Python & Data Engineering
인공지능 직무전환자 과정

- Day 4
Data Preprocessing & Visualization

Sanghyun Seo

shseo@dongguk.edu

0. Review

Course Descriptions

- Python & Data Engineering

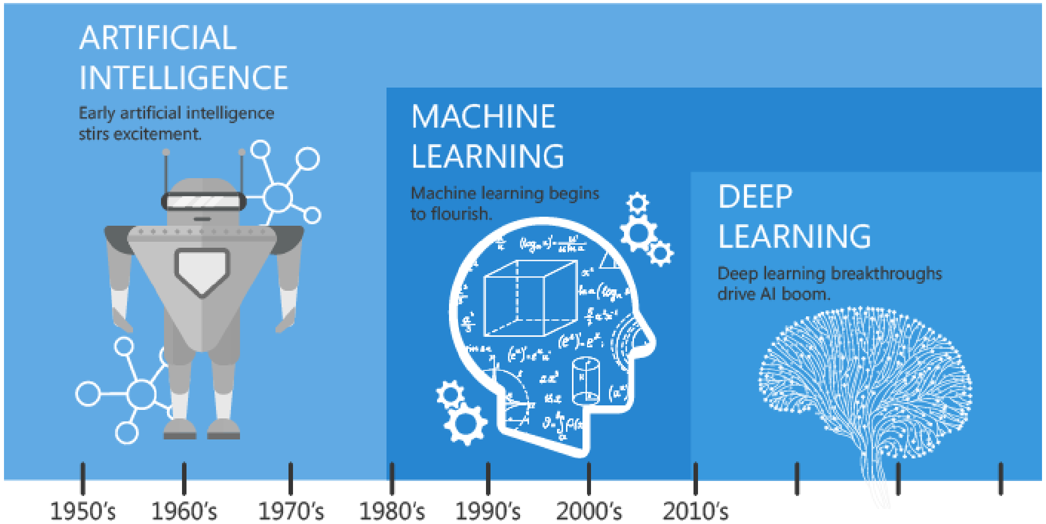
Day 1 – Python Basic	Day 2 – Advanced Python	Day 3 – Numpy, Pandas	Day 4 – Data Preprocessing, Visualization
• Getting started • Introduction to Python • Data Type & Variable • Flow control • Function • Python programming practice	• Review • File • Class • Exception Handling • Advanced python	• Review • Numpy Basic • Advanced Numpy • Pandas Basic • Advanced Pandas	• Review • Introduction to machine learning • Data preprocessing • Visualization • Course Summarization

Contents

1. Introduction to Machine Learning

Machine Learning

- Definition
  - Changes in a system that enable it to perform better on repetition of same task
  - " A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*. " *– Tom M. Mitchell –*

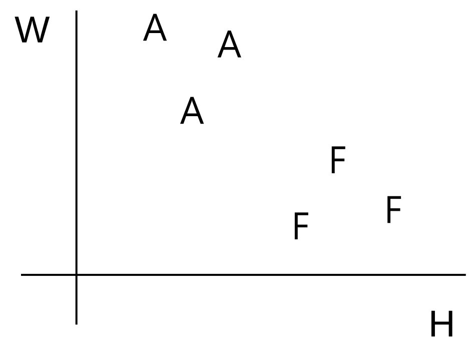


Ref: https://hackernoon.com/difference-between-artificial-intelligence-machine-learning-and-deep-learning-1pcv3zeg

Machine Learning

- Supervised learning
  - Given: **<data, label>** examples (labeled)
  - Learning: **<model>** - rules, trees, neural nets to give the right answer

H	W	Grade
185	65	F
162	80	A
175	70	F
165	92	A
…	…	…

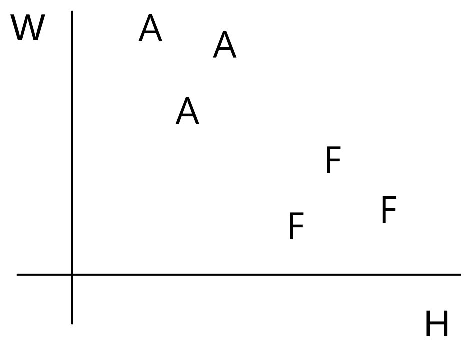


<187, 68> → ?

Machine Learning

- Unsupervised learning
  - Given: **<data>** examples (unlabeled)
  - Learning: **<groups>** of data

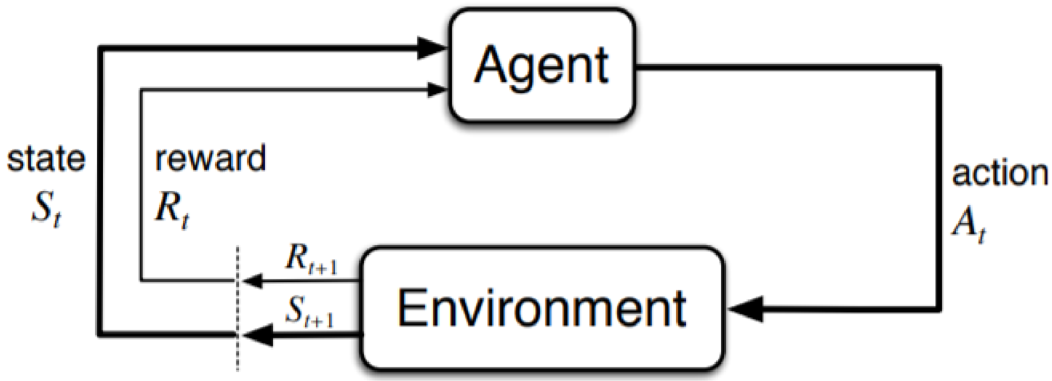
H	W
185	65
160	90
180	70
165	95
…	…



Group ?

Machine Learning

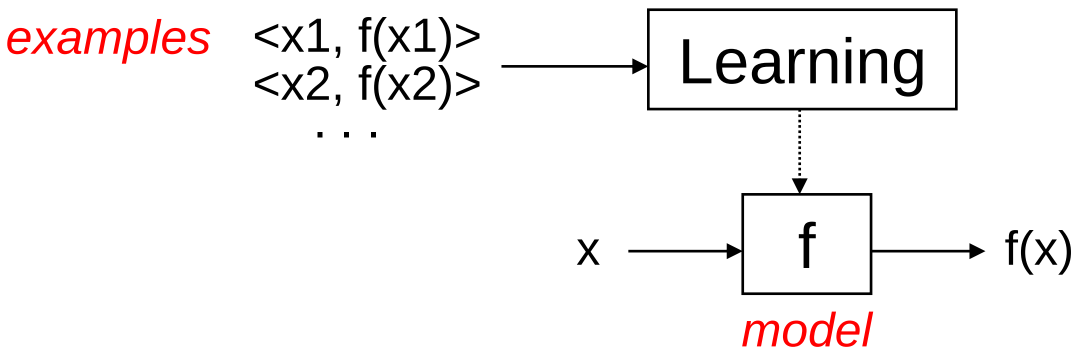
- Reinforcement learning
  - Given: **<action, reward>** experiences
  - Learning: **<rules>** for right action



Ref: http://incompleteideas.net/book/bookdraft2017nov5.pdf

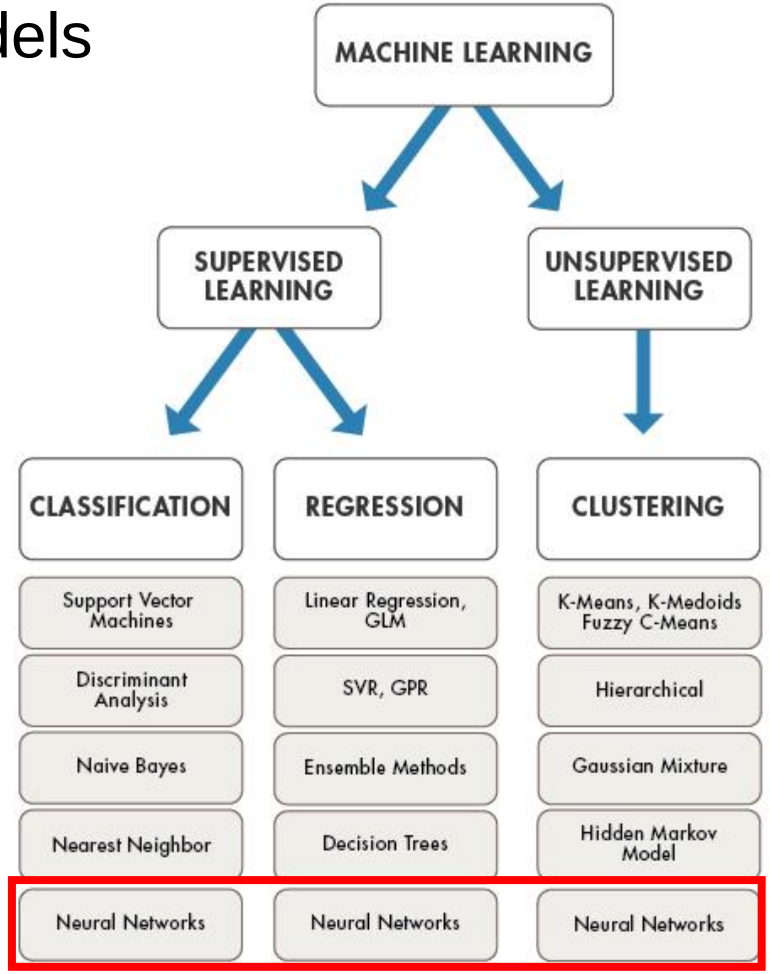
Machine Learning

- Example: <x, f(x)>
  - X: input, f(X): output (f: target function)
- Learning
  - Given a set of examples
  - Find target function f (model or classifier)



Machine Learning

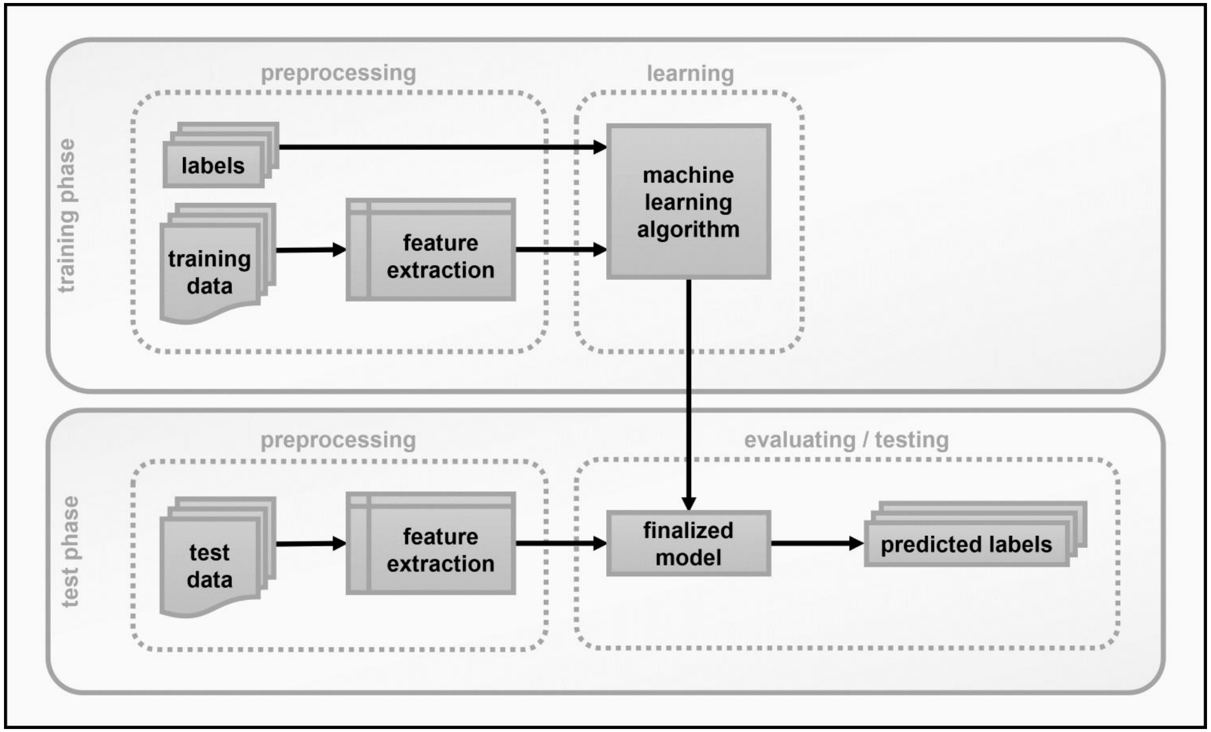
- Various Machine Learning Models
  - Classification
    - Nearest Neighbor
    - Decision Tree
    - Naïve Bayes
    - Support Vector Machine
    - Neural Networks
  - Regression
    - Linear Regression
    - Support Vector Regression
    - Neural Networks
  - Clustering
    - K-means
    - Density based
    - Gaussian Mixture



Ref: https://kr.mathworks.com/help/stats/machine-learning-in-matlab.html

Procedure of Machine Learning

- Data preprocessing
- Learning
- Evaluation(Testing) → Employment



Type of Dataset

- Structured data
  - Numerical, categorical, etc.
- Unstructured data
  - Images, text, audio, video, etc.

Unstructured data

The university has 5600 students.
John's ID is number 1, he is 18 years old and already holds a B.Sc. degree.
David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

Semi-structured data

```
<University>
 <Student ID="1">
   <Name>John</Name>
   <Age>18</Age>
   <Degree>B.Sc.</Degree>
 </Student>
 <Student ID="2">
   <Name>David</Name>
   <Age>31</Age>
   <Degree>Ph.D. </Degree>
 </Student>
....
</University>
```

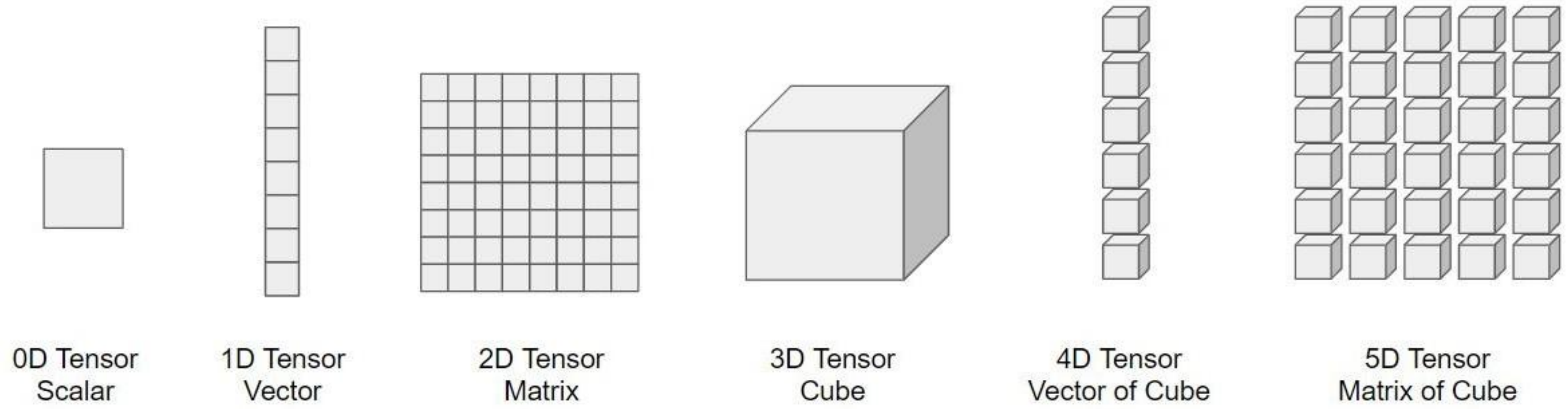
Structured data

ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Robert	51	Ph.D.
4	Rick	26	M.Sc.
5	Michael	19	B.Sc.

Ref:
https://www.researchgate.net/publication/236860222_Developing_Dynamic_Packaging_Applications_using_Semantic_Web_based_Integration/figures?lo=1

Type of Dataset

- 텐서 (Tensor)
  - 선형대수학/물리학에서, 선형 관계를 나타내는 미분기하학의 대상이다. 기본적인 예는 스칼라곱과 선형 변환이 있으며 스칼라와 벡터 또한 해당한다. 텐서는 기저를 선택하여 다차원 배열로 나타낼 수 있으며, 기저를 바꾸는 변환 법칙이 존재한다.
- Tensor in deep learning
  - 다차원의 데이터 구조를 표현한 것, 숫자를 담는 컨테이너
  - 다차원 배열

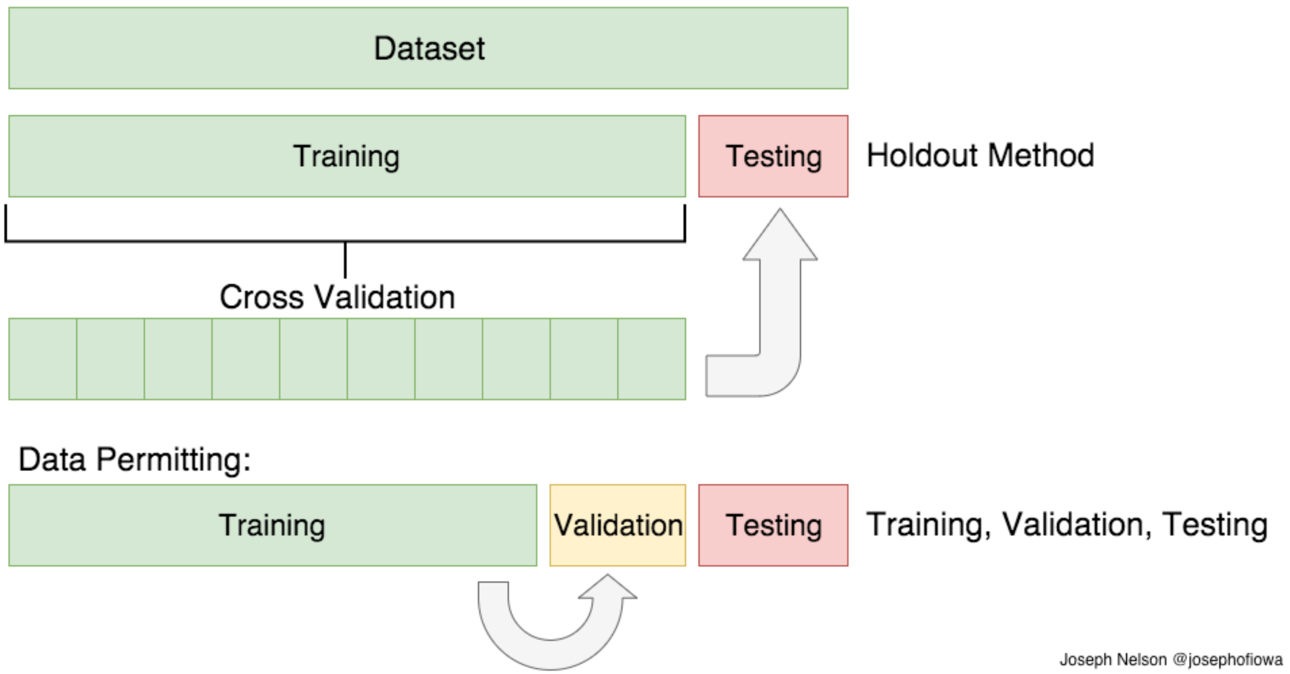


Data Preprocessing

- Data cleaning
  - Fill in the missing values
  - Handle the noise data, identify or remove outliers
- Data transformation
  - Normalization, standardization
  - Discretization
- Feature selection
- Dimensionality reduction
  - Principle Component Analysis

Data Preprocessing

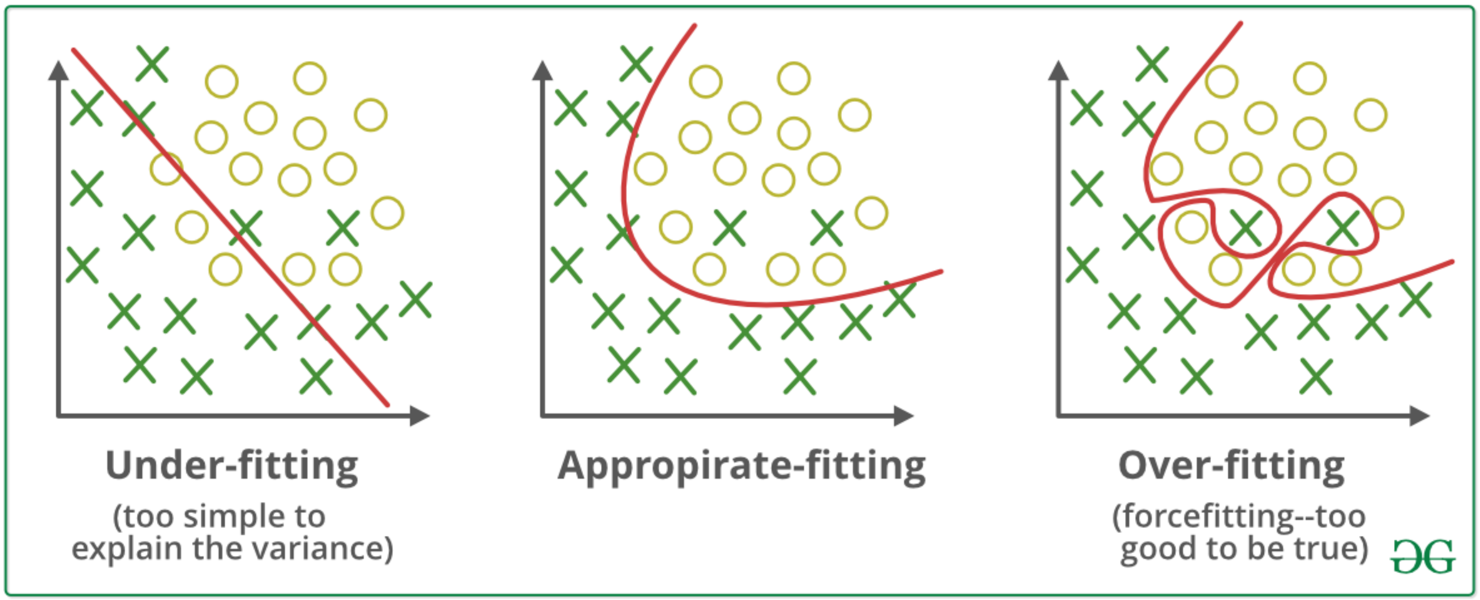
- 데이터 분할 (Dataset Split)
  - Training/Testing
  - Training/Validation/Testing



Ref: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

Evaluation

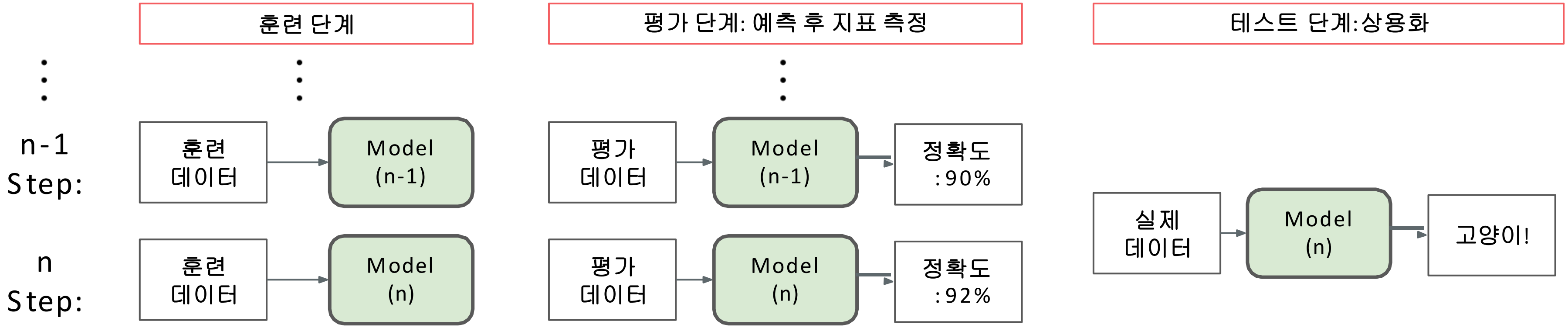
- 과적합(Overfitting)
  - Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data



Ref: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

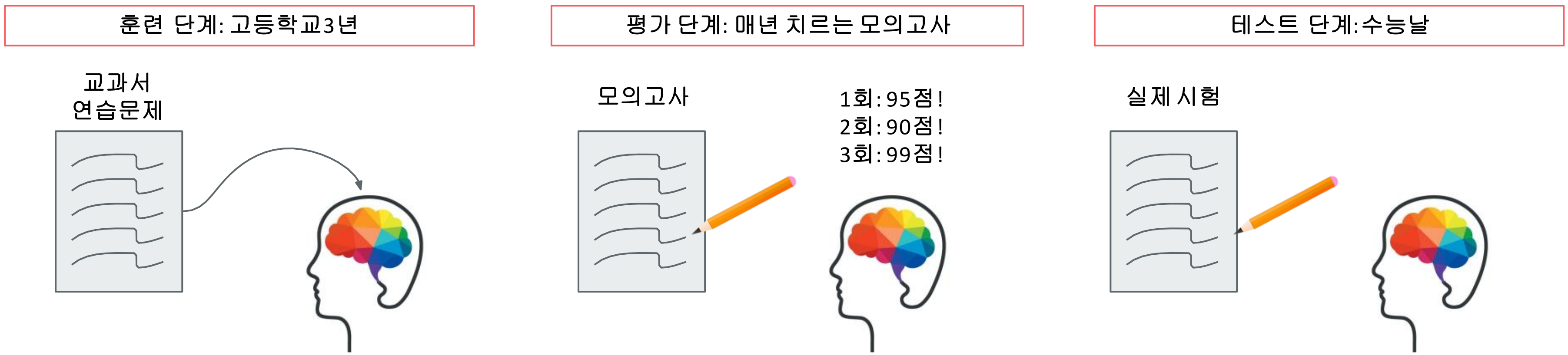
Evaluation

- 머신러닝의 목적 및 평가방법:
  - 목적: 처음보는 데이터도 잘 예측할 수 있게 한다(일반화 generalization 능력).
  - 훈련 단계(Train Phase): 훈련 데이터(Train data, 훈련에 사용되는 데이터)로 훈련
  - 평가 단계(Validation Phase): 일반화 능력을 평가, 특정 지표로 최적의 모델을 선택
  - 테스트 단계(Test Phase): 훈련/평가에 없는 실제 데이터로 테스트 후 상용화



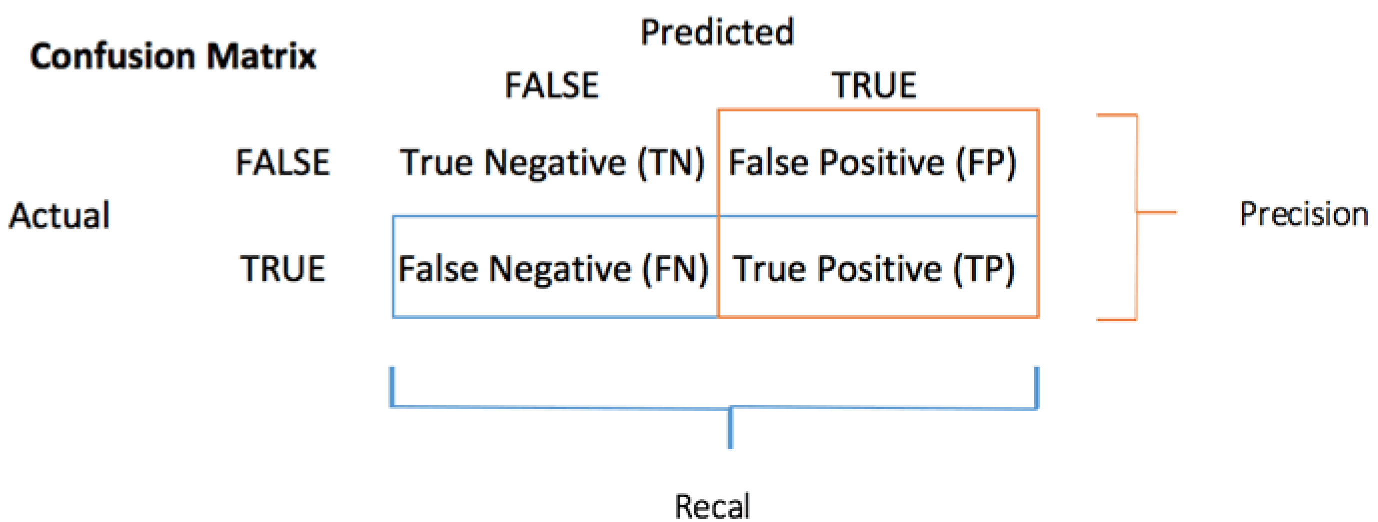
Evaluation

- 평가 단계의 의미:
  - 일반화generalization 능력을 측정한다!
  - 예시: 어떤 수학 지식을 “이해했다”라고 할때는 언제일까?
  - 수학 지식을 “이해했다” = 새로운 문제도 풀수 있다. = 일반화 능력
  - 평가 방법: 시험을 봐서 좋은 성적을 얻었을 때



Evaluation

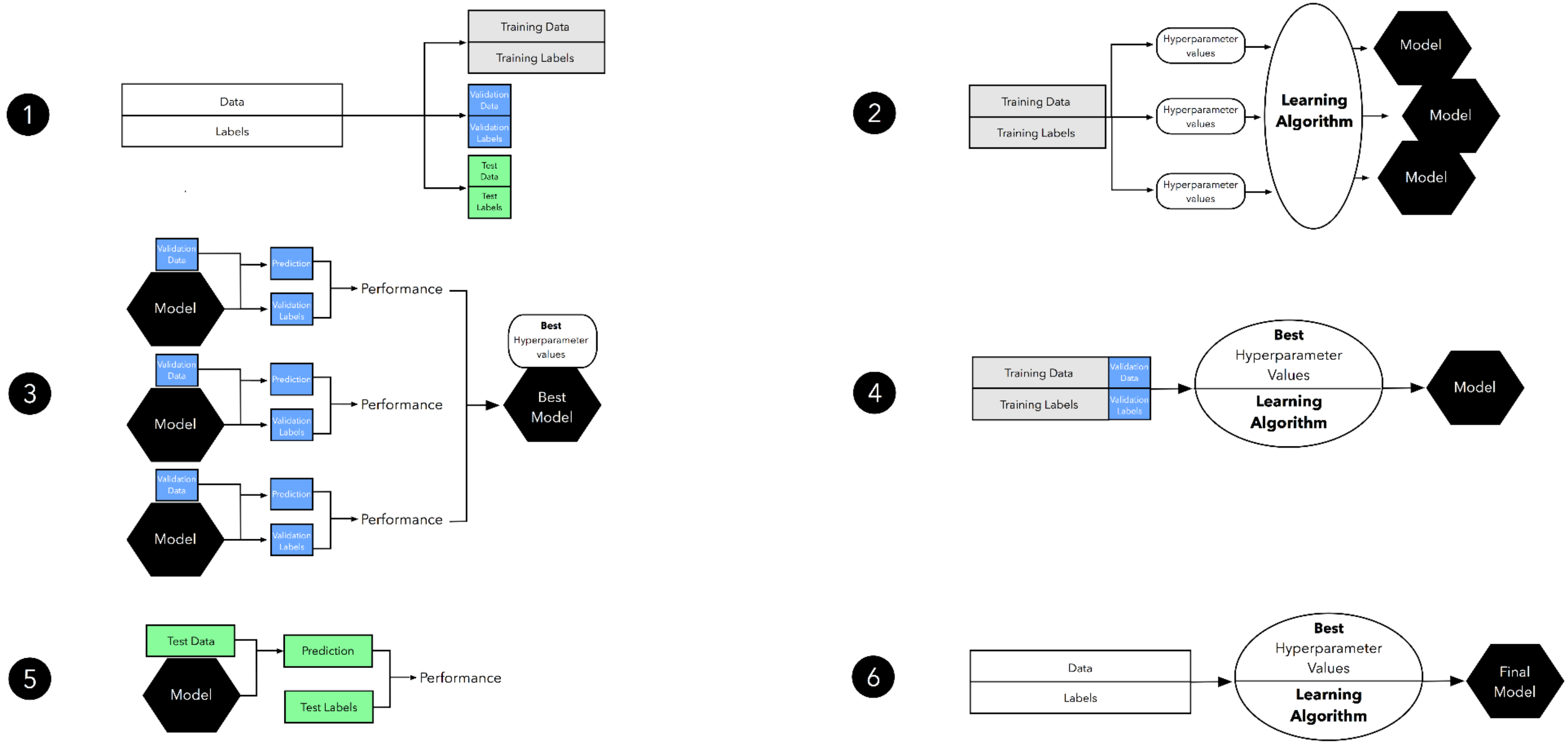
- Evaluation Metrics
  - Confusion matrix
  - $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
  - $Precsion = \frac{TP}{TP+FP}$
  - $Recall = \frac{TP}{TP+FN}$



Ref: https://www.guru99.com/confusion-matrix-machine-learning-example.html

Building Machine Learning Systems

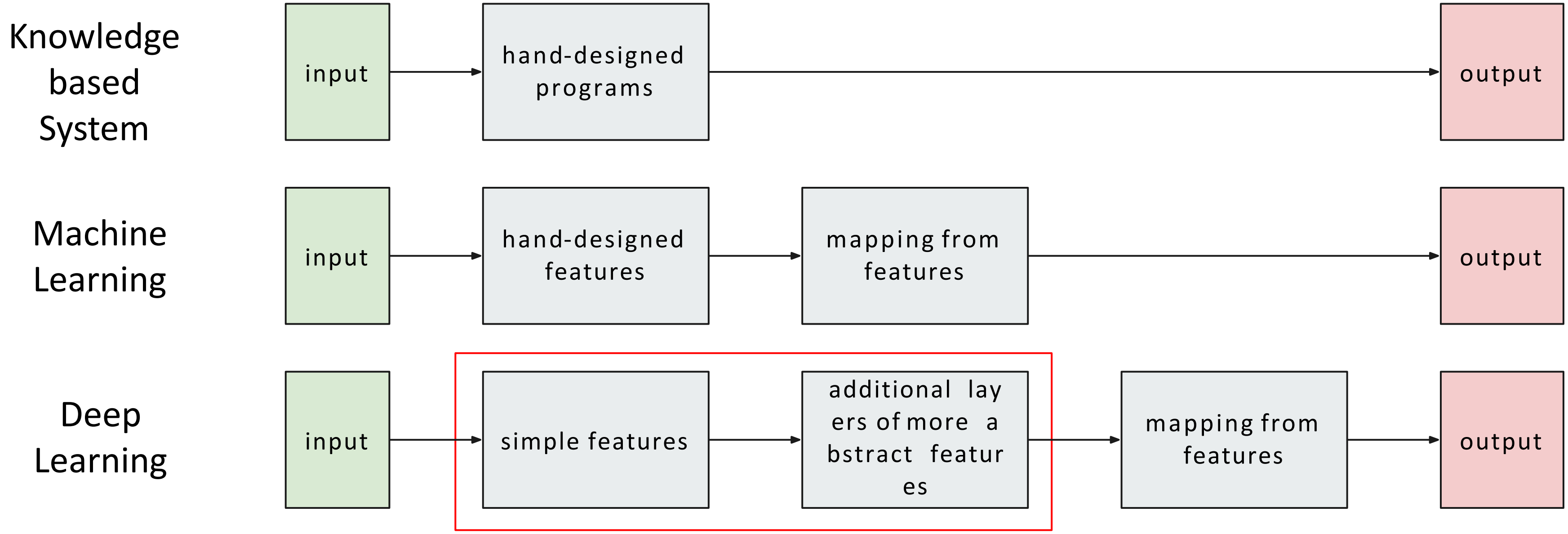
- (1) Data split → (2) learning with various hyperparameter → (3) evaluation on validation dataset → (4) find best model → (5) evaluation on test dataset → (6) deployment



Ref: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part3.html

Deep Learning

- Paradigm shift
  - Large-scale data (Big Data)
  - High computation power (GPU)
  - Deep Learning Framework (PyTorch, Tensorflow, etc.)



2. Data Preprocessing

Data Object

- A ***data object*** represents an entity
  - sales database: customers, store items, sales
  - medical database: patients, treatments
  - university database: students, professors, courses
  - Also called *samples, examples, instances, data points, objects, tuples*.
- Data objects are described by ***attributes***
  - Database rows → data objects; columns → attributes.

Id	Name	Gender	Age	GPA
1043028	Tom Cruise	M	28	3.14
2102019	Emma Stone	F	27	3.35
…	…	…	…	…

Attribute Types

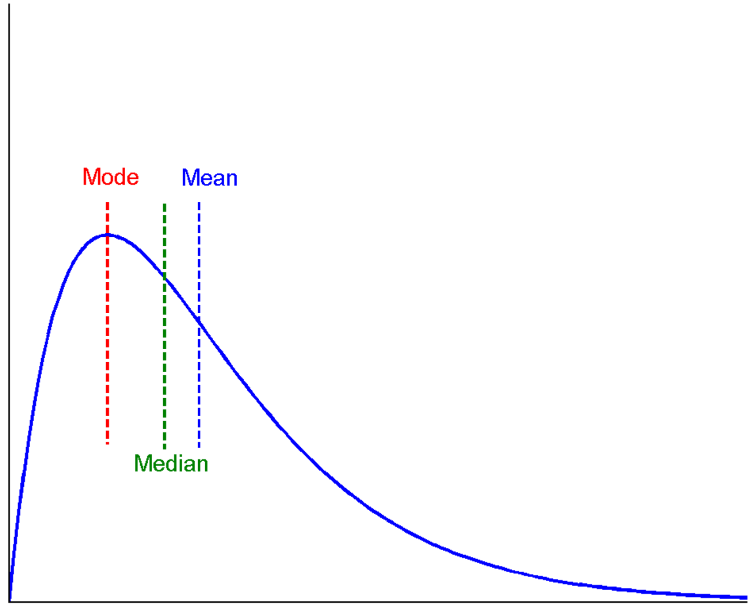
- 명목 (Nominal)
  - categories, states, or “names of things”
  - hair_color = {black, blond, brown}, occupation, zip code
- 이진 (Binary)
  - Nominal attribute with only 2 states (0/1, T/F, Y/N, +/-)
  - has_desease = {0, 1}, student?,
- 순서 (Ordinal)
  - Values have a meaningful order (ranking)
  - size = {small, medium, large}, grade, medal

Attribute Types

- 수치 (Numeric)
  - integer or real-valued
  - temperature = 36.8, age, weight, speed, salary
- 이산 (Discrete) vs. 연속 (Continuous)
  - Discrete: finite or countably infinite set of values
  - age = 25
  - Continuous: real numbers
  - weight = 72.3

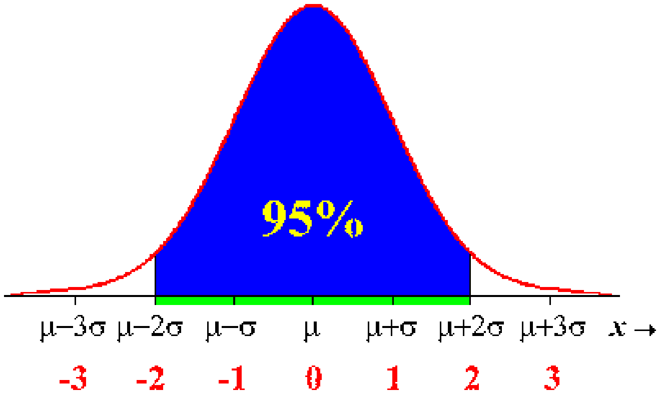
Data Summarization

- Motivation
  - To better understand the data: central tendency, variation and spread
- Mean : $\mu = \frac{\sum x}{N}$
- Median: middle value
- Mode: value that occurs most frequently in the data
- Max/Min

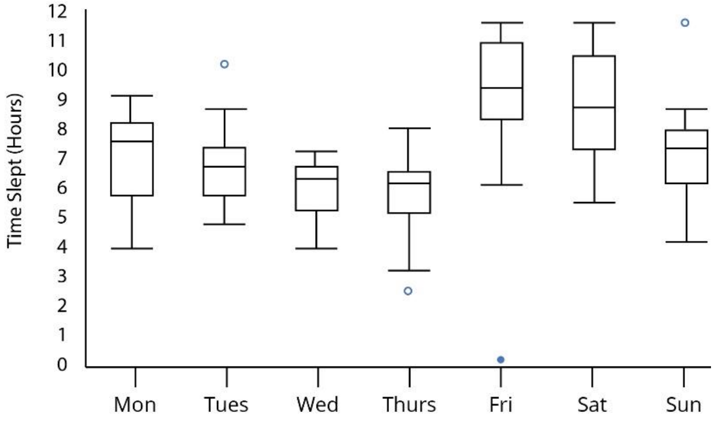


Data Summarization

- Variance $$\sigma^2 = \frac{1}{N}\sum_{i=1}^{n}(x_i - \mu)^2$$
- Standard deviation σ
  - square root of variance
  - For normal distribution, [μ–2σ, μ+2σ] contains about 95% of data

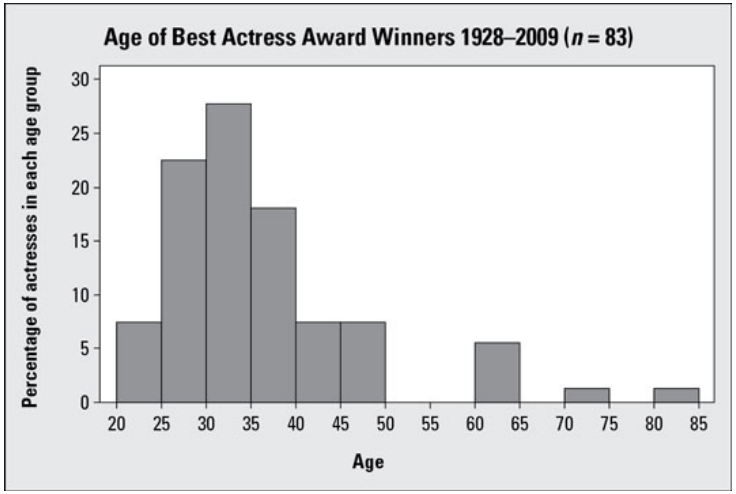


- Quartiles
  - Q1 (25th percentile), Q3 (75th percentile)
  - Boxplot: ends of the box are the quartiles, median is marked

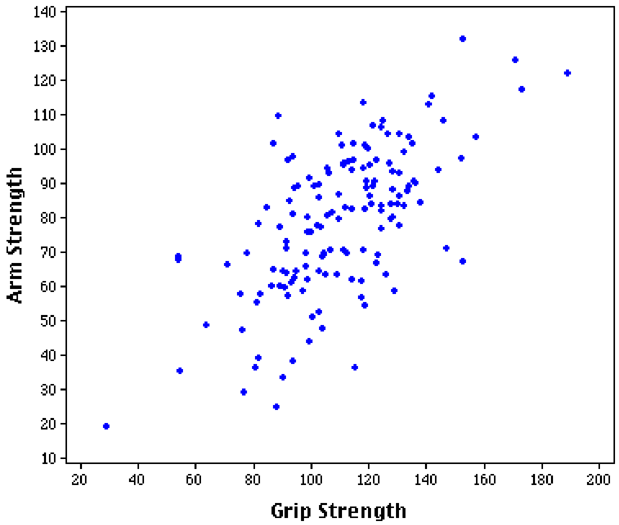


Data Summarization

- Histogram
  - display of tabulated frequencies
  - shows what proportion of cases fall into each of several categories



- Scatter plot
  - pair of values is treated as a pair of coordinates and plotted as points
  - provides a first look at bivariate data to see clusters of points, outliers, etc.



Why Data Preprocessing?

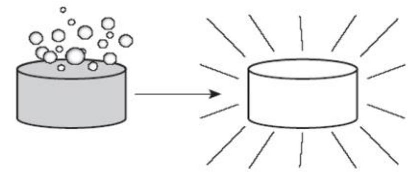
- Data in the real world is dirty
  - incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
    - e.g., occupation=“ ”
  - noisy: containing errors or outliers
    - e.g., Salary=“-10”
  - inconsistent: containing discrepancies in codes or names
    - e.g., “gender” vs. “sex”
    - e.g., sex=“woman” vs. sex=“female”
- No quality data, no quality mining results!
  - Quality decisions must be based on quality data
    - Noisy or missing data may cause misleading statistics
  - → Data warehouse needs consistent integration of quality data

Why Data Preprocessing?

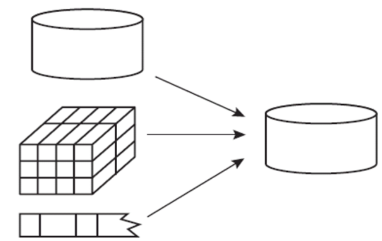
- Incomplete data may come from
  - “Not Applicable” data value when collected
  - Human/hardware/software problems
- Noisy data (incorrect values) may come from
  - Faulty data collection instruments
  - Human or computer error at data entry
  - Errors in data transmission
- Inconsistent data may come from
  - Different data sources
  - Functional dependency violation (e.g., modify some linked data)
- Duplicate records also need data cleaning

Major Tasks

- 데이터 정제 (Data cleaning)
  - Fill in missing values, smooth noisy data, identify or remove outliers, resolve inconsistencies



- 데이터 통합 (Data integration)
  - Integration of multiple databases, data cubes, or files

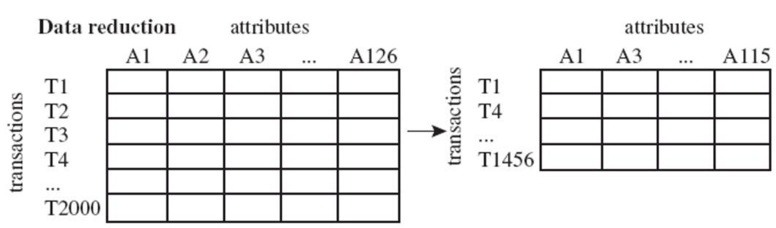


- 데이터 변형 (Data transformation)
  - Normalization, standardization
  - Discretization, Generalization

179, 164, 184, 158, … → 0.89, -0.61, 1.40, -1.28, …

18, 35, 47, 29, 63, 52, … → Y, Y, O, Y, O, O, …

- 데이터 축소 (Data reduction)
  - Obtains reduced representation
  - Sampling, dimensionality reduction



Data Cleaning

- Importance
  - “Data cleaning is one of the three biggest problems in data warehousing”—Ralph Kimball
  - “Data cleaning is the number one problem in data warehousing”—DCI survey
- Data cleaning tasks
  - Fill in missing values
  - Identify outliers and smooth out noisy data
  - Correct inconsistent data
  - Resolve redundancy caused by data integration

Handling Missing Data

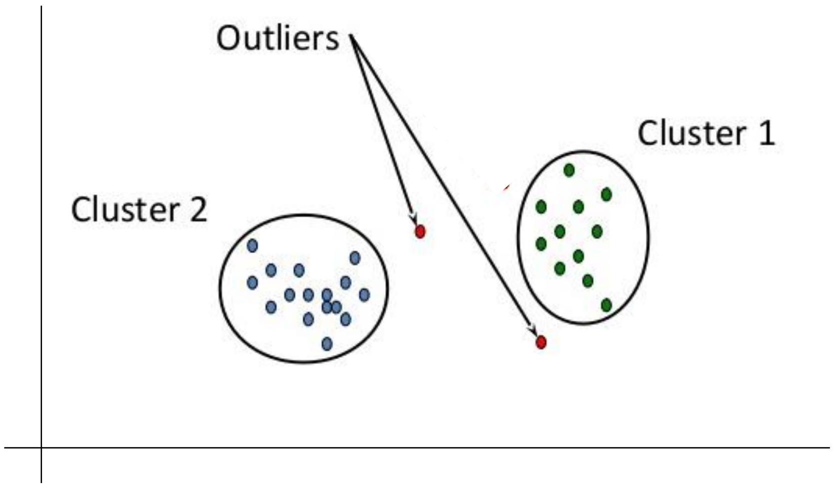
- Ignore the tuple
  - usually done when class label is missing (assuming the tasks in classification)
- Use a global constant
  - Ex> “unknown”, 0, or $-\infty$
- Use the attribute mean
- Use the attribute mean for all samples of the same class
  - Ex> For customer of “risk_high” class → fill in the average of “risk_high” people
- Use the most probable value
  - Inference-based such as Bayesian formula or decision tree

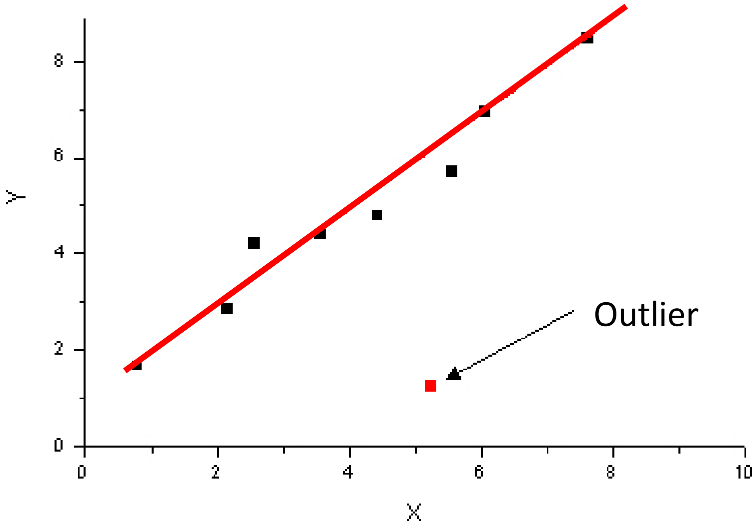
Handling Noisy Data

- Noise
  - Random error or variance in a measured variable
- Incorrect attribute values may due to
  - Data entry problems
  - Error in data collection / data transmission
  - Inconsistency in naming convention
- Handling Noisy Data
  - Binning
  - Outlier detection by clustering
  - Outlier detection by regression

Handling Noisy Data

- Outlier detection
  - Clustering
    - Similar values are organized into groups (clusters)
      → detect and remove outliers
  - Regression
    - Fit the data into regression functions
      → detect and remove outliers





Data Integration

- Data integration
  - Combines data from multiple sources into a coherent store
- Schema integration
  - Integrate metadata from different sources
  - Entity identification problem: identify real world entities from multiple data sources
    - Ex> customer_id $\equiv$ cust-No
- Detecting and resolving data value conflicts
  - For the same real world entity, attribute values from different sources are different
  - Possible reasons: different representations, different scales
    - Ex> 2.1 m vs. 210 cm

Data Integration

- Redundancy
  - One attribute may be a “derived” from another attribute
    - Ex> monthly sales vs. annual sales
- Detecting redundancy
  - Some redundancy can be detected by correlation analysis (how strongly one attribute implies the other)
  - Sample correlation coefficient

$$r_{A,B} = \frac{\sum (A - \bar{A})(B - \bar{B})}{(n-1)\sigma_A \sigma_B}$$

    - r > 0 : highly correlated (A increase → B increase)
    - r = 0 : independant
    - r > 0 : negatively correlated

Data Transformation

- Data transformation: Change data to appropriate form
  - Normalization:
    - Rescale the data into a small, specified range (Ex> [0, 1])
  - Standardization
    - Rescale the data to have 0 mean, 1 standard deviation
  - Discretization
    - Convert continuous values to discretized/nominal values
  - Generalization
    - Concept hierarchy climbing
  - Aggregation
    - Summarization, data cube construction

Normalization/Standardization

- Normalization

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Ex> \$12,000 ~ \$98,000 → [0, 1], then \$45,000 → 0.38

- Standardization: z-score

$$v' = \frac{v - \bar{A}}{\sigma_A}$$

  - Ex> If μ = 54,000, σ = 16,000, then \$45,000 → -0.56

Age	Salary
25	2000000
35	2500000
50	4000000

→

Age	Salary
-0.93	-0.80
-0.13	-0.32
1.06	1.12

Discretization

- Discretization
  - Dividing the range of the attribute into intervals

    → Interval labels can be used to replace actual data values

    → Reduce the number of values for a continuous attribute

    - [140, 220] → { <170, 170≤ }
    - (174, 159, 168, 182, 165, … } → (170≤, <170, <170, 170≤, <170, … }
- Concept hierarchy
  - Defines a discretization
  - Low level concepts → higher level concepts
    - Ex> Age (integer) → {young, middle-aged, senior}

      (18, 15, 27, 14, 19, 63, 32, …) → (Y, Y, M, Y, Y, S, M, …)
  - Can be automatically generated based on data distribution

Practice 1

- Building Good Training Sets
  - 결측치 처리 (Dealing with missing data)

```
import pandas as pd
from io import StringIO
import sys

csv_data = \
'''A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''

df = pd.read_csv(StringIO(csv_data))
Df

df.isnull().sum()

df.values
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

```
A    0
B    0
C    1
D    1
dtype: int64
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6., nan,  8.],
       [10., 11., 12., nan]])
```

Practice 1

- Building Good Training Sets
  - 결측치 처리 (Eliminating samples or features with missing values)

```
# remove rows that contain missing values
df.dropna(axis=0)

# remove columns that contain missing values
df.dropna(axis=1)

# drop rows where all colums are NaN
df.dropna(how="all")

# drop rows where NaN appear in specific columns (for example : "C")
df.dropna(subset=["C"])
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

	A	B	C	D
0	1.0	2.0	3.0	4.0
2	10.0	11.0	12.0	NaN

Practice 1

- Building Good Training Sets
  - 결측치 처리 (Imputing missing values)

```
df.values

# Impute missing values via the column mean
from sklearn.preprocessing import Imputer

imr = Imputer(missing_values='NaN', strategy='mean', axis=0)
imr = imr.fit(df.values)
```

```
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,  nan,   8.],
       [ 10.,  11.,  12.,  nan]])
```

→

```
array([[  1. ,   2. ,   3. ,   4. ],
       [  5. ,   6. ,   7.5,   8. ],
       [ 10. ,  11. ,  12. ,   6. ]])
```

Practice 1

- Building Good Training Sets
  - Handling categorical data

```
import pandas as pd

df = pd.DataFrame([['green', 'M', 10.1, 'class2'],
['red', 'L', 13.5, 'class1'],
['blue', 'XL', 15.3, 'class2']])

df.columns = ['color', 'size', 'price', 'classlabel']
df
```

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

  - Mapping ordinal features

```
size_mapping = {'XL': 3, 'L': 2, 'M': 1}
df['size'] = df['size'].map(size_mapping)
Df
```

	color	size	price	classlabel
0	green	1	10.1	class2
1	red	2	13.5	class1
2	blue	3	15.3	class2

```
inv_size_mapping = {v: k for k, v in size_mapping.items()}
df['size'] = df['size'].map(inv_size_mapping)
df
```

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

Practice 1

- Building Good Training Sets
  - Encoding class labels

```
import numpy as np
# create a mapping dict to convert class labels from strings to integers
class_mapping = {label: idx for idx, label in enumerate(np.unique(df['classlabel']))}
class_mapping

# to convert class labels from strings to integers
df['classlabel'] = df['classlabel'].map(class_mapping)
df

# reverse the class label mapping
inv_class_mapping = {v: k for k, v in class_mapping.items()}
df['classlabel'] = df['classlabel'].map(inv_class_mapping)
df

from sklearn.preprocessing import LabelEncoder
# Label encoding with sklearn's LabelEncoder
class_le = LabelEncoder()

y = class_le.fit_transform(df['classlabel'].values)
y

# reverse mapping
class_le.inverse_transform(y)
```

Practice 1

- Building Good Training Sets
  - Encoding class labels

```
import numpy as np
# create a mapping dict to convert class labels from strings to integers
class_mapping = {label: idx for idx, label in enumerate(np.unique(df['classlabel']))}
class_mapping
```

```
{'class1': 0, 'class2': 1}
```

```
# to convert class labels from strings to integers
df['classlabel'] = df['classlabel'].map(class_mapping)
df
```

	color	size	price	classlabel
0	green	M	10.1	1
1	red	L	13.5	0
2	blue	XL	15.3	1

```
# reverse the class label mapping
inv_class_mapping = {v: k for k, v in class_mapping.items()}
df['classlabel'] = df['classlabel'].map(inv_class_mapping)
df
```

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

```
from sklearn.preprocessing import LabelEncoder
# Label encoding with sklearn's LabelEncoder
class_le = LabelEncoder()

y = class_le.fit_transform(df['classlabel'].values)
y
```

```
array([1, 0, 1])
```

```
# reverse mapping
class_le.inverse_transform(y)
```

```
array(['class2', 'class1', 'class2'], dtype=object)
```

Practice 1

- Building Good Training Sets
  - Performing one-hot encoding on nominal features

```
df['size'] = df['size'].map(size_mapping)
X = df[['color', 'size', 'price']].values

color_le = LabelEncoder()
X[:, 0] = color_le.fit_transform(X[:, 0])
X
```

```
array([[1, 1, 10.1],
       [2, 2, 13.5],
       [0, 3, 15.3]], dtype=object)
```

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(categorical_features=[0])
ohe.fit_transform(X).toarray()
```

```
array([[ 0. ,  1. ,  0. ,  1. , 10.1],
       [ 0. ,  0. ,  1. ,  2. , 13.5],
       [ 1. ,  0. ,  0. ,  3. , 15.3]])
```

Practice 1

- Building Good Training Sets
  - Performing one-hot encoding on nominal features

```
# return dense array so that we can skip
# the toarray step
ohe = OneHotEncoder(categorical_features=[0], sparse=False)
ohe.fit_transform(X)

# one-hot encoding via pandas
pd.get_dummies(df[['price', 'color', 'size']])

# multicollinearity guard in get_dummies
pd.get_dummies(df[['price', 'color', 'size']], drop_first=True)

# multicollinearity guard for the OneHotEncoder
ohe = OneHotEncoder(categorical_features=[0])
ohe.fit_transform(X).toarray()[:, 1:]
```

Practice 1

- Building Good Training Sets
  - Performing one-hot encoding on nominal features

```
# return dense array so that we can skip
# the toarray step
ohe = OneHotEncoder(categorical_features=[0], sparse=False)
ohe.fit_transform(X)
```

```
array([[ 0. ,  1. ,  0. ,  1. , 10.1],
       [ 0. ,  0. ,  1. ,  2. , 13.5],
       [ 1. ,  0. ,  0. ,  3. , 15.3]])
```

```
# one-hot encoding via pandas
pd.get_dummies(df[['price', 'color', 'size']])

# multicollinearity guard in get_dummies
pd.get_dummies(df[['price', 'color', 'size']], drop_first=True)

# multicollinearity guard for the OneHotEncoder
ohe = OneHotEncoder(categorical_features=[0])
ohe.fit_transform(X).toarray()[:, 1:]
```

	price	size	color_blue	color_green	color_red
0	10.1	1	0	1	0
1	13.5	2	0	0	1
2	15.3	3	1	0	0

	price	size	color_green	color_red
0	10.1	1	1	0
1	13.5	2	0	1
2	15.3	3	0	0

```
array([[ 1. ,  0. ,  1. , 10.1],
       [ 0. ,  1. ,  2. , 13.5],
       [ 0. ,  0. ,  3. , 15.3]])
```

Practice 2

- Data transformation
  - Example - Load the wine dataset

```
import pandas as pd
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue',
                   'OD280/OD315 of diluted wines', 'Proline']
df_wine.head()
```

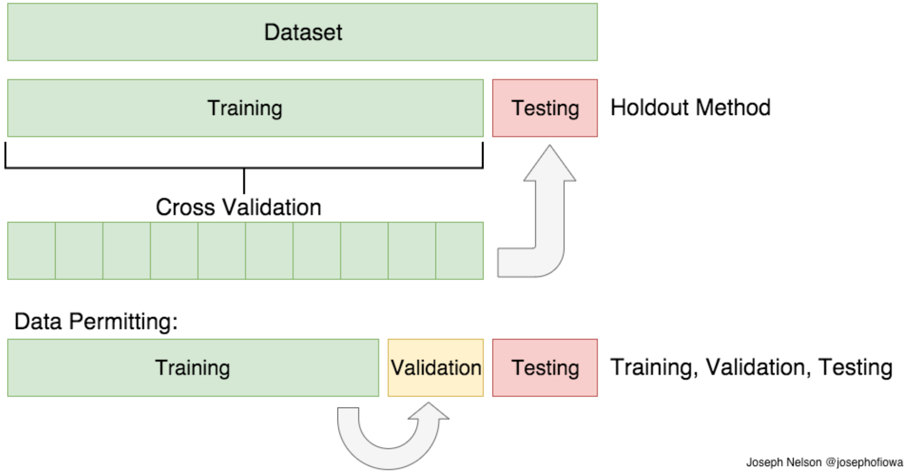
	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Practice 2

- Data transformation

```
from sklearn.model_selection import train_test_split

X = df_wine.iloc[:, 1:].values
y = df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    stratify=y,
                                                    random_state=0)

print("₩n Total dataset")
print(type(X))
print(X.shape)

print("₩n Training dataset")
print(type(X_train))
print(X_train.shape)

print("₩n Test dataset")
print(type(X_test))
print(X_test.shape)
```

```
 Total dataset
<class 'numpy.ndarray'>
(178, 13)

 Training dataset
<class 'numpy.ndarray'>
(124, 13)

 Test dataset
<class 'numpy.ndarray'>
(54, 13)
```




```
print("min: ", np.min(X_train))
print("max: ", np.max(X_train))
print("mean: ", np.mean(X_train))
X_train[0]
```

```
min:  0.13
max:  1680.0
mean:  69.73432382071961
array([1.362e+01, 4.950e+00, 2.350e+00, 2.000e+01, 9.200e+01, 2.000e+00,
       8.000e-01, 4.700e-01, 1.020e+00, 4.400e+00, 9.100e-01, 2.050e+00,
       5.500e+02])
```

Practice 2

- Data transformation
  - Brining features onto the same scale – Normalization
    - Change all values in the range [0, 1]

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  - Brining features onto the same scale – Standardization
    - Transform all values to have zero mean and unit variance

$$x_{new} = \frac{x - \mu}{\sigma}$$

```
ex = np.array([0, 1, 2, 3, 4, 5])

# normalize
print('normalized:', (ex - ex.min()) / (ex.max() - ex.min()))
# standardize
print('standardized:', (ex - ex.mean()) / ex.std())
```

➡

```
normalized: [ 0.   0.2  0.4  0.6  0.8  1. ]
standardized: [-1.46385011 -0.87831007 -0.29277002  0.29277002  0.87831007  1.46385011]
```

Practice 2

- Data transformation
  - Brining features onto the same scale

```
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
print(X[0:3,0:5])
```

```
[[  14.23    1.71    2.43   15.6    127.  ]
 [  13.2     1.78    2.14   11.2    100.  ]
 [  13.16    2.36    2.67   18.6    101.  ]]
```

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
X_norm = mms.fit_transform(X)
print(X_norm[0:3, 0:5])
```

```
[[ 0.84210526  0.1916996   0.57219251  0.25773196  0.61956522]
 [ 0.57105263  0.2055336   0.4171123   0.03092784  0.32608696]
 [ 0.56052632  0.3201581   0.70053476  0.41237113  0.33695652]]
```

```
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_std = mms.fit_transform(X)
print(X_std[0:3, 0:5])
```

```
[[ 1.51861254 -0.5622498   0.23205254 -1.16959318  1.91390522]
 [ 0.24628963 -0.49941338 -0.82799632 -2.49084714  0.01814502]
 [ 0.19687903  0.02123125  1.10933436 -0.2687382   0.08835836]]
```

3. Visualization

matplotlib

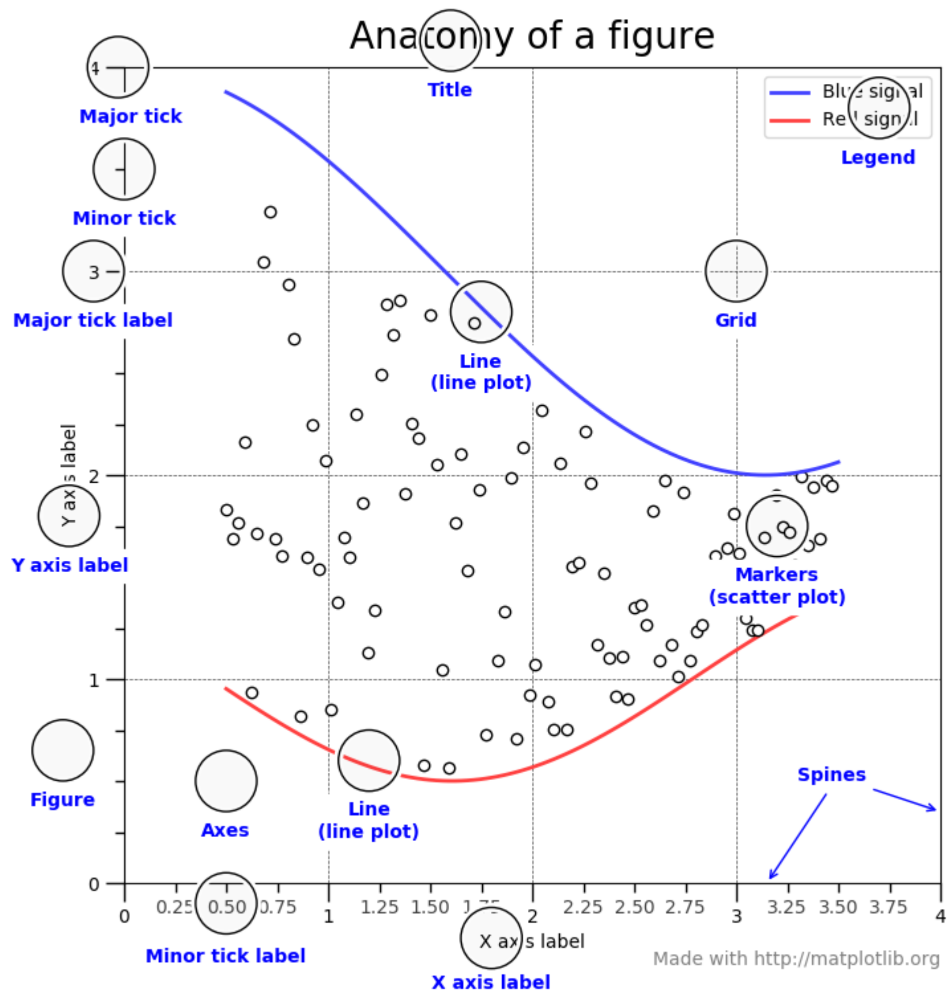
- matplotlib
  - python에서 가장 대표적인 시각화 패키지
  - seaborn, plotnine, plotly 등 다양한 시각화 패키지가 있음
  - naming convention

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

```
# 주피터 노트북에서 대화형 시각화를 사용하기 위해 포함되어야 하는 코드! 실
%matplotlib notebook
```

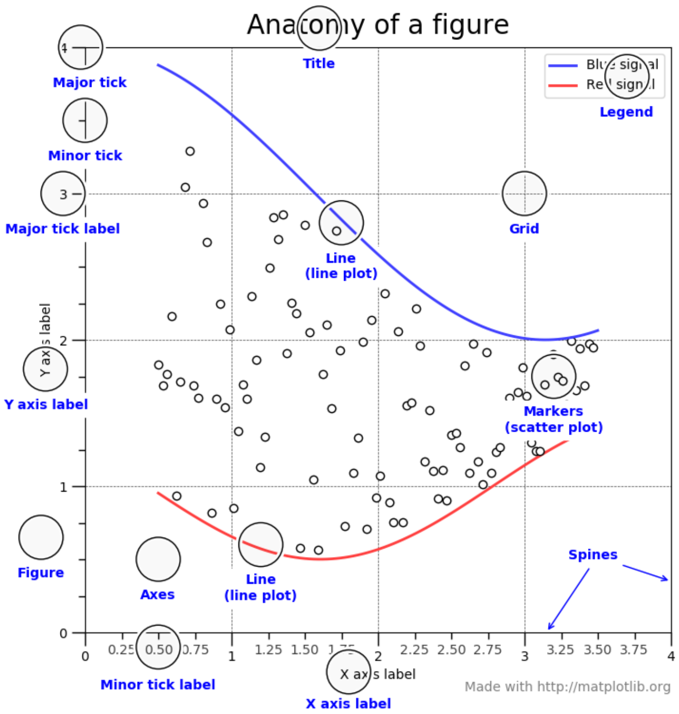
matplotlib

- matplotlib
  - anatomy of a figure
  - https://matplotlib.org/3.1.3/gallery/showcase/anatomy.html



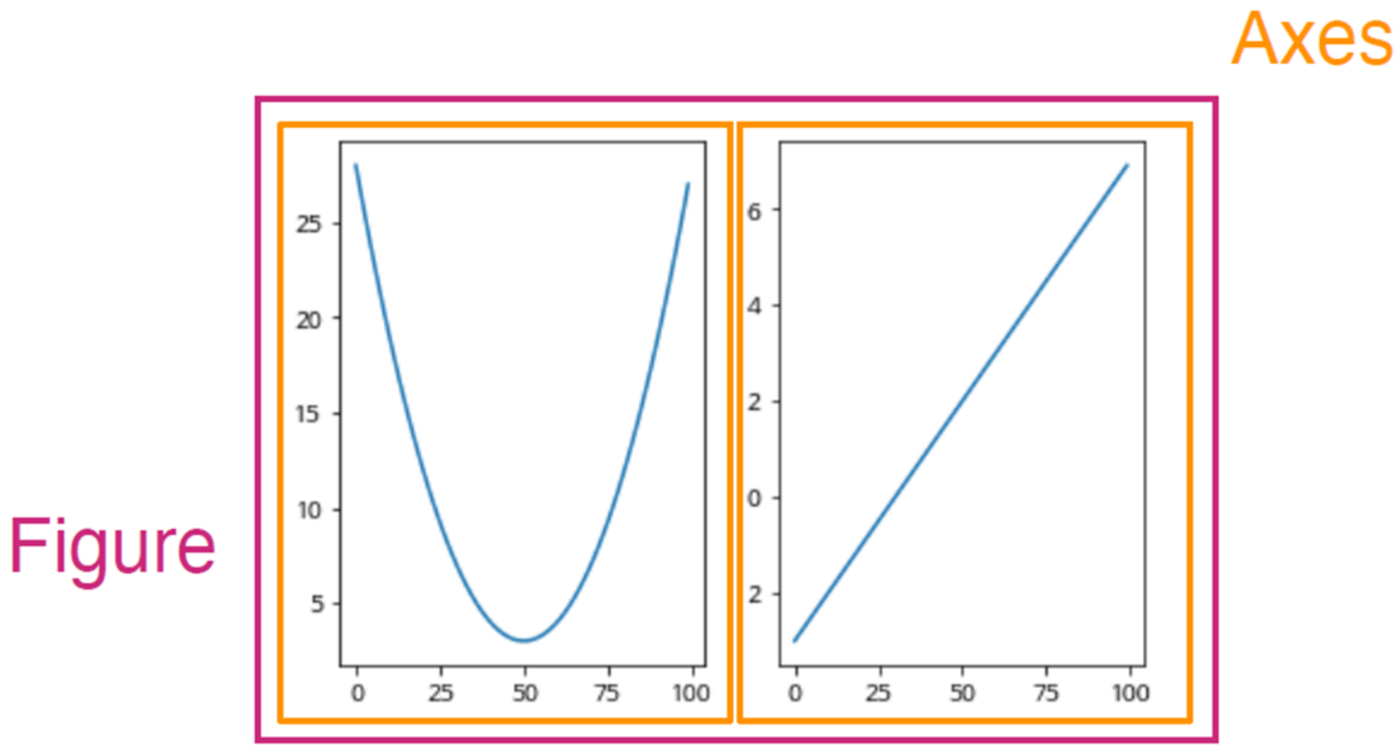
matplotlib

- **Spines**: Lines connecting the axis tick marks
- **Title**: Text label of the whole Figure object
- **Legend**: They describe the content of the plot
- **Grid**: Vertical and horizontal lines used as an extension of the tick marks
- **X/Y axis label**: Text label for the X/Y axis below the spines
- **Minor tick**: Small value indicators between the major tick marks
- **Minor tick label**: Text label that will be displayed at the minor ticks
- **Major tick**: Major value indicators on the spines
- **Major tick label**: Text label that will be displayed at the major ticks
- **Line**: Plotting type that connects data points with a line
- **Markers**: Plotting type that plots every data point with a defined marker



Figures & Axes

- 주요 구성요소: Figure & Axis



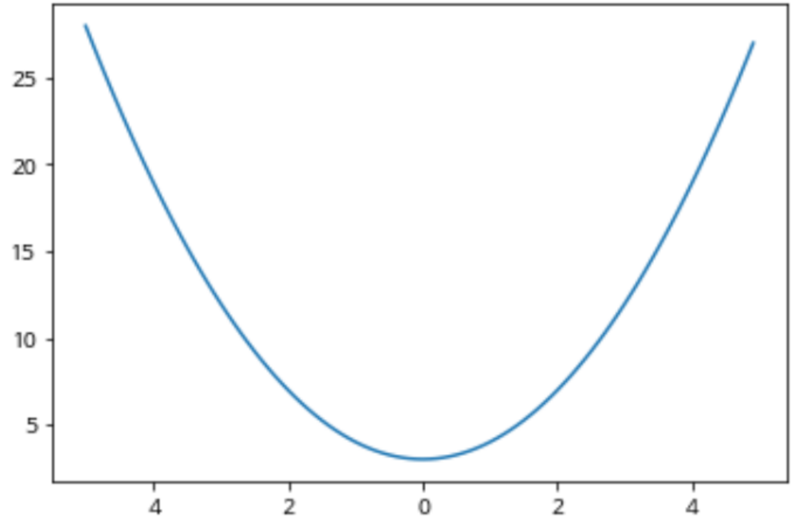
Figures & Axes

- the pyplot API

```
x = np.arange(-5, 5, 0.1)
y1 = x**2 + 3
y2 = x + 2
```

The pyplot API

```
plt.plot(x, y1)
plt.show()
```



Figures & Axes

- plt.figure
  - Making Figures
  - matplotlib.pyplot.figure

matplotlib.pyplot.figure

matplotlib.pyplot.figure(*num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs*) [source]

Create a new figure, or activate an existing figure.

- 'fig'를 object로 할당하여 사용 시 다양한 method 사용가능

```
[1]  import matplotlib.pyplot as plt
     import numpy as np
```

```
[2]  fig = plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```

Figures & Axes

- fig.add_subplot (arguments)

```
add_subplot(self, *args, **kwargs)
```

[source]

Add an Axes to the figure as part of a subplot arrangement.

Call signatures:

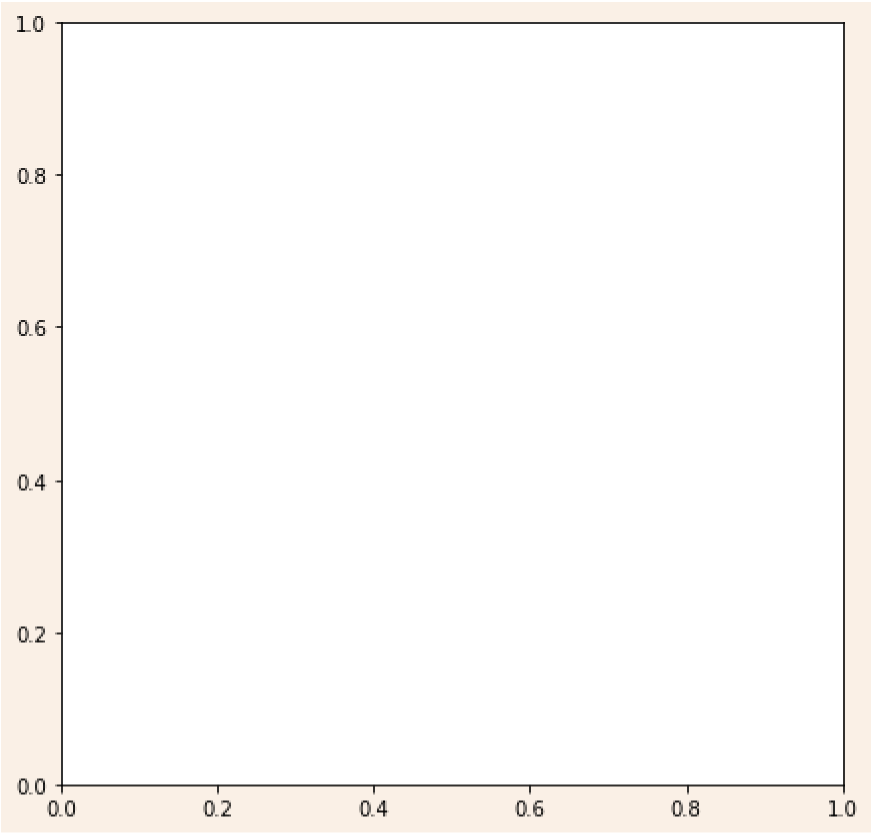
```
add_subplot(nrows, ncols, index, **kwargs)
add_subplot(pos, **kwargs)
add_subplot(ax)
add_subplot()
```

[5]

```
fig=plt.figure(figsize=(7, 7),
               facecolor='linen')
ax = fig.add_subplot(1, 1, 1)
```

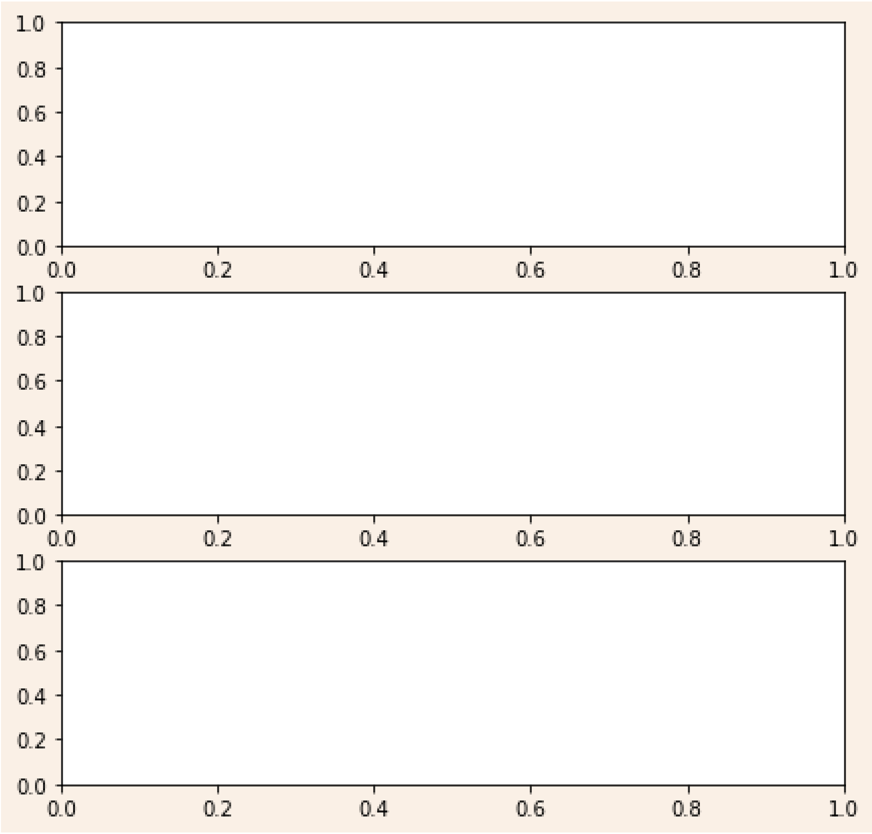
Figures & Axes

- fig.add_subplot (single ax)
  - ax = fig.add_subplot(1, 1, 1)
  - argument: (row, col, idx)



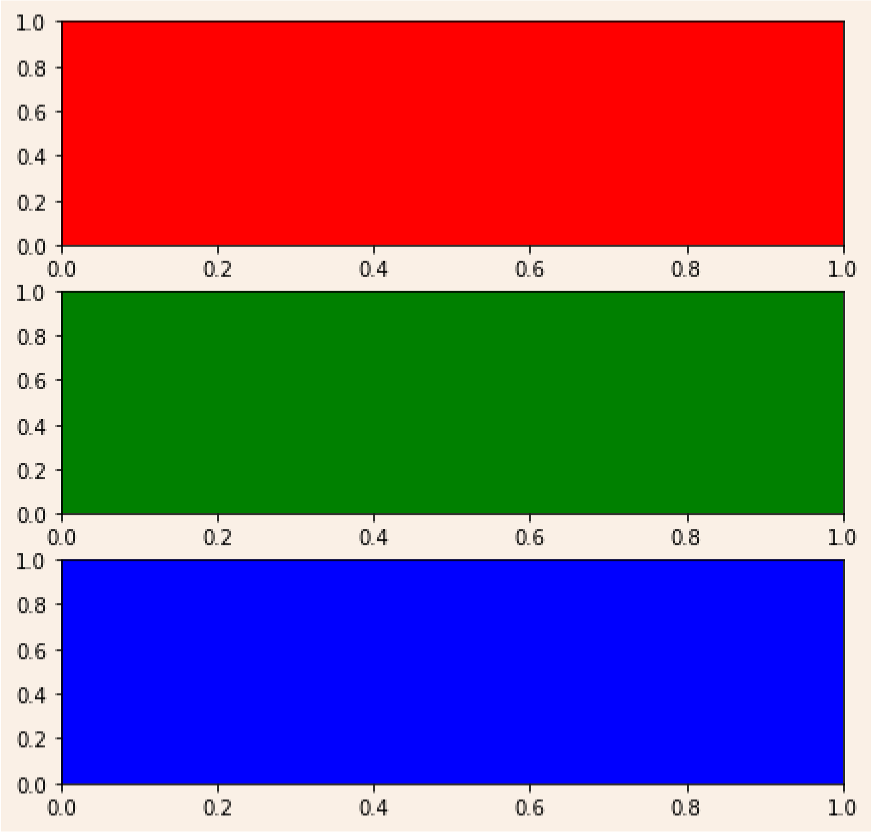
Figures & Axes

- fig.add_subplot (1D axes)
  - ax1 = fig.add_subplot(3, 1, 1)
  - ax2 = fig.add_subplot(3, 1, 2)
  - ax3 = fig.add_subplot(3, 1, 3)



Figures & Axes

- fig.add_subplot (1D axes)
  - ax1 = fig.add_subplot(3, 1, 1, facecolor=‘r’)
  - ax2 = fig.add_subplot(3, 1, 2, facecolor=‘g’)
  - ax3 = fig.add_subplot(3, 1, 3, facecolor=‘b’)



Figures & Axes

- HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision (ICCV 2019)

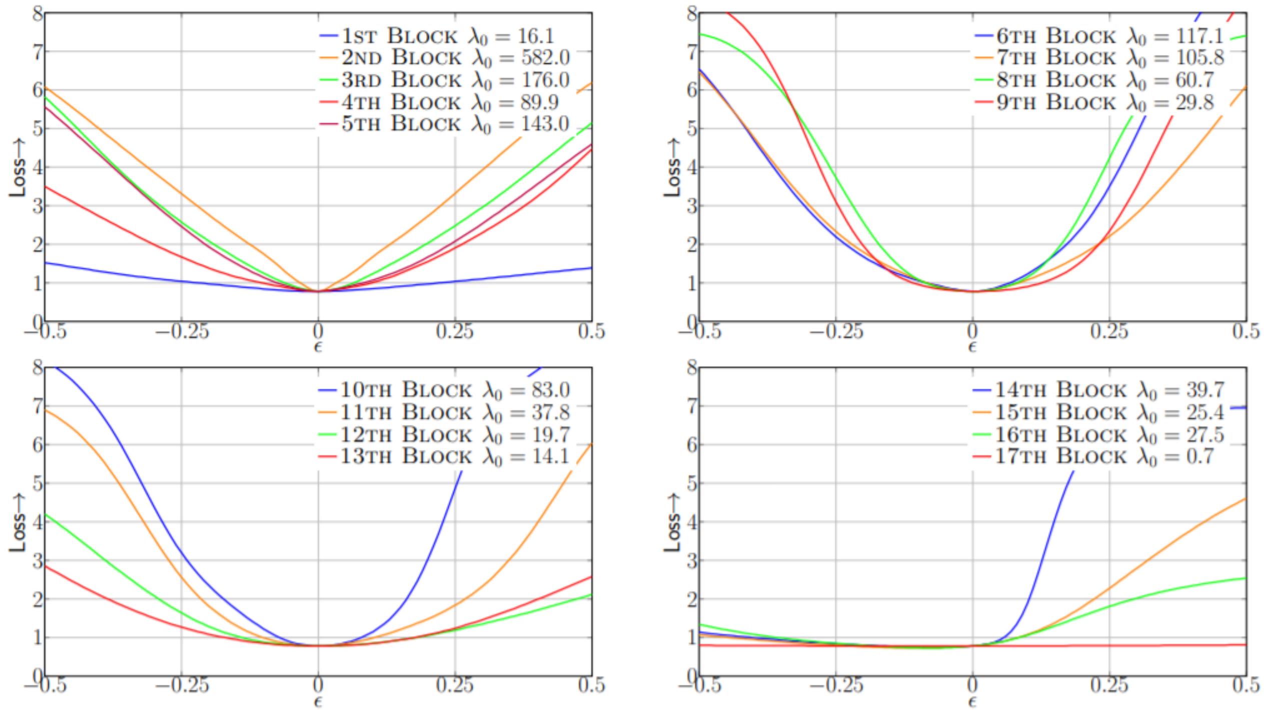
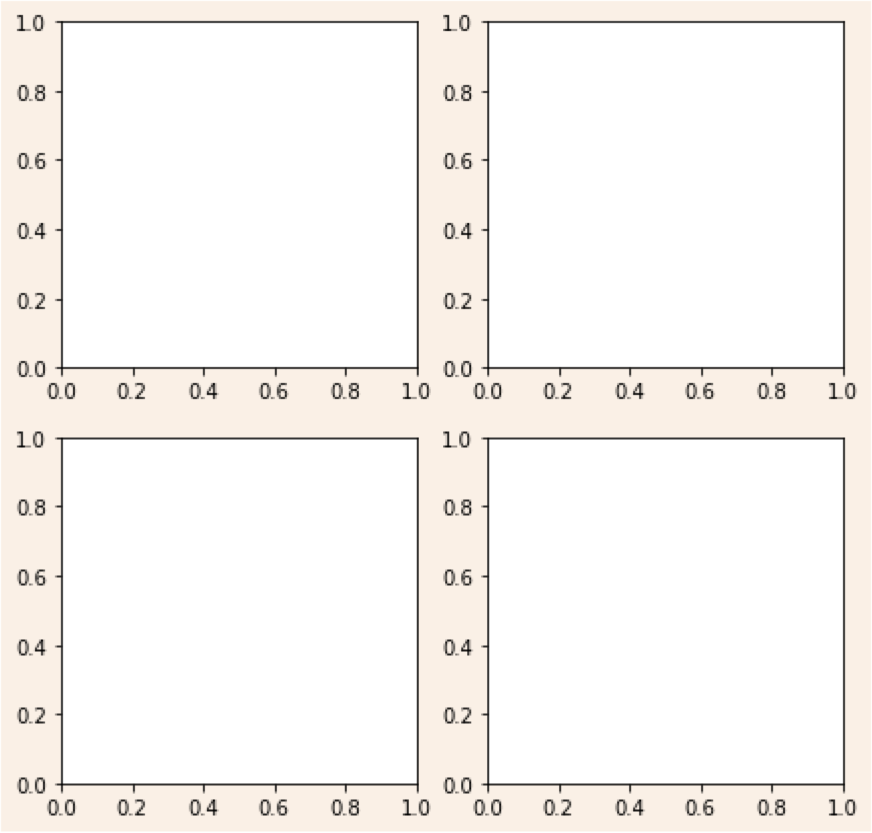


Fig. 3: *1-D loss landscape of all blocks of Inception-V3 on ImageNet along the first dominant eigenvector of the Hessian. Here ϵ is the scalar that perturbs the parameters of the corresponding block along the first dominant eigenvectors.*

Figures & Axes

- fig.add_subplot (2D axes)
  - axes grid [3, 3, i-th axes]

  - ax1 = fig.add_subplot(2, 2, 1)
  - ax2 = fig.add_subplot(2, 2, 2)
  - ax3 = fig.add_subplot(2, 2, 3)
  - ax4 = fig.add_subplot(2, 2, 4)



Figures & Axes

- HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision (ICCV 2019)

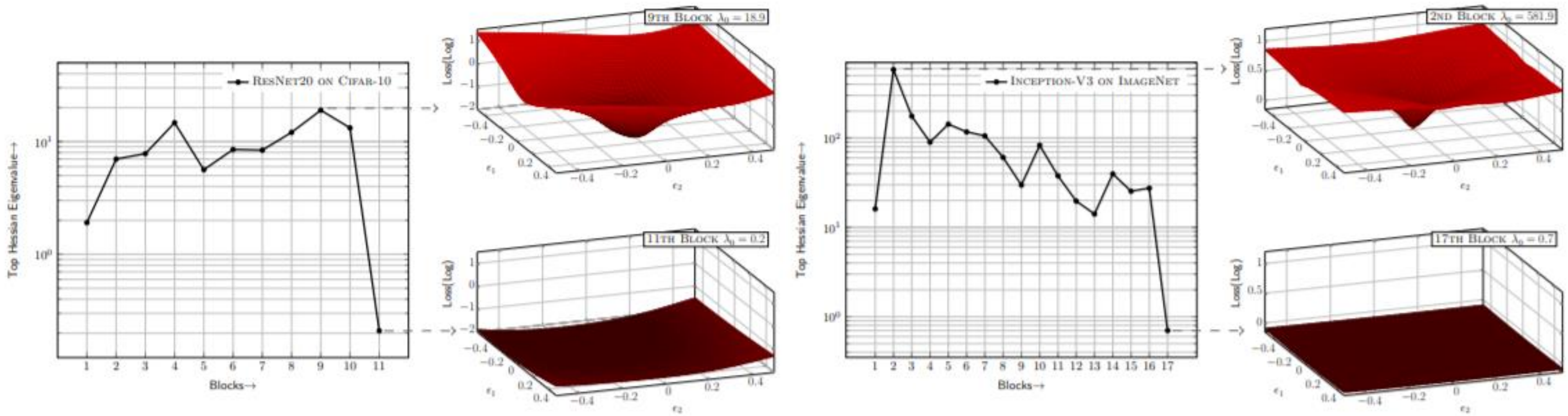
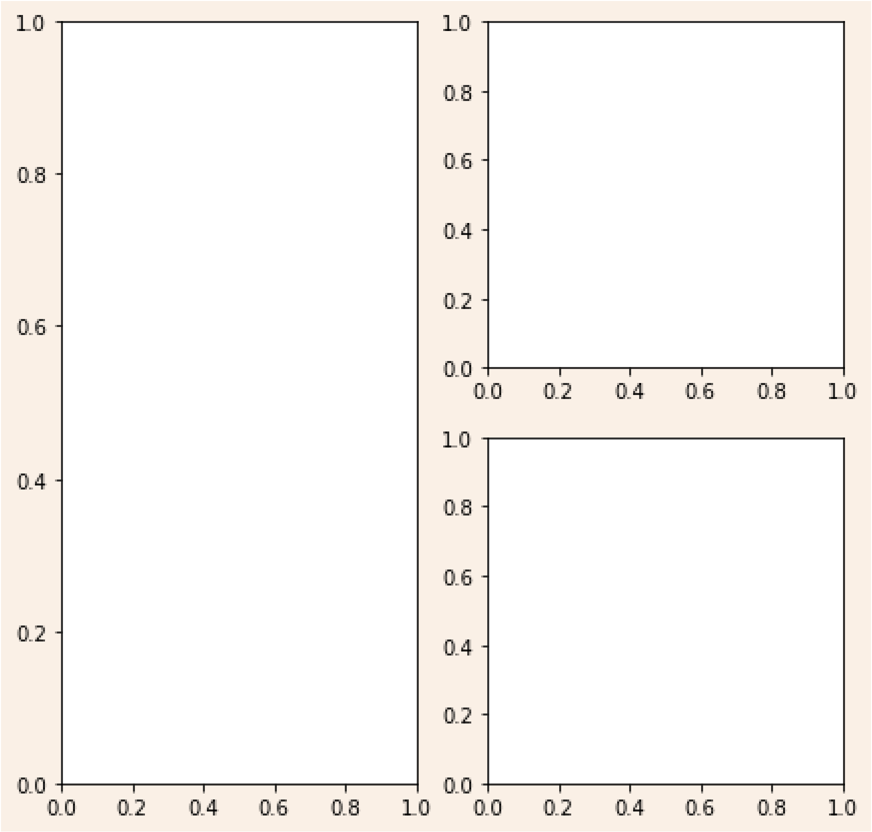


Fig. 1: *Top eigenvalue of each individual block of pre-trained ResNet20 on Cifar-10 (Left), and Inception-V3 on ImageNet (Right). Note that the magnitudes of eigenvalues of different blocks varies by orders of magnitude. See Figure 6 and 7 in appendix for the 3D loss landscape of other blocks.*

Figures & Axes

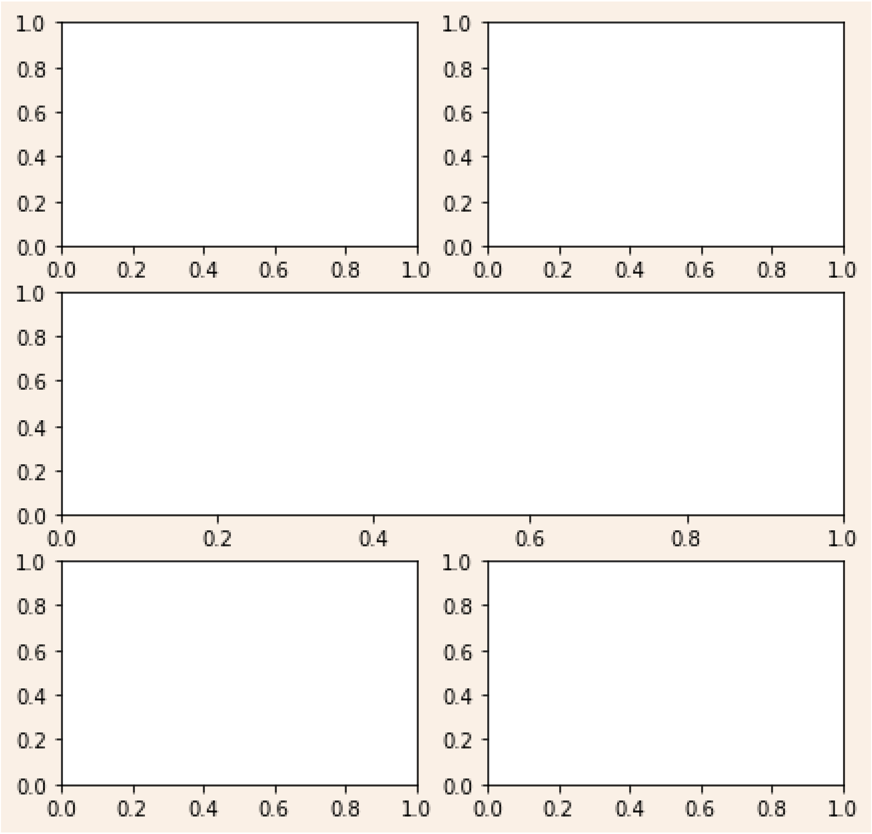
- fig.add_subplot (irregular arrangement)
  - axes grid [3, 3, i-th axes]

  - ax1 = fig.add_subplot(1, 2, 1)
  - ax2 = fig.add_subplot(2, 2, 2)
  - ax3 = fig.add_subplot(2, 2, 4)



Practice 3

- Drawing following figure and axes
  - fig.add_subplot (irregular arrangement)

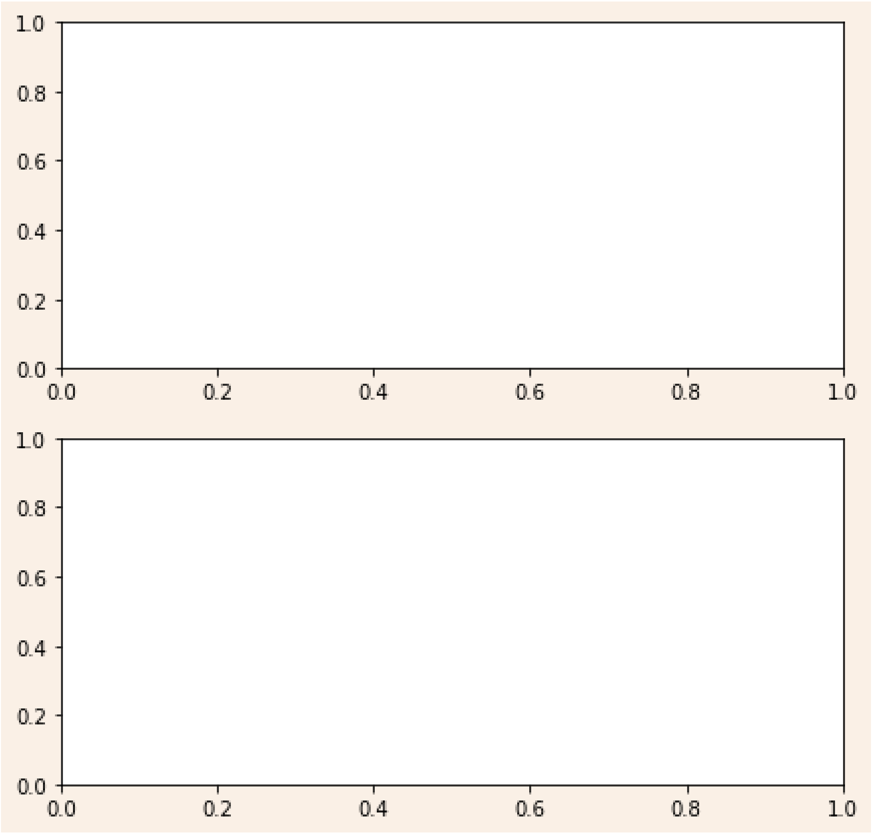


Figures & Axes

- plt.subplots (making fig and axes simultaneously)

```
fig=plt.figure(figsize=(7, 7),
               facecolor='linen')
ax1 = fig.add_subplot(2,1,1)
ax2 = fig.add_subplot(2,1,2)
print(fig)
print(ax1)
print(ax2)
```

```
Figure(504x504)
AxesSubplot(0.125,0.536818;0.775x0.343182)
AxesSubplot(0.125,0.125;0.775x0.343182)
```



Figures & Axes

- plt.subplots (making fig and axes simultaneously)

matplotlib.pyplot.subplots

`matplotlib.pyplot.subplots(nrows=1, ncols=1, *, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw)` [source]

Create a figure and a set of subplots.

This utility wrapper makes it convenient to create common layouts of subplots, including the enclosing figure object, in a single call.

- argument: (nrows=r, ncols=c, etc.)

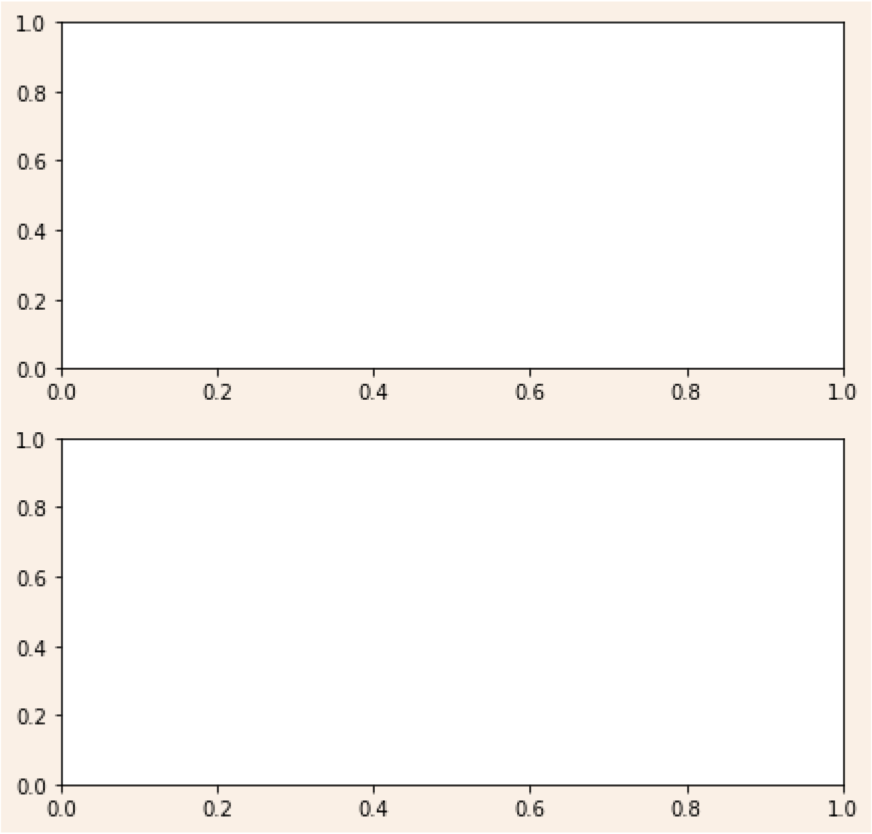
- fig, axes = plt.subplots(nrows=1, ncols=1)
- fig, axes = plt.subplots(1, 1)
- fig, axes = plt.subplots()

Figures & Axes

- plt.subplots (axes: ndarray type)

```python
fig, axes = plt.subplots(nrows=2,ncols=1,
                         figsize=(7, 7), facecolor='linen')
print(fig)
print(axes)
print(type(axes))
```

```
Figure(504x504)
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f6c041fdc88>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f6c042b7630>]
<class 'numpy.ndarray'>
```

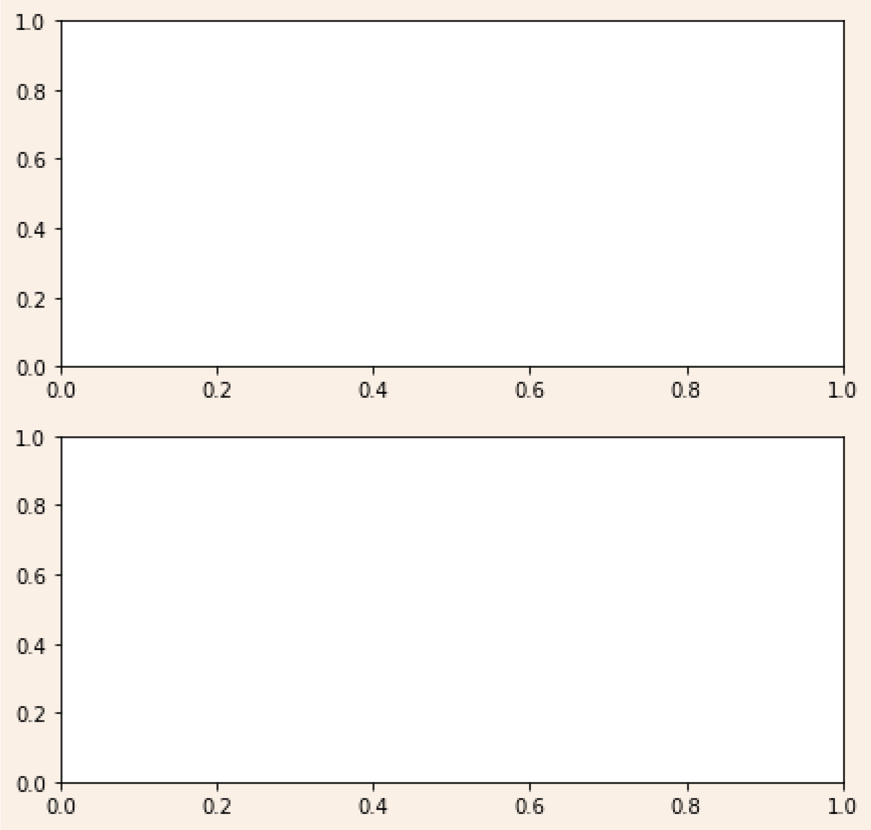


Figures & Axes

- plt.subplots (unpacking axes)

```
fig, (ax1, ax2) = plt.subplots(nrows=2,ncols=1,
                               figsize=(7, 7), facecolor='linen')
print(fig)
print(ax1)
print(ax2)
```

```
Figure(504x504)
AxesSubplot(0.125,0.536818;0.775x0.343182)
AxesSubplot(0.125,0.125;0.775x0.343182)
```

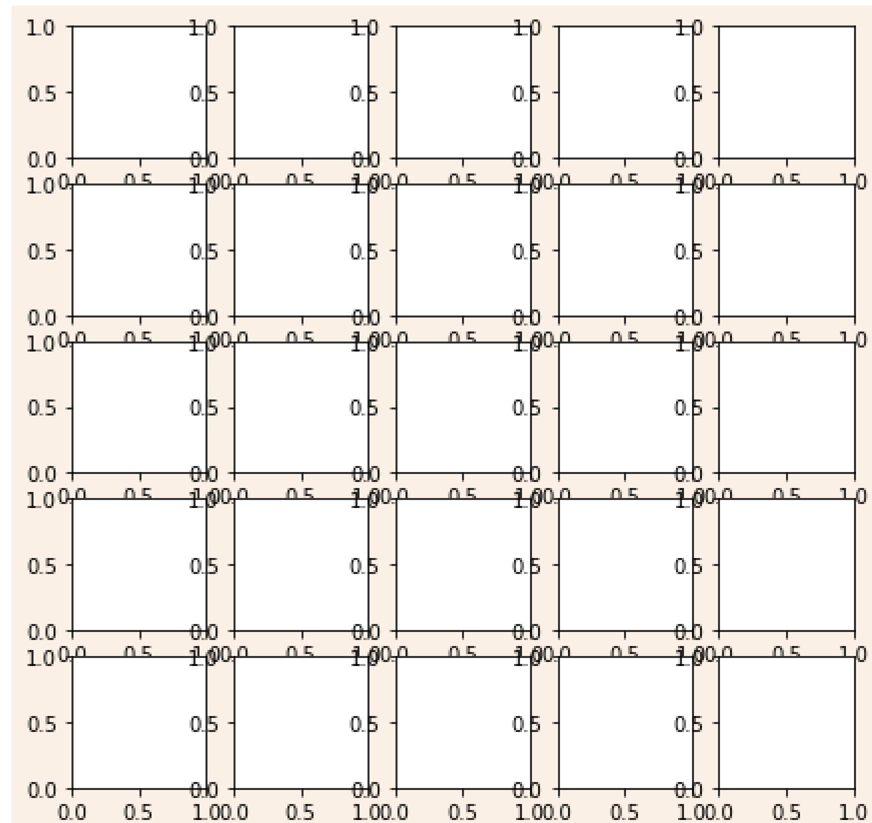


Figures & Axes

- plt.subplots (access via loop)

```python
fig, axes = plt.subplots(nrows=5,ncols=5,
                         figsize=(7, 7), facecolor='linen')
for ax in axes:
    print(ax)
```

```
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbfa7d30>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedc4fe978>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbcc04e0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbcf0860>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbca1be0>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbc55f60>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbc12320>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbbc46d8>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbb73a90>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbba8da0>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbb67160>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbb184e0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbac9860>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedba7cbe0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbaaff60>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedba6b320>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedba206a0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb9d1a20>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb983da0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb941160>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb8f44e0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb926860>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb8d7be0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb88cf60>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb84b320>]
```

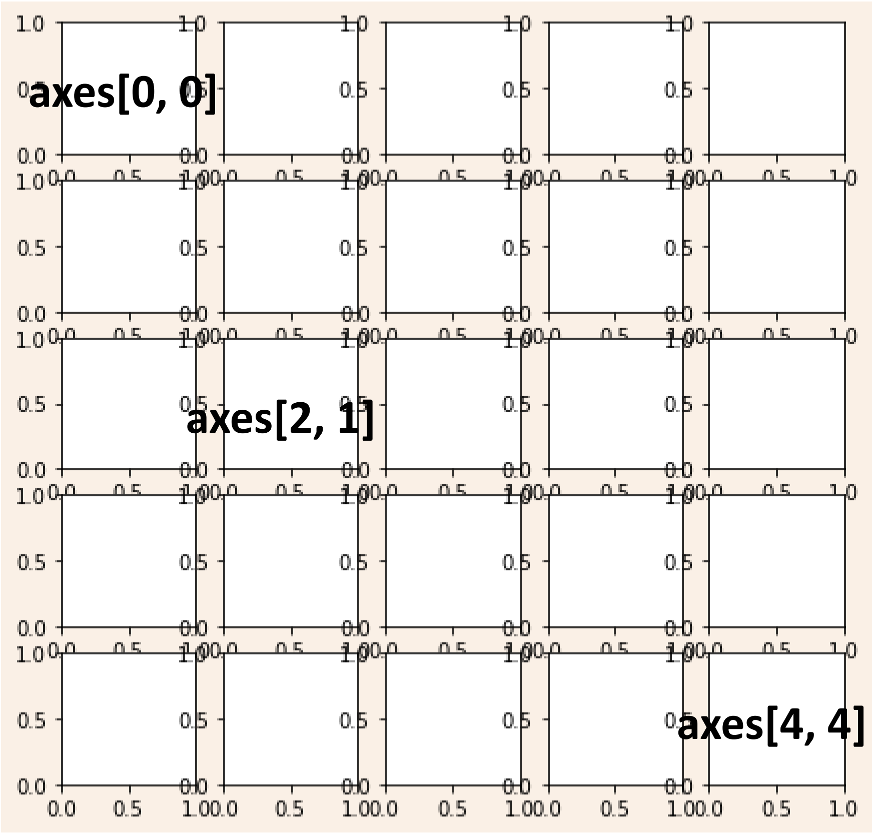


Figures & Axes

- plt.subplots (indexing 2D axes)

```python
fig, axes = plt.subplots(nrows=5,ncols=5,
                         figsize=(7, 7), facecolor='linen')
for ax in axes:
    print(ax)
```

```
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbfa7d30>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedc4fe978>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbcc04e0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbcf0860>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbca1be0>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbc55f60>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbc12320>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbbc46d8>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbb73a90>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbba8da0>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbb67160>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbb184e0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbac9860>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedba7cbe0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedbaaff60>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedba6b320>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedba206a0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb9d1a20>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb983da0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb941160>]
[<matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb8f44e0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb926860>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb8d7be0>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb88cf60>
 <matplotlib.axes._subplots.AxesSubplot object at 0x7fcedb84b320>]
```

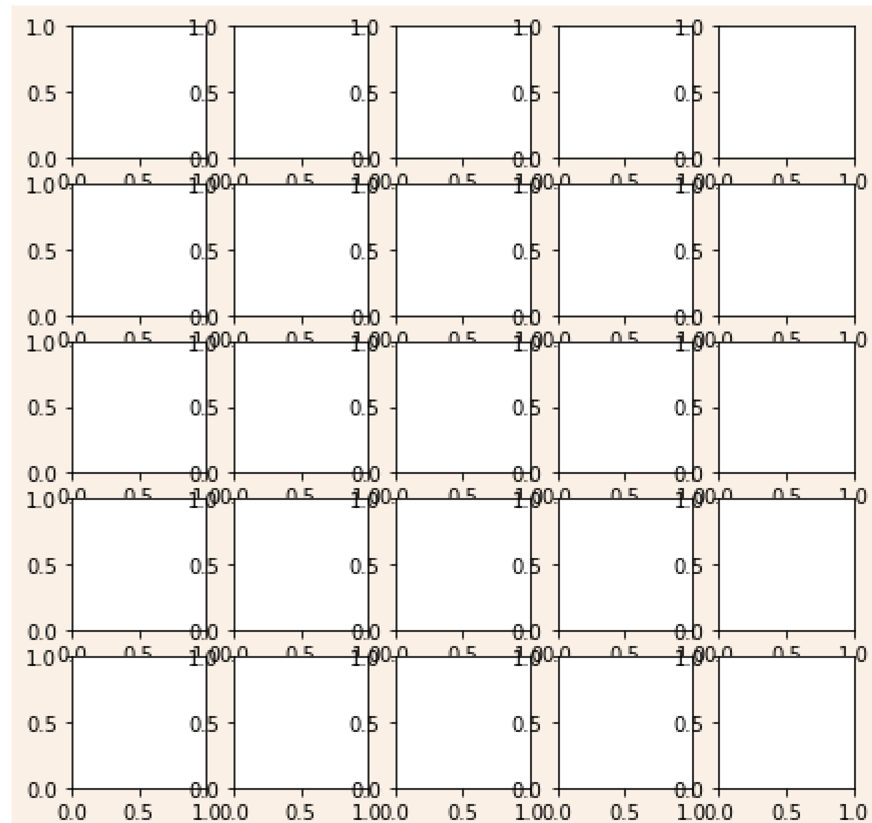


Figures & Axes

- plt.subplots (numpy flat & enumerate)

```
fig, axes = plt.subplots(nrows=5,ncols=5,
                         figsize=(7, 7), facecolor='linen')
for idx, ax in enumerate(axes.flat):
    print(idx, ax)
```

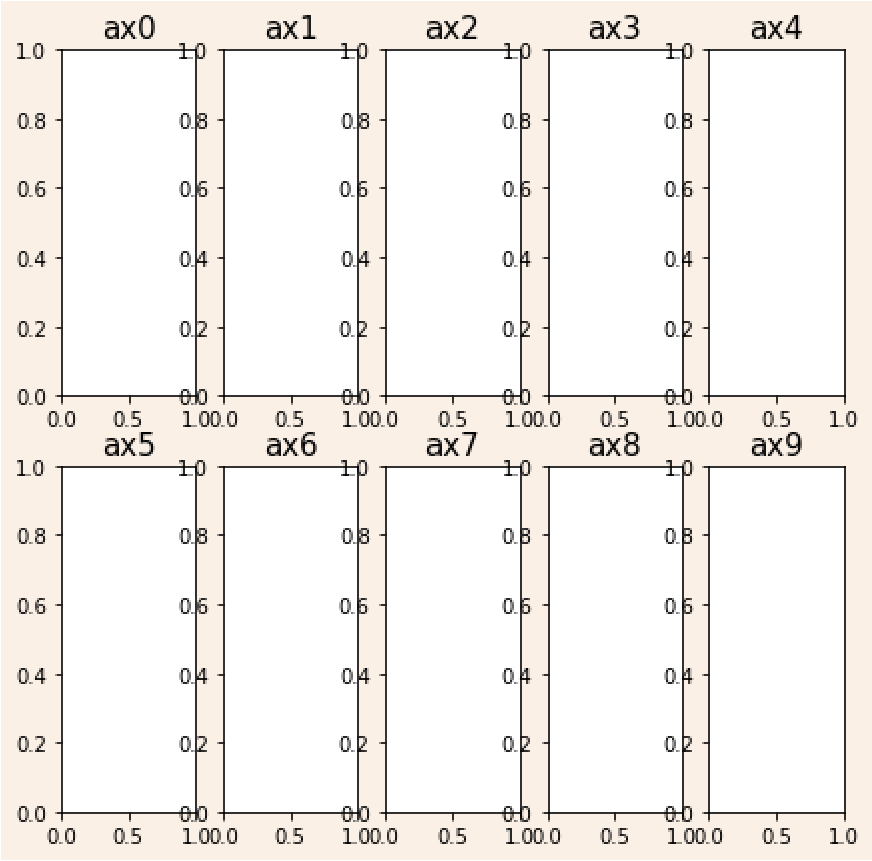
```
0 AxesSubplot(0.125,0.749828;0.133621x0.130172)
1 AxesSubplot(0.285345,0.749828;0.133621x0.130172)
2 AxesSubplot(0.44569,0.749828;0.133621x0.130172)
3 AxesSubplot(0.606034,0.749828;0.133621x0.130172)
4 AxesSubplot(0.766379,0.749828;0.133621x0.130172)
5 AxesSubplot(0.125,0.593621;0.133621x0.130172)
6 AxesSubplot(0.285345,0.593621;0.133621x0.130172)
7 AxesSubplot(0.44569,0.593621;0.133621x0.130172)
8 AxesSubplot(0.606034,0.593621;0.133621x0.130172)
9 AxesSubplot(0.766379,0.593621;0.133621x0.130172)
10 AxesSubplot(0.125,0.437414;0.133621x0.130172)
11 AxesSubplot(0.285345,0.437414;0.133621x0.130172)
12 AxesSubplot(0.44569,0.437414;0.133621x0.130172)
13 AxesSubplot(0.606034,0.437414;0.133621x0.130172)
14 AxesSubplot(0.766379,0.437414;0.133621x0.130172)
15 AxesSubplot(0.125,0.281207;0.133621x0.130172)
16 AxesSubplot(0.285345,0.281207;0.133621x0.130172)
17 AxesSubplot(0.44569,0.281207;0.133621x0.130172)
18 AxesSubplot(0.606034,0.281207;0.133621x0.130172)
19 AxesSubplot(0.766379,0.281207;0.133621x0.130172)
20 AxesSubplot(0.125,0.125;0.133621x0.130172)
21 AxesSubplot(0.285345,0.125;0.133621x0.130172)
22 AxesSubplot(0.44569,0.125;0.133621x0.130172)
23 AxesSubplot(0.606034,0.125;0.133621x0.130172)
24 AxesSubplot(0.766379,0.125;0.133621x0.130172)
```



Figures & Axes

- plt.subplots (axes indexing example)
  - ax.set_title()

```python
title_list = ['ax0', 'ax1', 'ax2', 'ax3', 'ax4',
              'ax5', 'ax6', 'ax7', 'ax8', 'ax9']

fig, axes = plt.subplots(nrows=2,ncols=5,
                         figsize=(7, 7), facecolor='linen')

for idx, ax in enumerate(axes.flat):
    ax.set_title(title_list[idx], fontsize=15)
```



Figures & Axes

- plt.subplot2grid (more complex arrangement)

matplotlib.pyplot.subplot2grid

matplotlib.pyplot.subplot2grid(*shape*, *loc*, *rowspan=1*, *colspan=1*, *fig=None*, ***kwargs*) [source]

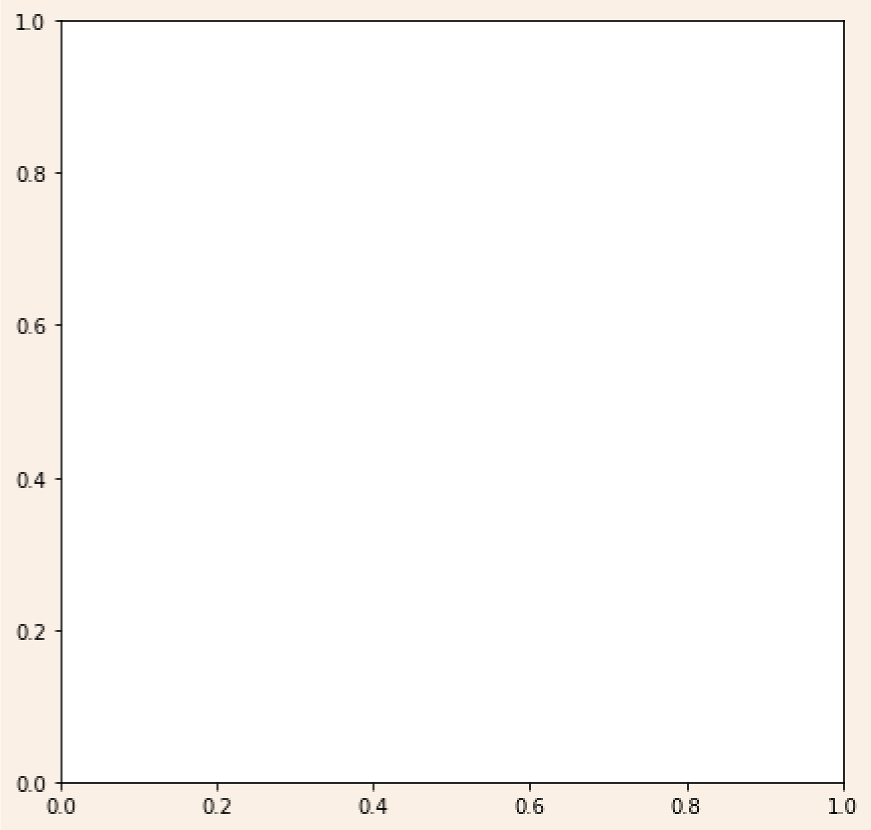
Create an axis at specific location inside a regular grid.

- fig = plt.figure()
- ax = plt.subplot2grid(shape=(,), loc=(,),
  rowspan= , colspan= ,
  fig=fig)

Figures & Axes

- plt.subplot2grid (simgle ax)

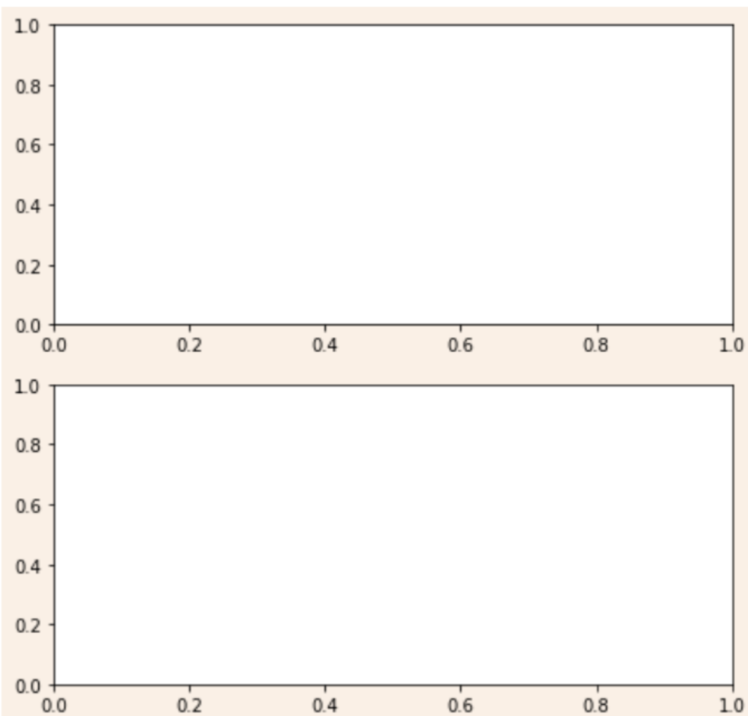
```
fig=plt.figure(figsize=(7, 7),
               facecolor='linen')

ax1 = plt.subplot2grid((1,1), (0,0), fig=fig)
```



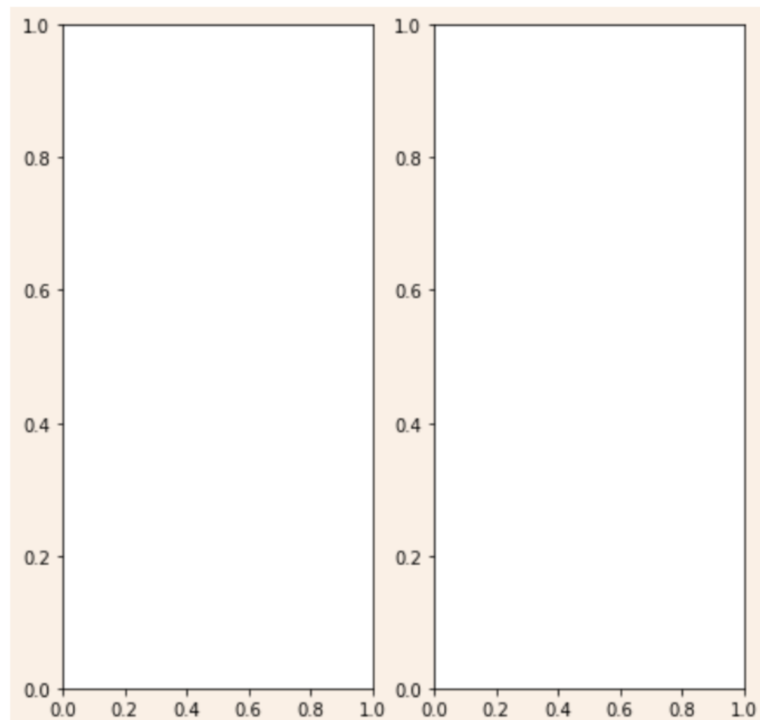
Figures & Axes

- plt.subplot2grid (axes arrangement)

```python
fig=plt.figure(figsize=(7, 7),
               facecolor='linen')

ax1 = plt.subplot2grid((2,1), (0,0), fig=fig)
ax2 = plt.subplot2grid((2,1), (1,0), fig=fig)
```



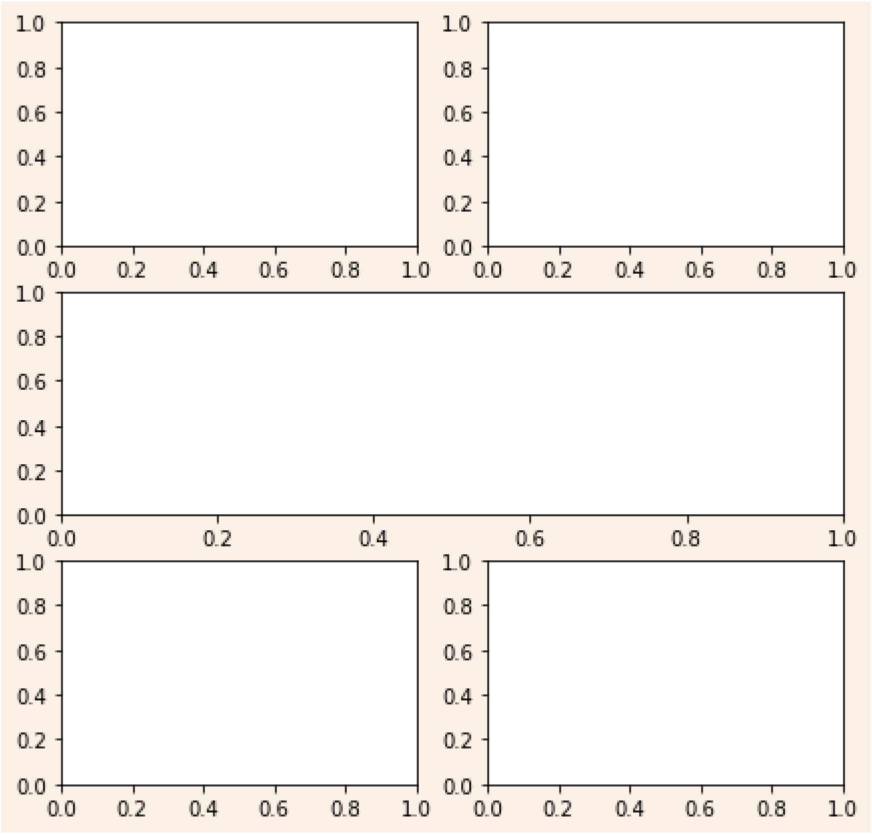

```python
fig=plt.figure(figsize=(7, 7),
               facecolor='linen')

ax1 = plt.subplot2grid((1,2), (0,0), fig=fig)
ax2 = plt.subplot2grid((1,2), (0,1), fig=fig)
```



Figures & Axes

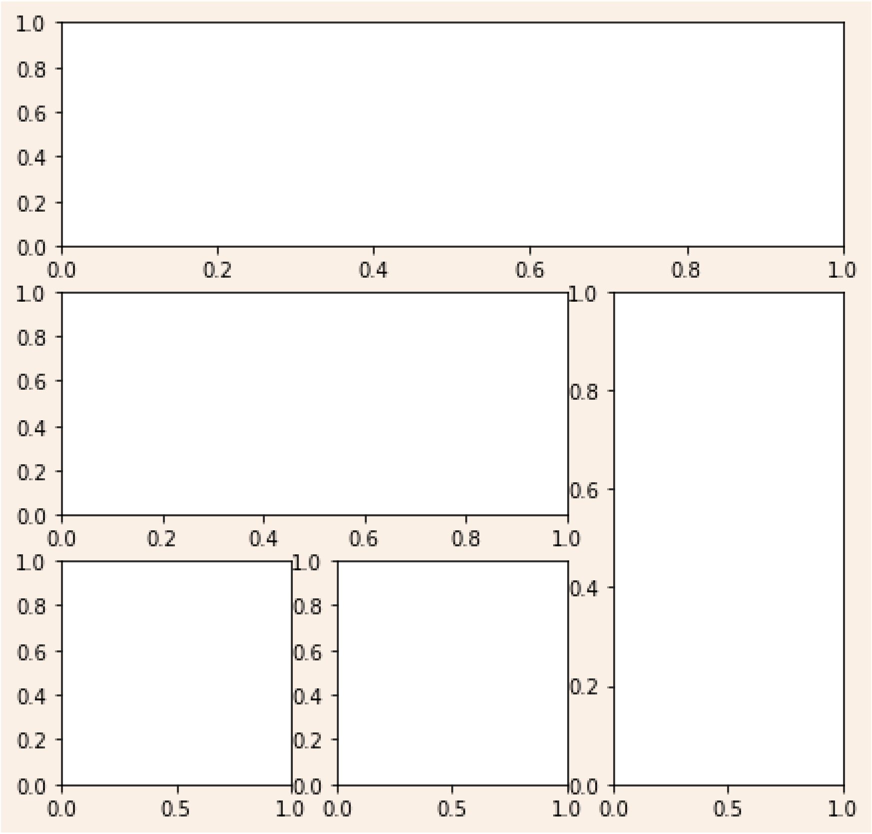
- plt.subplot2grid (more complex arrangement)

```
fig=plt.figure(figsize=(7, 7),
               facecolor='linen')

ax1 = fig.add_subplot(3, 2, 1)
ax2 = fig.add_subplot(3, 2, 2)
ax3 = fig.add_subplot(3, 1, 2)
ax4 = fig.add_subplot(3, 2, 5)
ax5 = fig.add_subplot(3, 2, 6)


ax1 = plt.subplot2grid((3,2), (0,0), colspan=1, fig=fig)
ax2 = plt.subplot2grid((3,2), (0,1), colspan=1, fig=fig)
ax3 = plt.subplot2grid((3,2), (1,0), colspan=2, fig=fig)
ax5 = plt.subplot2grid((3,2), (2,0), colspan=1, fig=fig)
ax6 = plt.subplot2grid((3,2), (2,1), colspan=1, fig=fig)
```



Practice 4

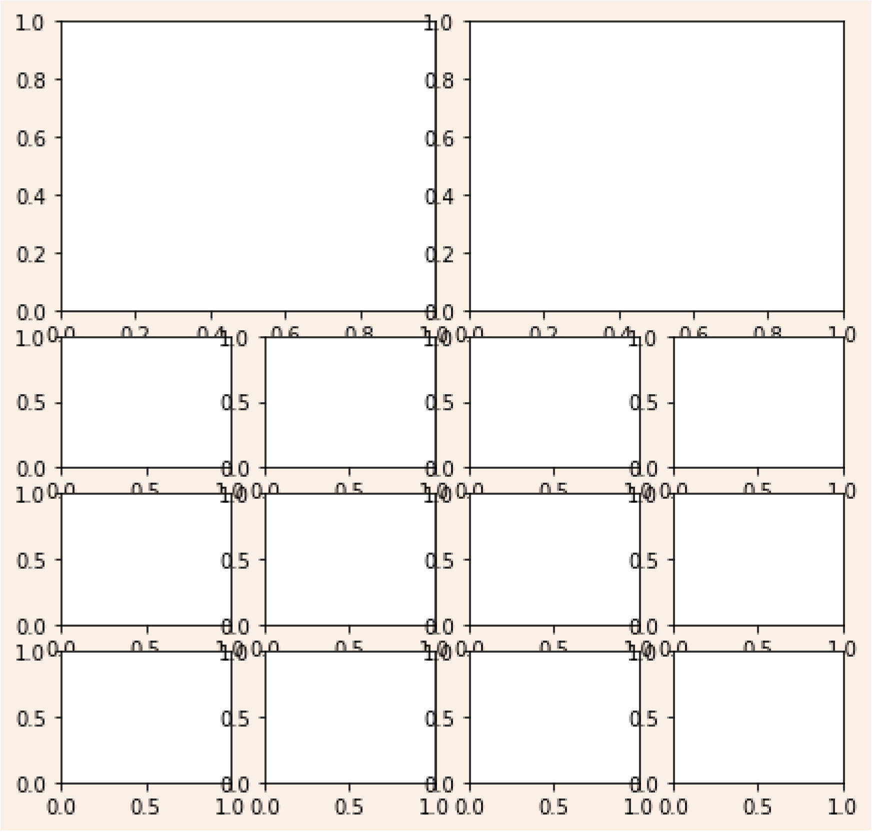
- Drawing following figure and axes
  - plt.subplot2grid (more complex arrangement)



Practice 5

- Drawing following figure and axes
  - Using for loop for deleting redundant code

```python
fig = plt.figure(figsize=(7, 7), facecolor='linen')
ax1 = plt.subplot2grid((5, 4), (0, 0),
                       rowspan=2, colspan=2)
ax2 = plt.subplot2grid((5, 4), (0, 2),
                       rowspan=2, colspan=2)

ax3 = plt.subplot2grid((5,4), (2,0))
ax4 = plt.subplot2grid((5,4), (2,1))
ax5 = plt.subplot2grid((5,4), (2,2))
ax6 = plt.subplot2grid((5,4), (2,3))

ax7 = plt.subplot2grid((5,4), (3,0))
ax8 = plt.subplot2grid((5,4), (3,1))
ax9 = plt.subplot2grid((5,4), (3,2))
ax10 = plt.subplot2grid((5,4), (3,3))

ax11 = plt.subplot2grid((5,4), (4,0))
ax12 = plt.subplot2grid((5,4), (4,1))
ax13 = plt.subplot2grid((5,4), (4,2))
ax14 = plt.subplot2grid((5,4), (4,3))
```



Figures & Axes

- fig.add_axes (arbitrary locations and sizes of axes)

add_axes(*self*, **args*, ***kwargs*) [source]

Add an axes to the figure.

Call signatures:

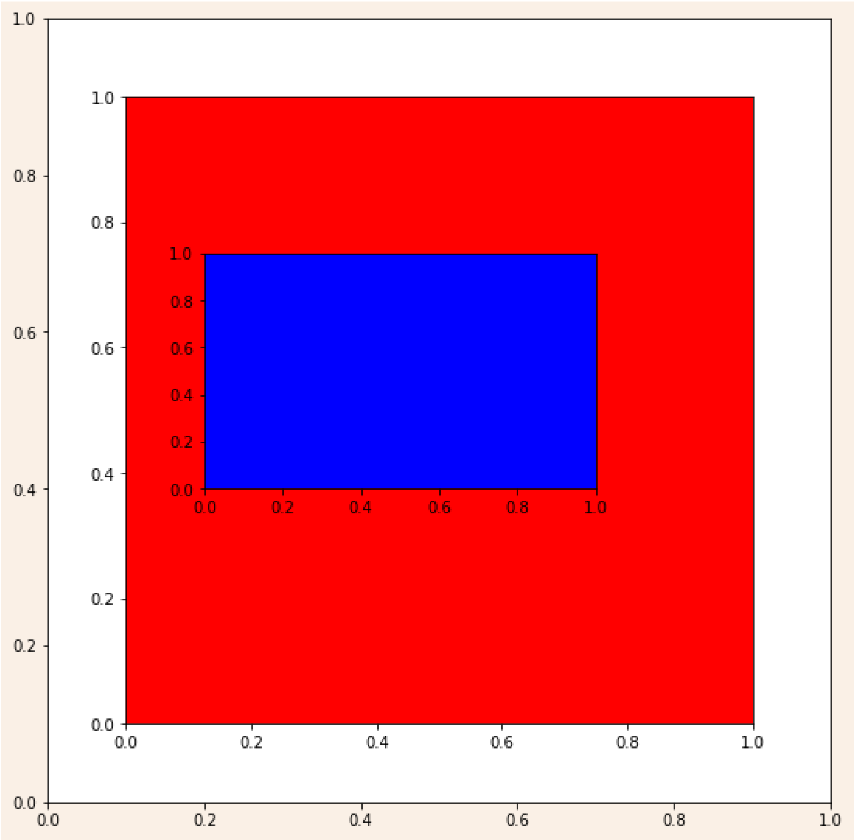
```
add_axes(rect, projection=None, polar=False, **kwargs)
add_axes(ax)
```

  - fig = plt.figure()
  - ax = fig.add_axes([left, bottom, width, height])

Figures & Axes

- fig.add_axes (arbitrary locations and sizes of axes)

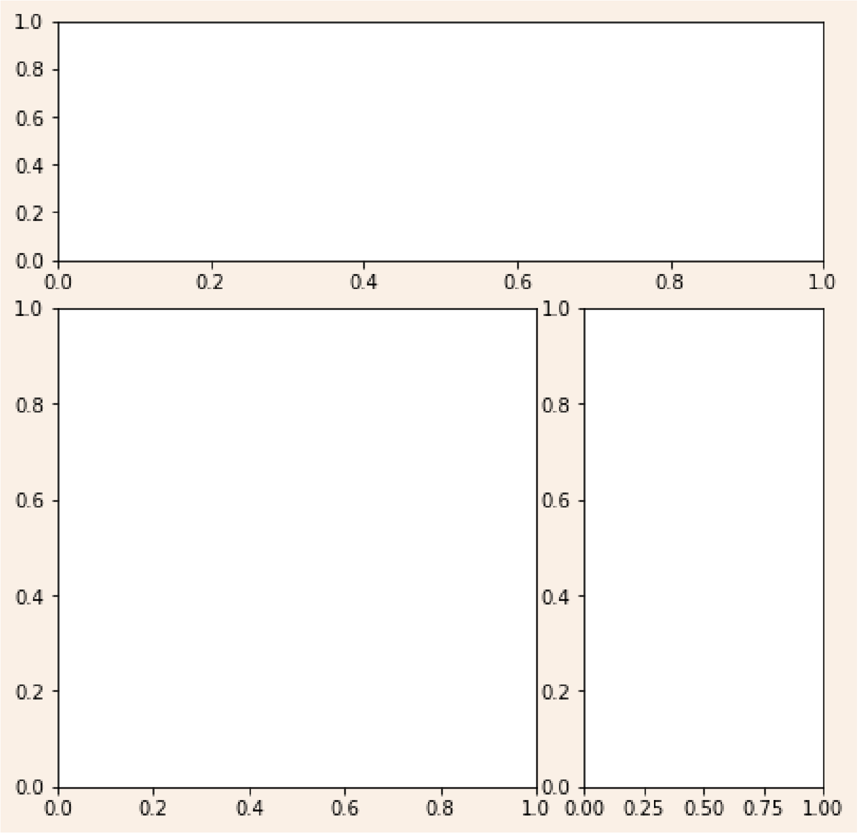
```
fig = plt.figure(figsize=(7, 7),
                 facecolor='linen')

rect1 = [0, 0, 1, 1]
rect2 = [0.1, 0.1, 0.8, 0.8]
rect3 = [0.2, 0.4, 0.5, 0.3]

ax1 = fig.add_axes(rect1)
ax2 = fig.add_axes(rect2, facecolor='r')
ax3 = fig.add_axes(rect3, facecolor='b')
```



Figures & Axes

- fig.add_axes (arbitrary locations and sizes of axes)

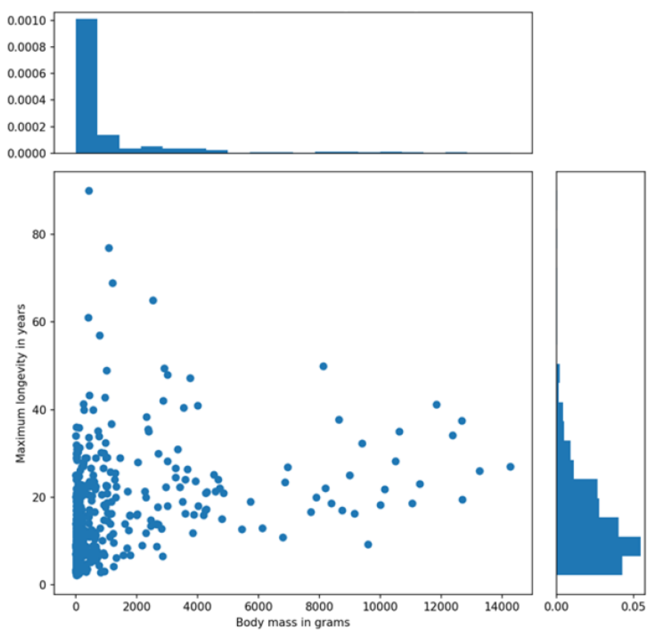
```
left, bottom = 0.1, 0.1
width1, height1 = 0.5, 0.5
spacing = 0.05

width2 = 1 - (2*left + width1 + spacing)
height2 = 1 - (2*bottom + height1 + spacing)

rect1 = [left, bottom, width1, height1]
rect2 = [left, bottom+height1+spacing, 1-2*left, height2]
rect3 = [left+width1+spacing, bottom, width2, height1]

fig = plt.figure(figsize=(7, 7),
                 facecolor='linen')

ax1 = fig.add_axes(rect1)
ax2 = fig.add_axes(rect2)
ax3 = fig.add_axes(rect3)
```

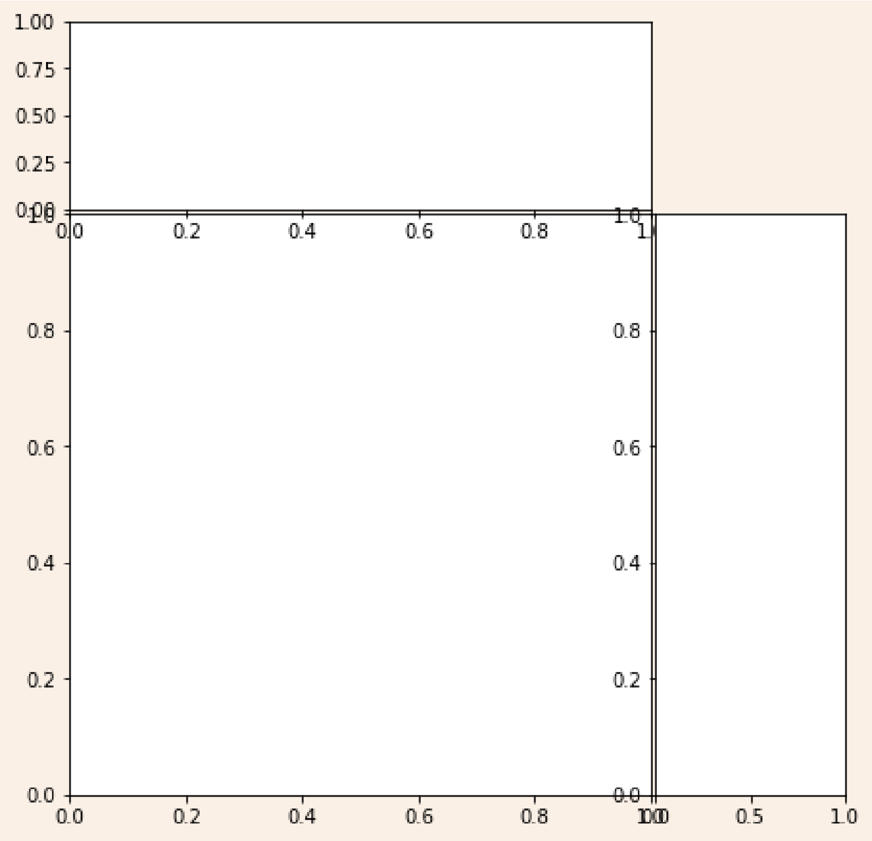


Practice 6

- Drawing following figure and axes
  - fig.add_axes (arbitrary locations and sizes of axes)

```
left, bottom = 0.1, 0.1
width1, height1 = 0.6, 0.6
spacing = 0.005
```

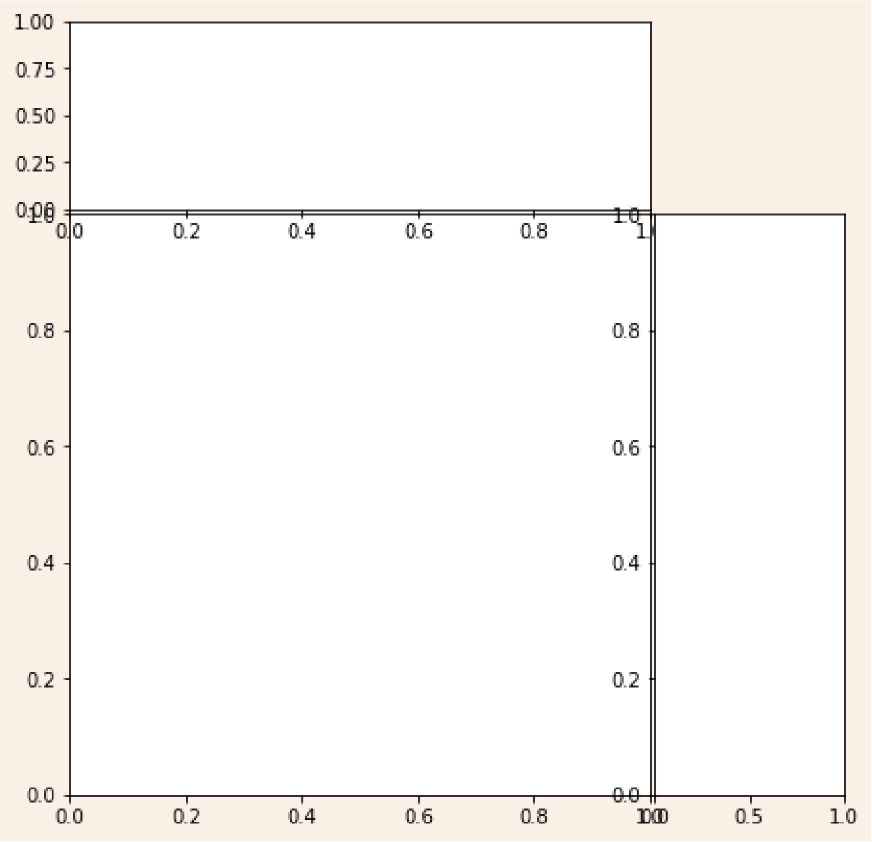




Practice 6

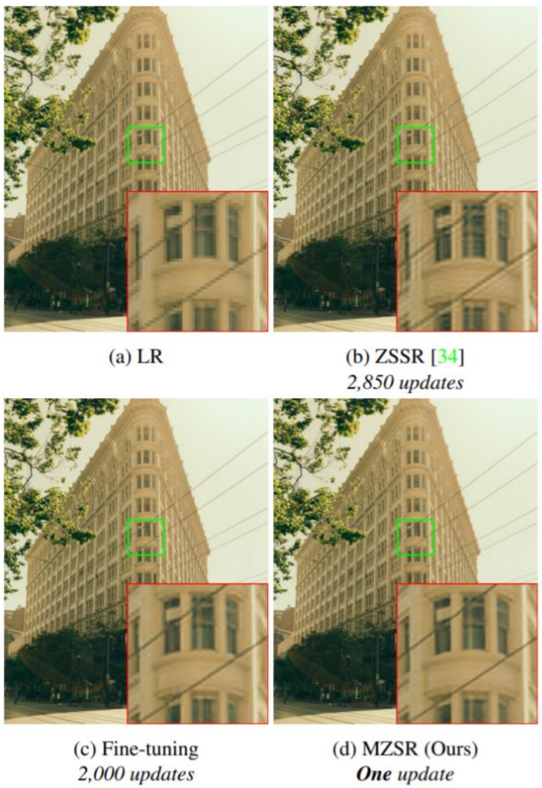
- Drawing following figure and axes
  - fig.add_axes (arbitrary locations and sizes of axes)

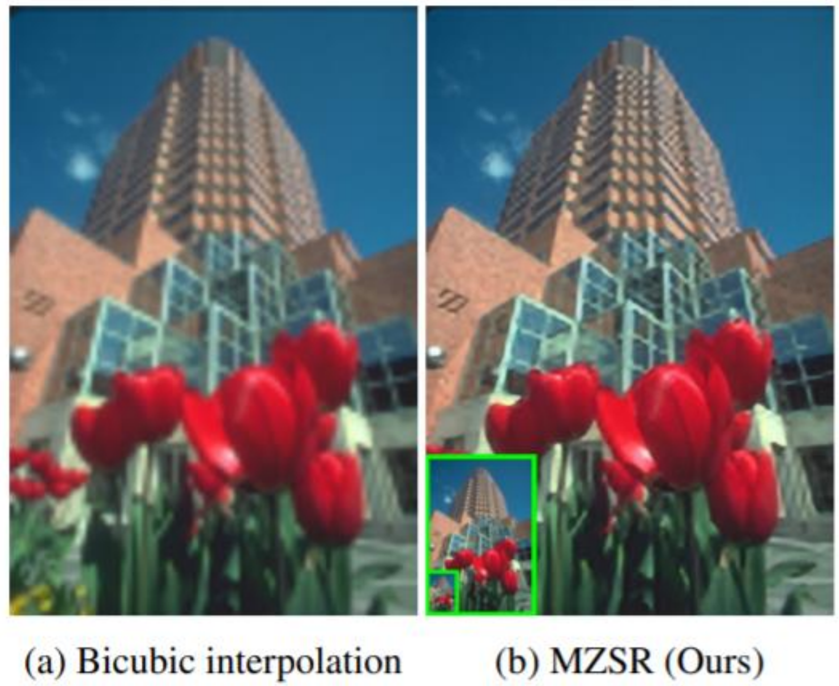
```
left, bottom = 0.1, 0.1
width1, height1 = 0.6, 0.6
spacing = 0.005
```



Figures & Axes

- fig.add_axes (zoom axes)



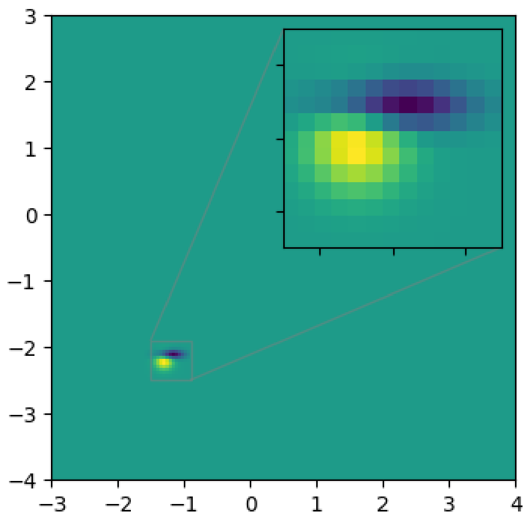


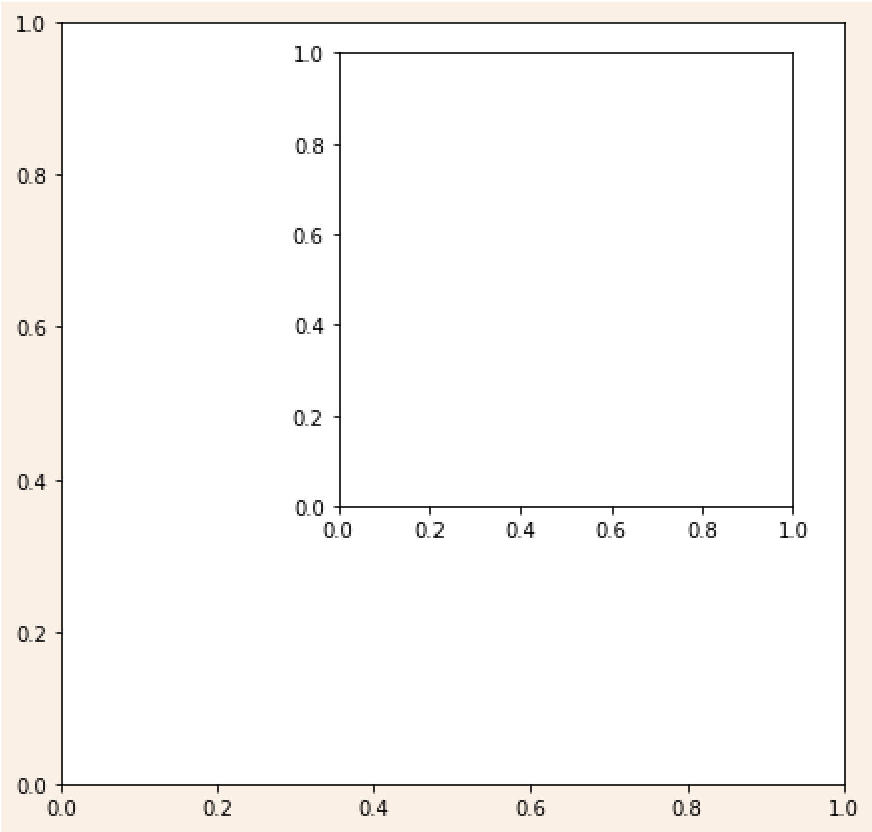
Soh, Jae Woong, Sunwoo Cho, and Nam Ik Cho. "Meta-transfer learning for zero-shot super-resolution." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

Figures & Axes

- fig.add_axes (zoom axes)

```
fig = plt.figure(figsize=(7, 7),
                 facecolor='linen')

ax = fig.add_subplot(1, 1, 1)
rect = [0.4, 0.4, 0.45, 0.45]
ax_zoom = fig.add_axes(rect)
```





Figures & Axes

- making figures and axes

  1) subplots
  - fig, axes = plt.subplots()

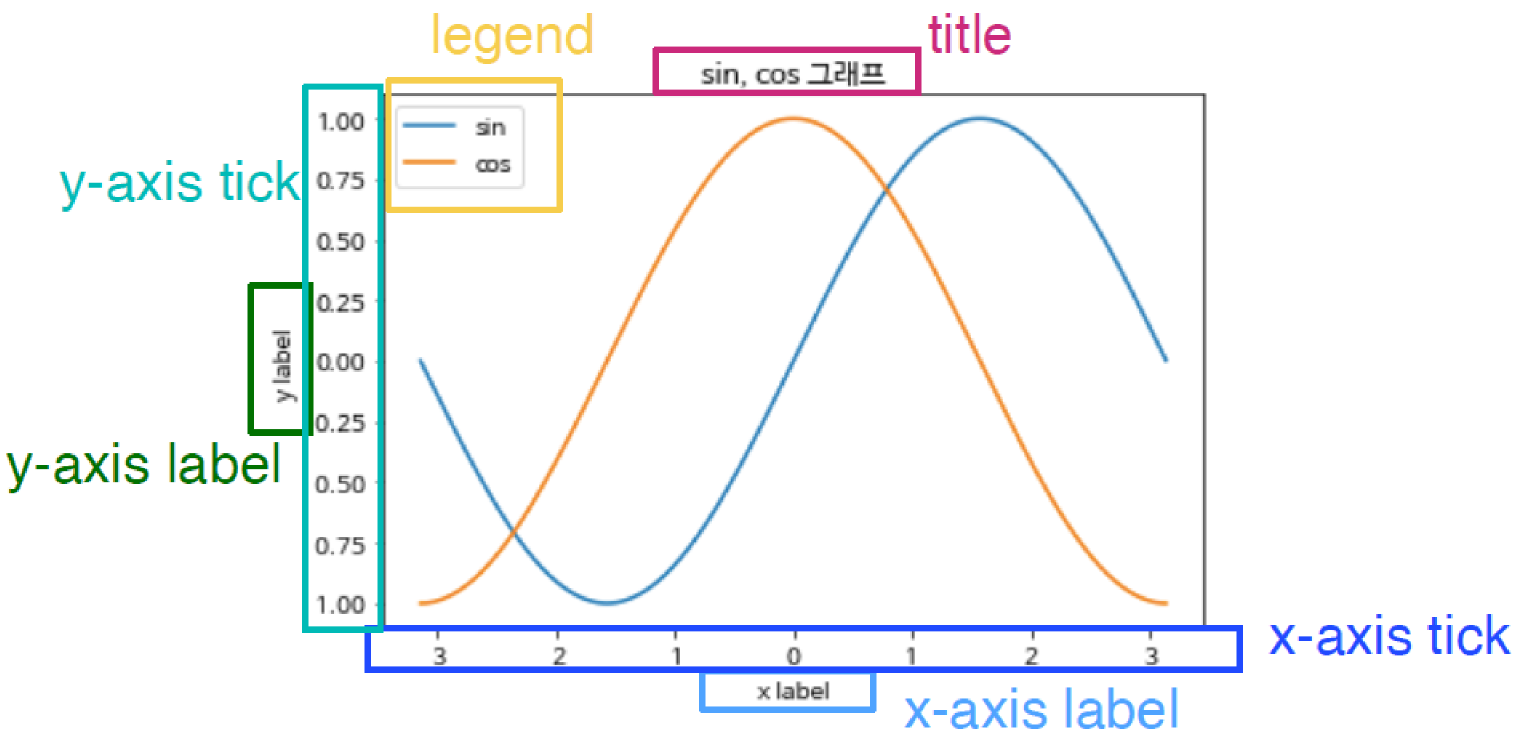
  2) add_subplot
  - fig= plt.figure()
  - ax = fig.add_subplot()

  3) subplot2grid
  - fig = plt.figure()
  - ax = plt.subplot2grid(fig=fig)

  4) add_axes
  - fig = plt.figure()
  - ax = fig.add_axes()

Axes Customizing

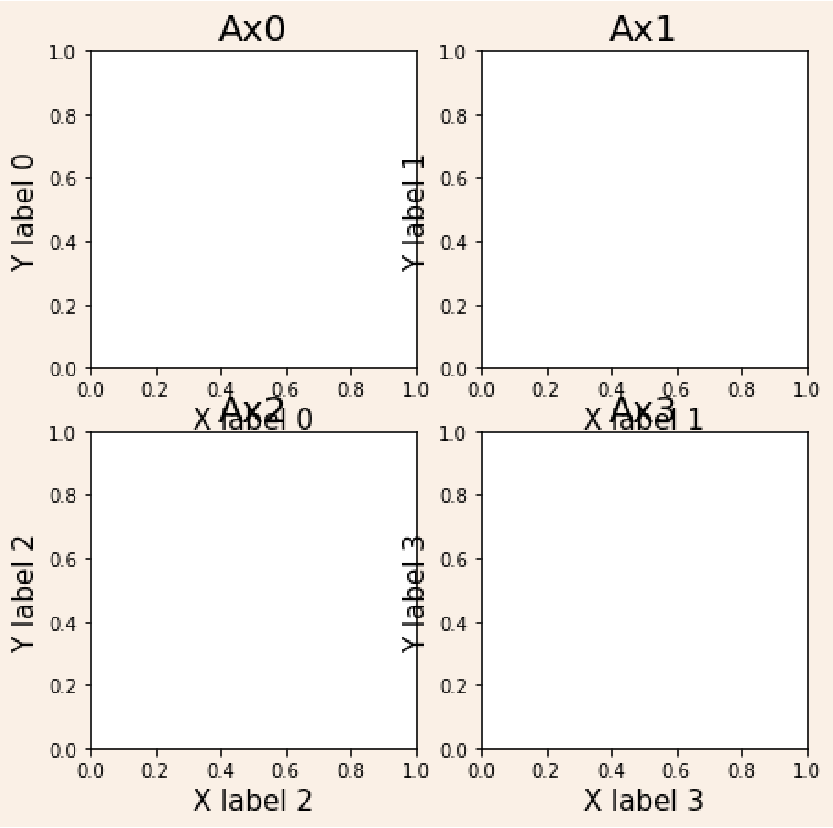
- Axis



Axes Customizing

- fig.tight_layout()

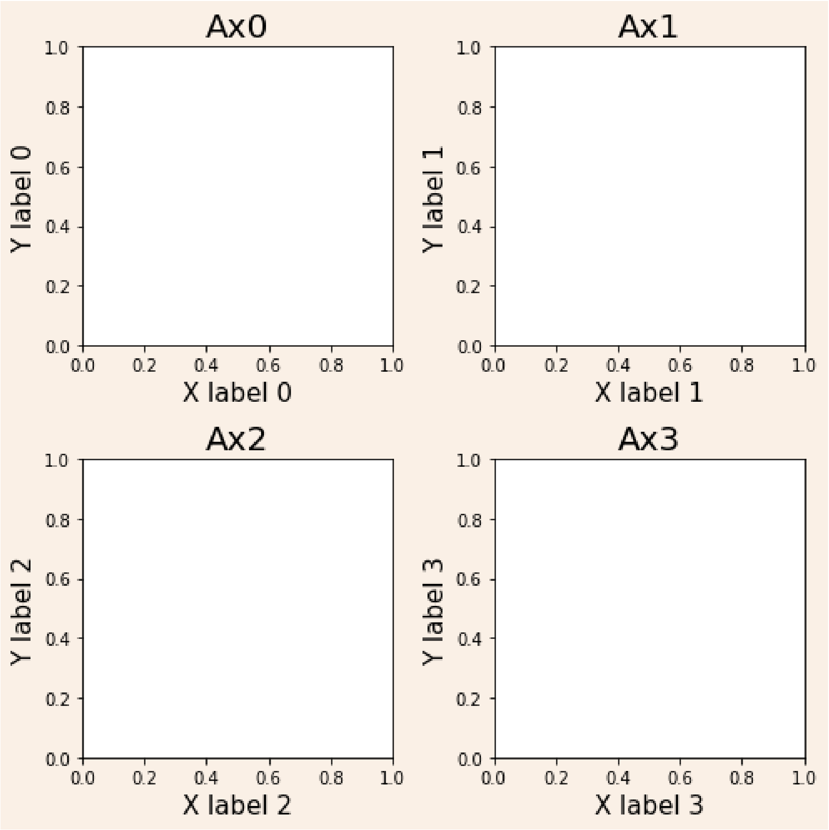
```
title_list = ['Ax' + str(i) for i in range(4)]
xlabel_list = ['X label ' + str(i) for i in range(4)]
ylabel_list = ['Y label ' + str(i) for i in range(4)]

fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.set_title(title_list[ax_idx], fontsize=20)
    ax.set_xlabel(xlabel_list[ax_idx], fontsize=15)
    ax.set_ylabel(ylabel_list[ax_idx], fontsize=15)
```



Axes Customizing

- fig.tight_layout(pad=1)

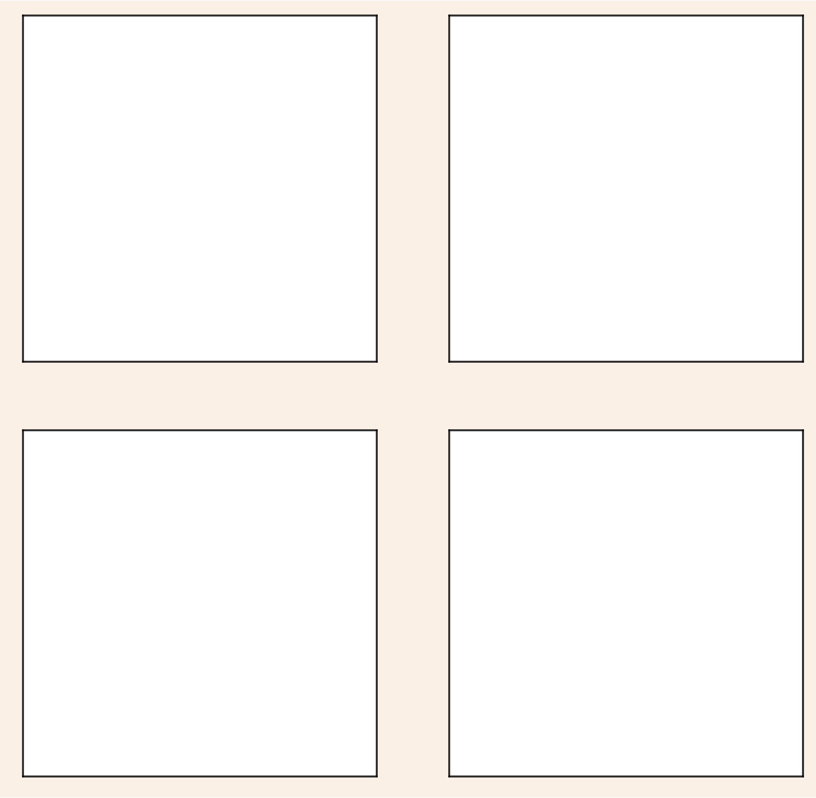
```python
title_list = ['Ax' + str(i) for i in range(4)]
xlabel_list = ['X label ' + str(i) for i in range(4)]
ylabel_list = ['Y label ' + str(i) for i in range(4)]

fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.set_title(title_list[ax_idx], fontsize=20)
    ax.set_xlabel(xlabel_list[ax_idx], fontsize=15)
    ax.set_ylabel(ylabel_list[ax_idx], fontsize=15)

fig.tight_layout()
```



Axes Customizing

- fig.subplots_adjust (more customized layout)
  - axis.set_visible()

```
fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```



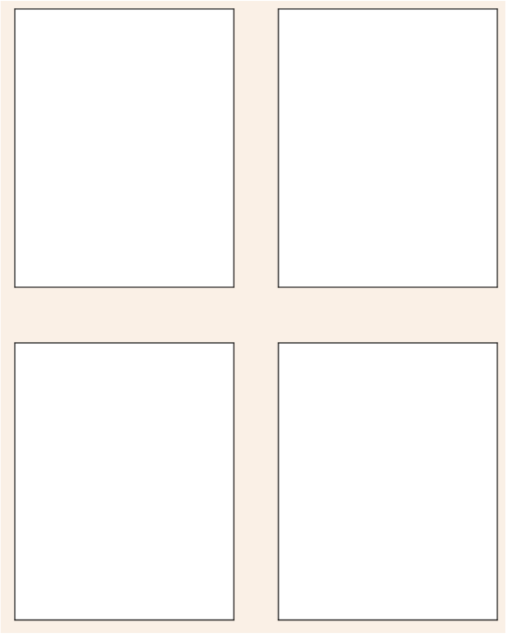
Axes Customizing

- fig.subplots_adjust (more customized layout)

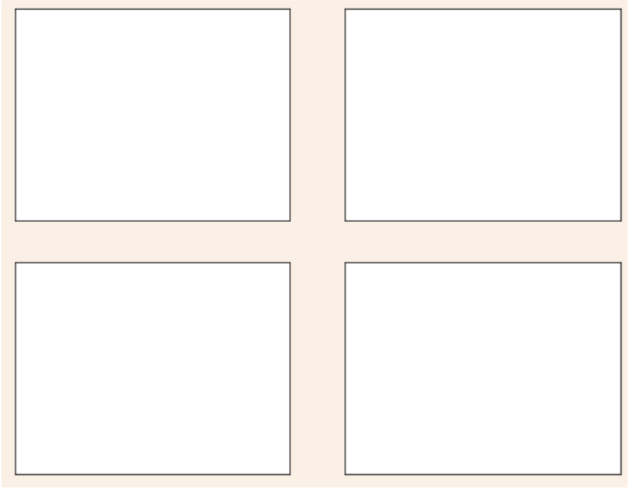
Axes Customizing

- fig.subplots_adjust (more customized layout)

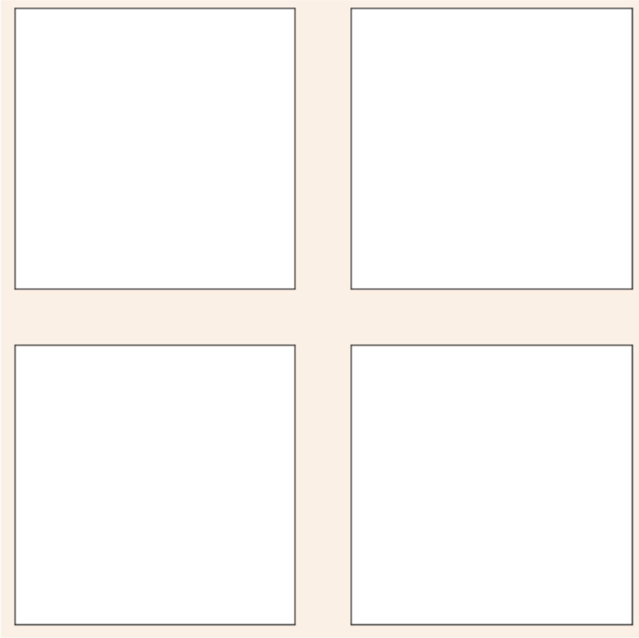
```
fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

fig.subplots_adjust(bottom=0.01, top=0.99)
```



```
fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

fig.subplots_adjust(left=0.01, right=0.99)
```



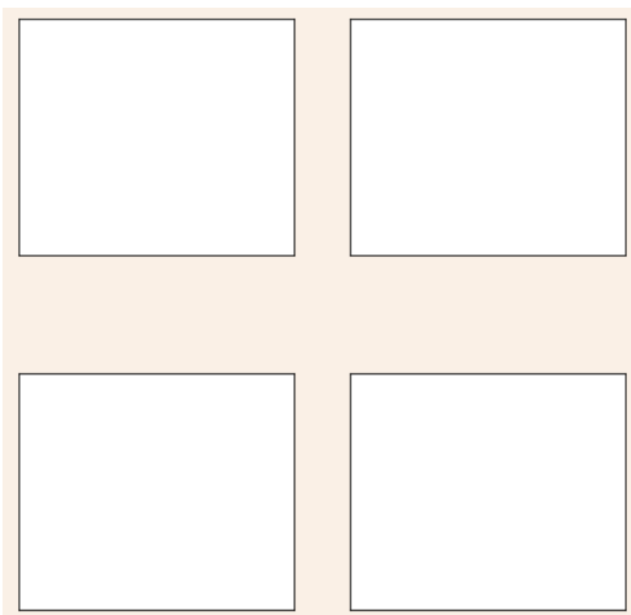
```
fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

fig.subplots_adjust(bottom=0.01, top=0.99, left=0.01, right=0.99)
```



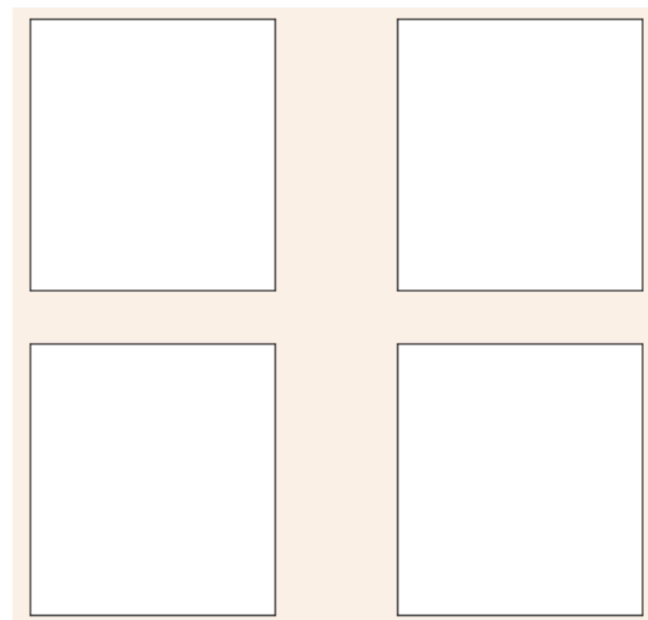
Axes Customizing

- fig.subplots_adjust (more customized layout)

```
fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

fig.subplots_adjust(hspace=0.5)
```



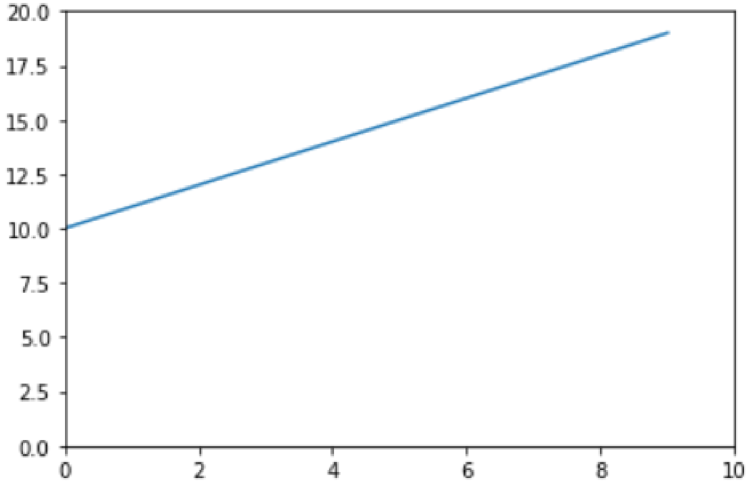
```
fig, axes = plt.subplots(2, 2, figsize=(7, 7),
                         facecolor='linen')

for ax_idx, ax in enumerate(axes.flat):
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

fig.subplots_adjust(wspace=0.5)
```



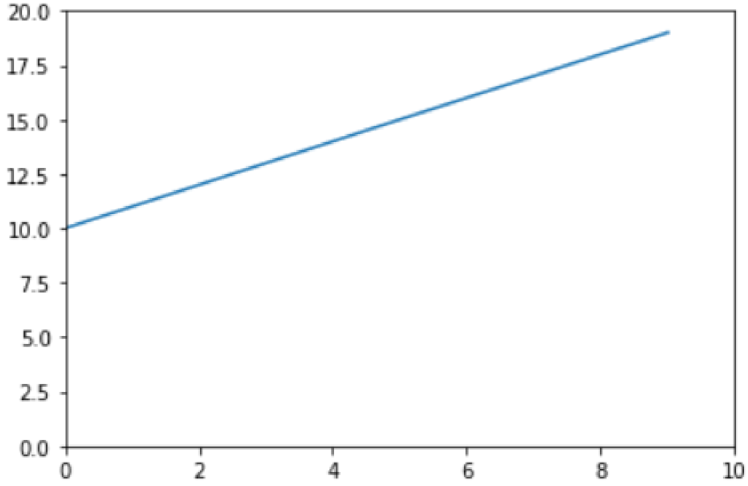
Axes Customizing

- 축지정
  - plt.axis([xmin, xmax, ymin, ymax])
  - plt.xlim, plt.ylim

```
plt.axis([0, 10, 0, 20])

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



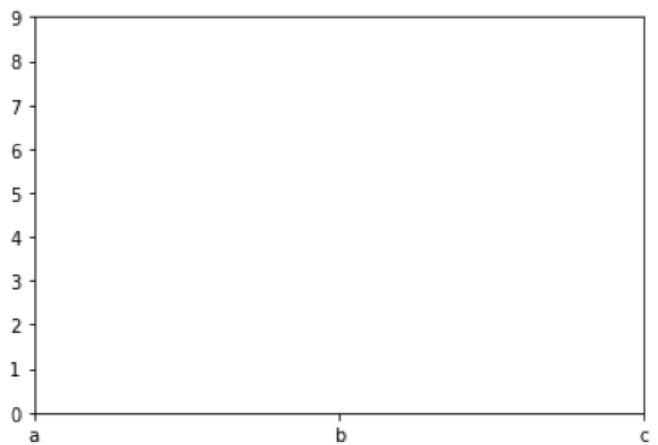

```
plt.xlim([0, 10])
plt.ylim([0, 20])

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



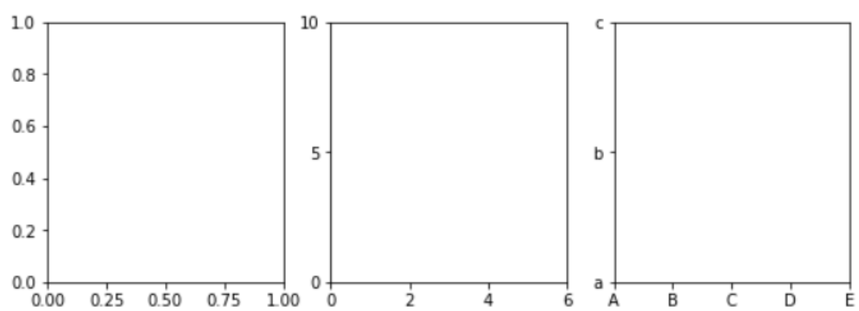
Axes Customizing

- Ticks
  - xticks(), yticks()
  - set_xtics / set_xticlabels
  - set_ytics / set_yticlabels

```
plt.xticks(np.arange(3), ['a', 'b', 'c'])
plt.yticks(np.arange(10))

plt.show()
```



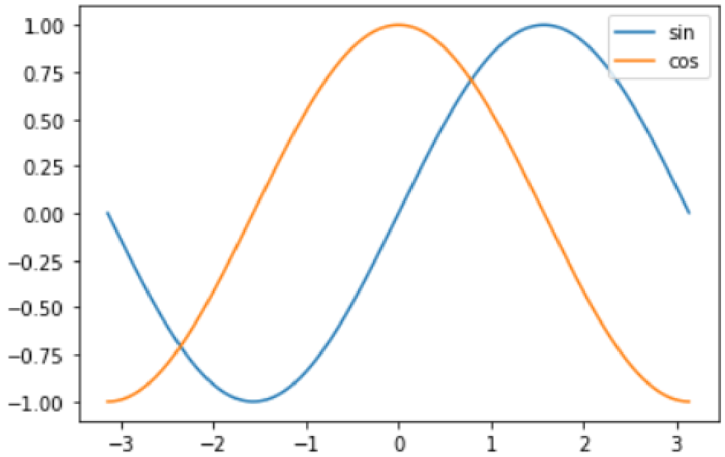

```
fig, axs = plt.subplots(1, 3, figsize=(9, 3))

axs[1].set_xticks([0,2,4,6])
axs[1].set_yticks([0,5,10])

axs[2].set_xticklabels(['A', 'B', 'C', 'D', 'E'])
axs[2].set_yticks([0,1,2])
axs[2].set_yticklabels(['a', 'b', 'c'])

plt.show()
```



Axes Customizing

- Legend
  - plt.legend()

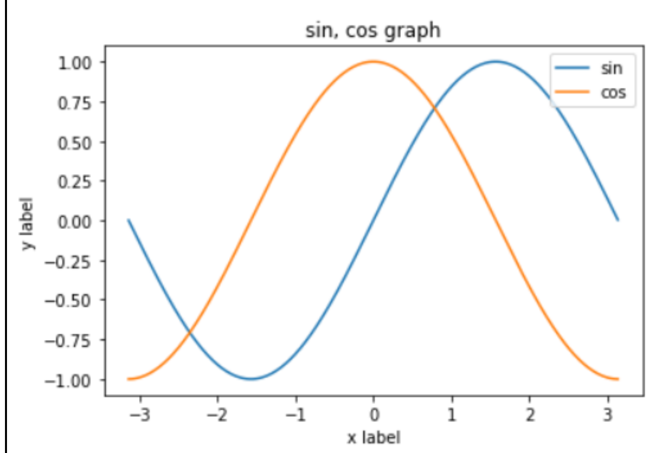
```
fig, ax = plt.subplots()

ax.plot(x, y1, label = 'sin')
ax.plot(x, y2, label = 'cos')

ax.legend(loc=1)

plt.show()
```



Axes Customizing

- Text
  - plt.title()
  - plt.xlabel()
  - plt.ylabel()

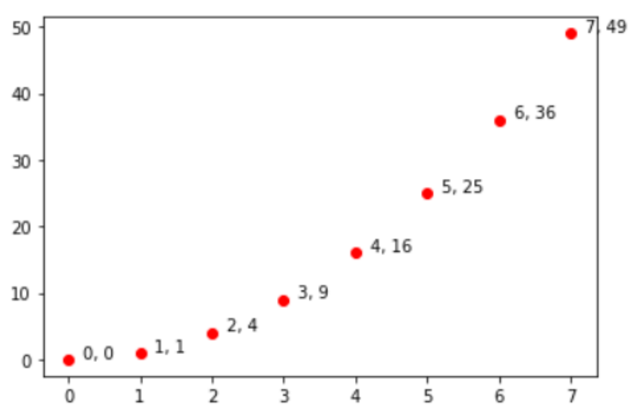
```python
fig, ax = plt.subplots()

ax.plot(x, y1, label = 'sin')
ax.plot(x, y2, label = 'cos')

ax.legend(loc=1)

plt.title('sin, cos graph') # title

plt.xlabel('x label') # x label
plt.ylabel('y label') # y label

plt.show()
```



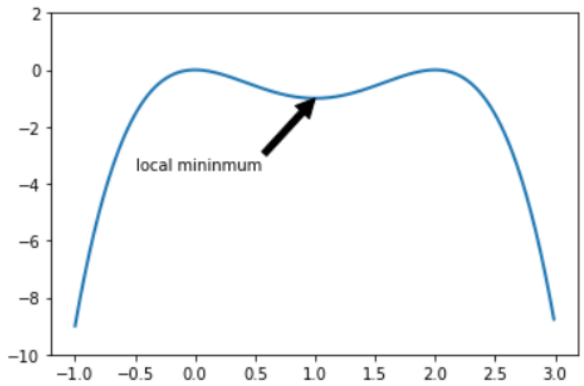
Axes Customizing

- Text
  - text()
  - annotate()

```python
x = np.arange(8)
y = x**2

fig, ax = plt.subplots()

ax.plot(x, y, 'ro')

for x_, y_ in zip(x, y):
    ax.text(x_+0.2, y_+0.3, '%d, %d' % (int(x_), int(y_)))
```



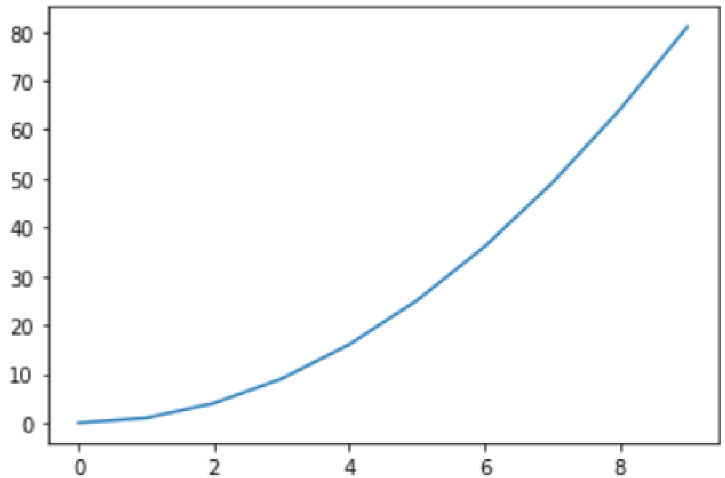

```python
x = np.arange(-1, 3, 0.01)
y = -x**4+4*x**3-4*x**2

fig, ax = plt.subplots()
ax.plot(x, y, lw=2)
ax.annotate('local mininmum', xy=(1, -1), xytext=(-0.5, -3.5),
            arrowprops=dict(facecolor='black'))

ax.set_ylim(-10,2)
plt.show()
```



Axes Customizing

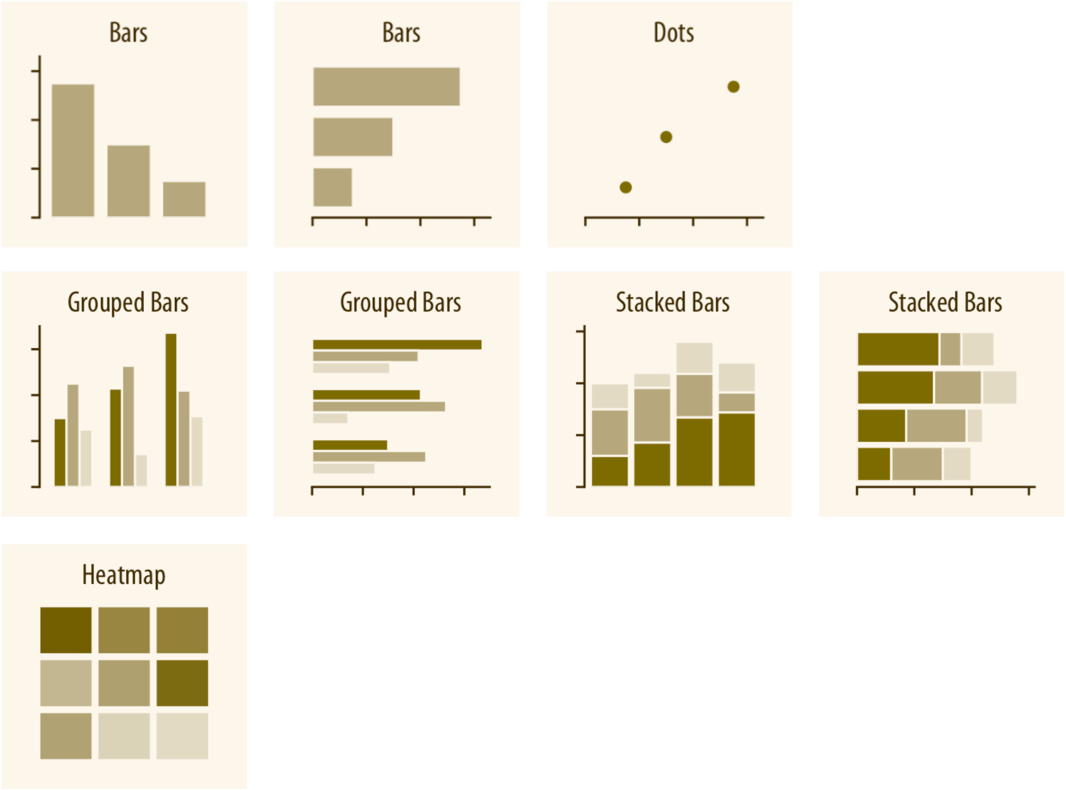
- Save figure
  - fig.savefig()

```python
fig, ax = plt.subplots()
x = np.arange(10)
y1 = x**2
ax.plot(x, y1, label = 'sin')
fig.savefig('./image_matplot_tmp.jpg')
```



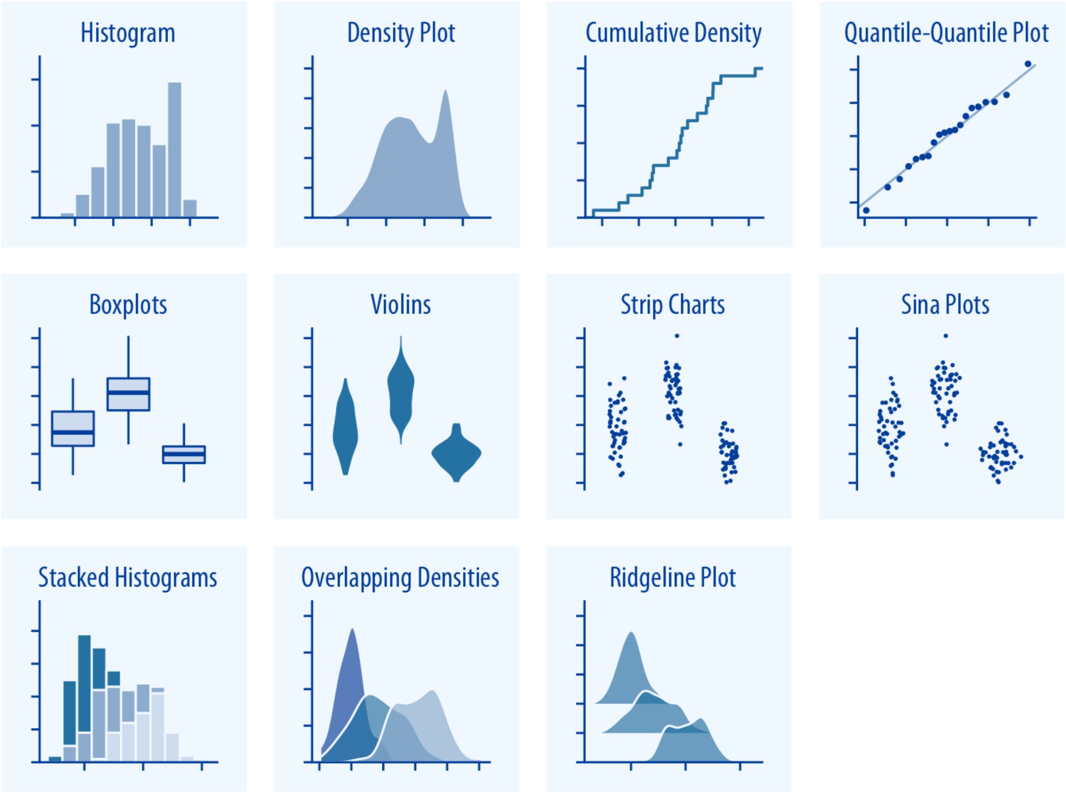
Directory of Visualizations

- Amounts
  - bars
  - dots
  - grouped bars
  - stacked bars
  - heatmap



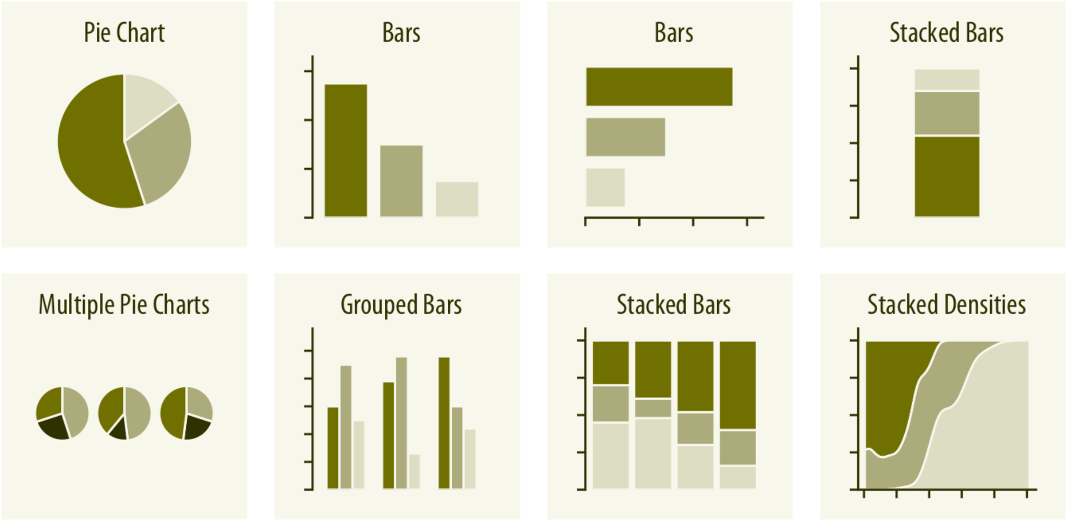
Directory of Visualizations

- Distributions
  - histogram
  - density plot
  - cumulative density
  - boxplots
  - violins
  - strip charts
  - sina plots
  - ridgeline plot



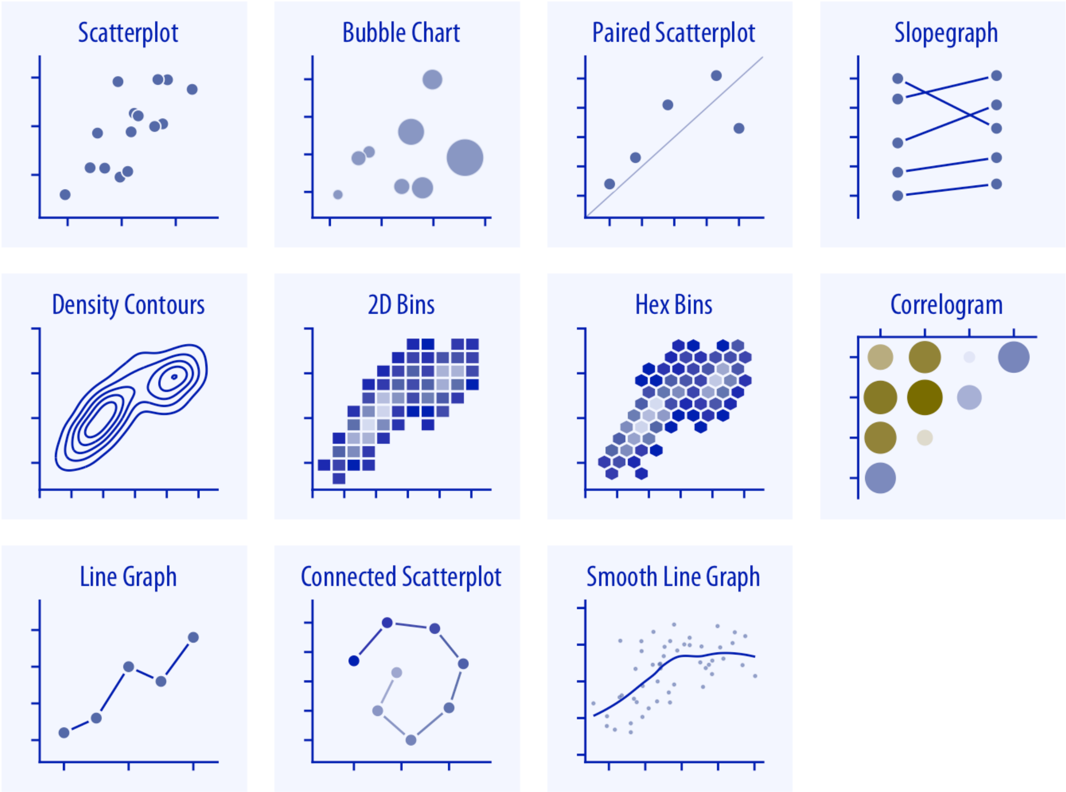
Directory of Visualizations

- Proportions
  - pie chart
  - bars
  - stacked bars
  - multiple pie carts
  - stacked densities



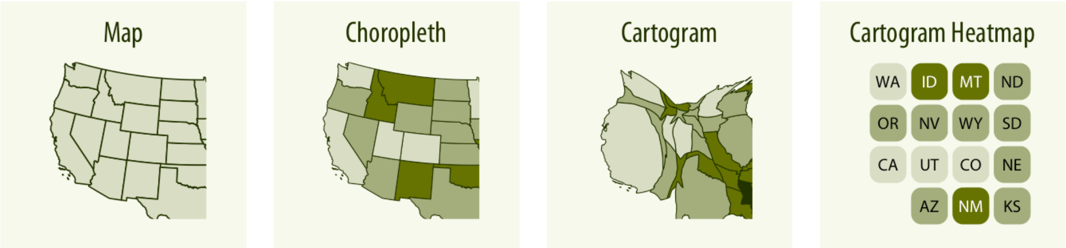
Directory of Visualizations

- x-y relationships
  - scatter plot
  - bubble chart
  - paired scatter plot
  - density contours
  - line graph



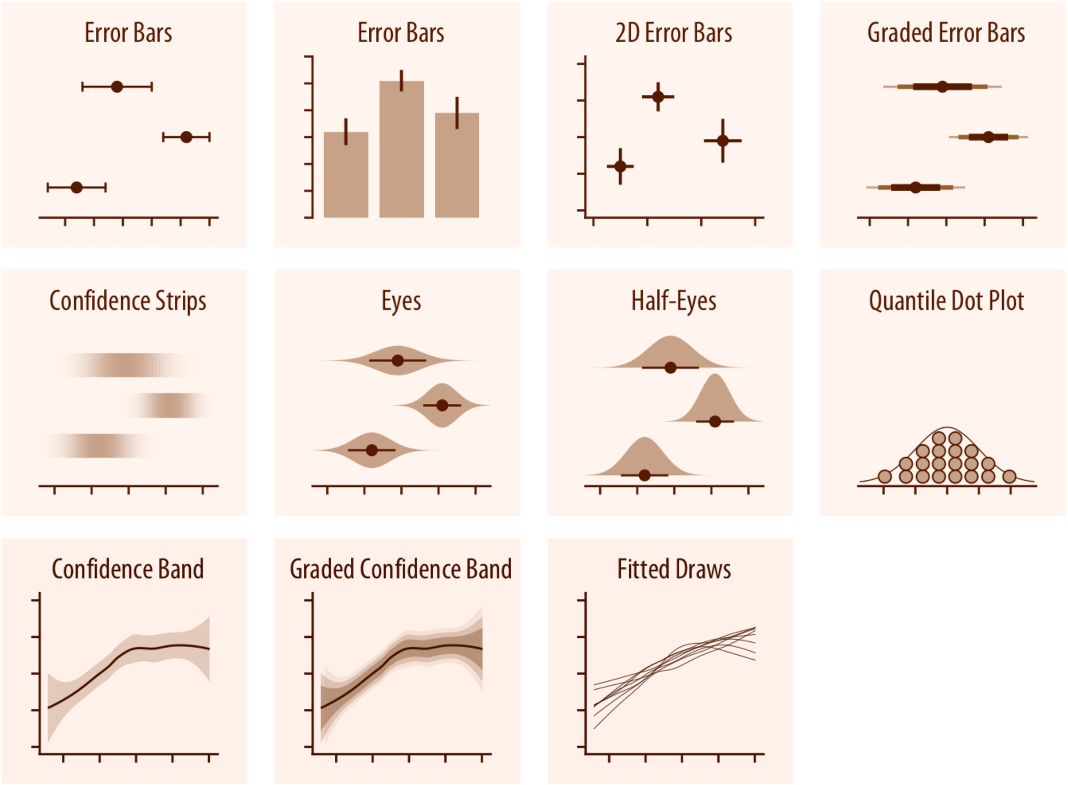
Directory of Visualizations

- geospatial data
  - map
  - choropleth
  - cartogram



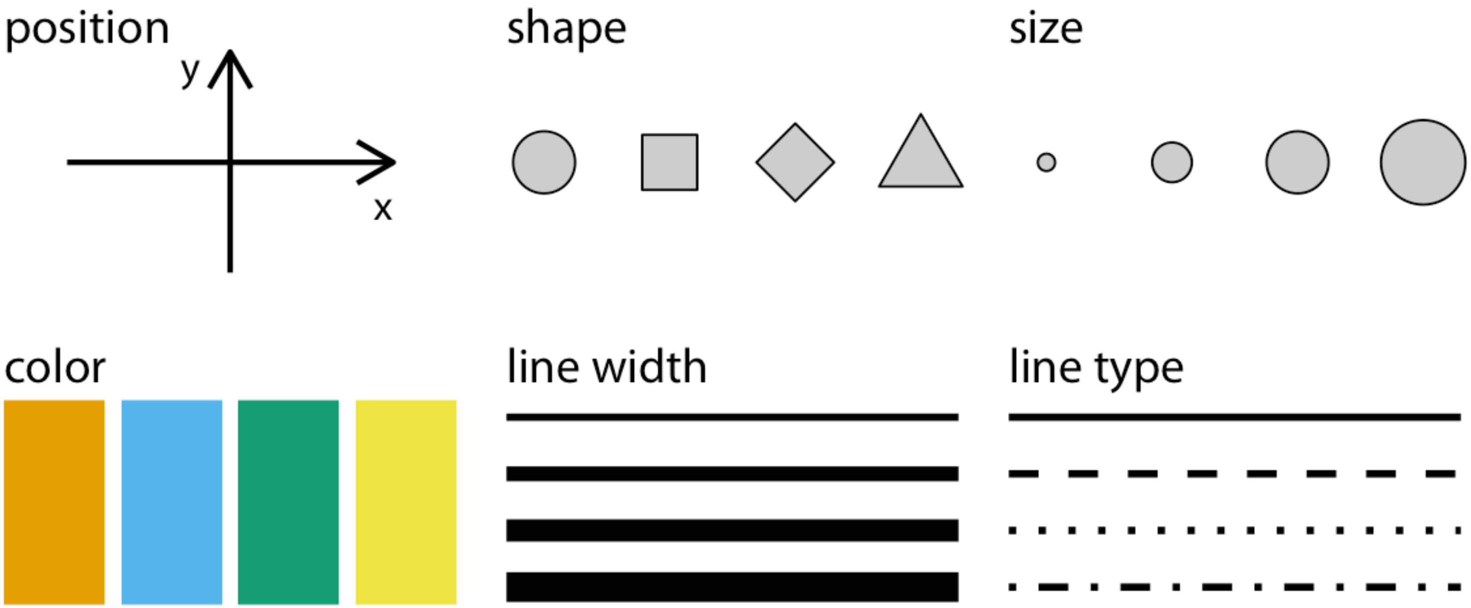
Directory of Visualizations

- uncertainty
  - error bars
  - eyes
  - confidence band



Graph

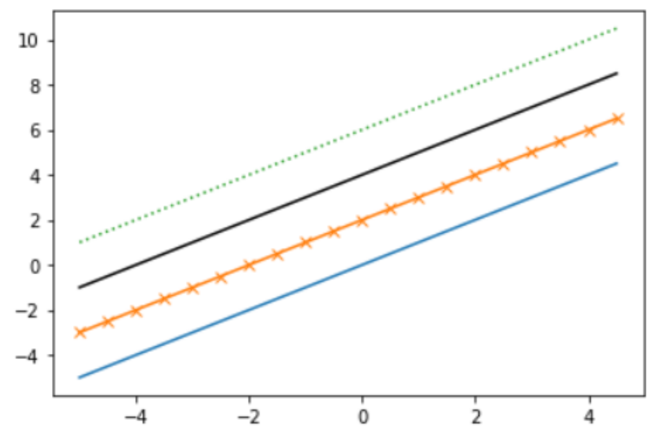
- Commonly used aesthetics in data visualization
  - position, shape, size, color, line width, line type
  - Some of these aesthetics can represent both continuous and discrete data (position, size, line width, color) while others can usually only represent discrete data (shape, line type)



Graph

- Line plot

```
x = np.arange(-5, 5, 0.5)
y1 = x
y2 = x+2
y3 = x+4
y4 = x+6

fig, ax = plt.subplots()
ax.plot(x, y1)
ax.plot(x, y2, marker='x')
ax.plot(x, y3, color='k')
ax.plot(x, y4, linestyle='dotted')
plt.show()
```

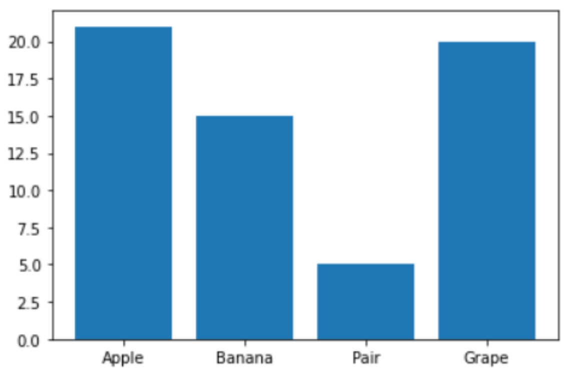


Graph

- Bar plot

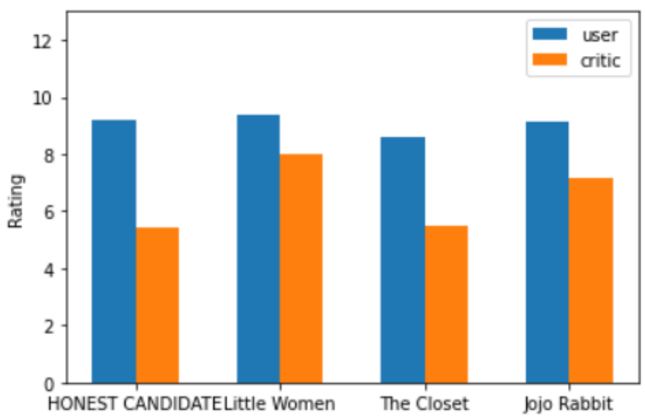
```python
data = {'Apple': 21, 'Banana': 15, 'Pair': 5, 'Grape': 20}
names = list(data.keys())
values = list(data.values())

fig, ax = plt.subplots()
ax.bar(names, values)
```

<BarContainer object of 4 artists>




```python
labels = ['HONEST CANDIDATE', 'Little Women', 'The Closet', 'Jojo Rabbit']
user = [9.2, 9.4, 8.6, 9.16]
critic = [5.4, 8, 5.5, 7.17]
x = np.arange(len(labels))  # the label locations

width = 0.3  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, user, width, label='user')
rects2 = ax.bar(x + width/2, critic, width, label='critic')

ax.set_ylim(0, 13)
ax.set_ylabel('Rating')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
```

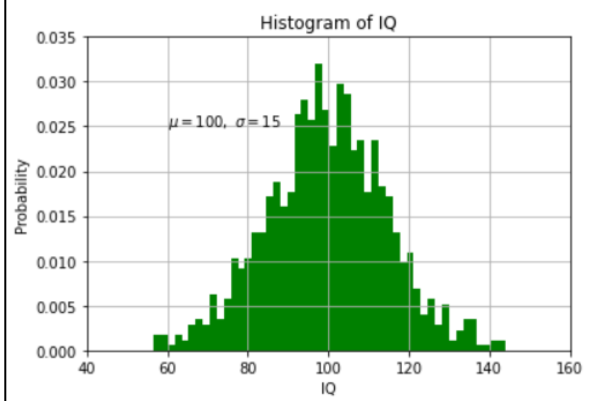
<matplotlib.legend.Legend at 0x7fc1ed3ccf50>



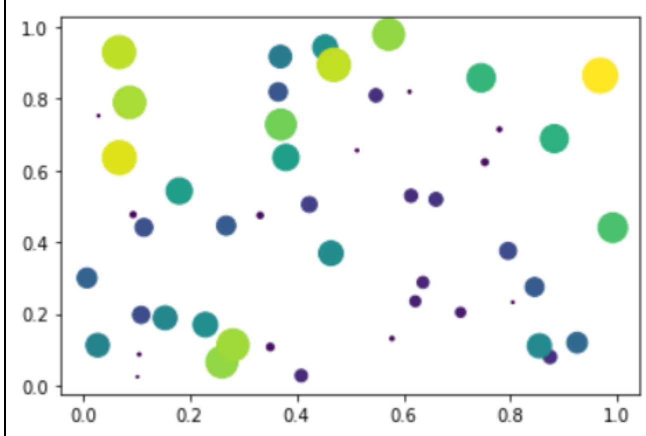
Graph

- Histogram & Scatter plots

```python
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(1000)

# the histogram of the data
plt.hist(x, 50, density=True, facecolor='g')

plt.xlabel('IQ')
plt.ylabel('Probability')
plt.title('Histogram of IQ')

plt.text(60, .025, r'$₩mu=100,₩ ₩sigma=15$')
plt.xlim(40, 160)
plt.ylim(0, 0.035)
plt.grid(True)
plt.show()
```




```python
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
area =(20 * np.random.rand(N))**2

fig, ax = plt.subplots()
ax.scatter(x, y, s=area, marker='o', c=area)

plt.show()
```



Thank you!
Q&A