Python Libraries for Machine Learning

Useful libraries for us

- numpy: data structures (esp. array and matrix) management
- scipy: algorithms, scientific computing
- matplotlib: graphs
- pandas: data frame management
- scikit-learn: machine learning, text processing
- nltk: natural language processing

numpy

- N-dimensional array objects
- Linear algebra, Fourier transform, random number generation

First steps…

```
import numpy as np

A = np.array([[1, 2, 3], [4, 5, 6]])

print(A)
```

```
[[1 2 3]

 [4 5 6]]
```

```
Af = np.array([1, 2, 3], float)
```

Ranges

```
np.arange(0, 1, 0.2)

array([0. , 0.2, 0.4, 0.6, 0.8])
```

Linear model

```
#creates a linear model consisting of 4
datapoints in the range [0, 2*π]

np.linspace(0, 2*np.pi, 4)

array([0., 2.0943951 , 4.1887902 ,
6.28318531])
```

Initializing matrices

```
A = np.zeros((2,3))

A

array([[0., 0., 0.],
       [0., 0., 0.]])
```

- Try also np.ones, np.diag

Random

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

array([[0.70837754, 0.66890078, 0.19124653],
       [0.64323459, 0.85562671, 0.83175077]])
```

Saving/loading to/from file

```
np.savetxt("a_out.txt", a)

b = np.loadtxt("a_out.txt")
```

File format

```
0.000000000000000000e+00 0.000000000000000000e+00 0.000000000000000000e+00

0.000000000000000000e+00 0.000000000000000000e+00 0.000000000000000000e+00
```

Modifying arrays

```
A = np.zeros((2, 2))

A[0, 0] = 1

A

array([[1., 0.],

       [0., 0.]])
```

Reshaping arrays

```
a = np.arange(10).reshape((2,5))
```

First it creates an array of 10 items, then it reshapes in a 2x5 matrix

Attributes

```
a.ndim    # returns the number of array dimensions (e.g., 2)
a.shape   # returns shape of array (e.g., 2x5)
a.size    # returns the number of of elements (e.g., 10)
a.T       # transposes the array
a.dtype   # returns the array data type
```

Array operations

```
a = np.arange(4)
# array([0, 1, 2, 3])

b = np.array([2, 3, 2, 4])

a*b
#array([0, 3, 4,12])

b - a
# array([2, 2, 0, 1])

c = [2, 3, 4, 5]
a*c
#array([0, 3, 8,15])
```

Operations with scalars

```
A = np.ones((3,3))

print(3 * A - 1)
```

```
[[2. 2. 2.]
 [2. 2. 2.]
 [2. 2. 2.]]
```

Matrix operations

- inner product
- outer product
- dot product (matrix multiplication)
- matrix element multiplication (matmul or * operator)
- **numpy automatically converts lists into numpy arrays**

Matrix operations

```
# note: numpy automatically converts lists
u = [1, 2, 3]
v = [1, 1, 1]


np.dot(u, v)
#6
```

Matrix operations

```
A = np.ones((3, 2))

B = np.ones((2, 3))

np.dot(A, B)

array([[2., 2., 2.],

       [2., 2., 2.],

       [2., 2., 2.]])
```

Matrix operations

```
np.dot(B, A)

array([[3., 3.],
       [3., 3.]])
```

Slicing arrays

```
a[2, :]  # third row, all columns

a[1:3]  # 2nd, 3rd row, all columns

a[:, 2:4]  # all rows, columns 3 and 4
```

Iterating

- Over rows

```
for row in A:
    print(row)
```

As flat values

```
for element in A.flat:
    print(element)
```

```
1.0
1.0
1.0
1.0
1.0
1.0
```

Linear algebra

- linalg.inv(A) inverse matrix
- linalg.solve(A,b) solves the system Ax=b
- linalg.svd(A, full) singular value decomposition

scipy

Algorithms and mathematical tools built to work with NumPy arrays.

- linear algebra - scipy.linalg
- statistics - scipy.stats
- optimization - scipy.optimize
- sparse matrices - scipy.sparse
- signal processing - scipy.signal

More

We’ll use scipy functions based on their need in our examples

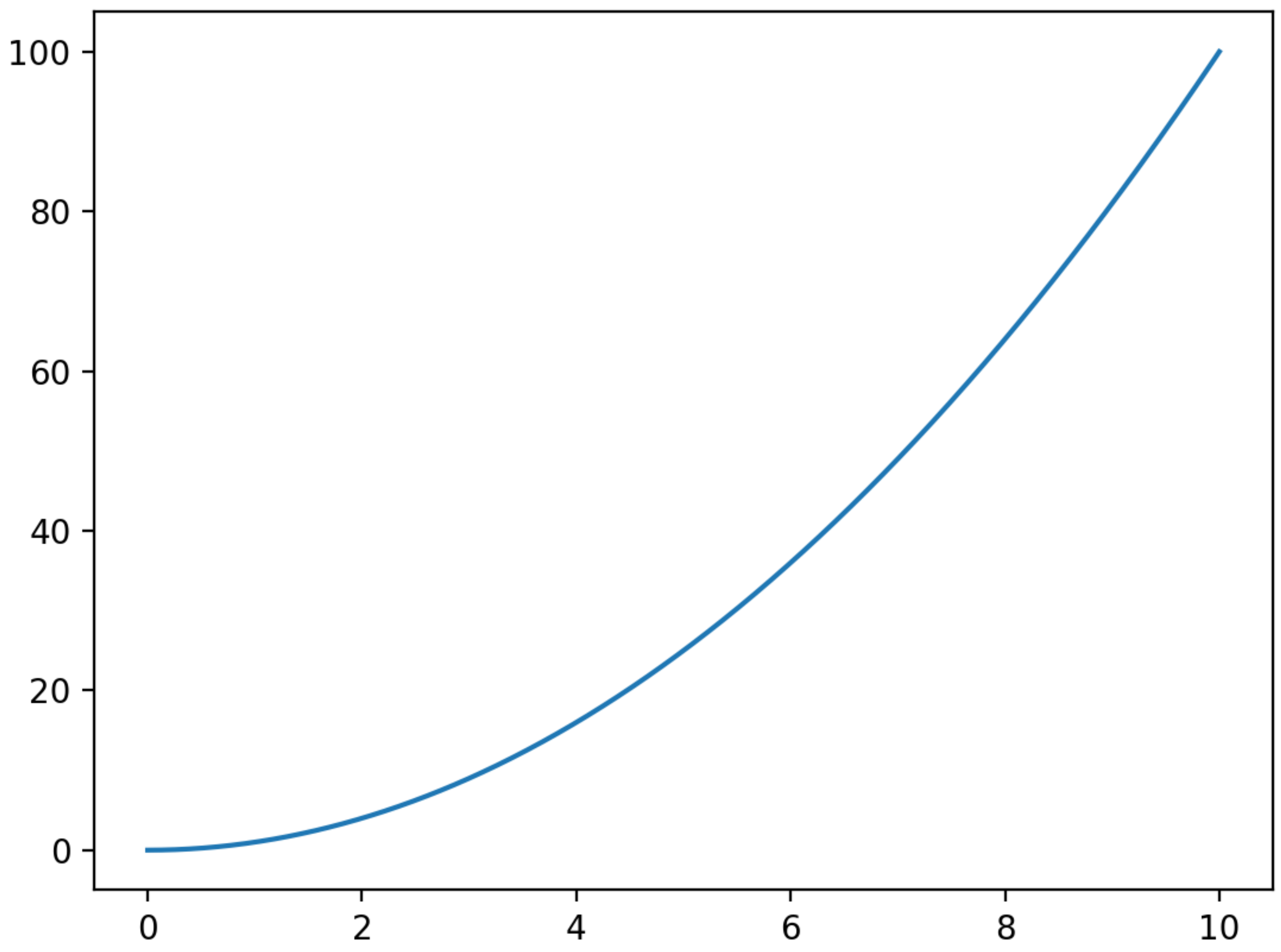
Matplotlib

- Plotting library for Python
- Works well with Numpy
- Syntax similar to Matlab

First example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
```

Result



Modifying the plot

```
plt.xlim((1, 5)) # range for the x axis
plt.ylim((0, 30))  # range for the y axis
plt.xlabel( 'my x label') # x label
plt.ylabel( 'my y label') # y label
plt.title('This is the plot title')  # plot title
plt.savefig('line_plot_plus2.pdf')  #saving the plot in a file
```

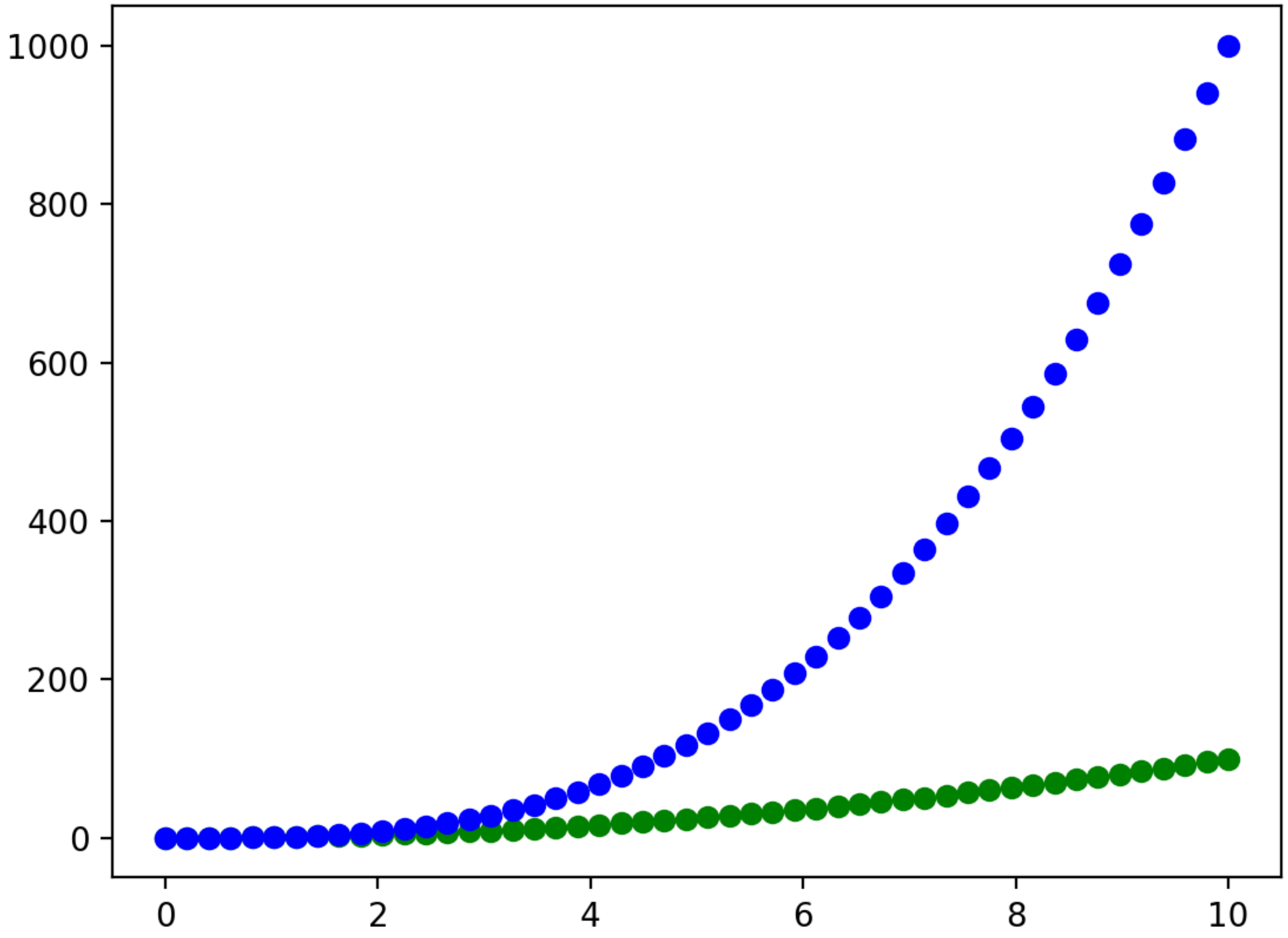
Scatterplot

- Use the third parameter to change the format, see details in https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

- For example “bo” indicates blue circles, “go” green circles

Example

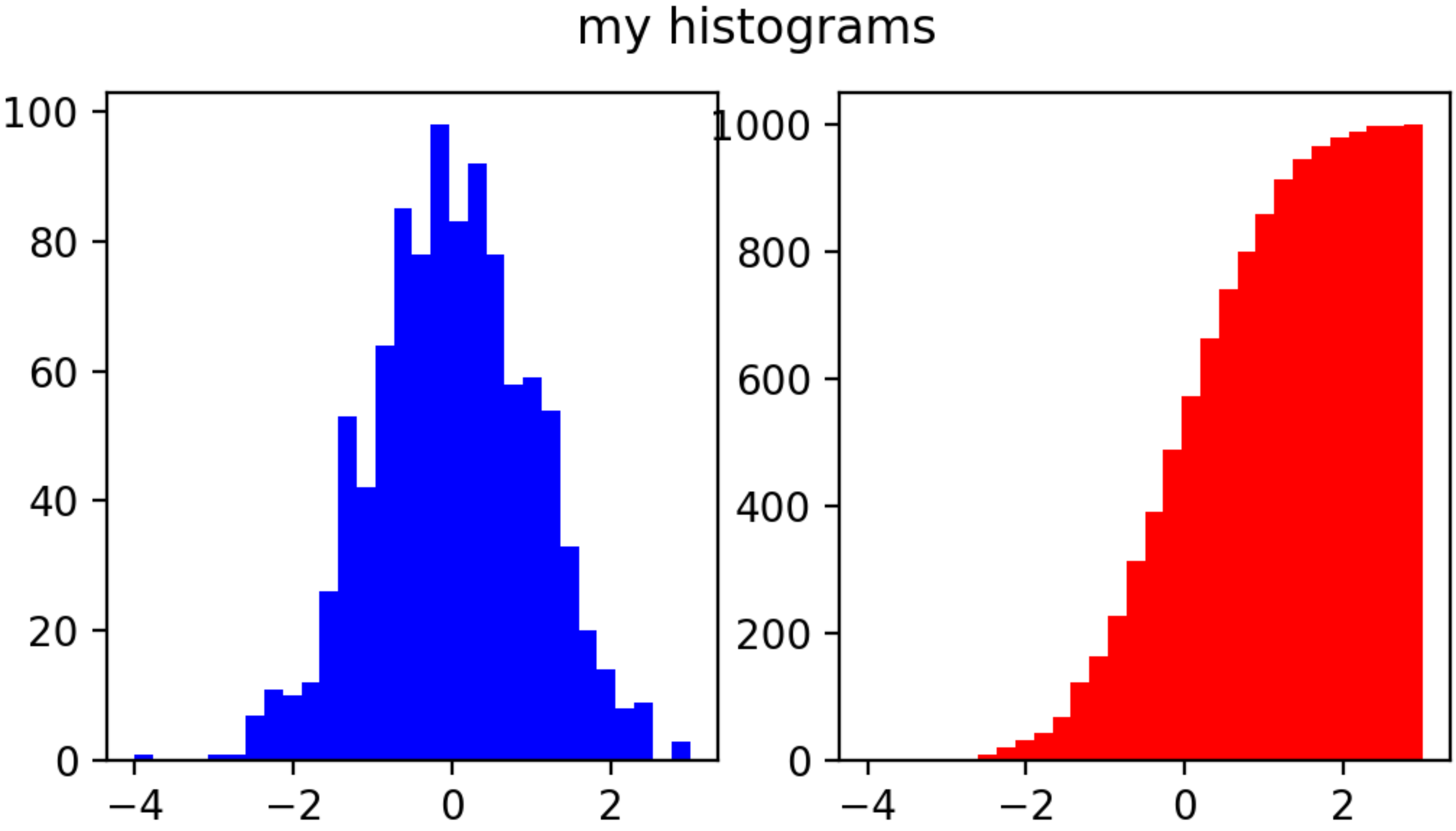
```
y1 = np.power(x, 2)
y2 = np.power(x, 3)
plt.plot(x,y1,”go")
plt.plot(x,y2,"bo")
```



Histogram

```python
import numpy as np
import matplotlib.pyplot as plt

data = np.random.randn(1000) # 1000 random numbers

# creates a figure with subplots, 1 row, 2 columns
# the result is composed of 3 objects, the figure (f1) and the two subfigures (ax1 and ax2)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(6,3))

# histogram on the first subfigure, with 30 bins, values normalized
ax1.hist(data, bins=30, normed=True, color='b')

# cumulative histogram
ax2.hist(data, bins=30, normed=True, color='r', cumulative=True)

# title for the whole figure
f.suptitle("my histograms")
```

Result



Pandas

- General purpose data manipulation lib for Python
- Makes Python very similar to languages like R or Matlab
- We will mainly see how to handle dataframes

What is a dataframe?

- A dataset consisting of rows and columns
- Columns are heterogeneous
- Each column has a name, indicated in the first row
- Rows may have a name, indicated in the first column
- Typically, in machine learning each column of a dataframe is a variable, each row a sample

First steps in Pandas

Creating a data series, similar to an list in Python or in numpy

```
import numpy as np
import pandas as pd

s = pd.Series([1, 3, 5, np.nan, 6, 8])

 >>>s
 0    1.0
 1    3.0
 2    5.0
 3    NaN
 4    6.0
 5    8.0
 dtype: float64
```

Creating a Dataframe

Composed of 4 columns, named “A”, “B”, “C”, “D”, each containing 6 random numbers

```
df = pd.DataFrame(np.random.randn(6, 4), columns=list(“ABCD"))
df
           A         B         C         D
 0  0.284643  0.287625 -0.058835  0.422457
 1  0.271413 -1.257482  2.048370  0.520126
 2 -0.526431 -1.489253  2.163840 -0.329746
 3  0.601180 -0.816019 -2.189319  0.713336
 4  0.183926 -0.528493  1.498898 -0.287538
 5  2.412859  0.211086 -1.003173 -3.027977
```

Adding row indexes

As said before, not always necessary

```
#a range of six dates
dates = pd.date_range("20130101", periods=6)

#dataframe where row indexes are the dates created before
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list(“ABCD”))
df
                   A         B         C         D
 2013-01-01 -0.297087  1.857124  0.215378  0.560495
 2013-01-02  0.481740 -1.332195  0.264884 -0.753363
 2013-01-03 -0.032739 -0.722602 -1.857090  1.109269
 2013-01-04  0.176271 -0.308361 -1.125783  0.356313
 2013-01-05  0.369300 -0.468248  0.218948 -0.700234
 2013-01-06 -2.237125  0.889179  1.378012 -0.669981
```

Creating a dataframe as a dictionary

```
df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
        "F": "foo",
    }
)
```

Result

```
df2
     A          B    C  D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

Note: you can see how scalar are repeated to fill the dataframe

Column types

```
df2.dtypes

A            float64

B     datetime64[ns]

C            float32

D              int32

E           category

F             object

dtype: object
```

Utility functions

```
df.head(n) # first n rows
df.tail(n) # last n rows
df.columns # column names
df.index #row names
df.to_numpy() #convert to numpy array
df.T #transpose
```

Statistics

```
df.describe()
```

```
              A          B          C          D
count  6.000000   6.000000   6.000000   6.000000
mean  -0.256606  -0.014184  -0.150942  -0.016250
std    1.009803   1.170434   1.152639   0.797090
min   -2.237125  -1.332195  -1.857090  -0.753363
25%   -0.231000  -0.659013  -0.790493  -0.692671
50%    0.071766  -0.388304   0.217163  -0.156834
75%    0.321043   0.589794   0.253400   0.509450
max    0.481740   1.857124   1.378012   1.109269
```

Sorting

On the second (horizontal) axis (axis=1), descending

```
df.sort_index(axis=1, ascending=False)
```

```
                   D         C         B         A
2013-01-01  0.560495  0.215378  1.857124 -0.297087
2013-01-02 -0.753363  0.264884 -1.332195  0.481740
2013-01-03  1.109269 -1.857090 -0.722602 -0.032739
2013-01-04  0.356313 -1.125783 -0.308361  0.176271
2013-01-05 -0.700234  0.218948 -0.468248  0.369300
2013-01-06 -0.669981  1.378012  0.889179 -2.237125
```

Sorting by column

```
df.sort_values(by="B")
                   A         B         C         D
2013-01-02  0.481740 -1.332195  0.264884 -0.753363
2013-01-03 -0.032739 -0.722602 -1.857090  1.109269
2013-01-05  0.369300 -0.468248  0.218948 -0.700234
2013-01-04  0.176271 -0.308361 -1.125783  0.356313
2013-01-06 -2.237125  0.889179  1.378012 -0.669981
2013-01-01 -0.297087  1.857124  0.215378  0.560495
```

Selection

```
df["A"] #column "A"
df[0:2] #first two rows
df["20130102":"20130104"] #rows with a given data range
df.loc["20130102":"20130104", ["A", "B"]] #multiple selection on rows and columns
df.loc[:, ["A", "B"]] #multiple selection on columns only
```

Accessing elements

```
df.loc[dates[0], "A"] #by keys
df.iloc[1,3] #by index
df.iloc[1,1:2] #multiple elements
```

Conditional selection

```
df[df["A"] > 0] #on a single column

df[df > 0] #on all data
```

Adding a new column

```
s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range("20130102", periods=6))

df["F"] = s1
```

Missing data

Pandas uses np.nan to represent missing data.

```
#Adds an empty column with missing data
df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ["E"])

#Fills the first 2 lines
df1.loc[dates[0] : dates[1], "E"] = 1

df1
                   A         B         C         D    E
 2013-01-01 -0.297087  1.857124  0.215378  0.560495  1.0
 2013-01-02  0.481740 -1.332195  0.264884 -0.753363  1.0
 2013-01-03 -0.032739 -0.722602 -1.857090  1.109269  NaN
 2013-01-04  0.176271 -0.308361 -1.125783  0.356313  NaN
```

Missing data functions

```
df1.dropna(how="any")
#removes lines with missing numbers
```

```
df1.fillna(value=5)
# replaces missing by 5
```

```
pd.isna(df1)
# returns True or False for any value,
# indicating whether it is a missing data or not
```

Converting types

For example, from string into categorical

```
df["E"]=pd.Series(["Red","Green","Red","Blue","Blue","Green"])
df[“E"]=df["E"].astype("category")

df.dtypes
A      float64
B      float64
C      float64
D      float64
E     category
```

Concatenating data frames

```
df2=pd.concat([df,df])
#in this cases, the results contains twice the rows of df
```

Stats

```
#applies a function over columns - computes the mean for each column
df2.apply(np.mean)


#applies a function over columns - normalizes the values
df4=df2.apply(lambda x: (x-x.min()) / (x.max() - x.min()))
```

Reading/writing csv file

```
df.to_csv("foo.csv") #writes dataframe to a CSV file

dfx=pd.read_csv("foo.csv") #reads it
```

More…

https://pandas.pydata.org