CS-618A
Deep Learning
Late Spring 2023

Lecture 1
Introduction to Deep Learning

Introduction

- Focuses on why should you learn deep learning, and what you'll need to get started.
- Start to dig deep in fundamental concepts, such as machine learning, parametric and nonparametric models, and supervised and unsupervised learning.
- Walk you through using simple networks to make a prediction, as well as provide your first look at a neural network.
- Teach you how to evaluate the predictions made and identify errors to help train models in the next step.
- You'll build your first "deep" neural network, code and all.
- Give you the tools and resources you need to continue your deep learning journey.

Source: Grokking Deep Learning First Edition by Andrew Trask

Textbooks

- **Textbook** Grokking Deep Learning First Edition
- **Author:** by Andrew Trask
- **ISBN-13**: 978-1617293702

- **Textbook**: Python Machine Learning & Deep Learning with Scikit-Learn and TensorFlow2
- **Author**: Sebastian Raschka / Vahid Mirjalili
- **Print ISBN**-13: 978-1-1789955750

- **Reference:** Hands-On Data Science and Python Machine Learning
- **Author**: Frank Kane
- **Web ISBN-13**: 978-1-78728-022-9

- **Reference:** Python for Everybody: Exploring Data Using Python 3, 2nd Edition, 2016
- **Author:** Charles Severance

Python Libraries for Data Analysis and Machine Learning

Tools required

We will use Python as programing language during this course:

- Anaconda (https://www.anaconda.com/)
  - The open-source Anaconda
    is the easiest way to perform Python/R data science and machine learning on Linux, Windows, a nd Mac OS X.
  - It is the industry standard for developing, testing, and training on a single machine.
  - Please follow the instruction here to install the Anaconda (for Python 3.7)
    https://www.anaconda.com/distribution/#download-section
  - It provides different versions to suit different OS. Please select the one you are using.
- Jupyter Notebook (https://jupyter.org/) for Python environment.
  - The Jupyter
    Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
  - Get started learning Python: https://www.learnpython.org

What is Anaconda

- The open-source Anaconda is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 19 million users worldwide.
- It is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:
  - Quickly download 7,500+ Python/R data science packages
  - Analyze data with scalability and performance with Dask, **NumPy**, **pandas**, and Numba
  - Visualize results with **Matplotlib**, Bokeh, Datashader, and Holoviews
  - Develop and train machine learning and deep learning models with **scikitlearn**, TensorFlow, and Theano

Anaconda installation

- Please follow the instruction here to install the Anaconda (for Python 3.7)

https://www.anaconda.com/distribution/#download-section

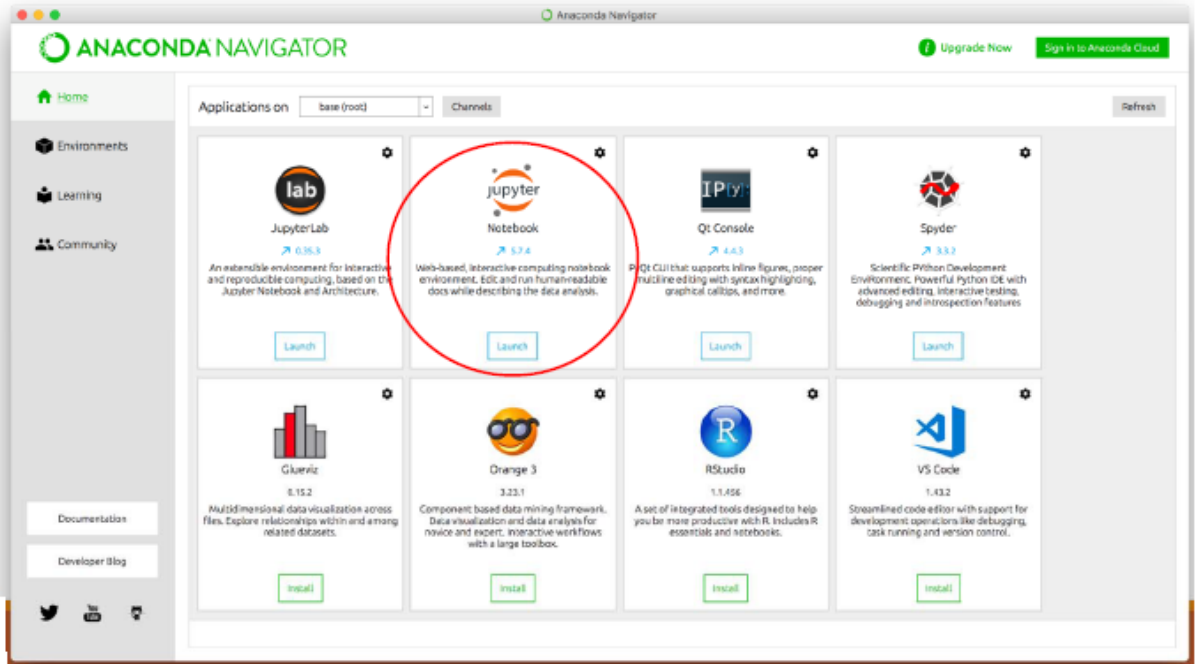
- It provides different versions to suit different OS. Please select the one you are using.
- Just install according to the default setting, and the environment variables will be automatically configured after installation.

Jupyter Notebook

- The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
- It includes: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.
- Jupyter Notebook is included in the Anaconda.

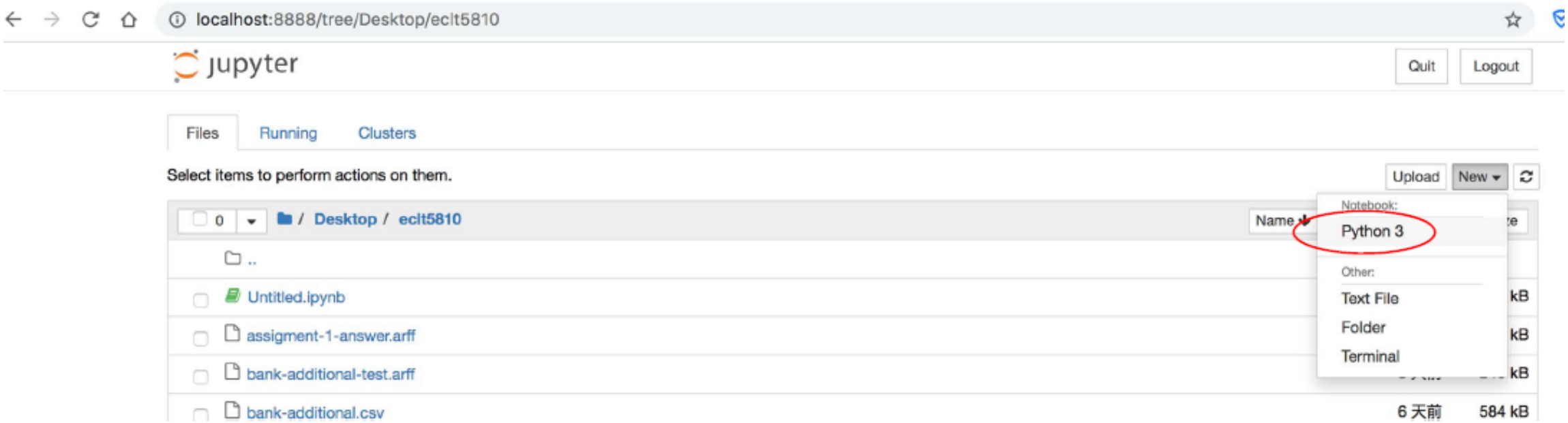
Basic Operation on Jupyter Notebook

After installing the Anaconda, open Anaconda-Navigator as below, and you can find the Jupyter Notebook on the Anaconda. Then click Launch.



Basic Operation on Jupyter Notebook

Jupyter Notebook is presented as a website. Select the path, then under the button “New”, choose “Python 3” to open a new python file.



Python libraries

- Python toolboxes/libraries for data processing:
  - NumPy (https://numpy.org/)
  - SciPy (https://scipy.org/)
  - Pandas (https://pandas.pydata.org/)
- Visualization libraries
  - Matplotlib (https://matplotlib.org/)
  - Seaborn (https://seaborn.pydata.org/)
- Machine learning & deep learning
  - Scikit-learn (https://scikit-learn.org/)
  - Tensorflow (https://www.tensorflow.org/)

35 comprehensive notebooks

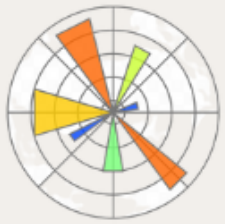
Data Computations

NumPy

Data Analysis

pandas

Data Visualization



matplotlib

Complete ML Package

Classical Machine Learning

scikit learn

Computer Vision
and
Natural Language Processing

Numby

- NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.
- pip install numpy
- import numpy as np

Source: https://numpy.org/

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

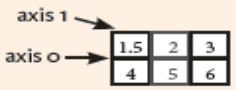
Use the following import convention:

```
>>> import numpy as np
```

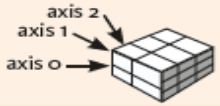
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                    Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)    Create an array of ones
>>> d = np.arange(10,25,5)             Create an array of evenly
                                       spaced values (step value)
>>> np.linspace(0,2,9)                 Create an array of evenly
                                       spaced values (number of samples)
>>> e = np.full((2,2),7)               Create a constant array
>>> f = np.eye(2)                      Create a 2X2 identity matrix
>>> np.random.random((2,2))            Create an array with random values
>>> np.empty((3,2))                    Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64          Signed 64-bit integer types
>>> np.float32        Standard double-precision floating point
>>> np.complex        Complex numbers represented by 128 floats
>>> np.bool           Boolean type storing TRUE and FALSE values
>>> np.object         Python object type
>>> np.string_        Fixed-length string type
>>> np.unicode_       Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape           Array dimensions
>>> len(a)            Length of array
>>> b.ndim            Number of array dimensions
>>> e.size            Number of array elements
>>> b.dtype           Data type of array elements
>>> b.dtype.name      Name of data type
>>> b.astype(int)     Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                          Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                   Subtraction
>>> b + a                              Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                        Addition
>>> a / b                              Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                     Division
>>> a * b                              Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                   Multiplication
>>> np.exp(b)                          Exponentiation
>>> np.sqrt(b)                         Square root
>>> np.sin(a)                          Print sines of an array
>>> np.cos(b)                          Element-wise cosine
>>> np.log(a)                          Element-wise natural logarithm
>>> e.dot(f)                           Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                             Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                              Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)               Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                            Array-wise sum
>>> a.min()                            Array-wise minimum value
>>> b.max(axis=0)                      Maximum value of an array row
>>> b.cumsum(axis=1)                   Cumulative sum of the elements
>>> a.mean()                           Mean
>>> b.median()                         Median
>>> a.corrcoef()                       Correlation coefficient
>>> np.std(b)                          Standard deviation
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()              Sort an array
>>> c.sort(axis=0)        Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                               Select the element at the 2nd index
  3
>>> b[1,2]                             Select the element at row 1 column 2
  6.0                                  (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                             Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                           Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                              Select all items at row 0
  array([[1.5, 2., 3.]])               (equivalent to b[0:1, :])
>>> c[1,...]                           Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                          Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                             Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]      Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]      Select a subset of the matrix's rows
  array([[ 4. ,5. , 6. , 4. ],         and columns
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)                Permute array dimensions
>>> i.T                                Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                          Flatten the array
>>> g.reshape(3,-2)                    Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                    Return a new array with shape (2,6)
>>> np.append(h,g)                     Append items to an array
>>> np.insert(a, 1, 5)                 Insert items in an array
>>> np.delete(a,[1])                   Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)       Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                   Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                         Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                   Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))             Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                         Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                     Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]   index
>>> np.vsplit(c,2)                     Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Scipy

- SciPy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems.
- pip install scipy
- import scipy sci

Source: https://scipy.org/

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.
- Provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc
- Allows handling missing data
- pip install pandas
- import padnas as pd

Source: https://pandas.pydata.org/

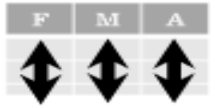
Data Wrangling

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

Pandas API Reference Pandas User Guide

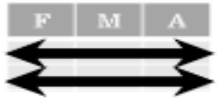
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



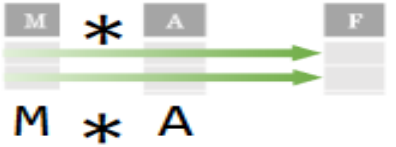
Each **variable** is saved in its own **column**

&



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

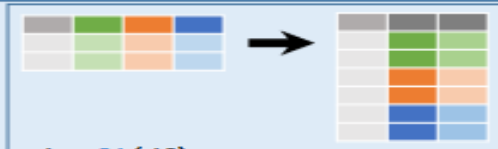
```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```
Create DataFrame with a MultiIndex

Method Chaining

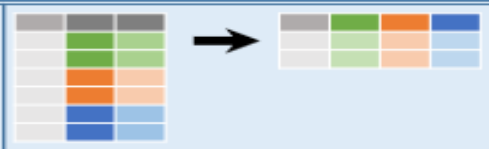
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
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'

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

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'].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 index level 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.

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)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

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

`rank(method='first')`
Ranks. Ties go to first value.

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

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

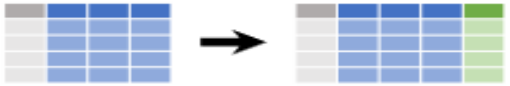
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

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

`df.fillna(value)`
Replace all NA/null data with value.

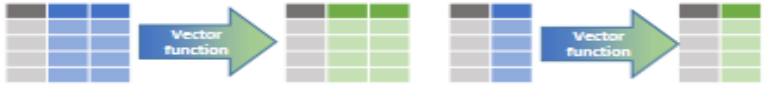
Make New Columns



`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)`
Element-wise max.

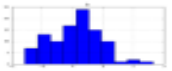
`clip(lower=-10,upper=10)`
Trim values at input thresholds

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

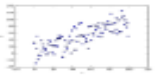
`abs()`
Absolute value.

Plotting

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



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



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

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

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

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

x1	x2	x3
A	1	T
B	2	F

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

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

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

Filtering Joins

x1	x2
A	1
B	2

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

x1	x2
C	3

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

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

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

matplotlib

- 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.
- pip install matplotlib
- import matplotlib.pyplot as plt

Source: https://matplotlib.org/

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

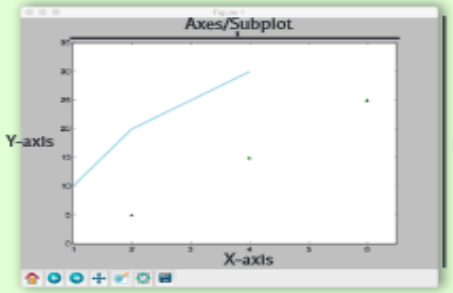
Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot 2D vector fields
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

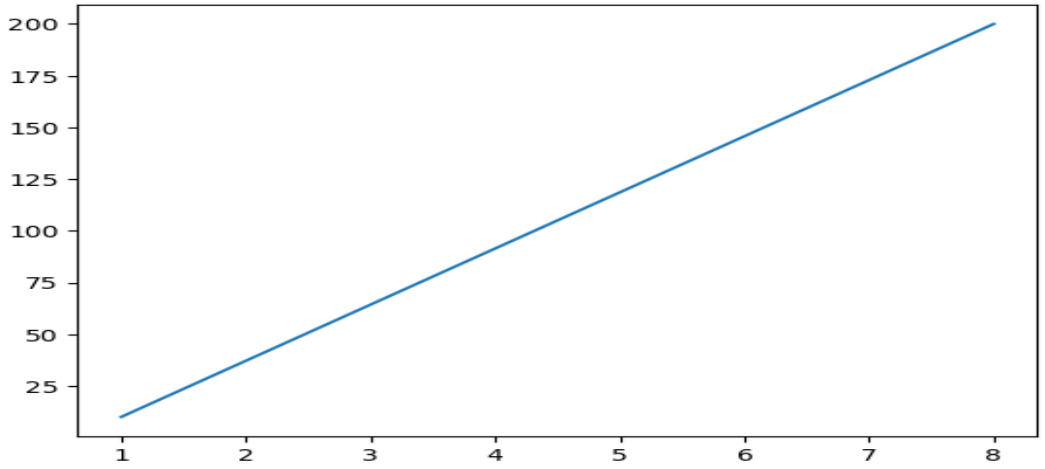
```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

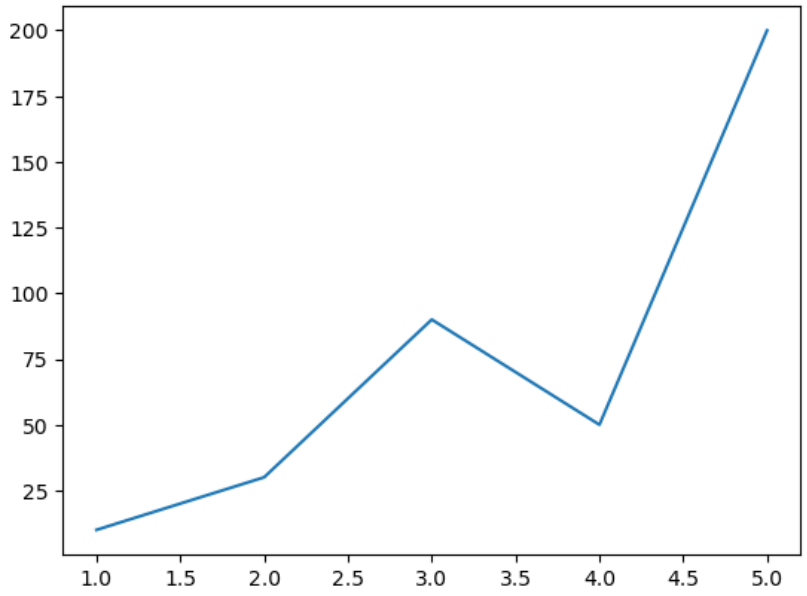
Matplotlib is Data Visualization libray this is python open source library Matplotlib helps you to show graphical representation of your solution

```
• import matplotlib.pyplot as plt
• import pandas as pd
• import numpy as np
• x=np.array([1,8])
• y=np.array([10,200])
• plt.plot(x,y)
• plt.show()
```



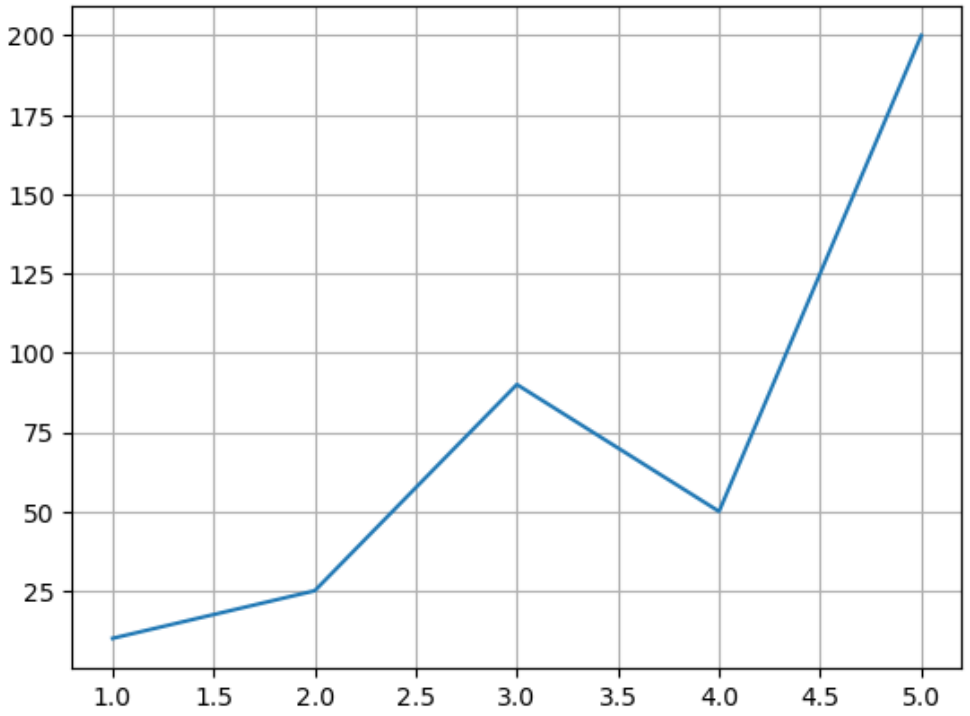

```
x=np.array([1,2,3,4,5])
y=np.array([10,30,90,50,200])

plt.plot(x,y)
plt.show()
```

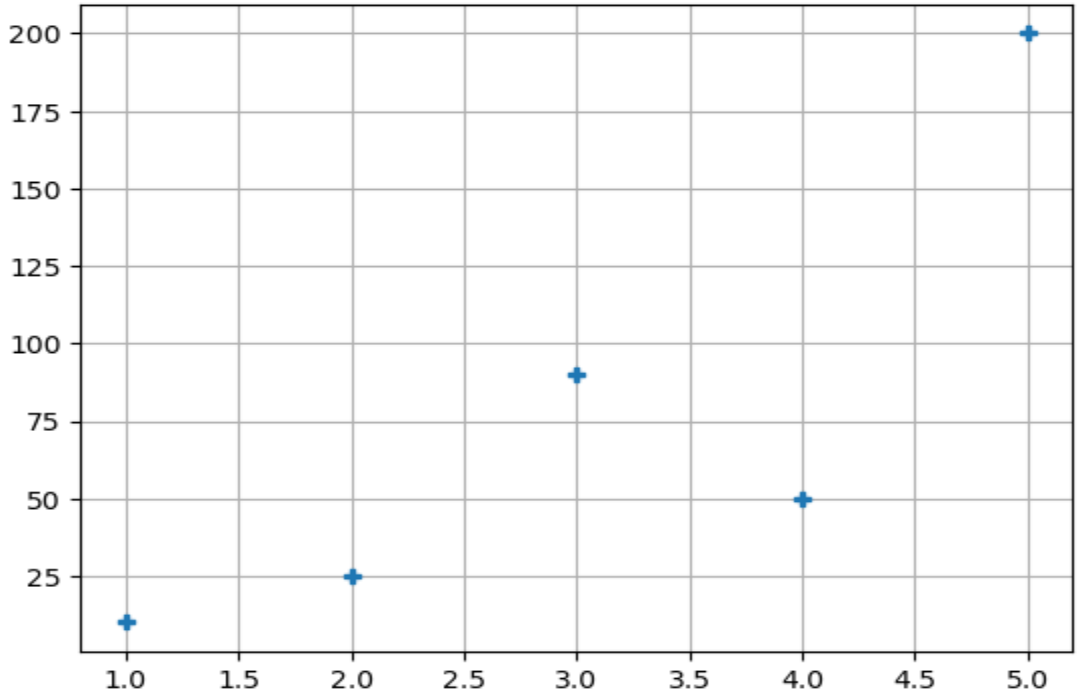


```
x=np.array([1,2,3,4,5])
y=np.array([10,25,90,50,200])

plt.plot(x,y)
plt.grid()
plt.show()
```



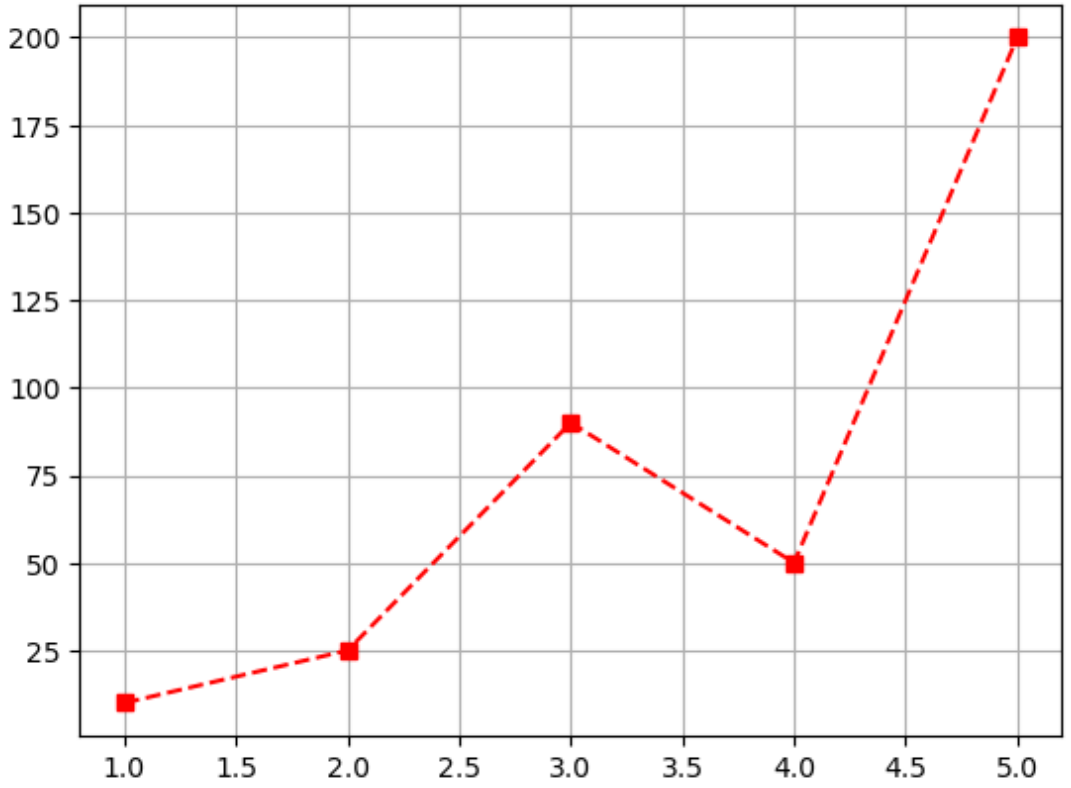

```
x=np.array([1,2,3,4,5])
y=np.array([10,25,90,50,200])
plt.plot(x,y,'P') #o represets Markers
plt.grid()
plt.show()
```

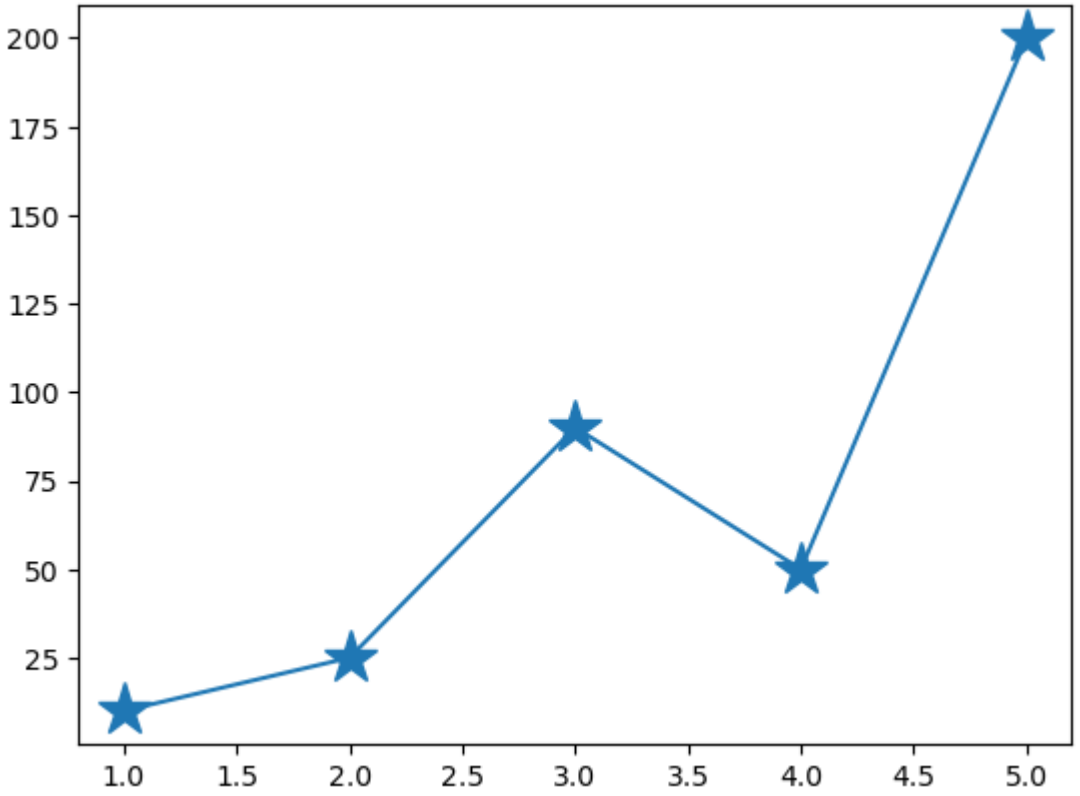


- x=np.array([1,2,3,4,5])
- y=np.array([10,25,90,50,200])
- plt.plot(x,y,'s--r') #o represets Markers
- plt.grid()
- plt.show()

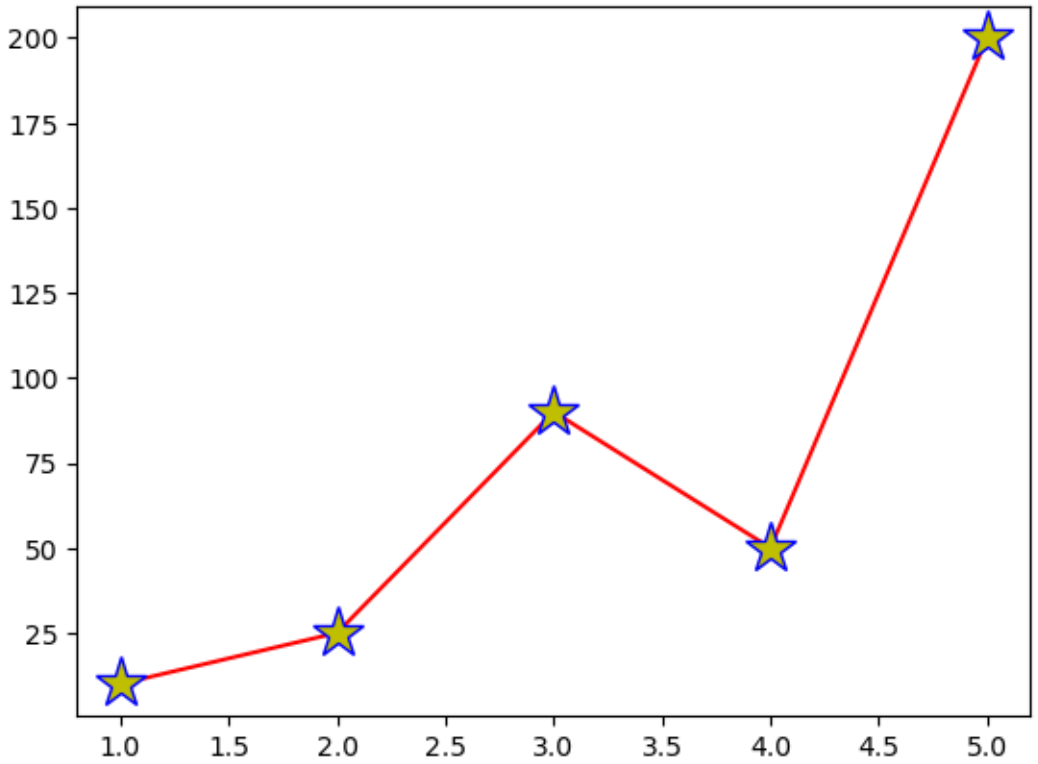
```
x=np.array([1,2,3,4,5])
y=np.array([10,25,90,50,200])

plt.plot(x,y,marker='*',ms=20) #marker_size
plt.show()
```

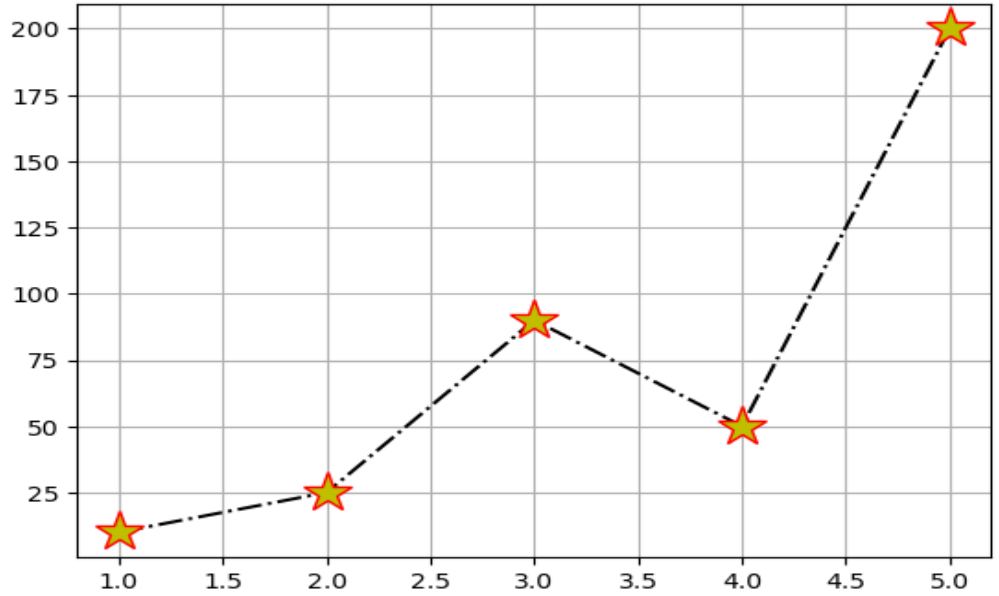




```
x=np.array([1,2,3,4,5])
y=np.array([10,25,90,50,200])
# ms is Marker size
# mec is Marker edge color
# mfc marker face color
plt.plot(x,y,marker='*',ms=20,mec='b',mfc='
y',color="red")
plt.show()
```




```
x=np.array([1,2,3,4,5])
y=np.array([10,25,90,50,200])
# ms is Marker size
# mec is Marker edge color
# mfc marker face color
# ls line style
plt.plot(x,y,marker='*',ms=20,mec='r',mfc='
y',color="black",ls='-.')
plt.grid()
plt.show()
```



Seaborn

- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- pip install seaborn
- import seaborn as sns

Source: https://seaborn.pydata.org/

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

`>>> f, ax = plt.subplots(figsize=(5,6))`	Create a figure and one subplot

Seaborn styles

`>>> sns.set()`	(Re)set the seaborn default
`>>> sns.set_style("whitegrid")`	Set the matplotlib parameters
`>>> sns.set_style("ticks", {"xtick.major.size":8, "ytick.major.size":8})`	Set the matplotlib parameters
`>>> sns.axes_style("whitegrid")`	Return a dict of params or use with `with` to temporarily set the style

Context Functions

`>>> sns.set_context("talk")`	Set context to `"talk"`
`>>> sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth":2.5})`	Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

`>>> sns.set_palette("husl",3)`	Define the color palette
`>>> sns.color_palette("husl")`	Use with `with` to temporarily set palette
`>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]`	
`>>> sns.set_palette(flatui)`	Set your own color palette

3 Plotting With Seaborn

Axis Grids

`>>> g = sns.FacetGrid(titanic, col="survived", row="sex")`	Subplot grid for plotting conditional relationships
`>>> g = g.map(plt.hist,"age")`	
`>>> sns.factorplot(x="pclass", y="survived", hue="sex", data=titanic)`	Draw a categorical plot onto a Facetgrid
`>>> sns.lmplot(x="sepal_width", y="sepal_length", hue="species", data=iris)`	Plot data and regression model fits across a FacetGrid
`>>> h = sns.PairGrid(iris)`	Subplot grid for plotting pairwise relationships
`>>> h = h.map(plt.scatter)`	Plot pairwise bivariate distributions
`>>> sns.pairplot(iris)`	Grid for bivariate plot with marginal univariate plots
`>>> i = sns.JointGrid(x="x", y="y", data=data)`	
`>>> i = i.plot(sns.regplot, sns.distplot)`	
`>>> sns.jointplot("sepal_length", "sepal_width", data=iris, kind='kde')`	Plot bivariate distribution

Categorical Plots

Scatterplot

`>>> sns.stripplot(x="species", y="petal_length", data=iris)`	Scatterplot with one categorical variable
`>>> sns.swarmplot(x="species", y="petal_length", data=iris)`	Categorical scatterplot with non-overlapping points

Bar Chart

`>>> sns.barplot(x="sex", y="survived", hue="class", data=titanic)`	Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

`>>> sns.countplot(x="deck", data=titanic, palette="Greens_d")`	Show count of observations

Point Plot

`>>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette={"male":"g", "female":"m"}, markers=["^","o"], linestyles=["-","--"])`	Show point estimates and confidence intervals as rectangular bars

Boxplot

`>>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic)`	Boxplot
`>>> sns.boxplot(data=iris,orient="h")`	Boxplot with wide-form data

Violinplot

`>>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)`	Violin plot

Regression Plots

`>>> sns.regplot(x="sepal_width", y="sepal_length", data=iris, ax=ax)`	Plot data and a linear regression model fit

Distribution Plots

`>>> plot = sns.distplot(data.y, kde=False, color="b")`	Plot univariate distribution

Matrix Plots

`>>> sns.heatmap(uniform_data,vmin=0,vmax=1)`	Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

`>>> g.despine(left=True)`	Remove left spine
`>>> g.set_ylabels("Survived")`	Set the labels of the y-axis
`>>> g.set_xticklabels(rotation=45)`	Set the tick labels for x
`>>> g.set_axis_labels("Survived", "Sex")`	Set the axis labels
`>>> h.set(xlim=(0,5), ylim=(0,5), xticks=[0,2.5,5], yticks=[0,2.5,5])`	Set the limit and ticks of the x-and y-axis

Plot

`>>> plt.title("A Title")`	Add plot title
`>>> plt.ylabel("Survived")`	Adjust the label of the y-axis
`>>> plt.xlabel("Sex")`	Adjust the label of the x-axis
`>>> plt.ylim(0,100)`	Adjust the limits of the y-axis
`>>> plt.xlim(0,10)`	Adjust the limits of the x-axis
`>>> plt.setp(ax,yticks=[0,5])`	Adjust a plot property
`>>> plt.tight_layout()`	Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

`>>> plt.show()`	Show the plot
`>>> plt.savefig("foo.png")`	Save the plot as a figure
`>>> plt.savefig("foo.png", transparent=True)`	Save transparent figure

Close & Clear

Also see Matplotlib

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear an entire figure
`>>> plt.close()`	Close a window

Scikit-learn

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license
- pip install scikit-learn

Source: https://scikit-learn.org/stable/

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Tensorflow

- create machine learning models for desktop, mobile, web, and cloud
- pip install tensorflow
- import tensorflow as tf

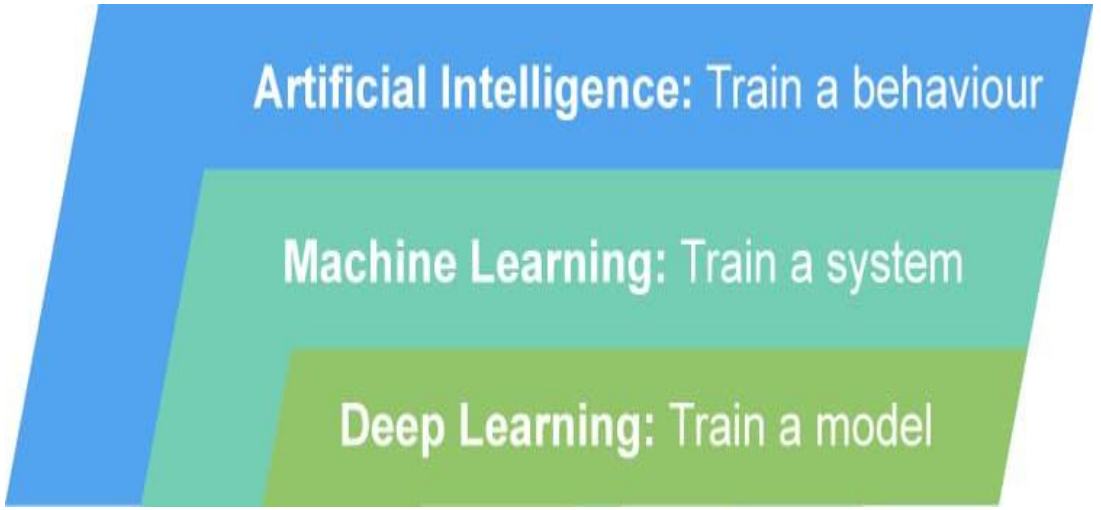
Source: https://www.tensorflow.org/learn

Import Python libraries

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Machine learning

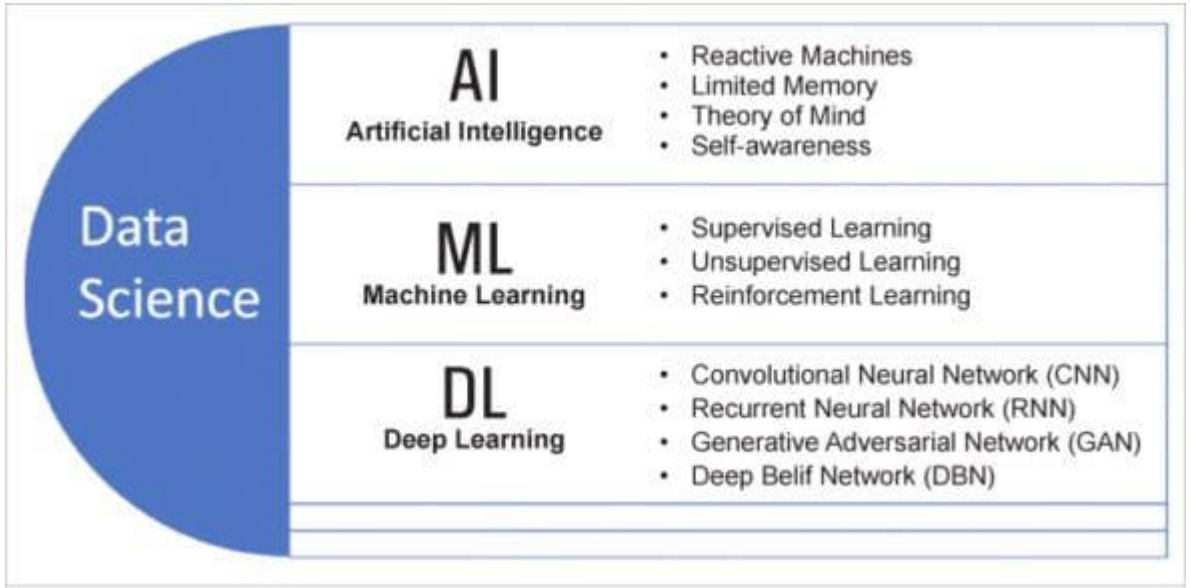
- Learning from data rather than through programmed rules/logic using traditional programming
- “A field of study that gives computers the ability to learn without being explicitly programmed.” Arthur Samuel



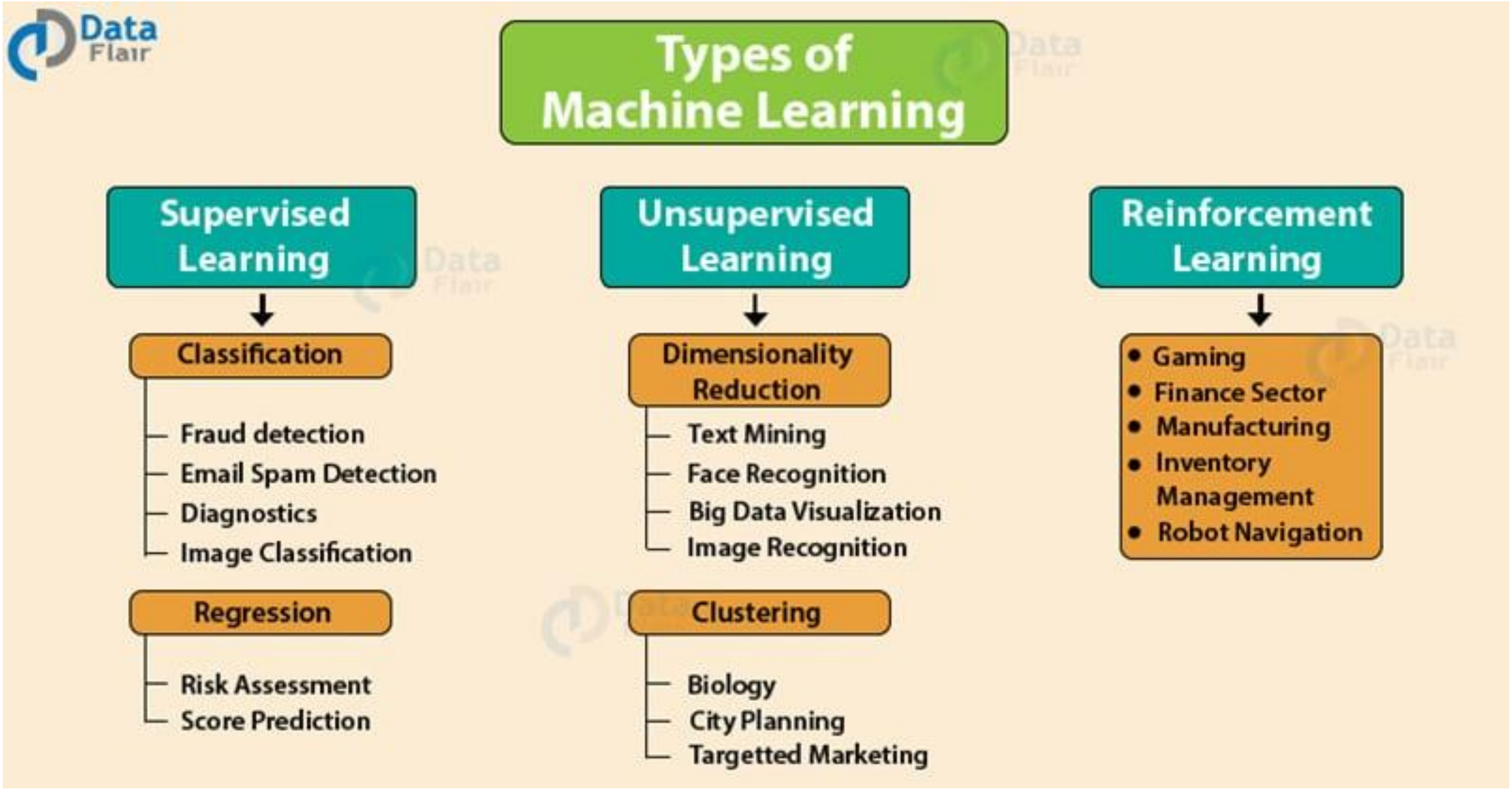
Source: https://www.opensourceforu.com/2022/08/ai-ml-and-dl-whats-the-difference/

Deep learning

- Deep learning is a specific kind of machine learning.
- Which is a field dedicated to the study and development of machines that can learn
- Deep learning is used to solve practical tasks in a variety of fields such as computer vision (image), natural language processing (text), and automatic speech recognition (audio).
- It works technically in the same way as machine learning does, but with different capabilities and approaches.
- Deep learning is implemented with the help of Neural Networks.

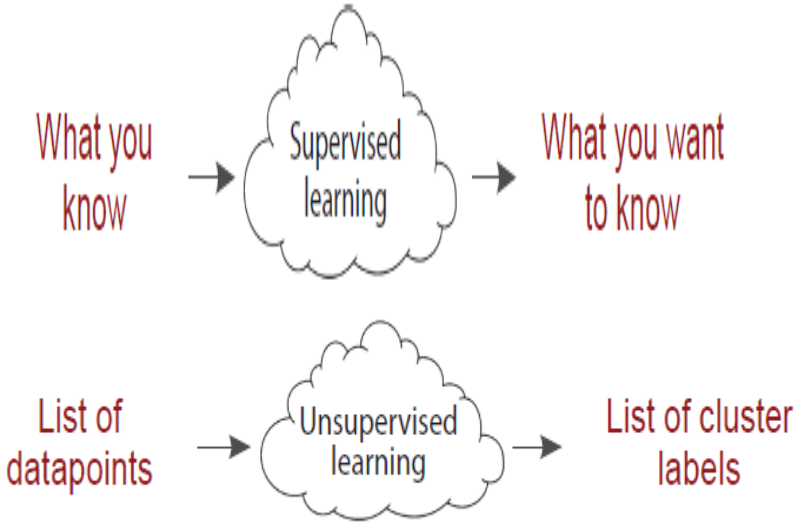


Source: https://www.opensourceforu.com/2022/08/ai-ml-and-dl-whats-the-difference/


Data Flair
Types of Machine Learning
Supervised Learning
Classification
Fraud detection
Email Spam Detection
Diagnostics
Image Classification
Regression
Risk Assessment
Score Prediction
Unsupervised Learning
Dimensionality Reduction
Text Mining
Face Recognition
Big Data Visualization
Image Recognition
Clustering
Biology
City Planning
Targetted Marketing
Reinforcement Learning
Gaming
Finance Sector
Manufacturing
Inventory Management
Robot Navigation

Types of machine learning

- Supervised machine learning
  - Supervised learning is a method for transforming one dataset into another.
  - a supervised learning algorithm might try to use one to predict he other.
- Unsupervised machine learning
  - It transform one dataset into another. But the dataset that it transforms into is not previously known
  - Unlike supervised learning, there is no “right answer” that you’re trying to get the model to duplicate
  - For example, *clustering a dataset into groups* is a type of unsupervised learning
- Reinforcement learning
  - Reinforcement Learning(RL) is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.
  - Learn from mistakes



Source: https://www.opensourceforu.com/2022/08/ai-ml-and-dl-whats-the-difference/

Supervised learning

- This is the most commonly used type of machine learning. It involves training a model on a labeled dataset, where each example in the dataset is paired with a label or target variable. The goal is to learn a mapping between the input data and the output labels, so that the model can make accurate predictions on new, unseen data. Examples of supervised learning algorithms include regression, decision trees, random forests, support vector machines (SVMs), and neural networks.
- Supervised learning is a type of machine learning where a model is trained on a labeled dataset, where each example in the dataset is paired with a label or target variable. The goal is to learn a mapping between the input data and the output labels, so that the model can make accurate predictions on new, unseen data.
- The basic workflow for supervised learning involves splitting the dataset into training and testing sets, training the model on the training set, evaluating its performance on the testing set, and then fine-tuning the model as needed. Common supervised learning algorithms include linear regression, logistic regression, decision trees, random forests, support vector machines (SVMs), and neural networks.

Supervised learning can be further divided into two categories:

1) Regression: In regression, the target variable is continuous, and the goal is to predict a numerical value. Examples of regression problems include predicting the price of a house based on its features, or predicting a person's income based on their education level and work experience.
2) Classification: In classification, the target variable is categorical, and the goal is to predict which class or category a new input belongs to. Examples of classification problems include predicting whether an email is spam or not, or classifying images of animals based on their species.

Unsupervised learning

- In unsupervised learning, the model is trained on an unlabeled dataset, meaning there are no target variables to learn from. The goal is to identify patterns or structure in the data without the help of explicit labels. Unsupervised learning can be used for tasks like clustering, dimensionality reduction, and anomaly detection. Examples of unsupervised learning algorithms include k-means clustering, principal component analysis (PCA), and autoencoders.

- Unsupervised learning is particularly useful when you have a large amount of unstructured or unlabeled data that you want to make sense of. By identifying patterns and relationships in the data, unsupervised learning can help you gain new insights and identify opportunities for further analysis.

Unsupervised learning can be used for a variety of tasks, including:

1. Clustering: Grouping similar data points together based on their distance or similarity.
2. Anomaly detection: Identifying rare or unusual data points that may be indicative of errors or fraud.
3. Dimensionality reduction: Reducing the number of features in a dataset while retaining the most important information.

Some common unsupervised learning algorithms include

1. K-means clustering: A simple algorithm that groups data points into k clusters based on their distance to the cluster center.
2. Hierarchical clustering: A clustering algorithm that groups data points into a tree-like hierarchy of clusters.
3. Principal component analysis (PCA): A dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while retaining as much of the original information as possible.

Reinforcement learning

- Reinforcement learning involves training a model to make a sequence of decisions in an environment, with the goal of maximizing a reward signal. The model learns through trial and error, receiving feedback in the form of rewards or penalties for its actions. Reinforcement learning can be used for tasks like game playing, robotics, and self-driving cars. Examples of reinforcement learning algorithms include Q-learning and policy gradient methods.
- The reinforcement learning process can be modeled as a Markov decision process (MDP), where the agent interacts with an environment over a sequence of time steps. At each time step, the agent observes the current state of the environment, takes an action, and receives a reward signal and the next state of the environment.

Reinforcement learning

- The goal of the agent is to learn a policy that maps states to actions in order to maximize the expected cumulative reward over time. The policy can be learned using various techniques such as Q-learning, policy gradients, and actor-critic methods.
- Reinforcement learning has been successfully applied to a variety of tasks, including game playing, robotics, and autonomous driving. One of the most well-known examples of reinforcement learning is AlphaGo, an AI system developed by DeepMind that defeated the world champion in the game of Go.

How can machines learn

- A computer learns something about the structures that represent the information in the raw data.
- Structural descriptions are the models we build to contain the information extracted from the raw data.
- We can use those models to predict unknown data.
- Each model type has a different way of applying rules to known data to predict unknown data.
- Samuel defined machine learning as : “The field of study that gives computers the ability to learn without being explicitly programmed.”

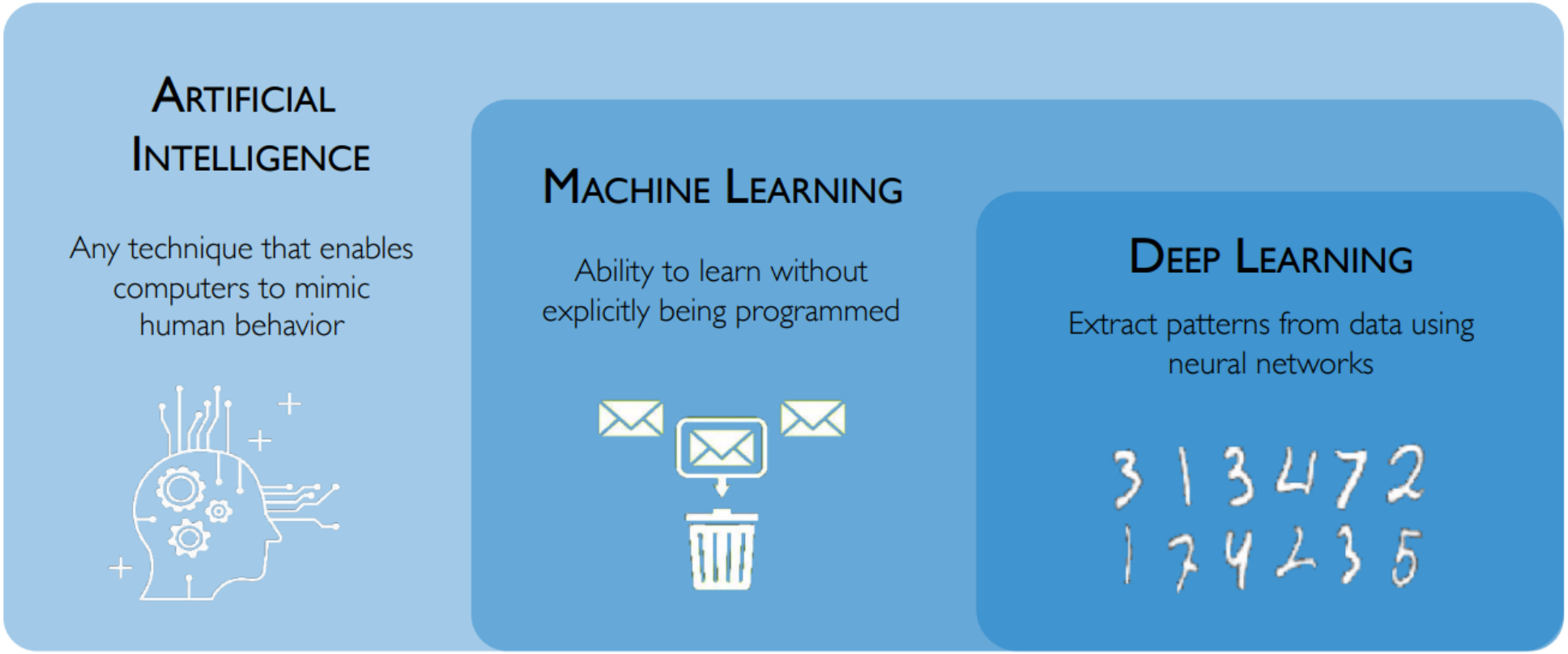
Source: Deep Learning Practitioners Approach

Deep learning

- “A new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations”.
- 90% of deep learning applications use supervised learning.
- Supervised learning is a process that, you collect some of inputs and outputs, and the inputs are feed into a machine to learn the correct output.
- If the output is correct, you don’t do anything. else the output is wrong; you tweak the parameter of the machine and correct the output toward the result that you need.
- The trick here is how you figure out how to tweak this parameter and in which direction

Source: https://www.opensourceforu.com/2022/08/ai-ml-and-dl-whats-the-difference/

Deep learning



Source: http://introtodeeplearning.com/

Introduction to Deep Learning

- Deep learning is a subset of machine learning that involves training artificial neural networks to learn from large amounts of data. It is a type of artificial intelligence that is modeled after the way the human brain works, and it has revolutionized many fields such as computer vision, natural language processing, speech recognition, and robotics.

- Deep learning is important because it enables machines to learn and make predictions based on complex patterns in data, without being explicitly programmed. This allows for more accurate and efficient solutions to complex problems, and has led to breakthroughs in areas such as image and speech recognition, natural language processing, and autonomous driving.

Why deep learning is important:

1. Accuracy: Deep learning has been shown to produce highly accurate results in many applications, often outperforming traditional machine learning methods.
2. Automation: Deep learning enables automation of tasks that would otherwise require significant human involvement. For example, deep learning models can be used to automatically classify images, transcribe speech, or generate natural language responses.
3. Unstructured data: Deep learning is particularly useful for handling unstructured data such as images, video, and text, where traditional machine learning techniques may struggle.
4. Scalability: Deep learning models can be trained on large amounts of data, making them suitable for big data applications.
5. Real-time processing: Deep learning models can make predictions in real-time, making them useful for applications that require immediate decisions or actions.
6. Cross-disciplinary applications: Deep learning has applications across many different fields, including healthcare, finance, transportation, and entertainment.

Summary

- Deep learning is important because it has revolutionized many fields such as computer vision, natural language processing, speech recognition, and robotics, among others. It has enabled machines to learn and make predictions based on complex patterns in data, without being explicitly programmed. This allows for more accurate and efficient solutions to complex problems, and has led to breakthroughs in areas such as image and speech recognition, natural language processing, and autonomous driving.
- In summary, deep learning is important because it provides powerful tools for analyzing and making predictions based on complex data, with applications across a wide range of fields.

Applications of deep learning

- Computer vision: Image classification, object detection, facial recognition, and autonomous driving.
- Natural language processing: Sentiment analysis, machine translation, and speech recognition.
- Healthcare: Medical image analysis, diagnosis, and drug discovery.
- Robotics: Object recognition, motion planning, and control.

Deep learning Frameworks

- Deep learning frameworks are software tools that provide a high-level interface for building and training deep neural networks. These frameworks abstract away many of the details of the low-level implementation, such as how the computations are performed on GPUs or distributed computing systems, and allow users to easily define and train complex models.

There are many deep learning frameworks available for developing and training deep neural networks. Some popular deep learning frameworks include:

- TensorFlow: Developed by Google, TensorFlow is an open-source software library for dataflow and differentiable programming across a range of tasks.
- PyTorch: Developed by Facebook, PyTorch is a Python-based scientific computing package that provides two high-level features: Tensor computation (like NumPy) with strong acceleration via graphics processing units (GPU) and deep neural networks built on a tape-based autodiff system.
- Keras: Developed by Google, Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
- Caffe: Developed by the Berkeley Vision and Learning Center (BVLC), Caffe is a deep learning framework made with expression, speed, and modularity in mind.
- MXNet: Developed by Apache, MXNet is a flexible deep learning framework that supports multiple programming languages and allows users to easily switch between imperative and symbolic programming.
- Theano: Theano is a deep learning framework that was one of the first to gain popularity. While it has been largely superseded by TensorFlow and PyTorch, it still has a strong following and can be useful for certain applications.

These frameworks provide developers with powerful tools for building, training, and deploying deep learning models.

TensorFlow

- TensorFlow: Developed by Google, TensorFlow is one of the most popular deep learning frameworks used today. It offers a wide range of tools and features for building and training deep neural networks, including support for distributed computing and deployment on mobile devices.
- TensorFlow is designed to be flexible and scalable, and supports a wide range of neural network architectures, from simple feedforward networks to complex convolutional and recurrent networks. It also includes support for distributed computing, allowing users to train models on large-scale clusters of CPUs or GPUs.

- One of the key features of TensorFlow is its ability to define computational graphs, which represent the flow of data through a neural network. These graphs can be optimized for performance, allowing TensorFlow to efficiently execute computations on a variety of hardware, including CPUs, GPUs, and custom hardware like Google's Tensor Processing Units (TPUs).
- TensorFlow provides a high-level API called Keras, which makes it easy to build and train deep neural networks using a simple, intuitive interface. It also provides a low-level API, allowing users to define their own custom operations and perform low-level optimizations.
- In addition to its core functionality, TensorFlow also includes a number of useful tools and libraries, such as TensorFlow Data Validation, TensorFlow Probability, and TensorFlow Hub, that can be used to develop and deploy deep learning models for a variety of applications.
- Overall, TensorFlow is a powerful and versatile deep learning framework that has become a standard tool in the field of artificial intelligence and machine learning.

PyTorch

- PyTorch is an open-source deep learning framework developed by Facebook AI Research (FAIR). It is one of the most popular and widely used deep learning frameworks, particularly in the research community.
- PyTorch is designed to be flexible and intuitive, and provides a dynamic computational graph that allows users to define and modify models on the fly. This makes it easy to experiment with different architectures and model designs, and enables fast prototyping of new ideas.

- PyTorch also includes support for distributed computing, allowing users to train models on large-scale clusters of CPUs or GPUs. It provides a number of high-level APIs for building and training deep neural networks, including TorchScript, which allows users to export PyTorch models to other languages like C++ or JavaScript.
- One of the key features of PyTorch is its strong emphasis on Pythonic code. PyTorch is built on top of Python and integrates well with other Python libraries and tools, making it easy to use for a wide range of applications. PyTorch also provides a number of useful tools and libraries, such as TorchVision for computer vision tasks and TorchText for natural language processing.
- PyTorch has gained a large and active community of users and developers, and is widely used in both industry and academia for a variety of applications, including computer vision, natural language processing, and reinforcement learning. Its combination of flexibility, ease of use, and strong community support has made it a popular choice for deep learning research and development.

Deep learning application

- image classification
- speech recognition
- handwriting transcription
- machine translation
- text-to-speech conversion
- Digital assistants such as Google Now and Amazon Alexa
- autonomous driving
- ad targeting, as used by Google, Baidu, and Bing
- search results on the web
- Ability to answer natural-language questions

Source: deep learning with python by Francois Chollet

Deep learning projects

1. Image classification: Train a deep neural network to classify images into different categories, such as animals, plants, or vehicles. You can use popular datasets like CIFAR-10, ImageNet, or the MNIST dataset of handwritten digits to build your model.
2. Object detection: Use a deep learning model to detect and localize objects within an image or video stream. Popular approaches include the YOLO and Faster R-CNN algorithms.
3. Text classification: Train a deep neural network to classify text documents into different categories, such as spam vs. non-spam or sentiment analysis. You can use popular datasets like the IMDB movie review dataset or the 20 Newsgroups dataset to build your model.

Continue

1. Generative models: Use deep learning models to generate new data that is similar to a given dataset, such as generating new images, music, or text. Popular generative models include Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).
2. Reinforcement learning: Train a deep neural network to learn how to make decisions in an environment based on rewards or punishments. You can build a model to play games like Atari or create a model to control a robot in a simulated environment.
3. Recommendation systems: Use deep learning models to build personalized recommendation systems for products, movies, or music. You can use collaborative filtering or content-based approaches to build your model.

Advantage of deep learning

- The key advantage of deep learning is its ability to learn features directly from raw data, without the need for human-crafted features. This is in contrast to traditional machine learning, where features must be carefully designed by human experts in order to achieve good performance.
- Deep learning is based on artificial neural networks, which are modeled after the structure and function of the human brain. Each layer in a neural network consists of a set of interconnected neurons that perform computations on the input data. By stacking multiple layers together, neural networks can learn increasingly complex representations of the input data.
- Deep learning has achieved state-of-the-art results in a wide range of applications, including computer vision, natural language processing, speech recognition, and robotics. Some of the most notable applications of deep learning include image classification, object detection, speech synthesis, machine translation, and autonomous driving.
- The success of deep learning is largely due to the availability of large amounts of data and powerful computing resources, which have enabled the training of extremely large neural networks. In recent years, deep learning has become a key technology driving advances in AI and is being used to solve some of the most challenging problems in science, engineering, and medicine.

- Data limitations: Deep learning models require large amounts of labeled data to achieve good performance, but in many domains, such data is either scarce or expensive to obtain.
- Interpretability: Deep learning models are often considered to be black boxes, making it difficult to understand how they arrived at their predictions. This can be a barrier to their adoption in fields such as healthcare and finance, where interpretability is important.
- Overfitting: Deep learning models are prone to overfitting, where they memorize the training data instead of learning general patterns. This can lead to poor performance on new, unseen data.
- Computational requirements: Training deep learning models requires significant computational resources, including powerful CPUs or GPUs and large amounts of memory. This can be a barrier to entry for researchers and organizations with limited resources.
- Robustness: Deep learning models can be sensitive to changes in the input data, such as noise or adversarial attacks. Ensuring that models are robust to such perturbations is an active area of research.
- Generalization: Deep learning models can struggle to generalize to new data that is significantly different from the training data. This is particularly true in domains where the distribution of data is constantly changing, such as in natural language processing.
- Privacy concerns: Deep learning models trained on sensitive data can raise privacy concerns if they are used to infer sensitive information about individuals or groups. Ensuring that models are trained in a privacy-preserving manner is an important challenge for the field

What is Neural network

- "The fundamental unit of a neural network is a *node*, which is loosely based on the biological neuron in the brain"
- Researchers estimate that there are more than 500 trillion connections between neurons in the human brain. while the largest artificial neural networks today don't even come close to approaching this number.
- Input data
  - It's a number that you recorded in the real world somewhere.
  - that is easily knowable, like today's temperature, a basketball player's average, or stock price.
- Prediction
  - A *prediction* is what the neural network tells you, *given the input data*, such as given the stock price, will increase or decrease
- Is this prediction always right?
  - No. Sometimes a neural network will make mistakes, but it can learn from them.
  - For example, if it predicts too high, it will adjust its weight to predict lower next time, and vice versa.
- How does the network learn?
  - Trial and error! First, it tries to make a prediction. Then, it sees whether the prediction was too high or too low.
  - Finally, it changes the weight (up or down) to predict more accurately the next time it sees the same input.

Source: deep learning with python by Francois Chollet

What is Neural network

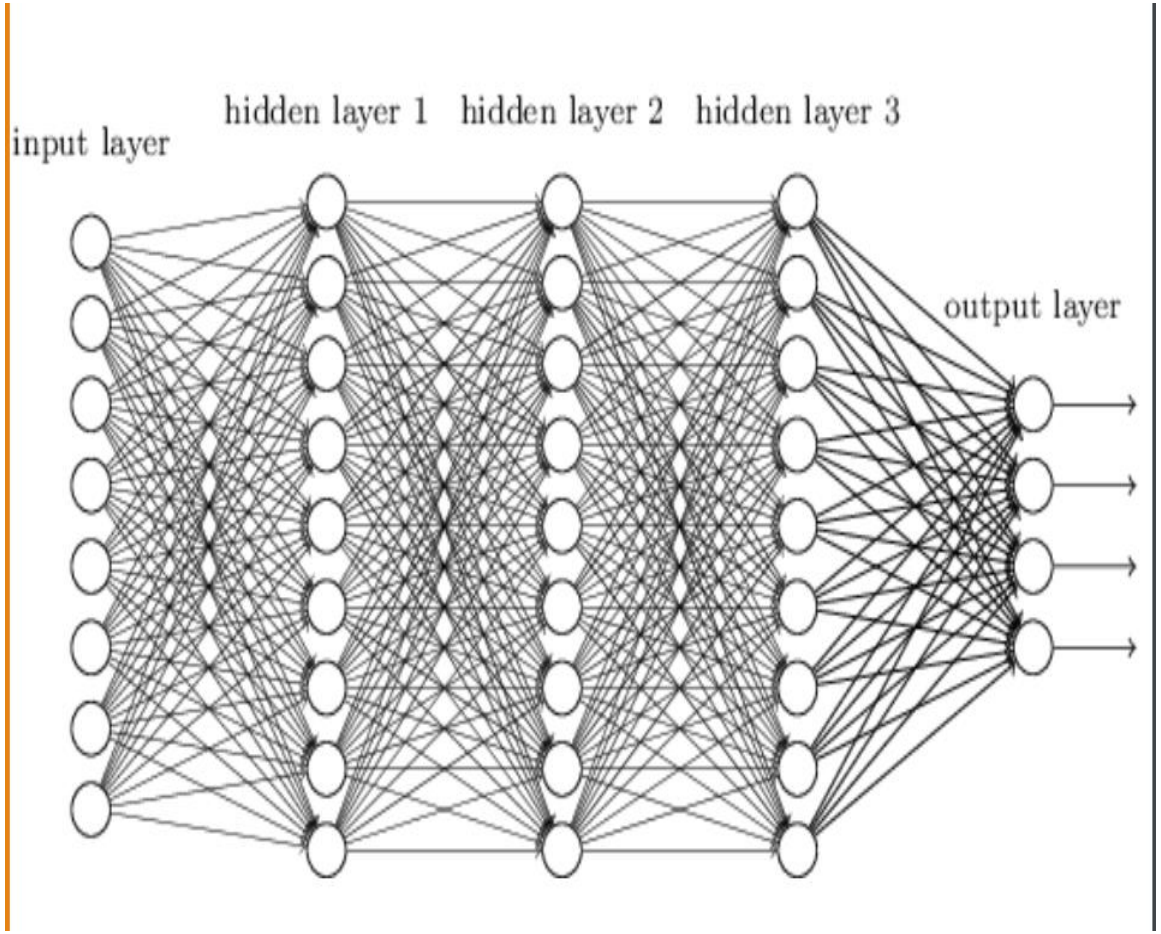
- One useful definition specifies that deep learning deals with a “neural network with more than two layers.”
- More neurons than previous networks
- More complex ways of connecting layers/neurons in NNs
- Explosion in the amount of computing power available to train
- Automatic feature extraction

Source: deep learning with python by Francois Chollet

Deep learning challenges

- Lack of processing power
- Lack of data
- Overfitting (Deep learning's greatest weakness)

  Overfitting is an error that occurs in data modeling as a result of a particular function aligning too closely to a minimal set of data points.



Thank You