머신러닝을 위한 데이터 시각화

Matplotlib 기본 활용 준비

matplotlib 임포트와 쥬피터 노트북에 플롯 삽입 준비

```
1  import matplotlib as mpl
2  import matplotlib.pyplot as plt
3
4  %matplotlib inline
```

matplotlib를 import할 때 일반적으로 사용하는 별칭

%(매직 명령어) : 파이썬 시스템 제어를 돕는 기능

notebook 또는 inline 중에 선택 가능

Matplotlib 기본 활용 준비

플롯 스타일과 설정

```
1  plt.style.available
```

```
['bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark-palette',
 'seaborn-dark',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'seaborn',
 'Solarize_Light2',
 'tableau-colorblind10',
 '_classic_test']
```

```
1  plt.style.use("seaborn-whitegrid")
```

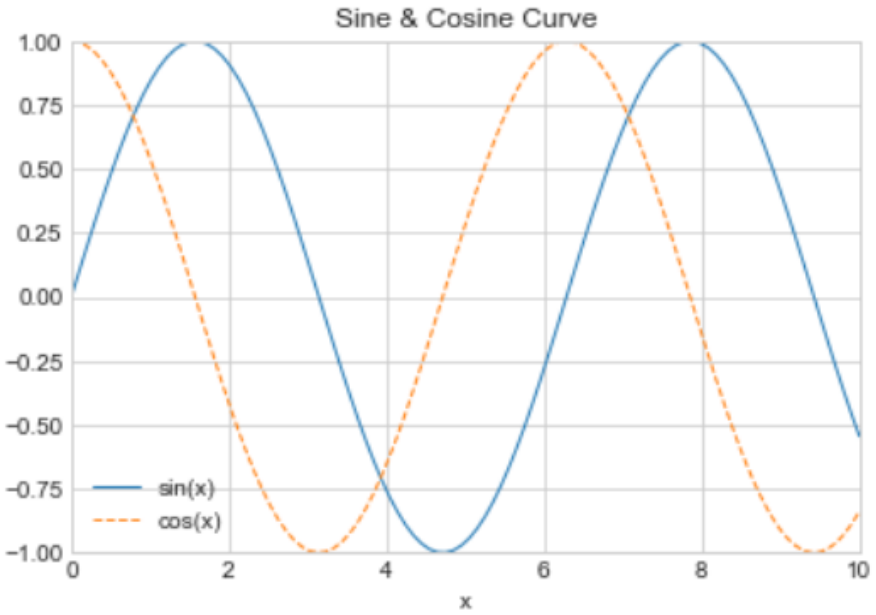
스타일 적용

라인 플롯

라인 플롯과 주요 속성 설정

```
1  import numpy as np
2
3  x = np.linspace(0, 10, 100)
4
5  plt.plot(x, np.sin(x), "-", linewidth=1, label="sin(x)")
6  plt.plot(x, np.cos(x), "--", linewidth=1, label="cos(x)")
7  plt.xlim(0, 10)
8  plt.ylim(-1, 1);
9  plt.title("Sine & Cosine Curve")
10 plt.xlabel("x")
11 plt.legend();
```

데이터 생성



plt.plot() : 라인 플롯을 그리는 함수
(x값, y값, 라인 스타일(컬러,선 모양, 포인트 등), 선 굵기, 범례)

plt.xlim(최소값, 최대값) : 플롯의 x축 범위 지정
plt.ylim(최소값, 최대값) : 플롯의 y축 범위 지정
plt.title(문자열) : 플롯의 제목
plt.label(문자열) / plt.ylabel(문자열) : 축 이름
Plt.legend() : 범례 표시

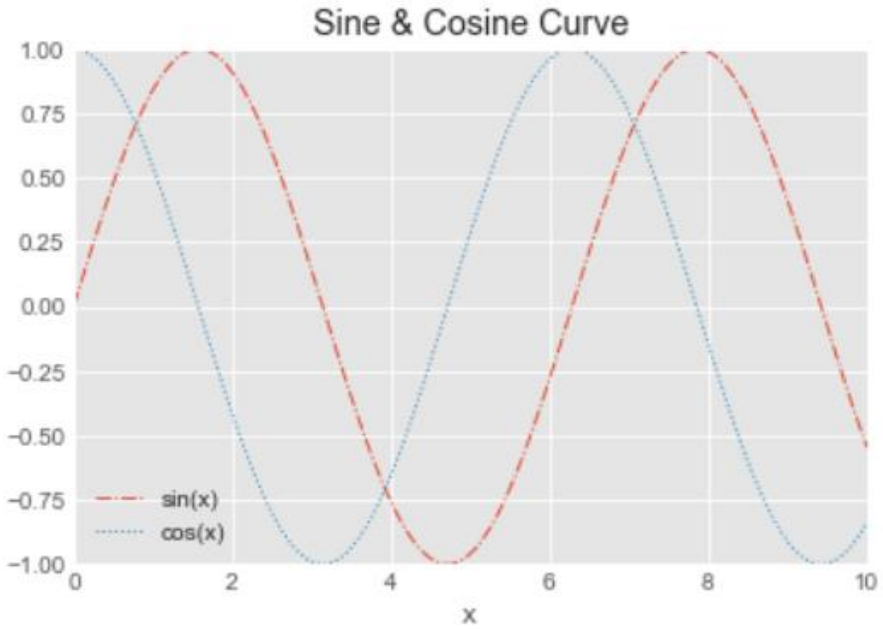
라인 플롯

스타일 컨텍스트 매니저를 이용한 플롯 스타일의 일시적 설정 변경

```
1  with plt.style.context("ggplot"):
2      plt.plot(x, np.sin(x), '-.', linewidth=1, label="sin(x)")
3      plt.plot(x, np.cos(x), ':', linewidth=1, label="cos(x)")
4      plt.xlim(0, 10)
5      plt.ylim(-1, 1)
6      plt.title("Sine & Cosine Curve")
7      plt.xlabel("x")
8      plt.legend();
```

ggplot 스타일 내에서만 적용

plt.style.context() 로 플롯 스타일의 일시적 변경



라인 플롯

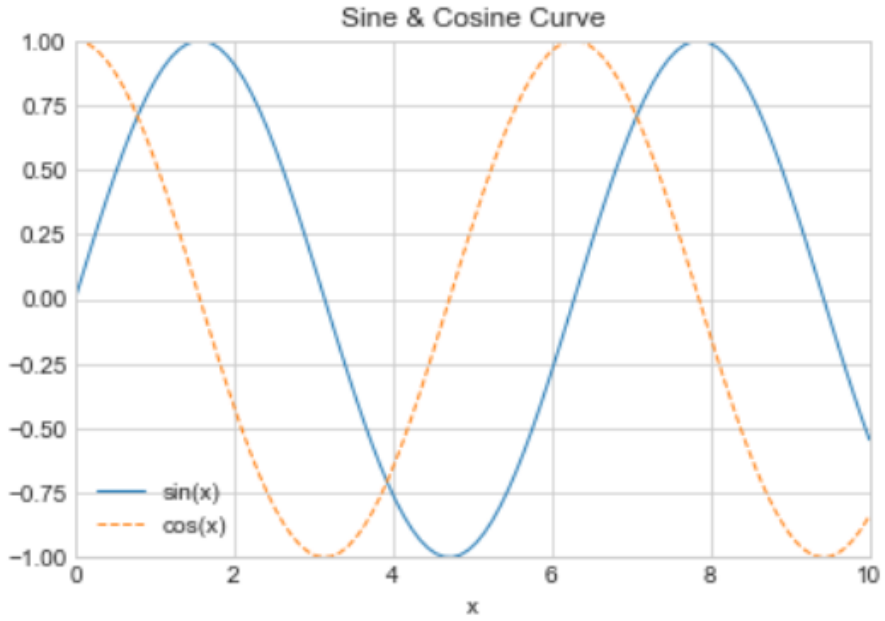
객체지향 인터페이스 Figure와 Axes 객체의 사용

```
1  fig = plt.figure()
2  ax = plt.axes()
3
4  x = np.linspace(0, 10, 100)
5
6  ax.plot(x, np.sin(x), "-", linewidth=1, label="sin(x)")
7  ax.plot(x, np.cos(x), "--", linewidth=1, label="cos(x)")
8  ax.set_xlim([0, 10])
9  ax.set_ylim([-1, 1])
10 ax.set_title("Sine & Cosine Curve")
11 ax.set_xlabel("x")
12 ax.legend();
```

플롯이 출력되는 캔버스 정보를 저장하는 객체

캔버스의 축(위치) 정보를 저장하는 객체

plt 대신 axe 객체를 통한 스타일 설정
(함수명 앞에 'set_' 이 포함됨에 유의)



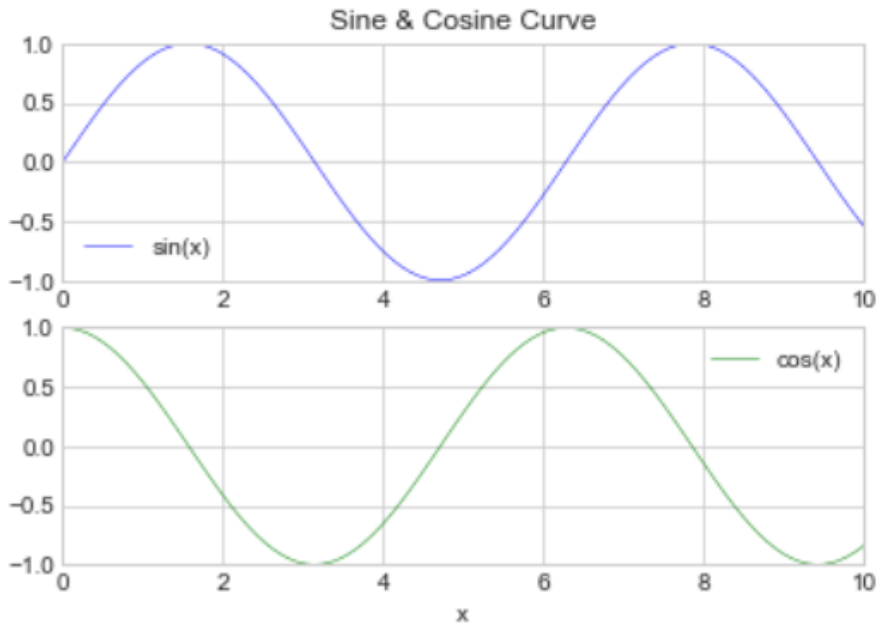
라인 플롯

2행 1열로 2개의 플롯을 그리기 위한
면 객체 및 축(위치)정보 객체 설정

```
1  fig, ax = plt.subplots(2, 1)
2
3  ax[0].plot(x, np.sin(x), 'b-'  linewidth=0.5, label="sin(x)")
4  ax[0].set_title("Sine & Cosine Curve")
5  ax[0].set(xlim=[0, 10], ylim=[-1, 1])
6  ax[0].legend()
7
8  ax[1].plot(x, np.cos(x), 'g-', linewidth=0.5, label="cos(x)")
9  ax[1].set(xlim=[0, 10], ylim=[-1, 1], xlabel="x")
10 ax[1].legend();
```

선 스타일 : blue, 실선

여러 개의 속성 한꺼번에 지정



라인 플롯

매트랩 스타일 pyplot 인터페이스의 사용

fig, ax 객체 생성없이
plt로 서브플롯 생성

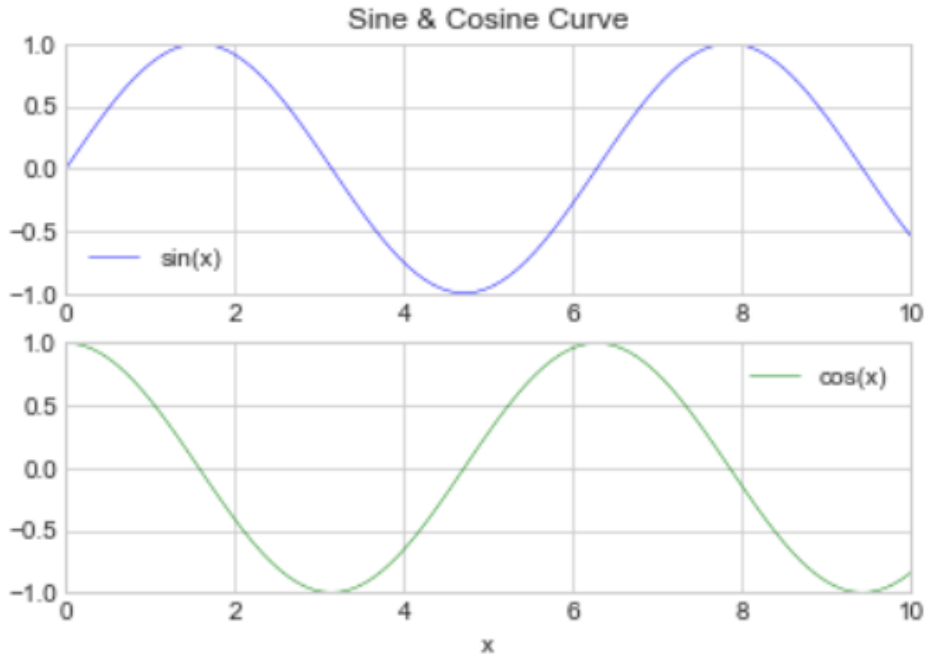
```
1  plt.subplot(2, 1, 1)
2  plt.plot(x, np.sin(x), 'b-', linewidth=0.5, label="sin(x)")
3  plt.title("Sine & Cosine Curve")
4  plt.axis([0, 10, -1, 1])
5  plt.legend()
6
7  plt.subplot(2, 1, 2)
8  plt.plot(x, np.cos(x), 'g-', linewidth=0.5, label="cos(x)")
9  plt.xlabel("x")
10 plt.axis([0, 10, -1, 1])
11 plt.legend();
```

2행 1열 플롯 중 1번째 서브플롯

축의 값 범위 설정

2행 1열 플롯 중 2번째 서브플롯

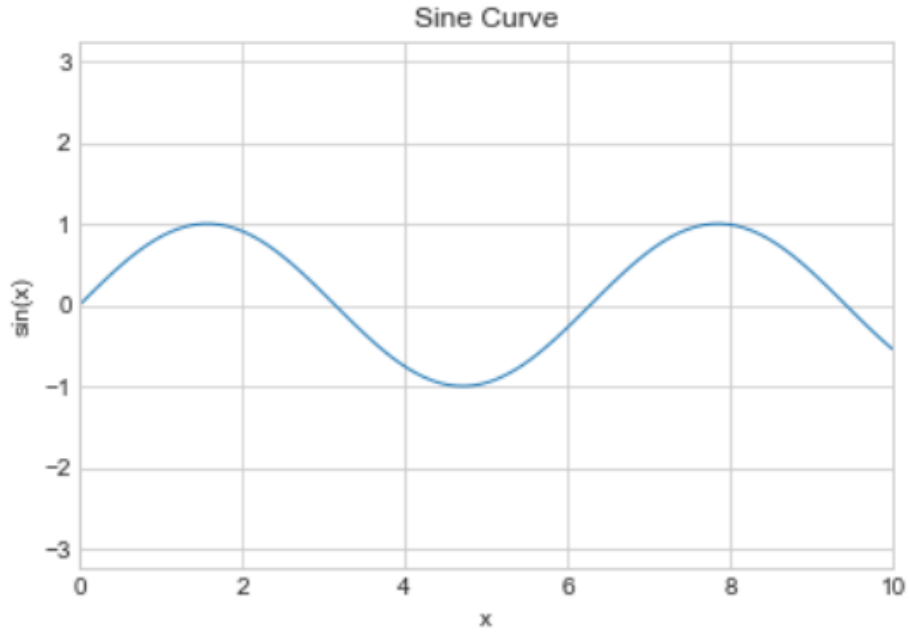
plt.subplot(행 개수, 열 개수, 플롯 번호)



라인 플롯

```
1  plt.plot(x, np.sin(x), "-", linewidth=1)
2  plt.title("Sine Curve")
3  plt.xlabel("x")
4  plt.ylabel("sin(x)")
5  plt.margins(0, 0)
6  plt.axis("equal");
```

플롯의 범위나 스케일이 변하더라도
그래프 모양(종횡비)를 일정하게 유지

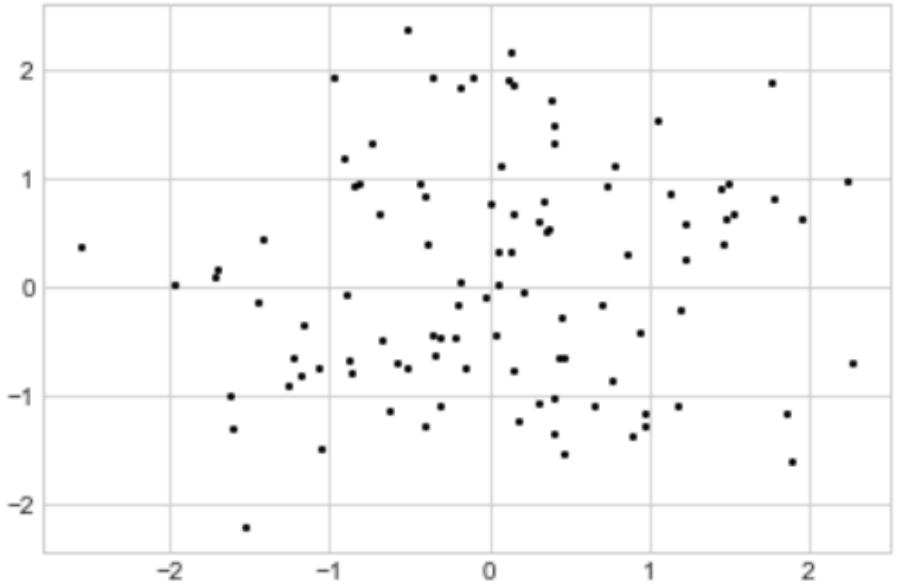


스캐터 플롯

plt.plot()을 이용한 스캐터 플롯

```
1  np.random.seed(0)
2
3  x = np.random.randn(100)
4  y = np.random.randn(100)
5
6  plt.plot(x, y, "ko", ms=2)
```

마커 사이즈

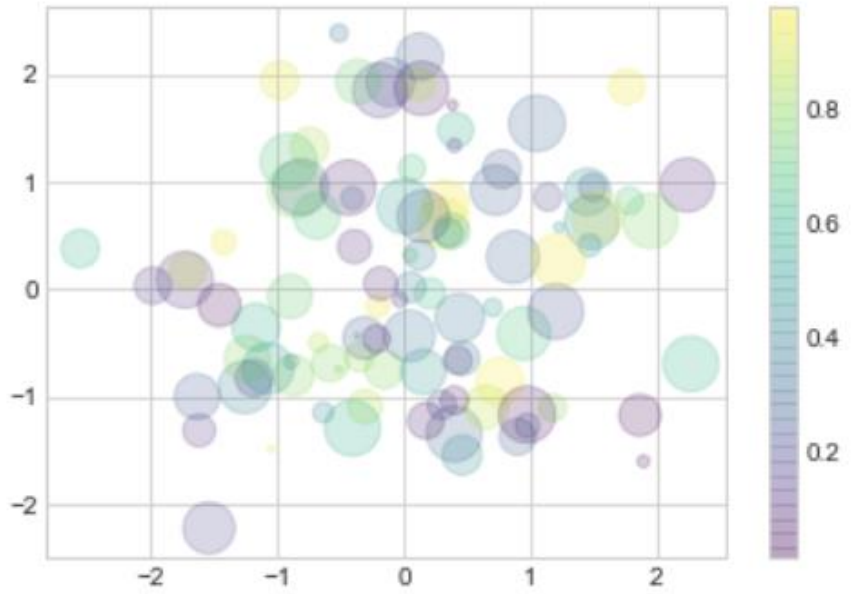


스캐터 플롯

plt.scatter()를 이용한 스캐터 플롯

```
1  np.random.seed(0)
2
3  x = np.random.randn(100)
4  y = np.random.randn(100)
5  c = np.random.rand(100)
6  s = np.random.rand(100) * 500
7
8  plt.scatter(x, y, c=c, s=s, alpha=0.2, cmap='viridis')
9  plt.colorbar();
```

plt.scatter(x값, y값, 컬러, 사이즈, 알파, 컬러맵)
plt.colorbar() : 컬러바 표시



스캐터 플롯

Seaborn 라이브러리 내 'iris' 데이터셋 불러오기

```
1 from seaborn import load_dataset
2 iris = load_dataset("iris")
3 iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

```
1 iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

스캐터 플롯

```
1  import pandas as pd
2
3  iris["species_cat"] = pd.Categorical(iris.species)
4  iris.info()
```

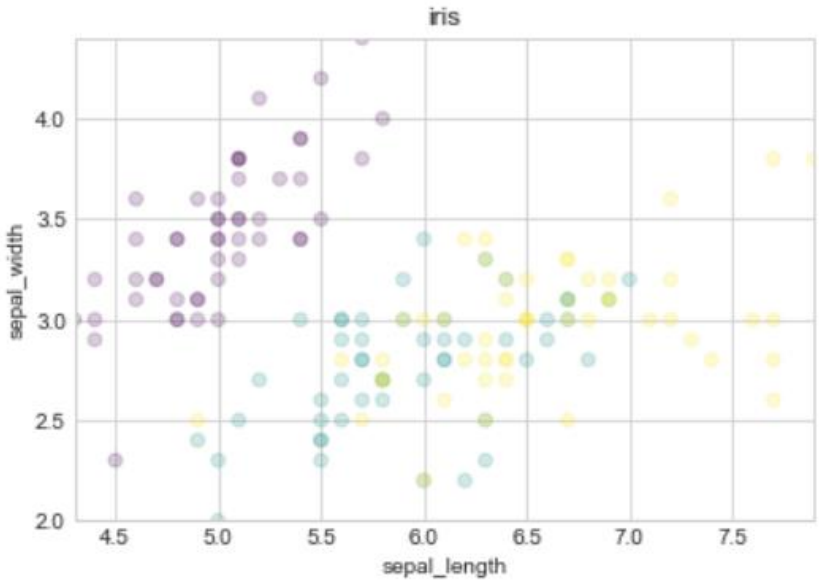
컬럼을 범주형 데이터로 변환

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
species_cat     150 non-null category
dtypes: category(1), float64(4), object(1)
memory usage: 6.2+ KB
```

스캐터 플롯

```
1  plt.scatter(iris.sepal_length, iris.sepal_width,
2              c=iris.species_cat.cat.codes, alpha=0.2, cmap="viridis")
3  plt.title("iris")
4  plt.xlim(iris.sepal_length.min(), iris.sepal_length.max())
5  plt.ylim(iris.sepal_width.min(), iris.sepal_width.max())
6  plt.xlabel(iris.columns[0])
7  plt.ylabel(iris.columns[1]);
```

Category 타입은 cat객체라는 수치속성을 가지고 있어 컬러 지정 가능



스캐터 플롯

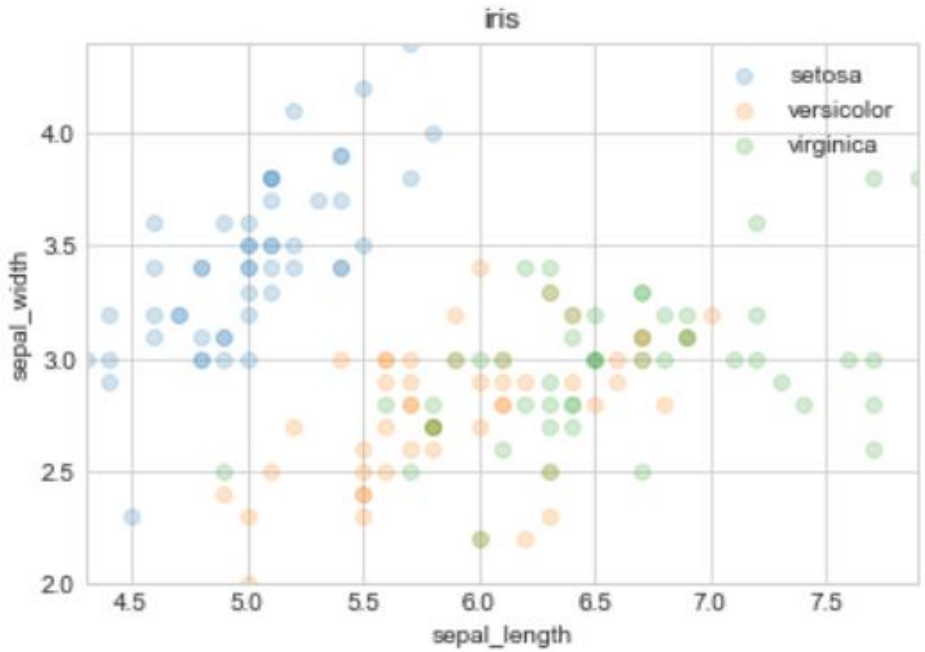
Category 타입 데이터는 unique() 함수로
3가지 품종에 대한 고유한 값을 얻을 수 있음

```
1  species = iris.species.unique()
2  species
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

Species를 루프로 하나씩 꺼내서 그려보기

```
1  for sp in species:
2      x = iris.loc[iris["species"] == sp, "sepal_length"]
3      y = iris.loc[iris["species"] == sp, "sepal_width"]
4      plt.scatter(x, y, label=sp, alpha=0.2)
5  plt.title("iris")
6  plt.xlim(iris.sepal_length.min(), iris.sepal_length.max())
7  plt.ylim(iris.sepal_width.min(), iris.sepal_width.max())
8  plt.xlabel(iris.columns[0])
9  plt.ylabel(iris.columns[1])
10 plt.legend();
```



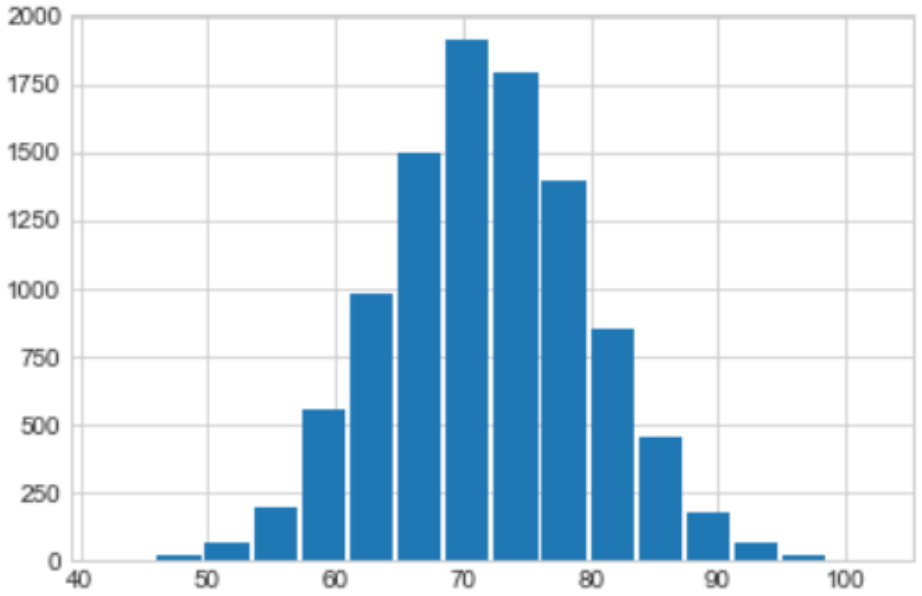
히스토그램

데이터의 분포 형상을 식별하는 히스토그램

```
1  np.random.seed(0)
2
3  mu = 72
4  sigma = 8
5  x = np.random.normal(mu, sigma, 10000)
6
7  plt.hist(x, rwidth=0.9, bins=16);
```

평균 72, 표준편차 8인
1만건의 데이터 생성

plt.hist(데이터, 막대의 너비, 계급개수)



히스토그램

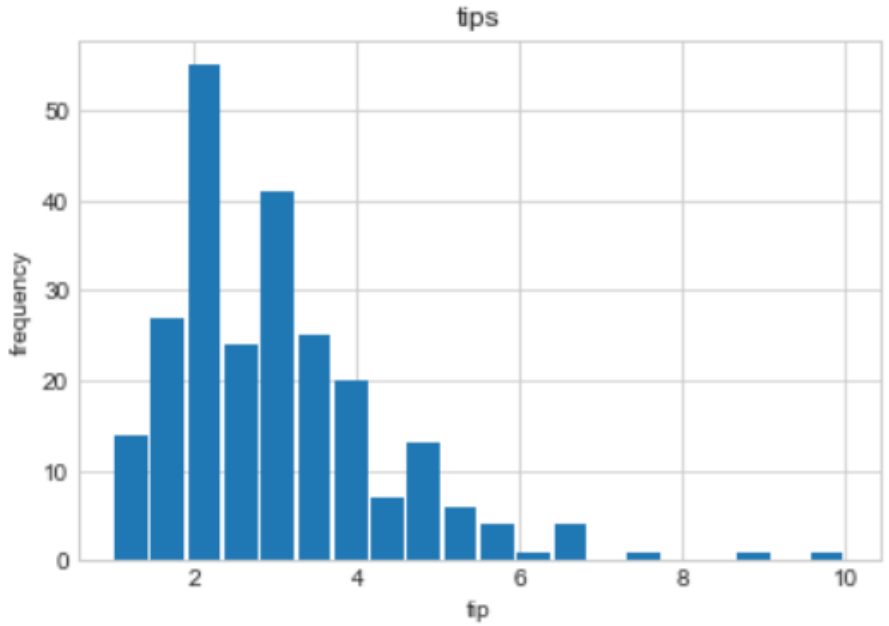
```
1  tips = load_dataset("tips")
2  tips.info()
```

bill, tip, 성별, 흡연 여부 등의 정보로 구성

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           244 non-null float64
sex           244 non-null category
smoker        244 non-null category
day           244 non-null category
time          244 non-null category
size          244 non-null int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.2 KB
```

히스토그램

```
1  plt.hist(x = tips.tip, rwidth=0.9, bins = 20);
2  plt.title("tips")
3  plt.xlabel("tip")
4  plt.ylabel("frequency");
```

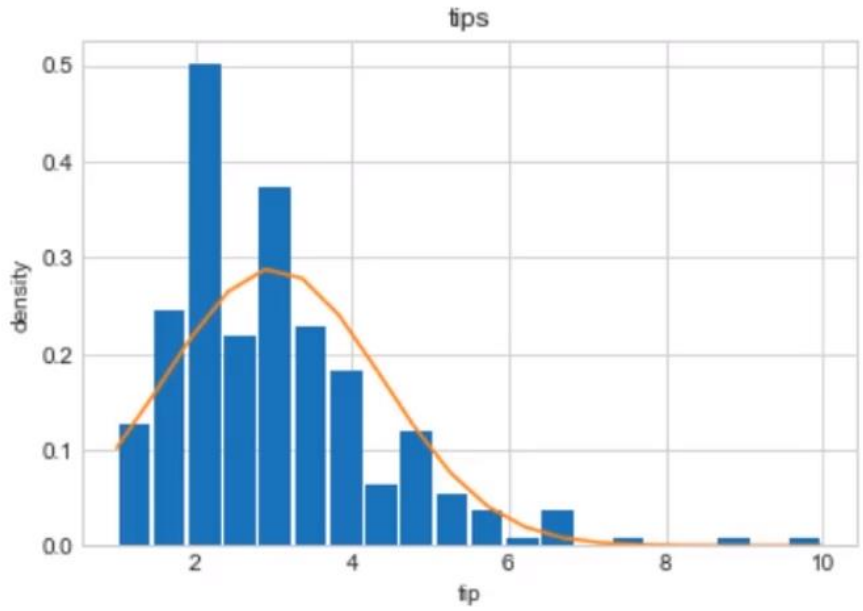


히스토그램

상대도수 히스토그램과 확률밀도 함수

```
1  import scipy as sp
2
3  bins = 20
4  plt.hist(x = tips.tip, rwidth=0.9, bins = bins, density=True);
5  plt.title("tips")
6  plt.xlabel("tip")
7  plt.ylabel("density")
8
9  mu, sigma = tips.tip.mean(), tips.tip.std()
10 tip_min = tips.tip.min()
11 tip_max = tips.tip.max()
12 x = np.linspace(tip_min, tip_max, bins)
13 y = sp.stats.norm(mu, sigma).pdf(x)
14 plt.plot(x, y);
```

Y축을 빈도 수 대신 확률로 표시



확률밀도함수 값 계산은 scipy 라이브러리의 stats.norm(mu, sigma).pdf(x)를 활용

박스 플롯

박스 플롯과 사분위수

박스플롯 : 데이터의 분포 파악 및 이상치 발견에 직관적인 도구

describe() : 데이터의 요약 통계 출력
round(1) : 소수점 2번째 자리에서 반올림

```
1  tips.loc[tips.sex == "Male", "tip"].describe().round(1)
```

```
count    157.0
mean       3.1
std        1.5
min        1.0
25%        2.0
50%        3.0
75%        3.8
max       10.0
Name: tip, dtype: float64
```

```
1  tips.loc[tips.sex == "Female", "tip"].describe().round(1)
```

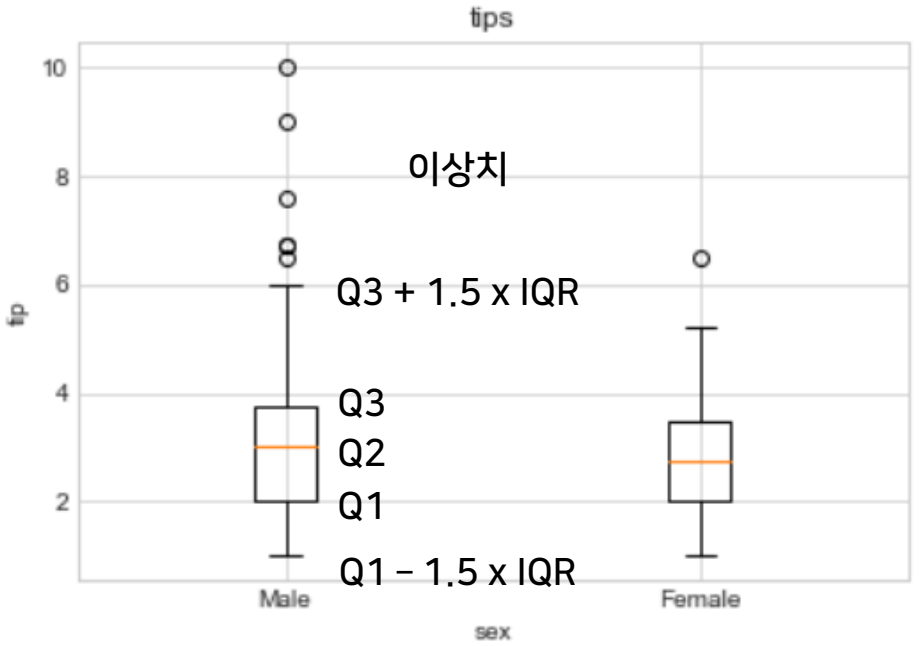
```
count    87.0
mean      2.8
std       1.2
min       1.0
25%       2.0
50%       2.8
75%       3.5
max       6.5
Name: tip, dtype: float64
```

박스 플롯

박스 플롯과 IQR

```
1  labels = []
2  tip_list = []
3
4  for label, df_per_sex in tips.groupby("sex"):
5      labels.append(label)
6      tip_list.append(df_per_sex["tip"].tolist())
7
8  plt.boxplot(tip_list, labels=labels)
9  plt.title("tips")
10 plt.xlabel("sex")
11 plt.ylabel("tip");
12
```

plt.boxplot(y값, labels(레이블(x축 정보)))



Interquartile Range(IQR) = $Q_3 - Q_1$

Lower 1.5 * IQR whisker = $Q_1 - 1.5 \times IQR$

Upper 1.5 * IQR whisker = $Q_3 + 1.5 \times IQR$

이미지 플롯

plt.imshow() 를 이용한 이미지 플롯

```
1  from sklearn.datasets import load_digits
2
3  digits = load_digits()
4  digits.keys()
```

숫자 손글씨 데이터 로딩 (bunch 객체로 되어있음)

```
dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

```
1  digits.images[0]
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

이미지에 대한 픽셀 정보(8x8)
(각 숫자는 밝기를 나타냄)

이미지 플롯

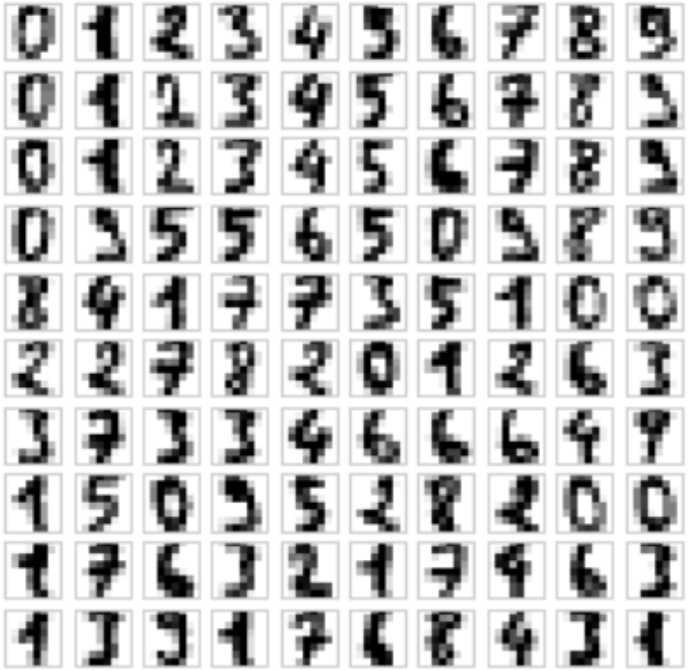
100개의 서브플롯 생성

```
1  fig, ax = plt.subplots(10, 10, figsize=(5, 5))
2  for i, ax_i in enumerate(ax.flat):
3      ax_i.imshow(digits.images[i], cmap="binary")
4      ax_i.set(xticks=[], yticks=[])
```

눈금 등의 정보는 기재하지 않고 공간만 구성

ax.flat : 2차원 구조를 1차원 구조로 변환

imshow(픽셀 데이터, 컬러맵)



주요 정리

1. Matplotlib 라이브러리를 이용해 시각화 작업을 할 경우, Figure와 Axes 객체와 같은 객체지향 인터페이스를 사용하거나, 활성화된 그림이나 축을 사용하는 매트랩 스타일의 pylab 인터페이스를 사용할 수 있다.
2. 라인 플롯은 함수를 시각화하기 위한 플롯으로, plot( ) 함수를 사용하며, 선의 색상, 스타일과 같은 속성을 데이터에 매핑해 사용한다.
3. 스캐터 플롯은 데이터의 산포도에 대한 직관을 얻을 수 있는 플롯으로, scatter( ) 함수를 사용하며, 각 점의 크기, 색상과 같은 속성을 데이터에 매핑해 사용한다.
4. 히스토그램은 계급 구간별 빈도와 밀도를 쉽게 이해할 수 있는 플롯으로, hist( ) 함수를 사용하며, scipy.stats 패키지의 norm.pdf( ) 함수를 이용해 근사값을 같이 표시할 수 있다.
5. 박스 플롯은 사분위수와 이상치의 존재 여부를 쉽게 파악할 수 있는 플롯으로, boxplot( ) 함수를 사용하며, 이미지 플롯은 이미지 데이터를 적재해 시각화하는 imshow( ) 함수를 사용한다.