[0.11413139 0.91544601 0.31044065 0.96895694 0.64600164 0.08063942
 0.65753678 0.94372576 0.96397951 0.88515821]
```

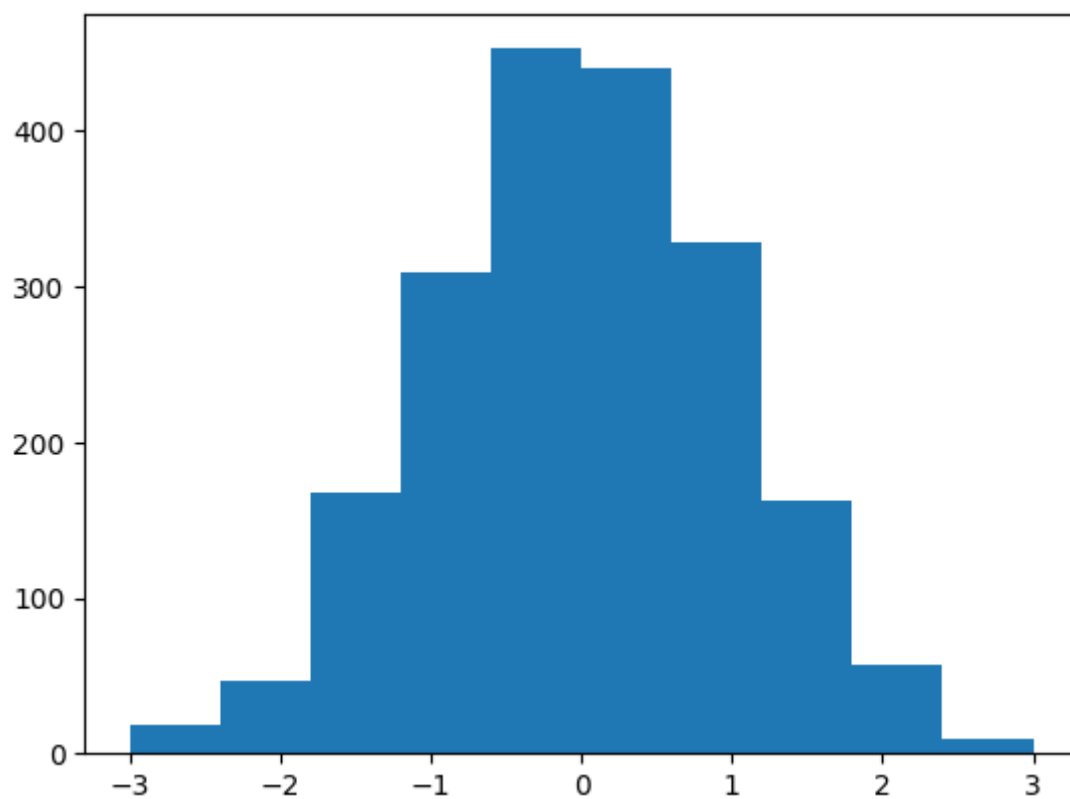



```
(array([479., 471., 431., 458., 470., 485., 489., 453., 499., 494., 502.,
        493., 474., 472., 478., 478., 485., 484., 465., 475., 465.]),
 array([3.97768540e-05, 4.76501326e-02, 9.52604884e-02, 1.42870844e-01,
        1.90481200e-01, 2.38091556e-01, 2.85701912e-01, 3.33312267e-01,
        3.80922623e-01, 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]]))
```

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 (1)

2. Struktura

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

Data Wrangling with pandas Cheat Sheet

<http://pandas.pydata.org>

[Pandas API Reference](#) [Pandas User Guide](#)

Creating DataFrames

	a	b	c
1	6	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
1	6	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 = 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.

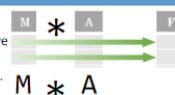
```
df = (pd.melt(df)
      .rename(columns={
          'variable': 'var',
          'value': 'val'})
      .query('val >= 200'))
```

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:
Each variable is saved in its own column

Each observation is saved in its own row

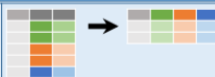
Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



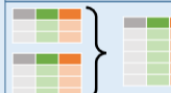
Reshaping Data – Change layout, sorting, reindexing, renaming



`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')`

	Logic in Python (and pandas)	
<	Less than	!=
>	Greater than	df.column.isin(values)
==	Equals	pd.isnull(obj)
<=	Less than or equals	pd.notnull(obj)
>=	Greater than or equals	df.isna(), df.any(), df.all()

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

Summarize Data

`df['v'].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['v'].nunique()`
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.

`min()`
Minimum value in each object.
`max()`
Maximum value in each object.
`mean()`
Mean value of each object.
`var()`
Variance of each object.
`std()`
Standard deviation of 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 values in column named 'ind'.

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.

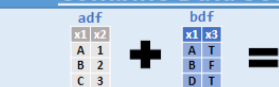
Make New Columns

`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 the individual Series for the individual Series. Examples:
`max(axis=1)`
Element-wise max.
`clip(lower=-10, upper=10)`
Trim values at input thresholds.
`min(axis=1)`
Element-wise min.
`abs()`
Absolute value.

Combine Data Sets



`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

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

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

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

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

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

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

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

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

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

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

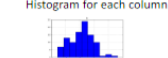
`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

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

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w', y='h')`
Scatter chart using pairs of points



```
import pandas as pd
```

```
data = pd.DataFrame({
    "sex": np.random.choice(["M", "F"], 100),
    "age": np.random.uniform(low = 18, high=90, size = 100),
```

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

```

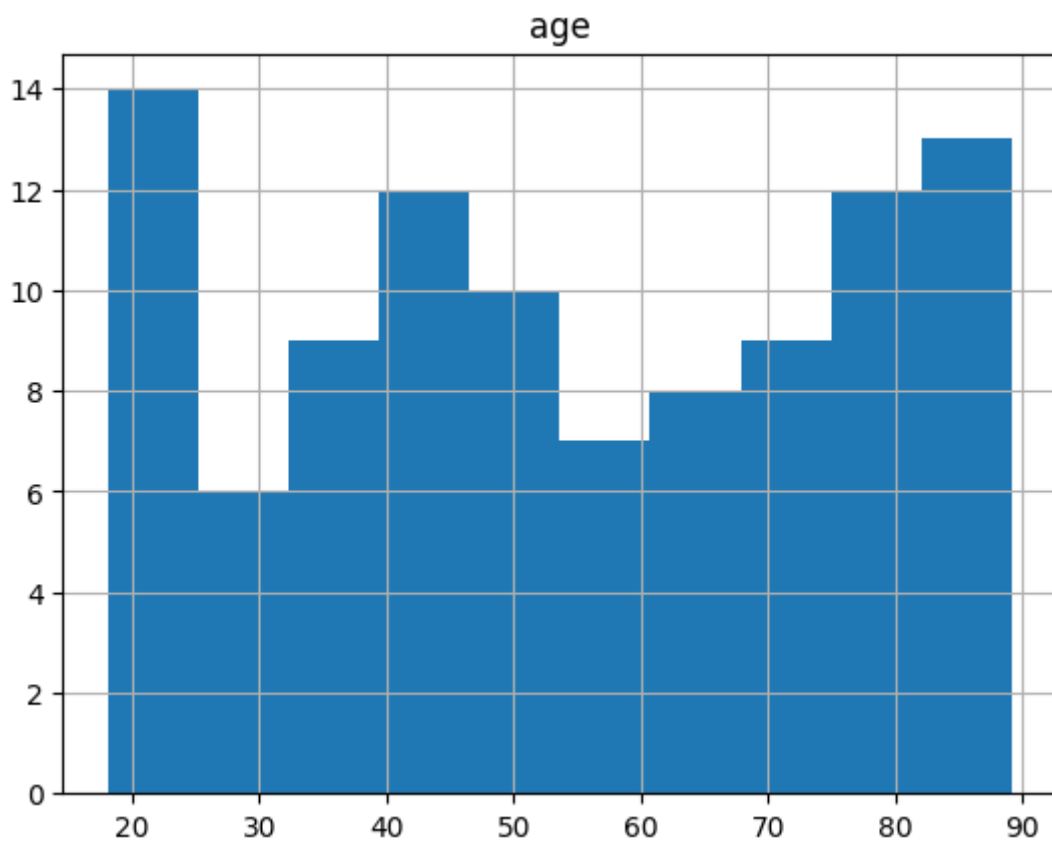
}, index = np.arange(100)
)
print(data)
data.hist("age")
---
```

	sex	age
0	M	67.273757
1	M	30.689785
2	M	52.553379
3	M	89.154159
4	F	47.129377
..
95	M	56.412083
96	M	73.289160
97	M	28.874982
98	M	66.894197
99	M	58.513988

```

[100 rows x 2 columns]

array([[<Axes: title={'center': 'age'}>]], dtype=object)
```



```
data.pivot_table(values="age", index="sex", aggfunc = {np.mean, np.std})
```



mean

std



sex



F	59.967502	21.731762
M	50.860775	21.446421