고용노동부 한국산업인력공단 국가인적자원개발컨소시엄

설계

Data Science 입문 with Python

2018.7.9 ~ 7.10

백명숙

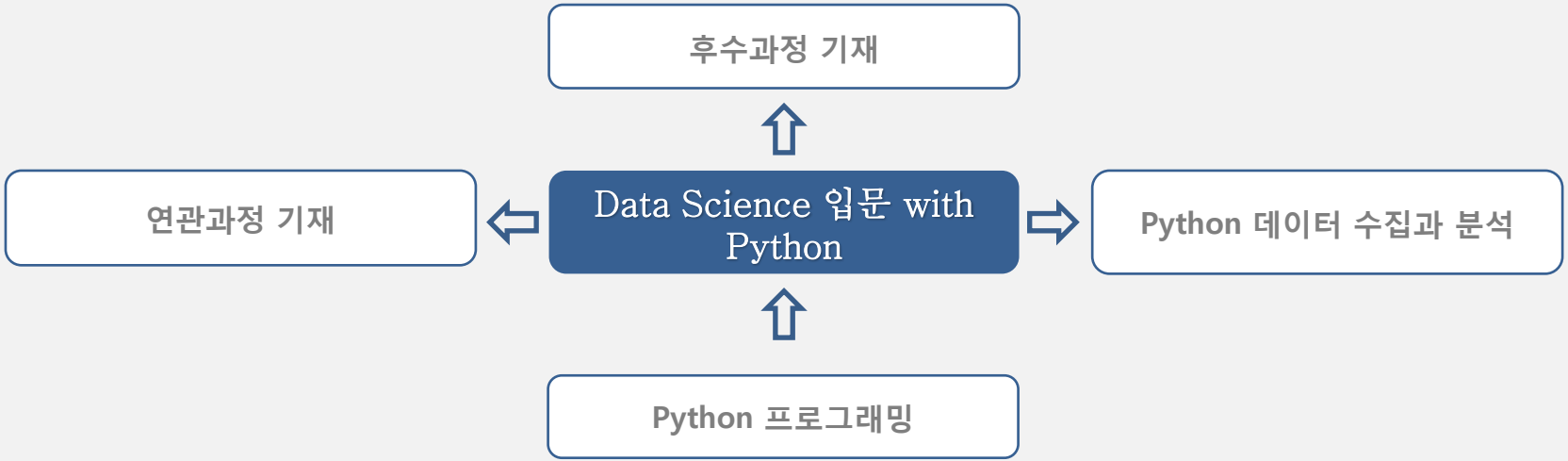
과정개요

■ 데이터 과학의 개념을 알아보고, 데이터 분석에서 주로 사용하는 파이썬의 핵심 내용을 살펴본다. CSV, JSON 파일, 수치형 및 테이블형 빅 데이터를 NumPy, Pandas를 사용해 분석하고 Matplotlib를 사용하여 그래프를 만들어 보면서 실제로 데이터를 어떻게 분석하고 활용할 수 있는지 이해하는 과정입니다.

과정목표

■ 데이터 과학을 위한 입문 과정으로 데이터를 분석하고 , 시각화 할 수 있는 능력을 키우는 것이 이 과정의 목표입니다.

러닝 맵


후수과정 기재
연관과정 기재
Data Science 입문 with Python
Python 데이터 수집과 분석
Python 프로그래밍

강사 프로필

성명	백명숙
소속 및 직함	*휴클라우드 전임강사*
주요 경력	일은시스템(제일은행 IT자회사), Sun MicroSystems 교육서비스, 인터넷커머스코리아
강의/관심 분야	Java, Framework, Python, Data Science, Machine Learning, Deep Learning
자격/저서/대외활동	Java기반 오픈소스 프레임워크/NCS 개발위원

학습 모듈(Learning Object) 및 목차

LO명	LO별 목차
데이터 분석의 개요	탐색적 데이터 분석의 개요 데이터 분석의 가치 데이터분석을 위한 Python의 핵심도구들
데이터분석 라이브러리 Numpy와 Pandas 사용하기	Numpy 라이브러리 개요 Numpy Array와 인덱싱 Numpy Array 관련 함수 Pandas 라이브러리 개요 Pandas의 DataFrame의 개요 DataFrame의 인덱싱과 데이터 분석용 함수
데이터 시각화 라이브러리 Matplotlib 사용하기	Matplolib 라이브러리 개요 Line plot 그리기 Bar plot 그리기 Histogram 그리기 Scatter plot 그리기 Plot 모양 변형하기
Pandas DataFrame의 merge와 grouping 기능 사용하기	DataFrame 합치기 (merging, concatenating) DataFrame 그룹핑 (grouping)
머신러닝 입문	머신러닝 개요 머신러닝의 분류

목 차

1.데이터 분석 이란?

데이터 분석이란?

- 데이터란, 어떤 사실에 대한 측정이나 관찰을 통해 모아 놓은 값들의 모음을 말함
  보통 '모음'이라는 사실을 강조하기 위해 'Data Set'이라는 표현도 많이 쓰임
- 분석이란, 어떤 대상을 더 잘 이해하기 위해 나누고 쪼개는 것을 말함
- 즉 데이터 분석이란, 어떤 대상에 대한 사실을 설명하는 값의 모음을 유효 적절한 방법으로 나누고 쪼개서 그 대상을 더 잘 이해하는 것이라고 할 수 있다.

탐색적 데이터 분석(EDA)이란?

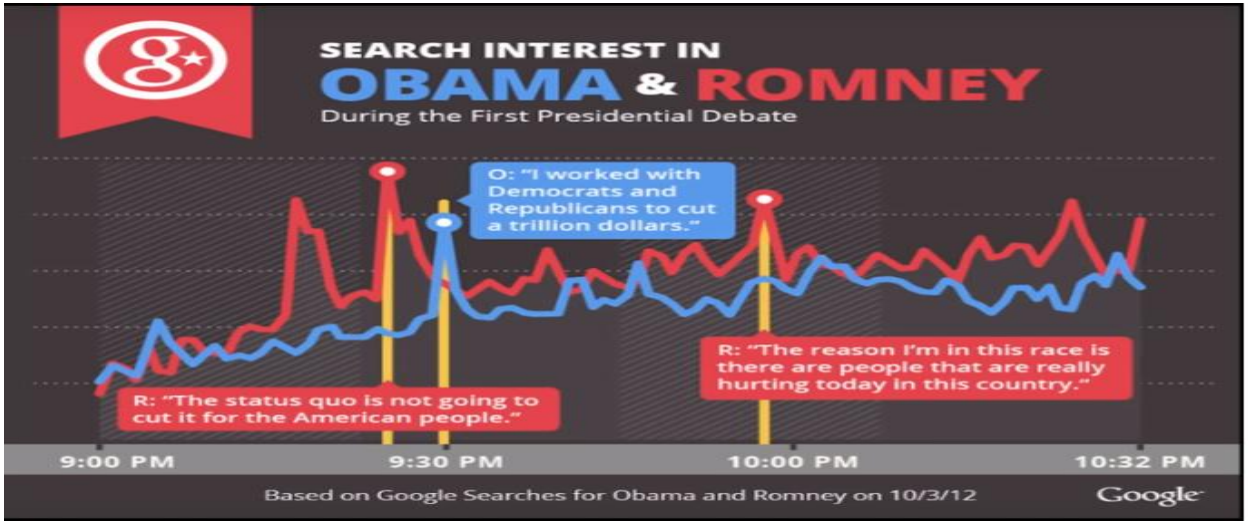
- **탐색적 데이터 분석(EDA, exploratory data analysis)**은 기계학습(machine learning), 자연어 처리(natural language processing), 패턴인식(pattern recognition) 등 모든 데이터 과학의 출발점이라고 할 수 있다.
- 각종 통계적 기법을 적용하여 데이터의 주요 특징을 발견하고, 이를 시각화하여 표현하는 일련의 과정을 포괄함
- 본 과정에서는 탐색적 데이터 분석 방법을 배울 것이며, 이를 곧 '데이터 분석'으로 지칭할 것임

데이터 분석의 가치

1. 버락 오바마 미국 대통령의 재선 사례

오바마 선거 캠프에서는 유권자들의 64%가 인터넷을 사용하여 대선 후보자들의 발언의 진위 여부를 조사한다는 연구 결과에 주목하여, 유권자들이 Google에서 어떤 검색어를 사용하였을 때 선거 캠프 웹 페이지로 가장 많이 유입되는 지를 데이터 분석을 통해 실시간으로 조사하였다.

데이터 분석에 기반한 이러한 전략은 **, 결과적으로 오바마의 재선을 성공시키는 주된 요인으로 작용함**



데이터 분석의 가치

2. 아마존의 추천 시스템 사례

온라인 서점으로 유명한 아마존에서는 고객들이 이전에 구매했던 서적 데이터에 대한 분석을 통해, 유사한 구매 성향의 고객들이 구매한 서적을 추천해 주는 방식으로 자체적인 시스템을 구축함

추천 시스템의 도입으로 인한 매출은, 아마존의 전체 매출의 35% 가량을 차지함

데이터 분석의 가치

3. 2014 브라질 월드컵에서의 독일 국가대표 축구팀 사례

독일 축구협회에서는 SAP AG 측으로 독일 국가대표 축구팀의 경기 영상을 촬영한 비디오 데이터와, 선수 개인별 기록 데이터, 팀 기록 데이터 등에 대한 분석을 의뢰하였습니다. SAP AG 측에서는 해당 데이터에 대한 분석을 통해 선수 개인의 기량을 어떻게 향상 시킬 수 있을지, 팀워크를 어떻게 향상 시킬 수 있을지에 대한 자세한 피드백을 제공하였습니다.

그 결과, 독일 대표팀은 준결승전에서 홈 팀인 브라질을 7:1로 대파하고, 결승전에서는 아르헨티나마저 꺾어 버리는 놀라운 성적을 보였습니다.

‘빅 데이터’의 등장

- 빅 데이터(Big Data)는 양이 너무 많고 복잡해서, 통상적으로 사용되는 데이터 분석 방법이 잘 적용되지 않는 데이터를 의미한다.
- 통계이론의 핵심적인 이론인 ‘큰 수의 법칙(law of large numbers)’에 의하면 표본의 크기가 커질수록 그 데이터의 평균이 실제 모집단 데이터의 평균에 가까워진다는 사실이 알려져 있습니다. 이는 양면으로 구성된 동전을 더 많이 던질수록, 전체 던진 횟수 대비 앞면이 나온 횟수의 비율이 (이론적으로 동전의 앞면이 나올 확률인) 0.5에 가까워진다는 결과를 통해 확인할 수 있습니다.
- **이 법칙을 데이터의 측면에서 바라보면, 어떤 대상과 관련된 사실의 단면을 보여주는 데이터의 양이 많을수록, 분석을 통해 대상에 대한 이해를 보다 사실에 가깝게 할 수 있을 가능성이 높아진다는 것을 입증하고 있습니다. 다시 말해, 데이터의 양이 많아지면서 빅 데이터에 가까워질수록, 분석 결과를 '일반화(generalization)'하기가 더 수월해진다는 것입니다.**
- 오늘날, 분야를 막론하고 빅 데이터에 대한 분석은 가히 일의 성패를 좌우할 수 있는 중요한 과제로 부각되고 있습니다.

왜 파이썬 인가?

- **다른 수많은 프로그래밍 언어들 중, 데이터 분석을 위해 Python을 추천하는 이유**
  - 언어 자체가 너무 쉽습니다.
  - 컴퓨터와 대화하듯이 프로그래밍이 가능합니다.
  - Numpy, Pandas, Matplotlib 등의 강력한 데이터 분석을 위한 라이브러리를 제공합니다.
  - 모두 오픈 소스(open source)입니다.

- R 언어의 경우에도 위와 같은 장점을 모두 가지고 있는 스크립트 언어이나, R의 경우 통계 분석 및 리서치 작업 등에 특화된 성격이 강합니다. 반면, **Python의 경우 다양한 목적으로 범용적으로 사용할 수 있는 언어입니다. 이는 Python으로 데이터 분석한 결과물을 웹 사이트를 통해 외부에 공개하고자 할 때도 매우 유리한 특징이라고 할 수 있습니다.**

데이터 분석을 위한 Python의 핵심도구

- Python shell이라고 부르는 기본적인 대화식 프로그래밍 툴을 제공하는데, **IPython**은 이 기본 툴에 몇 가지 강력한 기능을 덧붙인 Tool이라고 할 수 있습니다.
- **IPython Notebook**은 IPython의 대화식 프로그래밍 방식을 기본적으로 제공하면서, 여러분이 데이터 분석을 하는 과정을 노트 형식으로 보기 쉽게 기록하고 정리해 놓을 수 있도록 도와주는 강력한 툴입니다.
- **Numpy**

  Numpy는 주요한 Python 데이터 분석 라이브러리들의 기본 베이스가 되는 라이브러리입니다.

  Numpy는 특히 벡터(vector) 및 행렬(array) 연산에 있어 엄청난 편의성을 제공하는 라이브러리입니다.

데이터 분석을 위한 Python의 핵심도구

Pandas

Pandas는 고유하게 정의한 Series 및 DataFrame 등의 자료구조를 활용하여, 빅 데이터 분석에 있어 높은 수준의 퍼포먼스를 가능하게 하는 라이브러리입니다. 기존에 엑셀로 하던 모든 분석을 더 큰 스케일의 데이터에 적용할 수 있으며, 더 빠른 속도로 수행할 수 있습니다.

Matplotlib

Matplotlib은 데이터 시각화를 위한 라이브러리입니다. Matplotlib을 사용하면 데이터 분석 결과에 대한 시각화를 빠르고 깔끔하게 수행해 줍니다.

Scikit-Learn

회귀, 분류, 군집, 차원축소, 특성공학, 전처리, 교차검증, 파이프라인 등 머신러닝에 필요한 도구를 두루 갖추고 있습니다.

어떠한 데이터를 다루게 되나요?

MovieLens 1M dataset

MovieLens에서는 각 사용자가 여러 개의 영화에 대하여 매긴 평점을 사용하여, 해당 사용자가 시청한 적이 없는 영화에 대한 예상 평점을 산출하고 이를 기반으로 사용자에게 새로운 영화를 추천합니다.

Lending Club Loan dataset 2007-2015

미국의 peer-to-peer 대출 서비스인 Lending Club이라는 업체에서 2007년부터 2015년 사이에 발생한 대출 관련 정보를 데이터셋으로 구성하여 공개한 Lending Club Loan 데이터셋 입니다.

Deaths and Battles from Game of Thrones dataset

Kaggle에서 제공하는 데이터셋이며, 절찬리에 방영된 미국 TV 드라마 시리즈인 <왕좌의 게임(Game of Thrones>에서 벌어지는 전투와, 극중에서 사망하는 등장 인물들의 정보가 포함된 Deaths and Battles from Game of Thrones 데이터셋입니다.

2016 US Election dataset

Kaggle에서 제공하는 2016 US Election 데이터셋을 소개합니다. 미국 대선 후보들의 2016년 예비 선거(primary) 결과를 담은 데이터셋입니다. 미국의 지역 별 정치 여론을 잘 반영하고 있는 데이터셋 입니다.

어떠한 데이터를 다루게 되나요?

Bike Sharing Demand dataset

Kaggle에서 제공하는 Bike Sharing Demand 데이터셋을 소개합니다.

자전거 공유 시스템은 회원 가입, 임대 및 자전거 반환 절차가 도시 전역의 키오스크 위치 네트워크를 통해 자동화 되는 자전거 대여 방법입니다.

이 시스템을 사용하여 사람들은 한 곳에서 자전거를 빌려 필요할 때 다른 곳으로 돌려 보낼 수 있습니다.

현재 전세계에는 500 개가 넘는 자전거 공유 프로그램이 있습니다.

2. 데이터 분석 환경 구성 및 라이브러리 설치

데이터 분석을 위한 환경 구성하기

Scikit-learn 설치

scikit-learn은 두 개의 다른 파이썬 패키지인 **NumPy**넘파이와 **SciPy**싸이파이를 사용합니다. 그래프를 그리려면 matplotlib맷플롯립을, 대화식으로 개발하려면 IPython아이파이썬과 주피터 노트북도 설치해야 합니다. 그래서 필요한 패키지들을 모아놓은 파이썬 배포판을 설치하는 방법을 권장합니다. 다음은 대표적인 배포판들입니다.

Anaconda (https://www.continuum.io/anaconda-overview)

대용량 데이터 처리, 예측 분석, 과학 계산용 파이썬 배포판입니다. Anaconda아나콘다는 NumPy, SciPy, matplotlib, pandas판다스, IPython, 주피터 노트북, 그리고 scikit-learn을 모두 포함합니다. macOS, 윈도우, 리눅스를 모두 지원하며 매우 편리한 기능을 제공하므로 파이썬 과학 패키지가 없는 사람에게 추천하는 배포판 입니다. Anaconda는 상용 라이브러리인 인텔 MKL(Math Kernel Library) 라이브러리도 포함합니다. MKL을 사용하면(Anaconda를 설치하면 자동으로 사용할 수 있게 됩니다) scikit-learn의 여러 알고리즘이 훨씬 빠르게 동작합니다.

참고) MKL은 인텔 호환 프로세서를 위한 고성능 수학 라이브러리입니다. https://software.intel.com/en-us/intel-mkl

데이터 분석을 위한 환경 구성하기

- **Enthought Canopy** (https://www.enthought.com/products/canopy/)

  또 다른 과학 계산용 파이썬 배포판입니다. NumPy, SciPy, matplotlib, pandas, IPython을 포함하지만 무료 버전에는 scikit-learn이 들어 있지 않습니다. 학생과 학위 수여가 되는 기관 종사자는 Enthought Canopy의 유료 버전을 무료로 받을 수 있는 아카데믹 라이선스를 신청할 수 있습니다. Enthought Canopy는 파이썬 2.7.x에서 작동하며 macOS, 윈도우, 리눅스에서 사용할 수 있습니다.

- **Python(x,y)** (http://python-xy.github.io/)

  특별히 윈도우 환경을 위한 과학 계산용 무료 파이썬 배포판입니다. Python(x,y)는 NumPy, SciPy, matplotlib, pandas, IPython, scikit-learn을 포함합니다.

- **Python** (https://www.python.org/downloads/)

```
pip install numpy
pip install scipy
pip install pandas
pip install scikit-learn
pip install matplotlib
pip install seaborn
pip install jupyter
```

Jupyter Notebook

- 프로그램 코드 + 결과 + 문서를 위한 대화식 개발 환경
- 탐색적 데이터 분석(EDA)에 유리하여 많은 과학자 엔지니어들이 사용

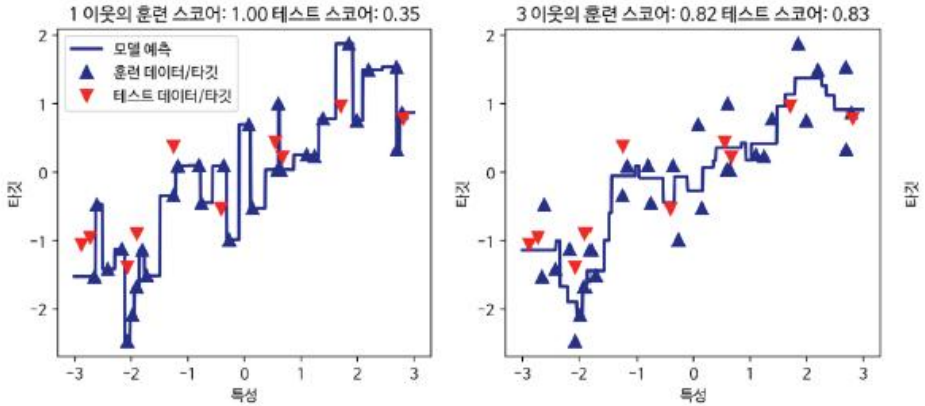
https://jupyter.org

KNeighborsRegressor 분석

In [25]:
```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# -3 과 3 사이에 1,000 개의 데이터 포인트를 만듭니다
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes):
    # 1, 3, 9 이웃을 사용한 예측을 합니다
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)

    ax.set_title(
        "{} 이웃의 훈련 스코어: {:.2f} 테스트 스코어: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train), reg.score(X_test, y_test)
    ax.set_xlabel("특성")
    ax.set_ylabel("타깃")
axes[0].legend(["모델 예측", "훈련 데이터/타깃", "테스트 데이터/타깃"], loc="best")
```

Out[25]: `<matplotlib.legend.Legend at 0x114152b00>`



Jupyter Notebook 실행 방법

```
jupyter notebook
```

Jupyter Notebook이 실행되면서 **http://localhost:8888**에서 실행되고 있다는 메시지를 확인

웹 브라우저를 실행하여 해당 URL로 접속하면, Jupyter Notebook의 GUI 화면을 확인

상단의 New 버튼을 클릭, 맨 하단의 Python 3 메뉴를 클릭하면, 새로운 노트북을 생성하고 편집가능

참고) IPython 노트북에서 여러 언어를 포괄하는 프로젝트인 주피터 노트북으로 이름이 바뀌었고 IPython은 주피터 노트북의 파이썬 커널을 의미하게 되었습니다. Jupyter 란 이름은 줄리아(Julia), 파이썬(Python), R의 합성어이고 목성의 발음과 같아 과학자들과 천문학자들에 대한 경의가 담겨 있습니다. 주피터 노트북 로고의 가운데 큰 원은 목성을 의미하며 주위 3개의 작은 원은 1610년 목성의 위성 3개를 최초로 발견한 갈릴레오 갈릴레이를 기리는 의미입니다.

Numpy

- NumPy(http://www.numpy.org/)는 파이썬으로 과학 계산을 하려면 꼭 필요한 패키지입니다. 다차원 배열을 위한 기능과 선형대수연산과 푸리에 변환 같은 고수준 수학 함수와 유사pseudo 난수 생성기를 포함합니다.
- scikit-learn에서 NumPy 배열은 기본 데이터 구조입니다. scikit-learn은 NumPy 배열 형태의 데이터를 입력으로 받습니다. 그래서 우리가 사용할 데이터는 모두 NumPy 배열로 변환되어야 합니다. NumPy의 핵심 기능은 다차원(n-차원) 배열인 ndarray 클래스입니다. 이 배열의 모든 원소는 동일한 데이터 타입이어야 합니다.


```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6]])
print("x:₩n{}".format(x))
```

Scipy

- SciPy(https://www.scipy.org/scipylib)는 과학 계산용 함수를 모아놓은 파이썬 패키지입니다. SciPy는 고성능 선형대수, 함수 최적화, 신호 처리, 특수한 수학 함수와 통계 분포 등을 포함한 많은 기능을 제공합니다. scikit-learn은 알고리즘을 구현할 때 SciPy의 여러 함수를 사용합니다. 그 중에서 가장 중요한 기능은 scipy.sparse입니다. 이 모듈은 scikit-learn에서 데이터를 표현하는 또 하나의 방법인 희소 행렬 기능을 제공합니다. 희소 행렬sparse matrix, 희박 행렬은 0을 많이 포함한 2차원 배열을 저장할 때 사용합니다.

```
from scipy import sparse
# 대각선 원소는 1이고 나머지는 0인 2차원 NumPy 배열을 만듭니다.
eye = np.eye(4) print("NumPy 배열:₩n{}".format(eye))
```

```
NumPy 배열:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

Scipy

```
In [4]: # NumPy 배열을 CSR 포맷의 SciPy 희박 행렬로 변환합니다.
        # 0이 아닌 원소만 저장됩니다.
        sparse_matrix = sparse.csr_matrix(eye)
        print("\nSciPy의 CSR 행렬:\n{}".format(sparse_matrix))

SciPy의 CSR 행렬:
  (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0
```

Compressed Sparse Row Format

대각 행렬의 위치

Coordinate Format

```
In [5]: data = np.ones(4)
        row_indices = np.arange(4)
        col_indices = np.arange(4)
        eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
        print("COO 표현:\n{}".format(eye_coo))

COO 표현:
  (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0
```

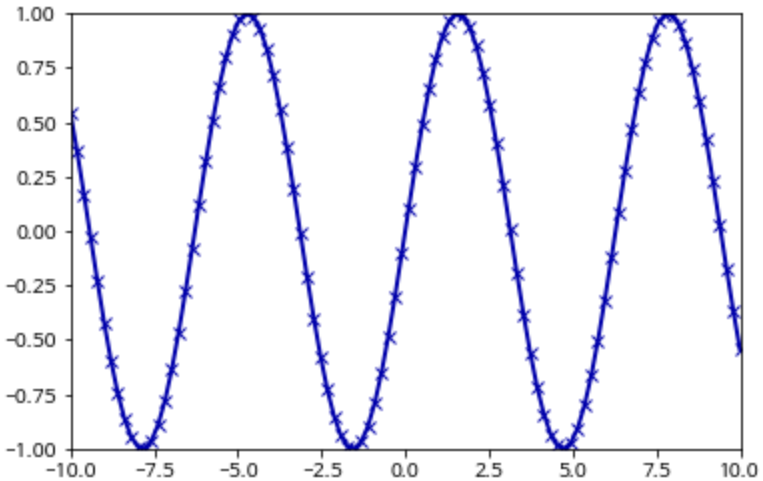
Matplotlib

- matplotlib(https://matplotlib.org/)은 파이썬의 대표적인 과학 계산용 그래프 라이브러리입니다. 선 그래프, 히스토그램, 산점도 등을 지원하며 출판에 쓸 수 있을 만큼의 고품질 그래프를 그려줍니다. 데이터와 분석 결과를 다양한 관점에서 시각화 해보면 매우 중요한 통찰을 얻을 수 있습니다.
- 주피터 노트북에서 사용할 때는 %matplotlib notebook이나 %matplotlib inline 명령을 사용하면 브라우저에서 바로 이미지를 볼 수 있습니다.

  대화식 환경을 제공하는 %matplotlib notebook 명령을 권장합니다.

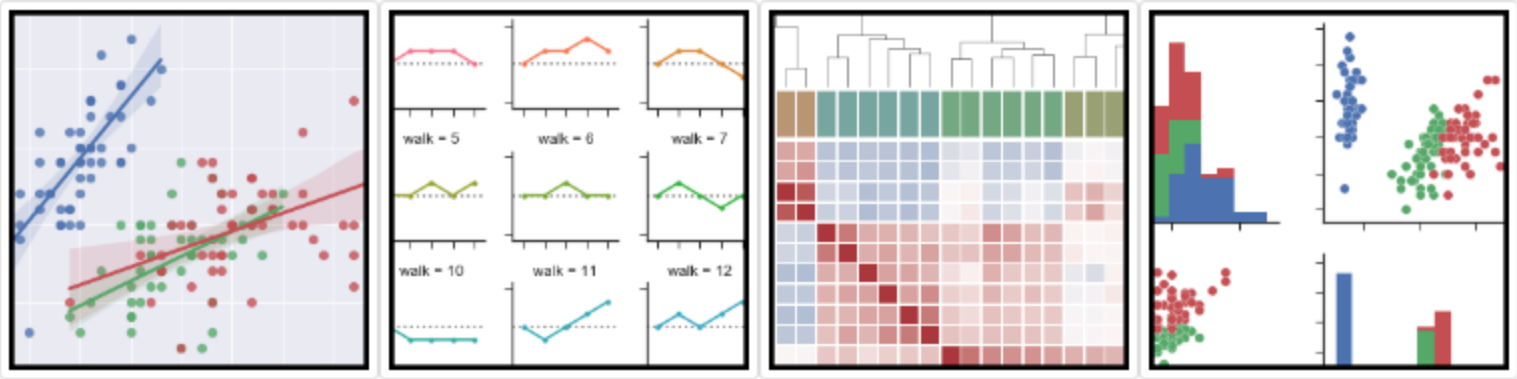
Matplotlib

```
%matplotlib inline
import matplotlib.pyplot as plt
# -10에서 10까지 100개의 간격으로 나뉘어진 배열을 생성합니다.
x = np.linspace(-10, 10, 100)
# 사인(sin) 함수를 사용하여 y 배열을 생성합니다.
y = np.sin(x)
# 플롯(plot) 함수는 한 배열의 값을 다른 배열에 대응해서 선 그래프를 그립니다.
plt.plot(x, y, marker="x")
```



Seaborn

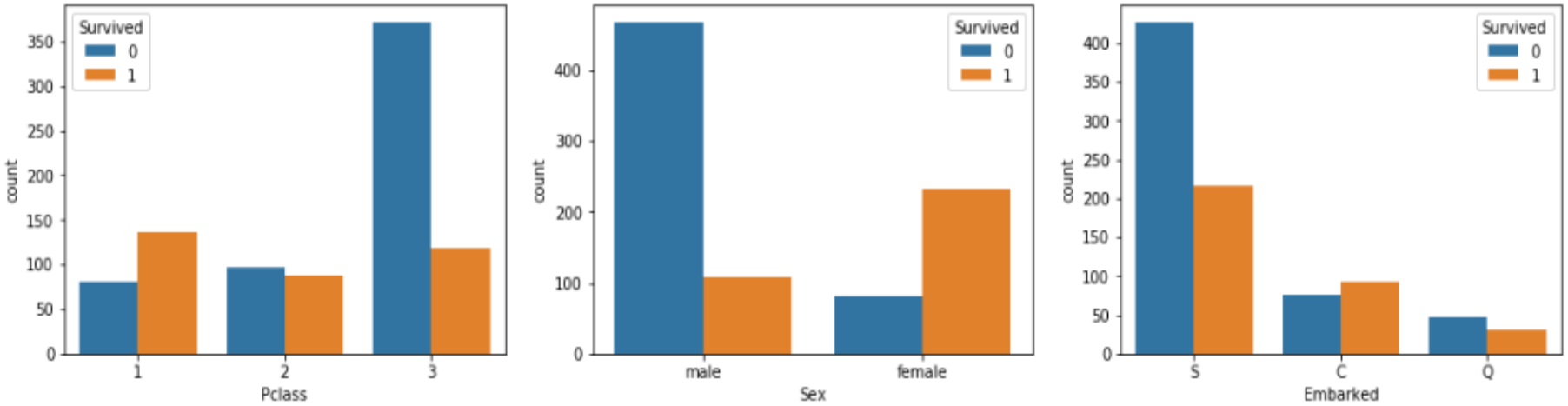
- Seaborn(https://seaborn.github.io/)은 Matplotlib을 기반으로 다양한 색상 테마와 통계용 챠트 등의 기능을 추가한 시각화 패키지이다. 기본적인 시각화 기능은 Matplotlib 패키지에 의존하며 통계 기능은 Statsmodels 패키지에 의존한다.
- Seaborn을 import 하면 색상 등을 Matplotlib에서 제공하는 기본 스타일이 아닌 Seaborn에서 지정한 기본 스타일로 바꾼다. 따라서 동일한 Matplotlib 명령을 수행해도 Seaborn을 import 한 것과 하지 않은 플롯은 모양이 다르다.



Seaborn

```
%matplotlib inline
import seaborn as sns

figure, (ax1, ax2, ax3) = plt.subplots(nrows=1,ncols=3)
figure.set_size_inches(18,4)
sns.countplot(data=train,x="Pclass",hue="Survived",ax=ax1)
sns.countplot(data=train,x="Sex",hue="Survived",ax=ax2)
sns.countplot(data=train,x="Embarked",hue="Survived",ax=ax3)
```



Pandas

- pandas(http://pandas.pydata.org/)는 데이터 처리와 분석을 위한 파이썬 라이브러리입니다. R의 data.frame을 본떠서 설계한 DataFrame이라는 데이터 구조를 기반으로 만들어졌습니다. 간단하게 말하면 pandas의 DataFrame은 엑셀의 스프레드시트와 비슷한 테이블 형태라고 할 수 있습니다. pandas는 이 테이블을 수정하고 조작하는 다양한 기능을 제공합니다. 특히, SQL처럼 테이블에 쿼리나 조인을 수행할 수 있습니다. 전체 배열의 원소가 동일한 타입이어야 하는 NumPy와는 달리 pandas는 각 열의 타입이 달라도 됩니다(예를 들면 정수, 날짜, 부동소숫점, 문자열). SQL, 엑셀 파일, CSV 파일 같은 다양한 파일과 데이터베이스에서 데이터를 읽어 들일 수 있는 것이 pandas가 제공하는 또 하나의 유용한 기능입니다.

Pandas

```
import pandas as pd
# 회원 정보가 들어간 간단한 데이터셋을 생성합니다.
data = {'Name': ["John", "Anna", "Peter", "Linda"], 'Location' : ["New York",
    "Paris", "Berlin", "London"], 'Age' : [24, 13, 53, 33] }
data_pandas = pd.DataFrame(data)
# IPython.display는 주피터 노트북에서 Dataframe을 출력해줍니다.
display(data_pandas)
```

	Age	Location	Name
0	24	New York	John
1	13	Paris	Anna
2	53	Berlin	Peter
3	33	London	Linda

Scikit-Learn

- scikit-learn은 2007년 구글 썸머 코드에서 처음 구현 됐으며 현재 파이썬으로 구현된 가장 유명한 기계 학습 오픈 소스 라이브러리다. scikit-learn의 장점은 라이브러리 외적으로는 scikit 스택을 사용하고 있기 때문에 다른 라이브러리와의 호환성이 좋다. 내적으로는 통일된 인터페이스를 가지고 있기 때문에 매우 간단하게 여러 기법을 적용할 수 있어 쉽고 빠르게 최상의 결과를 얻을 수 있다.

- 라이브러리의 구성은 크게 지도 학습, 비지도 학습, 모델 선택 및 평가, 데이터 변환으로 나눌 수 있다 (scikit-learn 사용자 가이드 참조, http://scikit-learn.org/stable/user_guide.html).

3. Python 데이터 분석 기초 라이브러리 – numpy 사용하기

numpy 개요

Numpy 란?

Numpy는 특히 벡터 및 행렬 연산에 있어 편의성을 제공하는 라이브러리로, 앞으로 많이 사용하게 될 Pandas와 Matplotlib의 기반이 되는 라이브러리이다.

Numpy는 기본적으로 **array(어레이)**라는 단위로 데이터를 관리하고, 이에 대한 연산을 수행함.
이 때 array는, 고등 수학에서 말하는 '행렬(matrix)'과 그 성격이 거의 유사함.

numpy array

- numpy import 하기

```
import numpy as np
```

- array 정의하기

```
arr = np.array([1,2,3,4])
```

이 때 Python 리스트 혹은 기존에 정의된 다른 array가 함수의 인자로 입력됩니다.

array를 생성할 때 인자로 입력되는 Python list는 1차원, 2차원 이상의 차원이 될 수 있음.

- array 타입과 크기 확인

```
arr.dtype
arr.shape
```

- array를 정의하는 특이한 함수들

```
np.zeros(),  np.ones(),  np.eye(), np.arange()
```

numpy array

- array의 데이터형

```
arr = np.array(data, dtype=np.float64)
```

array를 정의할 때 사용자가 데이터 형을 따로 명시하지 않은 경우, numpy에서는 가장 적절한 데이터 형을 자동으로 인식하여 해당 array에 부여함

array를 정의할 때 데이터 형을 직접 지정할 경우, dtype 인자의 값을 명시하면 됨

numpy array

- numpy에서 사용 가능한 데이터 형(dtype) 정리

데이터형	데이터형 코드	설명
int8, int16, int32, int64	i1, i2, i4, i8	부호가 있는 [8, 16, 32, 64]비트 정수
uint8, uint16, uint32, uint64	u1, u2, u4, u8	부호가 없는 [8, 16, 32, 64]비트 정수
float16, float32, float64, float128	f2, f4, f8, f16	[16, 32, 64, 128]비트 실수
complex64, complex128, complex256	c8, c16, c32	[64, 128, 256]비트 복소수
bool	b	불리언 (True 또는 False)
object	O	Python 오브젝트 형
string_	S	문자열
unicode_	U	유니코드 문자열

int나 float 뒤의 숫자는, 메모리 상에서 몇 비트를 사용하는지를 나타내는 숫자.

데이터 형을 간단하게 나타내기 위해, numpy에서는 'i', 'u', 'f', 'c' 등과 같이 축약된 코드를 사용

데이터 형을 변환할 수도 있는데, 이 때는 **.astype()** 함수를 사용함

numpy array

- Array의 연산

  두 array 간에는 더하기, 빼기, 곱하기, 나누기 등의 연산을 수행 가능

  '+', '-', '*', '/' 등과 같이 여러분이 일반 숫자에 대해 사용하는 연산자 사용 가능

  모든 연산은 두 array 상의 동일한 위치의 성분끼리 이루어지게 됩니다.

  그러므로, 두 array의 모양이 같아야 계산이 가능합니다.

- Array 형태 변형하기

```
sap = np.array(['MMM','ABT','ABBT','ACN','ACE','ATVI','ADBE','ADT'])

#대상 배열의 모양을 바꾼다. reshape() 함수의 파라미터로 배열의 새차원을 정의할 수 있다.
sap2d = sap.reshape(2,4)

#2차원 배열에서는 행과 열이 뒤바뀐다.
sap2d.T
```

numpy array 인덱싱

기본 인덱싱

numpy array의 핵심은 인덱싱(indexing), 기본적인 인덱싱은 Python 리스트와 매우 유사함

```
arr[5]      #특정 인덱스 명시
arr[5:8]    #범위 형태의 인덱스
arr[:]      #전체 성분을 모두 선택
```

```
import numpy as np
```

```
arr = np.arange(10)
```

```
arr
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
arr[5]
```

```
5
```

```
arr[5:8]
```

```
array([5, 6, 7])
```

```
arr[:]
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

numpy array 인덱싱

- 2차원 array의 인덱싱

```
arr2d[2,:]      #인덱스가 2에 해당하는 3행이 1차원 array
arr2d[:,3]      #모든 행의 4열에 해당하는 1차원 array
arr2d[1:3,:]    #2행과 3행의 모든 열에 해당하는 2차원 array
arr2d[3,2]      #4행 3열에 위치한 하나의 엘리먼트
```

```
arr2d
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

```
arr2d[3,2]
```

```
15
```

```
arr2d[2,:]
```

```
array([ 9, 10, 11, 12])
```

```
arr2d[1:3,:]
```

```
array([[ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

```
arr2d[:,3]
```

```
array([ 4,  8, 12, 16])
```

```
arr2d[:,:2]
```

```
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10],
       [13, 14]])
```

numpy array 인덱싱

- boolean 인덱싱

```
names = np.array(["Charles", "soyul", "Hayoung", "Charles", "Hayoung", "soyul", "soyul"])
data = np.array([[ 0.57587275, -2.84040808, 0.70568712, -0.1836896 ],
                 [-0.59389702, -1.35370379, 2.28127544, 0.03784684],
                 [-0.28854954, 0.8904534 , 0.18153112, 0.95281901],
                 [ 0.75912188, -1.88118767, -2.37445741, -0.5908499 ],
                 [ 1.7403012 , 1.33138843, 1.20897442, -0.58004389],
                 [ 1.11585923, 1.02466538, -0.74409379, -1.55236176],
                 [-0.45921447, 2.53114818, 0.5029578 , -0.24088216]])
```

names array 내 각각의 성분이, **data** array의 각 행에 순서대로 대응된다고 가정합시다.
이름이 "Charles"인 사람의 행 데이터만을 추출하고 싶다면

```
names == "Charles"
>>> array([ True, False, False, True, False, False, False], dtype=bool)
```

names == "Charles"와 같은 조건식 형태의 코드를 실행하였을 때 생성되는 불리언 array를 다른 말로 '**마스크(mask)**' 마스크는 다른 array를 인덱싱 하는 데 사용할 수 있음

numpy array boolean 인덱싱

- boolean 인덱싱

```
data[names == "Charles", :]
data = np.array([[ 0.57587275, -2.84040808, 0.70568712, -0.1836896 ],
                 [ 0.75912188, -1.88118767, -2.37445741, -0.5908499 ]])
```

data[:, 3]을 실행하여 4열만의 값 중에서 data[:, 3] < 0 을 실행하여 0보다 작은 엘리먼트를 가져옴

```
data[:, 3] < 0
>>> array([ True, False, False, True, True, True, True], dtype=bool)
```

data array의 4열의 값이 0보다 작은 행(row)에 대해서는 모든 값을 0으로 대입하도록 함

```
data[data[:, 3] < 0, :] = 0
>>> array([ True, False, False, True, True, True, True], dtype=bool)
```

numpy array 관련 함수

- 한 개의 array에 적용되는 함수

  numpy에서 한 개의 array의 각 엘리먼트에 적용되는 함수들

함수	설명
abs	각 성분의 절댓값 계산
sqrt	각 성분의 제곱근 계산 (array ** 0.5 의 결과와 동일)
square	각 성분의 제곱 계산 (array ** 2 의 결과와 동일)
exp	각 성분을 무리수 e의 지수로 삼은 값을 계산
log, log10, log2	자연로그(밑이 e), 상용로그(밑이 10), 밑이 2인 로그를 계산
sign	각 성분의 부호 계산 (+인 경우 1, -인 경우 -1, 0인 경우 0)
ceil	각 성분의 소수 첫 번째 자리에서 올림한 값을 계산
floor	각 성분의 소수 첫 번째 자리에서 내림한 값을 계산
isnan	각 성분이 NaN(Not a Number)인 경우 True를, 그렇지 않은 경우 False를 반환
isinf	각 성분이 무한대(infinity)인 경우 True를, 그렇지 않은 경우 False를 반환
cos, cosh, sin, sinh, tan, tanh	각 성분에 대한 삼각함수 값을 계산

numpy array 관련 함수

■ 두 개의 array에 적용되는 함수

크기가 같은 두 개의 array **x**, **y**가 아래와 같이 주어졌을 때,

np.maximum() 함수를 적용하면 두 array를 동일한 위치의 성분끼리 비교하여 값이 더 큰 성분을 선택, 새로운 array의 형태로 제시함.

```
x = np.array([ 0.97121145, 1.74277758, 0.17706708, -1.14078851,
               1.02197222, -0.75747493, 0.4994057 , -0.03462392])
y = np.array([ 0.91984849, 1.98745872, -0.11232596, 1.47306221,
               1.24527437, -0.77047603, 0.30708743, -1.76476678])
np.maximum(x, y)
>>> array([ 0.97121145, 1.98745872, 0.17706708, 1.47306221, 1.24527437,
-0.75747493, 0.4994057 , -0.03462392])
```

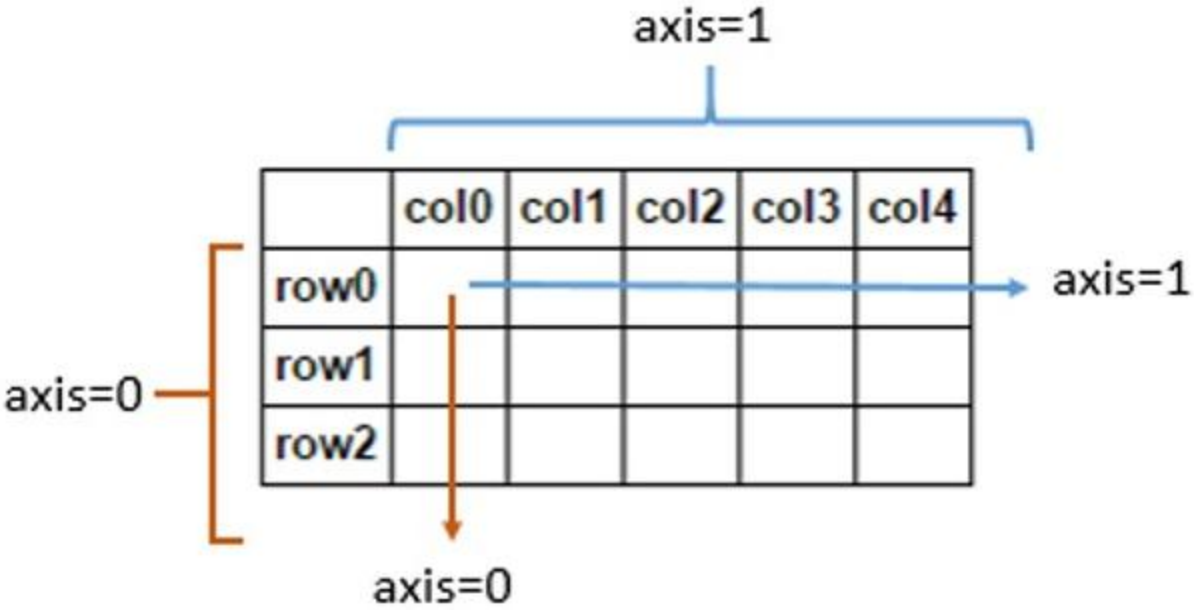
numpy array 관련 함수

- 두 개의 array에 적용되는 함수

함수	설명
add	두 array에서 동일한 위치의 성분끼리 더한 값을 계산 (arr1 + arr2의 결과와 동일)
subtract	두 array에서 동일한 위치의 성분끼리 뺀 값을 계산 (arr1 - arr2의 결과와 동일)
multiply	두 array에서 동일한 위치의 성분끼리 곱한 값을 계산 (arr1 * arr2의 결과와 동일)
divide	두 array에서 동일한 위치의 성분끼리 나눈 값을 계산 (arr1 / arr2의 결과와 동일)
maximum	두 array에서 동일한 위치의 성분끼리 비교하여 둘 중 최댓값을 반환
minimum	두 array에서 동일한 위치의 성분끼리 비교하여 둘 중 최솟값을 반환

numpy array 관련 함수

- axis=0 , axis=1 개념



numpy array 관련 함수

array에 적용되는 통계 함수

array의 합이나 평균을 계산할 때, 전체 성분 대신 '행 방향' 혹은 '열 방향'의 엘리먼트들 만 계산 가능

2차원 array에 대하여 **.sum(axis=0)**을 실행하면 각 열(col) 끼리의 합을 계산한 결과를 산출함.

반면 **.sum(axis=1)**을 실행하면 각 행(row)의 합을 계산한 결과를 산출함.

```
arr
```

```
array([[ 1.2351262 , -1.04245575,  1.11056994, -0.3262982 ],
       [ 0.73277381,  1.19512847, -0.60614721,  1.49860443],
       [ 0.35368461, -1.11190392,  0.18653943,  1.39460884],
       [-0.63825291,  2.09479354,  0.53836684, -1.15086368],
       [ 1.38042606, -0.34213266,  1.55187791, -1.48757167]])
```

```
#각 열(column) 별로 합계 구하기
arr.sum(axis=0)
```

```
array([ 3.06375777,  0.79342968,  2.78120692, -0.07152028])
```

```
#각 행(row) 별로 합계 구하기
arr.sum(axis=1)
```

```
array([ 0.97694219,  2.8203595 ,  0.82292897,  0.84404379,  1.10259964])
```

numpy array 관련 함수

■ array에 적용되는 통계 함수

통계 함수들은, 특정 조건을 만족하는 array 내 성분의 개수가 몇 개인지 셀 때도 유용하게 사용됨.

arr array에 대하여 arr > 0을 실행한 마스크를 얻으면, 해당 마스크에 대하여 (arr > 0).sum()을 실행 하여 마스크 내 True 값의 개수를 계산함으로써,

결과적으로 arr > 0 이라는 조건을 만족하는 성분의 합계를 산출할 수 도 있습니다.

함수	설명
sum	전체 성분의 합을 계산
mean	전체 성분의 평균을 계산
std, var	전체 성분의 표준편차, 분산을 계산
min, max	전체 성분의 최솟값, 최댓값을 계산
argmin, argmax	전체 성분의 최솟값, 최댓값이 위치한 인덱스를 반환
cumsum	맨 첫번째 성분부터 각 성분까지의 누적합을 계산 (0에서부터 계속 더해짐)
cumprod	맨 첫번째 성분부터 각 성분까지의 누적곱을 계산 (1에서부터 계속 곱해짐)

numpy array 관련 함수

- 1차원 array의 정렬

```
arr
```

```
array([-1.36306611,  0.81079459,  0.4270222 ,  1.85962427,  1.10513006,
        0.23784655, -1.85257902, -0.31059168])
```

```
#오름차순
np.sort(arr)
```

```
array([-1.85257902, -1.36306611, -0.31059168,  0.23784655,  0.4270222 ,
        0.81079459,  1.10513006,  1.85962427])
```

내림차순으로 정렬하고 싶다면, np.sort(arr)[::-1]을 실행하면 됨

```
#내림차순
np.sort(arr)[::-1]
```

```
array([ 1.85962427,  1.10513006,  0.81079459,  0.4270222 ,  0.23784655,
       -0.31059168, -1.36306611, -1.85257902])
```

numpy array 관련 함수

- 2차원 array의 정렬

```
arr
```

```
array([[-1.7166257 ,  1.78325628,  0.07492439],
       [-0.31123612,  0.36710066,  1.65704336],
       [-0.89232863,  0.91444772,  0.17163545],
       [-0.67307227,  0.07507323, -0.58988111],
       [ 0.62765514, -1.21787417, -0.24966279]])
```

np.sort(arr, axis=0)을 실행하면 '행 방향'으로 오름차순 정렬이 이루어지고,

axis 인자의 값을 1로 바꾸면 '열 방향'으로 오름차순 정렬이 이루어짐

```
#행(row) 방향으로 sort 하기
np.sort(arr,axis=0)
```

```
array([[-1.7166257 , -1.21787417, -0.58988111],
       [-0.89232863,  0.07507323, -0.24966279],
       [-0.67307227,  0.36710066,  0.07492439],
       [-0.31123612,  0.91444772,  0.17163545],
       [ 0.62765514,  1.78325628,  1.65704336]])
```

```
#열(column) 방향으로 sort 하기
np.sort(arr,axis=1)
```

```
array([[-1.7166257 ,  0.07492439,  1.78325628],
       [-0.31123612,  0.36710066,  1.65704336],
       [-0.89232863,  0.17163545,  0.91444772],
       [-0.67307227, -0.58988111,  0.07507323],
       [-1.21787417, -0.24966279,  0.62765514]])
```

numpy array 관련 함수 사용하기

- array 상에서 상위 5%에 위치하는 값 추출하기

large_arr을 먼저 내림차순으로 정렬한 뒤, **int(0.05 * len(large_arr))**을 통해 상위 5%에 해당하는 인덱스를 계산한 후, 이를 사용하여 정렬 결과에 대한 인덱싱을 수행하는 과정

```
np.sort(large_arr)[::-1][int(0.05 * len(large_arr))]
```

- array 상에서 중복된 값을 제외한 unique한 값 추출하기

중복된 성분을 포함하고 있는 array에 대하여, **np.unique()** 함수를 사용하면 중복된 값을 제외한 유니크한 값만을 추출할 수 있음.

```
names = np.array(["Charles","Soyul","Hayoung","Charles","Hayoung","Soyul","Soyul"])
ints = np.array([3,3,3,2,2,1,1,4,4])
```

```
#중복된 값을 제거함
np.unique(names)
```

```
array(['Charles', 'Hayoung', 'Soyul'],
      dtype='<U7')
```

```
np.unique(ints)
```

```
array([1, 2, 3, 4])
```

numpy를 사용한 데이터 분석 맛보기: MovieLens 1M 데이터셋 분석

- 파일 읽기

MovieLens 1M dataset의 경우, 콜론이 두 개 붙은 문자 '::'가 구분자임

해당 함수를 호출할 때, 데이터셋 파일의 경로, 사용할 구분자, array의 데이터형이 인자로 입력됨

```
data = np.loadtxt("data/movielens-1m/ratings.dat", delimiter="::", dtype=np.int64)
```

- 데이터 레이아웃

```
#UserID::MovieID::Rating::Timestamp
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
1::3408::4::978300275
1::2355::5::978824291
1::1197::3::978302268
1::1287::5::978302039
```

numpy를 사용한 데이터 분석 맛보기: MovieLens 1M 데이터셋 분석

- 각 사용자별 평점(rating)의 평균 구하기

```
user_ids = np.unique(data[:,0]) #unique한 UserID 가져오기
mean_rating_by_user_list = []
for user_id in user_ids:
    data_for_user = data[data[:,0] == user_id,:] #data[:,0] == user_id 인 row만 추출
    mean_rating_for_user = data_for_user[:,2].mean()  #평점의 평균을 구하여 저장
    mean_rating_by_user_list.append([user_id,mean_rating_for_user]) #userid와평균 저장
```

- List를 numpy array로 변환

```
mean_rating_by_user_array = np.array(mean_rating_by_user_list,dtype=np.float32)
```

- 파일 쓰기

np.savetxt() 함수를 사용하면, 현재 array 형태의 분석 결과물을 외부 파일로 쓸 수 있음

해당 함수를 호출할 때, 저장할 파일의 경로, 저장할 array, 저장 형태(format), 구분자 등을 인자로 입력됨

```
np.savetxt("mean_rating_by_user.csv", mean_rating_by_user_array, fmt='%.3f',
           delimiter=',')
```

4. 쉽고 강력한 데이터 분석 라이브러리
- pandas 사용하기

pandas의 고유한 자료구조 - Series와 DataFrame 이해하기

- Pandas 란?

pandas에서 정의한 자료 구조인 Series와 DataFrame은 빅 데이터 분석의 높은 성능을 발휘함.
Series는 동일한 데이터 형의 복수 개의 성분으로 구성된 자료 구조이며,
DataFrame은 서로 같거나 다른 데이터 형의 여러 개의 열에 대하여 복수 개의 성분으로 구성된 '표(table)와 같은 형태'의 자료 구조이다.

```
import pandas as pd
```

http://pandas.pydata.org/

pandas의 고유한 자료구조 - Series

Series

Series는 **pd.Series()** 함수를 사용하여 정의함.

Python list와 numpy array가 이 함수의 인자로 입력됨.

Series는 각 성분의 인덱스와, 이에 대응되는 값으로 구성되어 있음.

Series 생성 시 인덱스는 0으로 시작하는 정수 형태의 기본 인덱스가 부여됨.

기본 인덱스 대신 Series 생성 시 각 성분에 대한 인덱스를 사용자가 직접 명시할 수도 있음.

```
obj = pd.Series([4, 7, -5, 3])
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
```

Series **obj**의 인덱스만을 추출 : **obj.index** `RangeIndex(start=0, stop=4, step=1)`

Series **obj**의 값만을 추출 : **obj.values** `array([ 4,  7, -5,  3], dtype=int64)`

Series **obj**에 부여된 데이터형을 확인 : **obj.dtype** `dtype('int64')`

인덱스에 대한 이름을 지정 : **obj.name**과 **obj.index.name**에 값을 대입해 줌

pandas의 고유한 자료구조 - DataFrame

DataFrame

DataFrame은 **pd.DataFrame()** 함수를 사용하여 정의함.

Python 딕셔너리 혹은 numpy의 2차원 array가 이 함수의 인자로 입력됨.

```
data = {"names": ["soyul", "soyul", "soyul", "Charles", "Charles"],
        "year": [2014, 2015, 2016, 2015, 2016],
        "points": [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
```

DataFrame에서는 서로 다른 두 종류의 인덱스가 각각 행 방향과 열 방향에 부여되어 있으며, 교차하는 지점에 실제 값이 위치해 있음.

df

	names	points	year
0	Soyul	1.5	2014
1	Soyul	1.7	2015
2	Soyul	3.6	2016
3	Charales	2.4	2015
4	Charles	2.9	2016

pandas의 고유한 자료구조 - DataFrame

- **DataFrame**

행 방향의 인덱스를 '인덱스', 열 방향의 인덱스를 '컬럼'이라고 부름.

DataFrame의 인덱스 확인 : **df.index**

```
RangeIndex(start=0, stop=5, step=1)
```

DataFrame의 컬럼명 확인 : **df.columns**

```
Index(['names', 'points', 'year'], dtype='object')
```

DataFrame의 값 확인 : **df.values**

```
array([['Soyul', 1.5, 2014],
       ['Soyul', 1.7, 2015],
       ['Soyul', 3.6, 2016],
       ['Charales', 2.4, 2015],
       ['Charles', 2.9, 2016]], dtype=object)
```

DataFrame의 행,열 갯수 확인 : **df.shape**

pandas의 고유한 자료구조 - DataFrame

- **DataFrame 관련 함수와 변수 조회**

  IPython Notebook에서 **df.** 까지만 입력하고 **TAB** 키를 누르면 사용 가능한 모든 변수 및 함수가 표시됨

- **NaN (not a number)**

  NaN은 numpy나 python에서 'not a number'를 표시하는 약어로, 특정 위치에 값이 존재하지 않을 때 이를 표시하기 위한 기호이다.

  여러분이 데이터셋 파일을 DataFrame의 형태로 읽어 들이는 과정에서, 만약 특정 부분의 값이 포함되어 있지 않았을 경우 해당 위치에 NaN으로 표시된다.

- **describe 함수**

  df.describe() 함수를 실행하게 되면, 각 컬럼의 평균, 분산, 최소값/최대값 등 기본 통계량을 산출한 결과를 보여줍니다.

  데이터셋을 DataFrame 형태로 읽고 나서, 데이터셋을 전체적으로 살펴보고자 할 때 사용하기 유용함

DataFrame의 인덱싱

- **DataFrame의 기본 인덱싱 – 열(column) 선택하고 조작하기**

  df라는 이름의 DataFrame을 다음과 같이 정의합니다.

```
data = {"names": ["soyul", "soyul", "soyul", "Charles", "Charles"],
        "year": [2014, 2015, 2016, 2015, 2016],
      "points": [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data, columns=["year", "names", "points", "penalty"],
                        index=["one", "two", "three", "four", "five"])
```

df

	year	names	points	penalty
one	2014	soyul	1.5	NaN
two	2015	soyul	1.7	NaN
three	2016	soyul	3.6	NaN
four	2015	Charles	2.4	NaN
five	2016	Charles	2.9	NaN

DataFrame의 인덱싱

- **DataFrame의 기본 인덱싱 – 열(column) 선택하고 조작하기**

df에서 'year' 열 만을 가져올 경우 ,

'year' 열의 값들이 Series 형태로 인덱스와 함께 표시됨

```
df["year"]
```

```
one      2014
two      2015
three    2016
four     2015
five     2016
Name: year, dtype: int64
```

df에서 복수개의 열을 가져올 경우 ,

중괄호 안에 컬럼 이름으로 구성된 리스트가 들어감

```
df[["year", "points"]]
```

	year	points
one	2014	1.5
two	2015	1.7
three	2016	3.6
four	2015	2.4
five	2016	2.9

특정 열을 이렇게 선택한 뒤 값을 대입하면,

NaN 표시된 'penalty'열에 값을 지정함

```
df["penalty"] = [0.1, 0.2, 0.3, 0.4, 0.5]
```

DataFrame의 인덱싱

- **DataFrame의 기본 인덱싱 – 열 선택,조작**

df에 'zeros' 라는 열을 추가하는 경우

```
df["zeros"] = np.arange(5)
```

	year	names	points	penalty	zeros
one	2014	soyul	1.5	0.1	0
two	2015	soyul	1.7	0.2	1
three	2016	soyul	3.6	0.3	2
four	2015	Charles	2.4	0.4	3
five	2016	Charles	2.9	0.5	4

df에 새로운 열을 추가할 때 Series 형태로 추가할 수 있는데, 해당 Series에 명시된 인덱스와 DataFrame의 인덱스의 값을 비교하여 대응되는 인덱스의 값을 대입하는 형태로 이루어짐

```
val = pd.Series([-1.2, -1.5, -1.7],
       index=["two", "four","five"])
df["debt"] = val
```

	year	names	points	penalty	zeros	debt
one	2014	soyul	1.5	0.1	0	NaN
two	2015	soyul	1.7	0.2	1	-1.2
three	2016	soyul	3.6	0.3	2	NaN
four	2015	Charles	2.4	0.4	3	-1.5
five	2016	Charles	2.9	0.5	4	-1.7

DataFrame의 인덱싱

- **DataFrame의 기본 인덱싱 – 열(column) 선택하고 조작하기**

  기존의 열의 값을 사용하여 새로운 열의 값을 입력할 수도 있음

```
df["net_points"]  = df["points"] - df["penalty"]
df["high_points"] = df["net_points"] > 2.0
```

	year	names	points	penalty	zeros	debt	net_points	high_points
one	2014	soyul	1.5	0.1	0	NaN	1.4	False
two	2015	soyul	1.7	0.2	1	-1.2	1.5	False
three	2016	soyul	3.6	0.3	2	NaN	3.3	True
four	2015	Charles	2.4	0.4	3	-1.5	2.0	False
five	2016	Charles	2.9	0.5	4	-1.7	2.4	True

기존의 열을 DataFrame에서 삭제

```
del df["high_points"]
del df["net_points"]
```

DataFrame의 인덱싱

- **DataFrame의 기본 인덱싱 – 행(row) 선택하고 조작하기**

DataFrame의 행을 선택하고자 할 때, **.loc이나 .iloc**을 사용하는 것을 권장함

- **df.loc[]**

.loc은 실제 인덱스를 사용하여 행(row)를 가져올 때 사용함

인덱스가 'two'인 행의 값을 가져오거나, 인덱스가 'two' 부터 'four' 범위에 있는 모든 행의 값을 가져옴

```
df.loc["two"]
df.loc["two":"four"]
```

```
df.loc["two"]
```

```
Info
year        2015
names       soyul
points        1.7
penalty       0.2
debt         -1.2
Name: two, dtype: object
```

```
df.loc["two":"four"]
```

Info	year	names	points	penalty	debt
Order					
two	2015	soyul	1.7	0.2	-1.2
three	2016	soyul	3.6	0.3	NaN
four	2015	Charles	2.4	0.4	-1.5

DataFrame의 인덱싱

- **DataFrame의 기본 인덱싱 – 행(row) 선택하고 조작하기**
- **df.loc[]**

'two' 부터 'four'까지 인덱스의 값들 중 'points' 열의 값만을 가져오거나,
인덱스와 칼럼 모두 범위 인덱싱을 주어 가져올 수 있음

```
df.loc["two":"four", "points"]
df.loc["three":"four", "year":"penalty"]
```

```
df.loc["two":"four","points"]
```

```
Order
two      1.7
three    3.6
four     2.4
Name: points, dtype: float64
```

```
df.loc["three":"five","year":"penalty"]
```

	year	names	points	penalty
three	2016	soyul	3.6	0.3
four	2015	Charles	2.4	0.4
five	2016	Charles	2.9	0.5

DataFrame의 인덱싱

- **DataFrame의 기본 인덱싱 – 행(row) 선택하고 조작하기**
- **df.iloc[]**

.iloc은 numpy의 array 인덱싱 방식으로 행(row)를 가져올 때 사용함

인덱스가 3에 해당하는 4행을 모두 가져오거나,

행과 열에 대한 범위 인덱싱으로 가져올 수 있음

```
df.iloc[3]
df.iloc[3:5,0:2]
```

원하는 인덱스만을 명시할 수도 있음

```
df.iloc[[0,1,3],[1,2]]
```

```
df.iloc[3]
```

```
year          2015
names      Charles
points         2.4
penalty        0.4
zeros            3
debt          -1.5
net_points       2
high_points  False
Name: four, dtype: object
```

```
df.iloc[3:5,0:2]
```

Info	year	names
Order		
four	2015.0	Charles
five	2016.0	Charles

```
df.iloc[[0,1,3],[1,2]]
```

Info	names	points
Order		
one	soyul	1.5
two	soyul	1.7
four	Charles	2.4

DataFrame의 인덱싱

- **DataFrame의 boolean 인덱싱 – 열(column)이나 행(row)을 선택하고 조작하기**

  df["year"] > 2014 를 실행하면, 값이 2014보다 큰 위치에는 True, 그 이외에는 False가 들어간 boolean mask를 얻게 됩니다.

- **Mask**

  mask는 DataFrame을 인덱싱 하는 데 사용될 수 있음.

  'names'열의 값이 "soyul"인 행의 값 중에서

  "names"와 "points" 열에 해당하는 값들을 가져옴

```
df.loc[df["names"] == "soyul", ["names","points"]]
```

사용할 조건이 여러 개 필요한 경우 '&' 나 '|' 연산자를 사용함

```
df.loc[(df["points"] > 2) & (df["points"] < 3), :]
```

```
df.loc[df["names"] == "soyul",["names","points"]]
```

Info	names	points
Order		
one	soyul	1.5
two	soyul	1.7
three	soyul	3.6

```
df.loc[(df["points"] > 2) & (df["points"] < 3),:]
```

Info	year	names	points	penalty	debt
Order					
four	2015.0	Charles	2.4	0.4	-1.5
five	2016.0	Charles	2.9	0.5	NaN

DataFrame의 NaN 처리하기

- **데이터의 결측값 또는 이상치 처리하기**

pandas를 사용하여 읽어 들인 데이터셋 파일에 NaN의 형태의 값이 존재하거나, 정상 범주에서 벗어난 값이 포함된 경우가 있는데, 이러한 값들을 각각 **결측값(missing value)** 및 **이상치(outlier)**라고 함. 데이터 분석에 앞서 이들을 처리할 필요가 있음.

- **datetime 타입의 인덱스를 가지는 새로운 DataFrame 생성**

랜덤한 숫자로 구성된 DataFrame **df**를 정의하고, 인덱스와 컬럼을 명시함

pd.date_range()는 일자와 시각을 나타내는 데이터 형인 **datetime**으로 인덱스를 생성할 수 있는 함수

```
df = pd.DataFrame(np.random.randn(6, 4))
df.columns = ["A", "B", "C", "D"]
df.index = pd.date_range("20160701", periods=6)
```

	A	B	C	D
2016-07-01	-0.480360	-1.521118	0.580715	-2.882923
2016-07-02	0.391702	-0.099156	-1.255430	1.265961
2016-07-03	-0.540837	0.412312	-0.300376	-0.246781
2016-07-04	-2.806605	0.811436	1.094936	-0.601455
2016-07-05	1.331190	-0.657335	1.500431	-1.019643
2016-07-06	-0.381102	-0.011507	0.098860	2.711896

DataFrame의 NaN 처리하기

NaN 처리하기 – np.nan

NaN을 인위적으로 사용하려면

numpy에서 제공하는 np.nan을 사용합니다.

df에 'F' 열을 새로 정의하고, NaN 값을 추가함

```
df["F"] = [1.0, np.nan, 3.5, 6.1, np.nan, 7.0]
```

	A	B	C	D	F
2016-07-01	-0.480360	-1.521118	0.580715	-2.882923	1.0
2016-07-02	0.391702	-0.099156	-1.255430	1.265961	NaN
2016-07-03	-0.540837	0.412312	-0.300376	-0.246781	3.5
2016-07-04	-2.806605	0.811436	1.094936	-0.601455	6.1
2016-07-05	1.331190	-0.657335	1.500431	-1.019643	NaN
2016-07-06	-0.381102	-0.011507	0.098860	2.711896	7.0

NaN 값 삭제하기 – dropna(how="any")

행(row)의 값들 중에 NaN이 하나라도 포함되어 있는

경우 해당 행(row)을 DataFrame 상에서 삭제함

```
df.dropna(how="any")
```

	A	B	C	D	F
2016-07-01	-0.480360	-1.521118	0.580715	-2.882923	1.0
2016-07-03	-0.540837	0.412312	-0.300376	-0.246781	3.5
2016-07-04	-2.806605	0.811436	1.094936	-0.601455	6.1
2016-07-06	-0.381102	-0.011507	0.098860	2.711896	7.0

DataFrame의 NaN 처리하기

- **NaN 값 삭제하기 – dropna(how="all")**

  행(row)의 값들 모두가 NaN인 경우 해당 행(row) 을 DataFrame 상에서 삭제함

```
df.dropna(how="all")
```

- **NaN 값 대체하기 – fillna(value=0.5)**

  NaN을 실제 수치값으로 대체함.

```
df.fillna(value=0.5)
```

- **NaN 값 포함여부 체크 – isnull()**

  df.isnull() 함수는 DataFrame 상에서 NaN이 포함되어 있는 위치에 대한 불리언 마스크를 얻을 수 있음

  'F'열에 NaN을 포함하고 있는 행만을 선택함

```
df.loc[df.isnull()["F"], :]
```

	A	B	C	D	F
2016-07-01	-0.480360	-1.521118	0.580715	-2.882923	1.0
2016-07-02	0.391702	-0.099156	-1.255430	1.265961	0.5
2016-07-03	-0.540837	0.412312	-0.300376	-0.246781	3.5
2016-07-04	-2.806605	0.811436	1.094936	-0.601455	6.1
2016-07-05	1.331190	-0.657335	1.500431	-1.019643	0.5
2016-07-06	-0.381102	-0.011507	0.098860	2.711896	7.0

	A	B	C	D	F
2016-07-02	0.391702	-0.099156	-1.255430	1.265961	NaN
2016-07-05	1.331190	-0.657335	1.500431	-1.019643	NaN

DataFrame의 NaN 처리하기

- **NaN이 포함된 행(row)과 열(column) 삭제**

2016년 7월 1일 행 데이터를 선택하여 삭제함

두 개 이상의 인덱스를 기준으로 복수 개의 행을 삭제

```
df.drop(pd.to_datetime("20160701"))
df.drop([pd.to_datetime("20160702"),
         pd.to_datetime("20160704")])
```

	A	B	C	D	F
2016-07-01	-1.365534	-1.833548	-0.123941	-0.276072	1.0
2016-07-03	-1.890163	-0.852036	1.503335	0.000197	3.5
2016-07-05	-0.783819	0.536835	0.494487	1.013722	NaN
2016-07-06	-1.513752	-0.728070	0.993989	-0.629073	7.0

삭제할 'F' 열 컬럼을 정하고, **axis=1** 매개변수를 명시해 주면 삭제됨

```
df.drop("F", axis=1)
```

	A	B	C	D
2016-07-01	-1.365534	-1.833548	-0.123941	-0.276072
2016-07-03	-1.890163	-0.852036	1.503335	0.000197
2016-07-05	-0.783819	0.536835	0.494487	1.013722
2016-07-06	-1.513752	-0.728070	0.993989	-0.629073

DataFrame의 데이터 분석용 함수

통계함수

pandas의 DataFrame에서는 다양한 통계 함수를 지원하며, 데이터 분석을 할 때 유용하게 사용됨.

함수	설명
count	전체 성분의 (NaN이 아닌) 값의 갯수를 계산
min, max	전체 성분의 최솟, 최댓값을 계산
argmin, argmax	전체 성분의 최솟값, 최댓값이 위치한 (정수)인덱스를 반환
idxmin, idxmax	전체 인덱스 중 최솟값, 최댓값을 반환
quantile	전체 성분의 특정 사분위수에 해당하는 값을 반환 (0~1 사이)
sum	전체 성분의 합을 계산
mean	전체 성분의 평균을 계산
median	전체 성분의 중간값을 반환
mad	전체 성분의 평균값으로부터의 절대 편차(absolute deviation)의 평균을 계산
std, var	전체 성분의 표준편차, 분산을 계산
cumsum	맨 첫 번째 성분부터 각 성분까지의 누적합을 계산 (0에서부터 계속 더해짐)
cumprod	맨 첫번째 성분부터 각 성분까지의 누적곱을 계산 (1에서부터 계속 곱해짐)

DataFrame의 데이터 분석용 함수 – sum()

- **새로운 DataFrame 생성**

df라는 DataFrame을 정의하여, '행 방향' 또는 '열 방향'의 합을 구할 수 있음

```
data = [[1.4, np.nan], [7.1, -4.5],
        [np.nan, np.nan], [0.75, -1.3]]
df = pd.DataFrame(data,
                  columns=["one", "two"],
                  index=["a", "b", "c", "d"])
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

- **각 열(column)별의 합 (NaN 값은 skip 함)**

```
df.sum(axis=0)
```

```
one    9.25
two   -5.80
dtype: float64
```

- **각 행(row)별의 합 (NaN 값은 skip 함)**

```
df.sum(axis=1)
```

```
a    1.40
b    2.60
c     NaN
d   -0.55
dtype: float64
```

DataFrame의 데이터 분석용 함수 – sum(), mean()

- **특정 열(column)의 합**

'one' 열의 합만을 얻을 수 있음

```
df["one"].sum()
```

```
df["one"].sum()
9.25
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

- **특정 행(row)의 합**

'b' 행의 합 만을 얻을 수 있음

```
df.loc["b"].sum()
```

```
df.loc["b"].sum()
2.5999999999999996
```

- **행(row)별 평균 (NaN 값을 skip 하지 않음)**

```
df.mean(axis=1,skipna=False)
```

```
df.mean(axis=1,skipna=False)
a      NaN
b    1.300
c      NaN
d   -0.275
dtype: float64
```

DataFrame의 데이터 분석용 함수 – mean(), min()

- **통계함수를 사용하여 DataFrame의 결측값 채우기**

  'one' 열의 NaN은 나머지 값들의 평균값으로,

  'two' 열의 NaN은 해당 열의 최소값으로 대체함

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
one_mean = df.mean(axis=0)["one"]
df["one"] = df["one"].fillna(value=one_mean)
```

```
one_mean
```

```
3.0833333333333335
```

	one	two
a	1.400000	NaN
b	7.100000	-4.5
c	3.083333	NaN
d	0.750000	-1.3

```
two_min = df.min(axis=0)["two"]
df["two"] = df["two"].fillna(value=two_min)
```

```
two_min
```

```
-4.5
```

	one	two
a	1.400000	-4.5
b	7.100000	-4.5
c	3.083333	-4.5
d	0.750000	-1.3

DataFrame의 데이터 분석용 함수 – corr(), cov()

- **상관 계수(Correlation Coefficient)와 공분산(Covariance) 계산하기**

  데이터 분석 시 어느 한 변수가 다른 변수에 미치는 영향 등을 알고자 할 때는 상관 분석이 유용함

- **공분산(Covariance)**

  두 변수 중 X 변수가 변할 때, Y 변수가 얼마만큼 변하는지 그 양을 계산하려 할 때

  두 변수가 동시에 변하는 정도를 양(量)화 한 것이 공분산이다.

  공분산은 두 변수의 편차의 곱의 평균으로 계산한다.

Population Covariance = $Cov(X,Y) = \sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)]$

DataFrame의 데이터 분석용 함수 – corr(), cov()

- **상관계수란?**

두 변수가 어떠한 관계인지를 파악하는 분석이 상관계수 분석이다.

상관계수는 표준화된 공분산이다.

상관관계 수치는 보통 숫자 -1 부터 +1까지만 사용한다.

상관계수가 0에 가까울수록 상관관계가 약하다는 뜻이고,

-1에 가까울수록 음(-1)의 상관관계가 강하며,

+1에 가까울수록 양(+1)의 상관관계가 강하다는 뜻으로 해석할 수 있다.

Population Correation coefficient = $\rho = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$

DataFrame의 데이터 분석용 함수 – corr(), cov()

- 상관계수와 공분산 계산 함수

```
df2 = pd.DataFrame(np.random.randn(6, 4),
                   columns=["A", "B", "C", "D"],
                   index=pd.date_range("20160701", periods=6))
df2["A"].corr(df2["B"])        #'A' 열의 데이터와 'B' 열의 데이터 간의 상관계수
df2["B"].cov(df2["C"])         #'B'열 데이터와 'C'열 데이터 간의 공분산
```

	D	B	C	A	E	F
2016-07-05	0.890014	0.639285	-1.247287	-0.028576	5	alpha
2016-07-06	-0.705451	-0.090056	0.420597	0.222462	1	beta
2016-07-02	0.081823	-0.687809	-0.859128	0.662330	5	gamma
2016-07-01	-0.958517	1.884339	0.219757	-0.282696	1	gamma
2016-07-03	0.108020	0.335537	0.789473	1.428221	3	alpha
2016-07-04	0.064590	1.128564	-0.034507	-0.855283	1	gamma

```
#A열과 B열간의 상관관계
df2["A"].corr(df2["B"])
```

```
-0.59358428220823123
```

```
#공분산
df2["B"].cov(df2["C"])
```

```
0.15522757312166305
```

DataFrame의 데이터 분석용 함수 – sort_index()

- **정렬함수**

  pandas의 DataFrame에서는 인덱스 기준 정렬과 값 기준 정렬을 지원한다.

- 새로운 DataFrame 생성

  df2의 인덱스와 컬럼 순서를 무작위로 섞어서 인덱스와 컬럼의 순서를 불규칙하게 만듬

```
dates = df2.index
random_dates = np.random.permutation(dates)
df2 = df2.reindex(index=random_dates, columns=["D", "B", "C", "A"])
```

	D	B	C	A
2016-07-05	0.890014	0.639285	-1.247287	-0.028576
2016-07-03	0.108020	0.335537	0.789473	1.428221
2016-07-01	-0.958517	1.884339	0.219757	-0.282696
2016-07-04	0.064590	1.128564	-0.034507	-0.855283
2016-07-06	-0.705451	-0.090056	0.420597	0.222462
2016-07-02	0.081823	-0.687809	-0.859128	0.662330

DataFrame의 데이터 분석용 함수 – sort_index()

인덱스의 정렬 – 행 방향

```
#'행 방향'오름차순으로 정렬을 수행
df2.sort_index(axis=0)
```

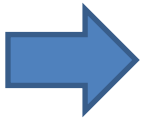
	D	B	C	A
2016-07-01	-0.958517	1.884339	0.219757	-0.282696
2016-07-02	0.081823	-0.687809	-0.859128	0.662330
2016-07-03	0.108020	0.335537	0.789473	1.428221
2016-07-04	0.064590	1.128564	-0.034507	-0.855283
2016-07-05	0.890014	0.639285	-1.247287	-0.028576
2016-07-06	-0.705451	-0.090056	0.420597	0.222462

```
#'행 방향'내림차순으로 정렬을 수행
df2.sort_index(axis=0, ascending=False)
```

	D	B	C	A
2016-07-06	-0.705451	-0.090056	0.420597	0.222462
2016-07-05	0.890014	0.639285	-1.247287	-0.028576
2016-07-04	0.064590	1.128564	-0.034507	-0.855283
2016-07-03	0.108020	0.335537	0.789473	1.428221
2016-07-02	0.081823	-0.687809	-0.859128	0.662330
2016-07-01	-0.958517	1.884339	0.219757	-0.282696

인덱스의 정렬 – 열 방향

```
#'열 방향'오름차순으로 정렬을 수행
df2.sort_index(axis=1)
```



	A	B	C	D
2016-07-05	-0.028576	0.639285	-1.247287	0.890014
2016-07-03	1.428221	0.335537	0.789473	0.108020
2016-07-01	-0.282696	1.884339	0.219757	-0.958517
2016-07-04	-0.855283	1.128564	-0.034507	0.064590
2016-07-06	0.222462	-0.090056	0.420597	-0.705451
2016-07-02	0.662330	-0.687809	-0.859128	0.081823

DataFrame의 데이터 분석용 함수

■ 열의 값으로 정렬

```
#D 열의 값을 기준으로 오름차순으로 정렬
df2.sort_values(by="D")
```

	D	B	C	A
2016-07-01	-0.958517	1.884339	0.219757	-0.282696
2016-07-06	-0.705451	-0.090056	0.420597	0.222462
2016-07-04	0.064590	1.128564	-0.034507	-0.855283
2016-07-02	0.081823	-0.687809	-0.859128	0.662330
2016-07-03	0.108020	0.335537	0.789473	1.428221
2016-07-05	0.890014	0.639285	-1.247287	-0.028576

```
#B 열의 값을 기준으로 오름차순으로 정렬
df2.sort_values(by="B",ascending=False)
```

	D	B	C	A
2016-07-01	-0.958517	1.884339	0.219757	-0.282696
2016-07-04	0.064590	1.128564	-0.034507	-0.855283
2016-07-05	0.890014	0.639285	-1.247287	-0.028576
2016-07-03	0.108020	0.335537	0.789473	1.428221
2016-07-06	-0.705451	-0.090056	0.420597	0.222462
2016-07-02	0.081823	-0.687809	-0.859128	0.662330

DataFrame의 데이터 분석용 함수

■ 여러 개의 열의 값으로 정렬

df2에 'E'열과 'F'열을 추가한 뒤,

'E' 열과 'F' 열의 값을 동시에 기준으로 삼아 각각 오름차순이 되도록 행들을 정렬하고 싶은 경우

```
df2["E"] = np.random.randint(0, 6, size=6)
df2["F"] = ["alpha", "beta", "gamma", "gamma", "alpha", "gamma"]
df2.sort_values(by=["E", "F"])
```

	D	B	C	A	E	F
2016-07-05	0.890014	0.639285	-1.247287	-0.028576	4	alpha
2016-07-03	0.108020	0.335537	0.789473	1.428221	3	beta
2016-07-01	-0.958517	1.884339	0.219757	-0.282696	0	gamma
2016-07-04	0.064590	1.128564	-0.034507	-0.855283	1	gamma
2016-07-06	-0.705451	-0.090056	0.420597	0.222462	0	alpha
2016-07-02	0.081823	-0.687809	-0.859128	0.662330	3	gamma

	D	B	C	A	E	F
2016-07-06	-0.705451	-0.090056	0.420597	0.222462	0	alpha
2016-07-01	-0.958517	1.884339	0.219757	-0.282696	0	gamma
2016-07-04	0.064590	1.128564	-0.034507	-0.855283	1	gamma
2016-07-03	0.108020	0.335537	0.789473	1.428221	3	beta
2016-07-02	0.081823	-0.687809	-0.859128	0.662330	3	gamma
2016-07-05	0.890014	0.639285	-1.247287	-0.028576	4	alpha

DataFrame의 데이터 분석용 함수

기타함수

df.unique() 함수는 지정한 행 또는 열에서 중복을 제거한 유니크한 값들만을 제시하는 함수

df.value_counts() 함수는 특정한 행 또는 열에서 값에 따른 갯수를 제시하는 함수

```
uniques = df2["F"].unique()
```

```
array(['alpha', 'beta', 'gamma'], dtype=object)
```

```
df2["F"].value_counts()
```

```
gamma    3
alpha    2
beta     1
Name: F, dtype: int64
```

	D	B	C	A	E	F
2016-07-06	-0.705451	-0.090056	0.420597	0.222462	0	alpha
2016-07-01	-0.958517	1.884339	0.219757	-0.282696	0	gamma
2016-07-04	0.064590	1.128564	-0.034507	-0.855283	1	gamma
2016-07-03	0.108020	0.335537	0.789473	1.428221	3	beta
2016-07-02	0.081823	-0.687809	-0.859128	0.662330	3	gamma
2016-07-05	0.890014	0.639285	-1.247287	-0.028576	4	alpha

DataFrame의 데이터 분석용 함수

기타함수

df.isin() 함수를 사용하면, 특정한 행 또는 열의 각 성분이 특정한 값들을 포함되어 있는지 확인가능

불리언 마스크를 반환하며, 이 불리언 마스크를 사용하여, 원하는 행만을 선택할 수도 있음.

```
df2["F"].isin(["alpha", "beta"])
df2.loc[df2["F"].isin(["alpha", "beta"]), :]
```

```
df2["F"].isin(["alpha","beta"])
```

```
2016-07-05     True
2016-07-06     True
2016-07-02    False
2016-07-01    False
2016-07-03     True
2016-07-04    False
Name: F, dtype: bool
```

```
df2.loc[df2["F"].isin(["alpha","beta"]),:]
```

	D	B	C	A	E	F
2016-07-05	0.890014	0.639285	-1.247287	-0.028576	5	alpha
2016-07-06	-0.705451	-0.090056	0.420597	0.222462	1	beta
2016-07-03	0.108020	0.335537	0.789473	1.428221	3	alpha

DataFrame의 데이터 분석용 함수

사용자 정의 함수를 DataFrame에 적용

사용자가 직접 정의한 함수를 DataFrame의 행 또는 열에 적용하는 것이 가능하다.

lambda 로 정의한 **func** 함수는, 해당 행 또는 열에서의 최대값 - 최솟값을 계산하는 함수

```
func = lambda x: x.max() - x.min()
```

새로운 DataFrame 생성

```
df3 = pd.DataFrame(np.random.randn(4, 3), columns=["b", "d", "e"],
                   index=["Seoul", "Incheon", "Busan", "Daegu"])
df3.apply(func, axis=0)     #각 행(row)별로  최대값에서 최소값을 뺀 값을 적용시킴
df3.apply(func, axis=1)     #각 열(column)별로  최대값에서 최소값을 뺀 값을 적용시킴
```

	b	d	e
Seoul	-1.073264	0.499519	-1.835678
Incheon	0.431766	-0.238659	1.465233
Busan	-1.193490	-0.804607	-0.542409
Daegu	-0.885650	1.033454	1.141613

```
df3.apply(func,axis=0)
```

```
b    1.625256
d    1.838061
e    3.300911
dtype: float64
```

```
df3.apply(func,axis=1)
```

```
Seoul      2.335197
Incheon    1.703892
Busan      0.651082
Daegu      2.027263
dtype: float64
```

pandas를 사용한 Lending Club Loan 데이터셋 분석

- **Lending Club Loan dataset의 주요 컬럼 요약**

loan_amnt: 대출자의 대출 총액

funded_amnt: 해당 대출을 위해 모금된 총액

issue_d: 대출을 위한 기금이 모금된 월

loan_status: 대출의 현재 상태*

title: 대출자에 의해 제공된 대출 항목명

purpose: 대출자에 의해 제공된 대출 목적

emp_length: 대출자의 재직 기간

grade: LC assigned loan grade**

int_rate: 대출 이자율

term: 대출 상품의 기간 (36-month vs. 60-month)

* 불량 상태(bad status): "Charged Off", "Default", "Does not meet the credit policy. Status:Charged Off", "In Grace Period", "Default Receiver", "Late (16-30 days)", "Late (31-120 days)"

** LC loan grade 참고: https://www.lendingclub.com/public/rates-and-fees.action

pandas를 사용한 Lending Club Loan 데이터셋 분석

파일 읽기

Lending Club Loan dataset 2007-2015를 사용함

CSV 파일은 **pd.read_csv()** 함수를 사용하여 읽음, 읽어들일 데이터 파일의 경로와 구분자 등을 입력함.

```
df = pd.read_csv("data/lending-club-loan-data/loan.csv", sep=",")
```

필요한 열 만을 발췌하고, 결측값(NaN) 제거하기

```
df2 = df[["loan_amnt", "loan_status", "grade", "int_rate", "term"]]
df2 = df2.dropna(how="any")
```

df2.head()

	loan_amnt	loan_status	grade	int_rate	term
0	5000.0	Fully Paid	B	10.65	36 months
1	2500.0	Charged Off	C	15.27	60 months
2	2400.0	Fully Paid	C	15.96	36 months
3	10000.0	Fully Paid	C	13.49	36 months
4	3000.0	Current	B	12.69	60 months

pandas를 사용한 Lending Club Loan 데이터셋 분석

- '36개월 대출'과 '60개월 대출'의 대출 총액 파악

```
term_to_loan_amnt_dict = {}
uniq_terms = df2["term"].unique()
for term in uniq_terms:
    loan_amnt_sum = df2.loc[df2["term"] == term, "loan_amnt"].sum()
    term_to_loan_amnt_dict[term] = loan_amnt_sum
term_to_loan_amnt = pd.Series(term_to_loan_amnt_dict)
```

```
term_to_loan_amnt_dict
```

```
{' 36 months': 7752507375.0, ' 60 months': 5341004575.0}
```

```
term_to_loan_amnt
```

```
 36 months    7.752507e+09
 60 months    5.341005e+09
dtype: float64
```

pandas를 사용한 Lending Club Loan 데이터셋 분석

- 각 불량 대출 들의 등급 분포 파악

```
total_status_category = df2["loan_status"].unique()
bad_status_category = total_status_category[[1, 3, 4, 5, 6, 8]]
```

total_status_category

```
array(['Fully Paid', 'Charged Off', 'Current', 'Default',
       'Late (31-120 days)', 'In Grace Period', 'Late (16-30 days)',
       'Does not meet the credit policy. Status:Fully Paid',
       'Does not meet the credit policy. Status:Charged Off', 'Issued']
```

bad_status_category

```
array(['Charged Off', 'Default', 'Late (31-120 days)',
       'Late (16-30 days)', 'In Grace Period',
       'Does not meet the credit policy. Status:Charged Off']
```

```
df2["bad_loan_status"] = df2["loan_status"].isin(bad_status_category)
```

df2.head()

	loan_amnt	loan_status	grade	int_rate	term	bad_loan_status
0	5000.0	Fully Paid	B	10.65	36 months	False
1	2500.0	Charged Off	C	15.27	60 months	True
2	2400.0	Fully Paid	C	15.96	36 months	False
3	10000.0	Fully Paid	C	13.49	36 months	False
4	3000.0	Current	B	12.69	60 months	False

pandas를 사용한 Lending Club Loan 데이터셋 분석

- 각 불량 대출 들의 등급 분포 파악

```
bad_loan_status_to_grades = df2.loc[df2["bad_loan_status"] == True, "grade"]
                            .value_counts()
```

bad_loan_status_to_grades

```
C    19054
D    15859
B    13456
E     9745
F     4383
A     3663
G     1269
Name: grade, dtype: int64
```

```
bad_loan_status_to_grades.sort_index()
```

```
A     3663
B    13456
C    19054
D    15859
E     9745
F     4383
G     1269
Name: grade, dtype: int64
```

pandas를 사용한 Lending Club Loan 데이터셋 분석

대출 총액과 대출 이자율 간의 상관관계 파악

```
df2["loan_amnt"].corr(df2["int_rate"])
```

```
0.14502309929883395
```

파일 쓰기

bad_loan_status_to_grades Series를 외부 파일로 쓰는 작업을 진행함.

Series.to_csv() 혹은 **df.to_csv()** 함수를 사용하면, 현재 Series 혹은 DataFrame 형태의 분석 결과물을 외부 파일로 손쉽게 쓸 수 있으며, 해당 함수를 호출할 때, 저장할 파일의 경로와 구분자를 인자로 같이 입력해야 한다.

```
bad_loan_status_to_grades.to_csv("bad_loan_status.csv", sep=",")
```

5. 데이터 시각화 라이브러리 – matplotlib 사용하기

matplotlib의 개요

- **matplotlib 란?**

matplotlib은 numpy나 pandas를 사용하여 데이터를 분석한 결과를 시각화 하는 데 사용되는 대표적인 Python 데이터 시각화 라이브러리입니다.

matplotlib에서는 DataFrame 혹은 Series 형태의 데이터를 가지고 다양한 형태의 플롯을 만들어 주는 기능을 지원합니다.

https://matplotlib.org/

matplotlib의 개요

플롯팅 옵션 지정

플롯을 그리기에 앞서, **%matplotlib**라는 플롯팅 옵션을 먼저 지정해야 jupyter notebook에서 그래프를 그려줄 수 있다.

%matplotlib nbagg 설정은, 생성되는 플롯을 인터랙티브 하게 조작할 수 있음.

%matplotlib inline 설정은 , 플롯을 일단 생성하면 이를 조작할 수 없음.

```
%matplotlib nbagg
```

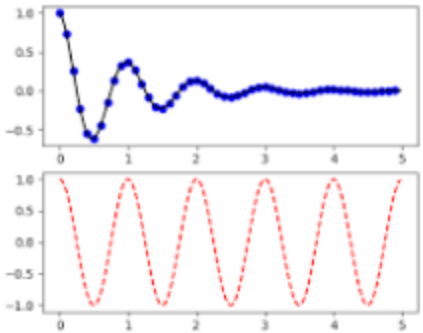
필요한 library import

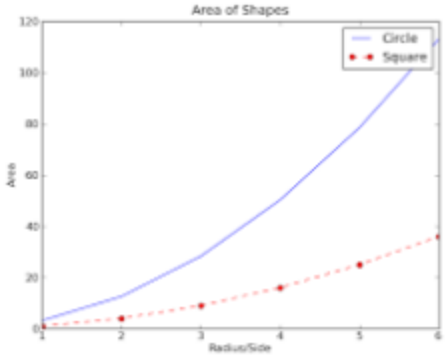
```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

라인 플롯(line plot)

라인 플롯(line plot)

라인 플롯은 연속적인 직선으로 구성된 플롯입니다. 어떤 특정한 독립변수 X가 변화함에 따라 종속변수 Y가 어떻게 변화하는지를 나타내고자 할 때 라인 플롯을 사용함.





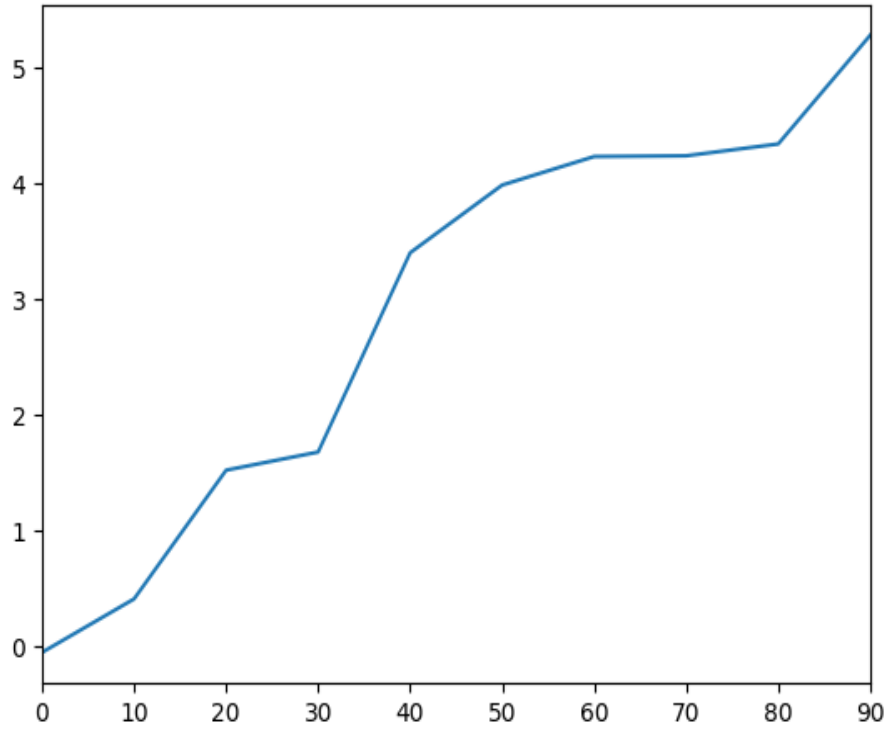
라인 플롯(line plot)

- **라인 플롯(line plot) – Series 사용**

랜덤한 값들로 구성된 Series **s**를 인덱스와 함께 생성하고 **s.plot()**을 실행하면, **s**의 인덱스와 값을 사용하여 라인 플롯을 그려줍니다.

```
s = pd.Series(np.random.randn(10).cumsum(),
              index=np.arange(0, 100, 10))
s.plot()
```

```
0    -0.048488
10    0.413296
20    1.524783
30    1.681463
40    3.402159
50    3.986920
60    4.233187
70    4.239451
80    4.341811
90    5.281117
dtype: float64
```



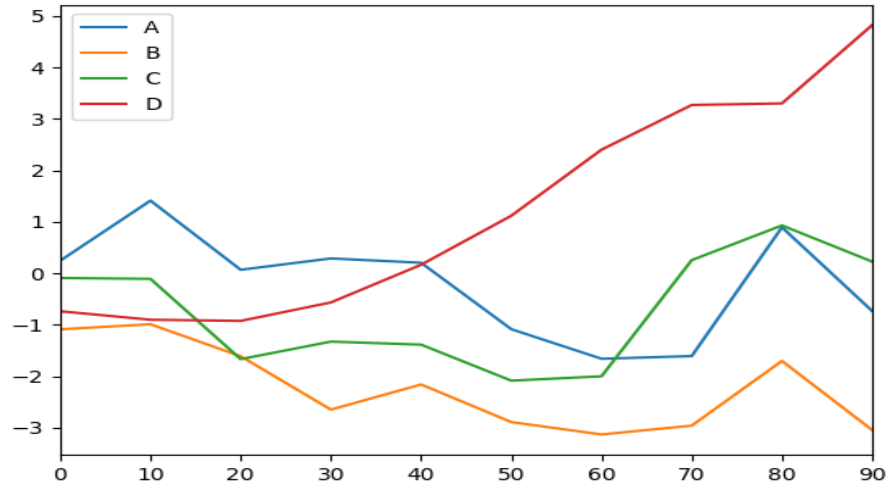
라인 플롯(line plot)

라인 플롯(line plot) – DataFrame 사용

랜덤한 값들로 구성된 DataFrame **df**을 인덱스, 컬럼과 함께 생성한 뒤 **df.plot()**을 실행하면,

df의 인덱스와 각 컬럼 값을 사용하여 여러 개의 라인 플롯을 그려줍니다.

```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(axis=0),
                  columns=["A", "B", "C", "D"],
                  index=np.arange(0, 100, 10))
df.plot()
```

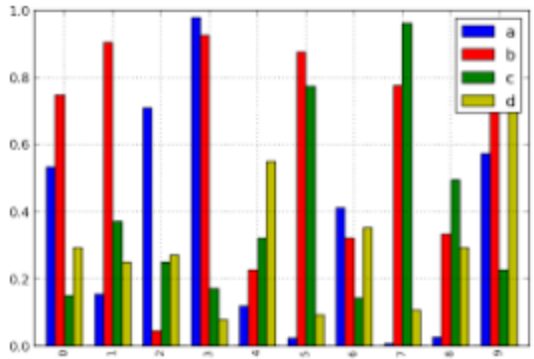


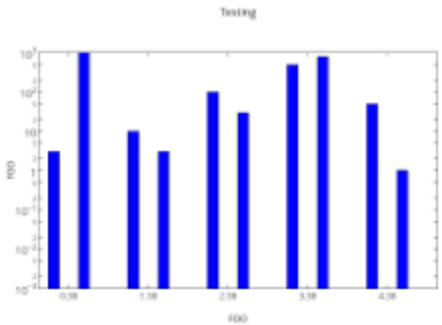
	A	B	C	D
0	0.246041	-1.087977	-0.090787	-0.737416
10	1.412900	-0.989792	-0.107997	-0.901651
20	0.068889	-1.611963	-1.668164	-0.924862
30	0.289113	-2.648715	-1.326974	-0.565940
40	0.205474	-2.160054	-1.385945	0.166205
50	-1.085889	-2.891230	-2.086098	1.119234
60	-1.660477	-3.131430	-2.001335	2.402729
70	-1.608062	-2.960515	0.256359	3.271790
80	0.894558	-1.702922	0.930967	3.301997
90	-0.732359	-3.045553	0.227134	4.822931

바 플롯(bar plot)

바 플롯(bar plot)

바 플롯은 막대 형태의 플롯입니다. 독립변수 X가 변화하면서 종속변수 Y가 변화하는 양상을 나타낼 때, X가 연속적인 숫자에 해당하는 경우 라인 플롯을 그렸다면, X가 유한 개의 값만을 가질 경우 바 플롯을 사용하면 유용합니다. 즉 라인 플롯은 변화하는 모습을 관찰할 때 사용하고, 바 플롯은 변화한 결과를 비교할 때 사용합니다.



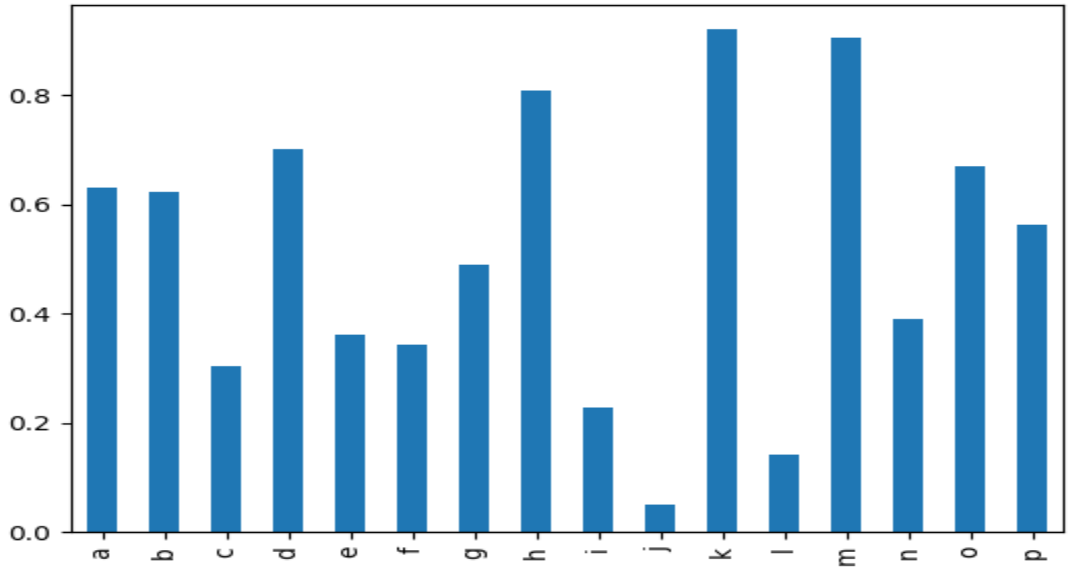


바 플롯(bar plot)

- **바 플롯(bar plot)** - 수직방향, Series 사용

랜덤한 값들로 구성된 Series **s2**를 인덱스와 함께 생성한 뒤 **s2.plot(kind="bar")**를 실행하면, **s2**의 인덱스와 값을 사용하여 수직 방향의 바 플롯을 그려준다.

```
s2 = pd.Series(np.random.rand(16), index=list("abcdefghijklmnop"))
s2.plot(kind="bar")
```



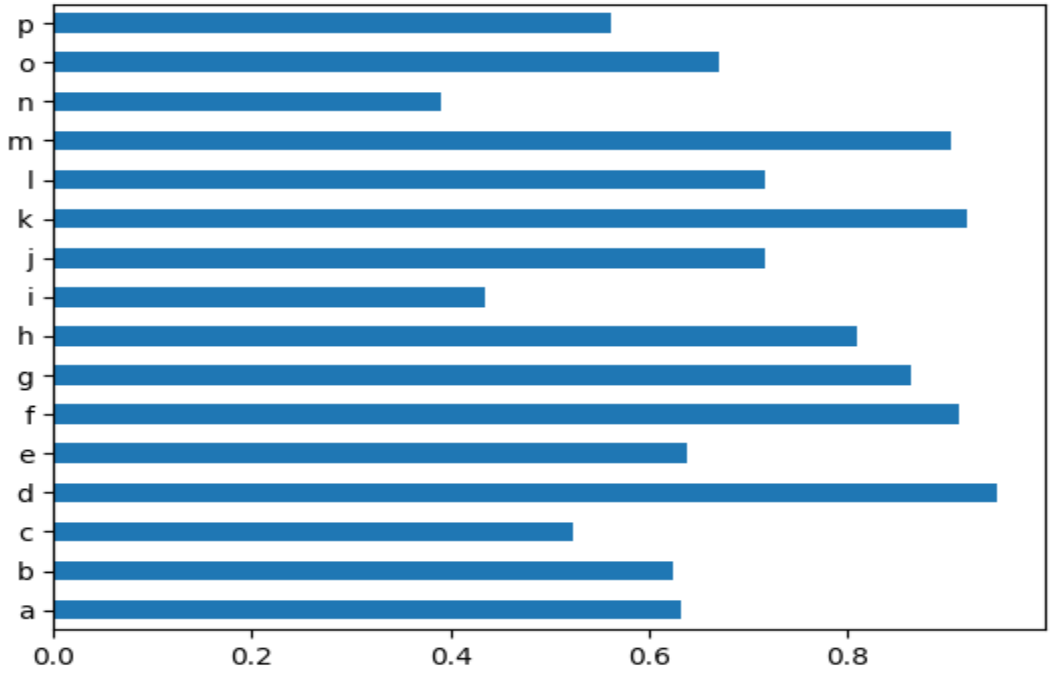

```
a    0.631815
b    0.623673
c    0.303475
d    0.700098
e    0.361139
f    0.343653
g    0.490511
h    0.809412
i    0.228673
j    0.050203
k    0.920162
l    0.140656
m    0.905064
n    0.390961
o    0.670953
p    0.562667
dtype: float64
```

바 플롯(bar plot)

- **바 플롯(bar plot) – 수평방향**, Series 사용

만약 바 플롯을 수평 방향으로 그리고자 할 경우, **s2.plot(kind="barh")**를 실행하면 됩니다.

```
s2.plot(kind="barh")
```



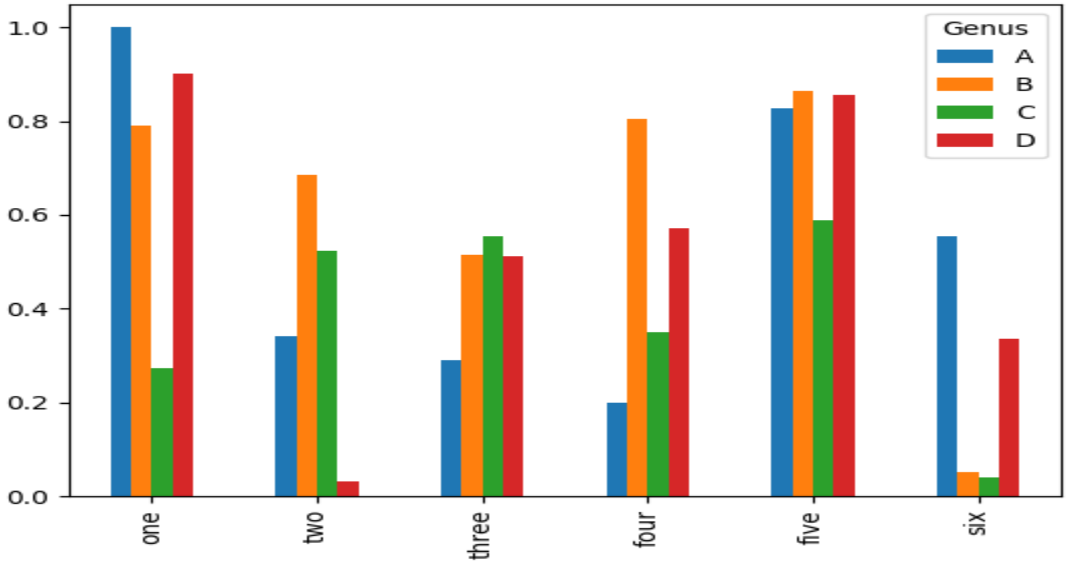

```
a    0.076422
b    0.563392
c    0.523028
d    0.951226
e    0.637694
f    0.912442
g    0.864007
h    0.410143
i    0.433987
j    0.717065
k    0.805000
l    0.717232
m    0.145570
n    0.388607
o    0.224175
p    0.139861
dtype: float64
```

바 플롯(bar plot)

바 플롯(bar plot) – 수직방향, DataFrame 사용

df2의 인덱스와 각 컬럼 값을 사용하여 여러 개의 바 플롯을 그려줍니다.

```
df2 = pd.DataFrame(np.random.rand(6, 4),
                   index=["one", "two", "three", "four", "five", "six"],
                   columns=pd.Index(["A", "B", "C", "D"], name="Genus"))
df2.plot(kind="bar")
```



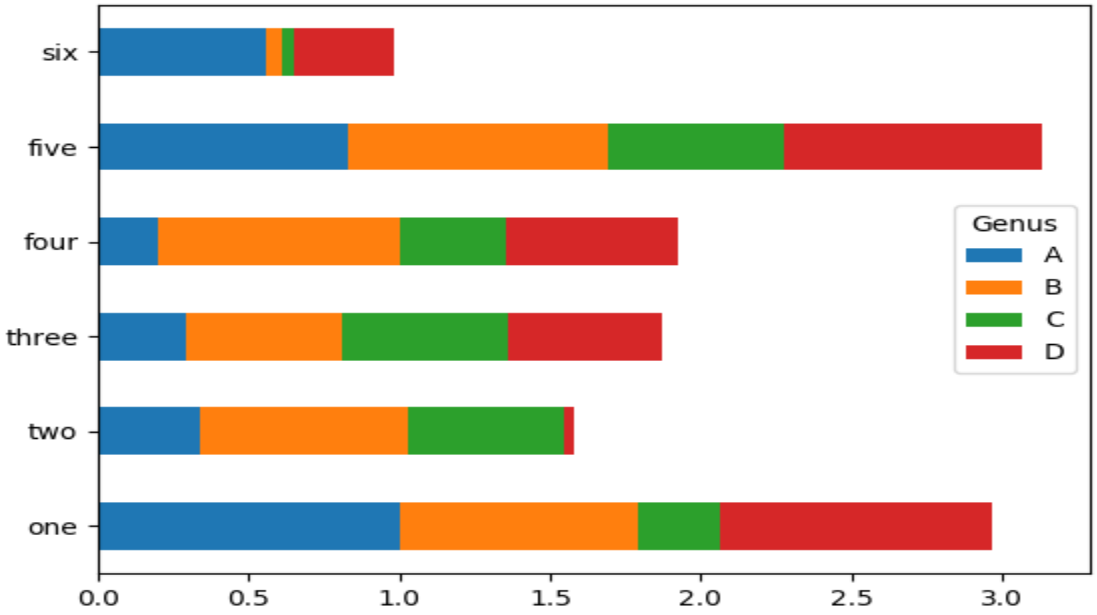
Genus	A	B	C	D
one	0.999805	0.790270	0.274315	0.901771
two	0.341059	0.684748	0.523636	0.032163
three	0.291255	0.515972	0.554363	0.513141
four	0.199427	0.805186	0.348938	0.571758
five	0.827911	0.862971	0.588690	0.855545
six	0.555743	0.052112	0.040661	0.336967

바 플롯(bar plot)

바 플롯(bar plot) – 수평방향, DataFrame 사용

바 플롯을 그릴 때 **stacked=True** 인자를 넣어주면, 하나의 인덱스에 대한 각 열의 값을 한 줄로 쌓아 표시해 줍니다. 이는 하나의 인덱스에 대응되는 각 열 값의 상대적 비율을 확인할 때 유용함.

```
df2.plot(kind="barh", stacked=True)
```

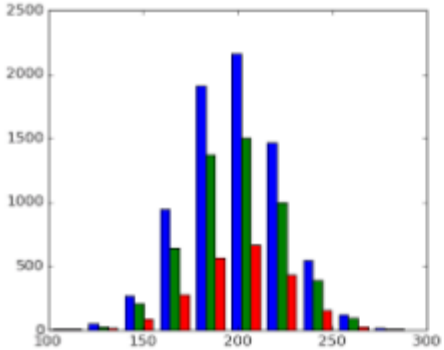


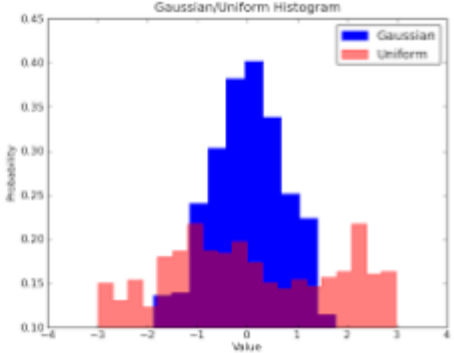
Genus	A	B	C	D
one	0.999805	0.790270	0.274315	0.901771
two	0.341059	0.684748	0.523636	0.032163
three	0.291255	0.515972	0.554363	0.513141
four	0.199427	0.805186	0.348938	0.571758
five	0.827911	0.862971	0.588690	0.855545
six	0.555743	0.052112	0.040661	0.336967

히스토그램(histogram)

히스토그램 (histogram)

히스토그램의 경우 어느 하나의 변수 X가 가질 수 있는 값의 구간을 여러 개 설정한 뒤, 각각의 구간에 속하는 갯수를 막대 형태로 나타낸 플롯입니다. Series로부터 히스토그램을 그릴 때는 인덱스를 따로 명시할 필요가 없으며, 값들만 가지고 있으면 됩니다.



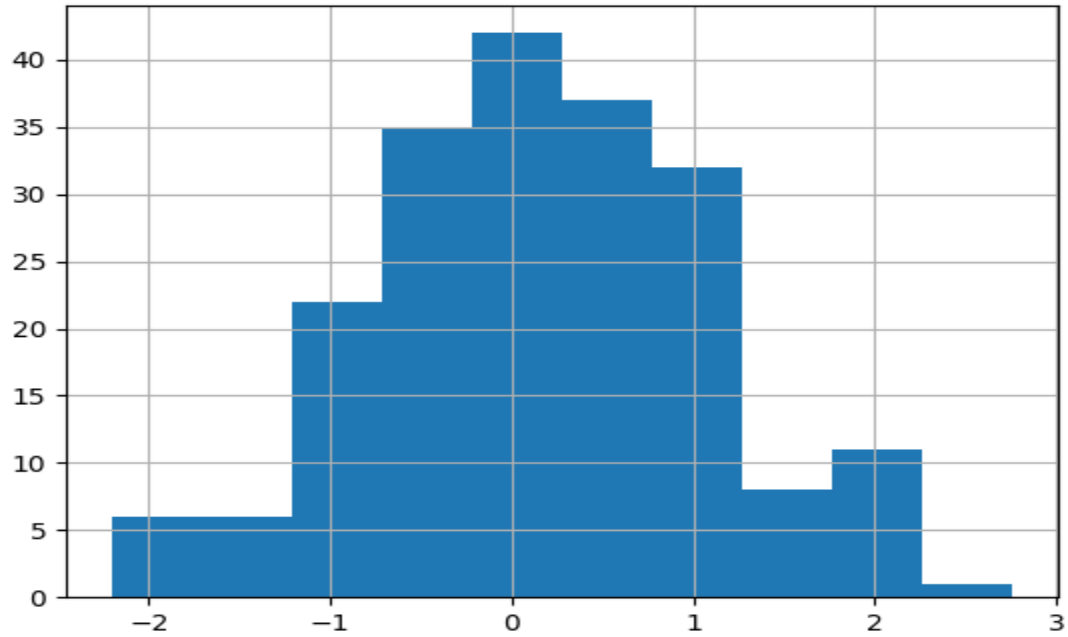


히스토그램(histogram)

히스토그램 (histogram)

랜덤한 값들로 구성된 Series s3를 생성한 뒤, s3.hist()를 실행하면, 히스토그램을 그려줌

```
s3 = pd.Series(np.random.normal(0, 1, size=200))
s3.hist()
```



0	-0.966593	170	-0.301480
1	-0.355063	171	0.846058
2	-1.051846	172	-0.095773
3	0.891772	173	-0.116472
4	0.333097	174	0.965010
5	1.146275	175	-0.909931
6	0.280591	176	0.379822
	...		
22	-0.926031	192	-0.592127
23	-0.814655	193	0.206081
24	0.542441	194	0.405933
25	-0.036267	195	-0.473236
26	-0.371562	196	2.063566
27	0.292268	197	-0.844352
28	1.274431	198	-0.407641
29	0.897567	199	0.010817

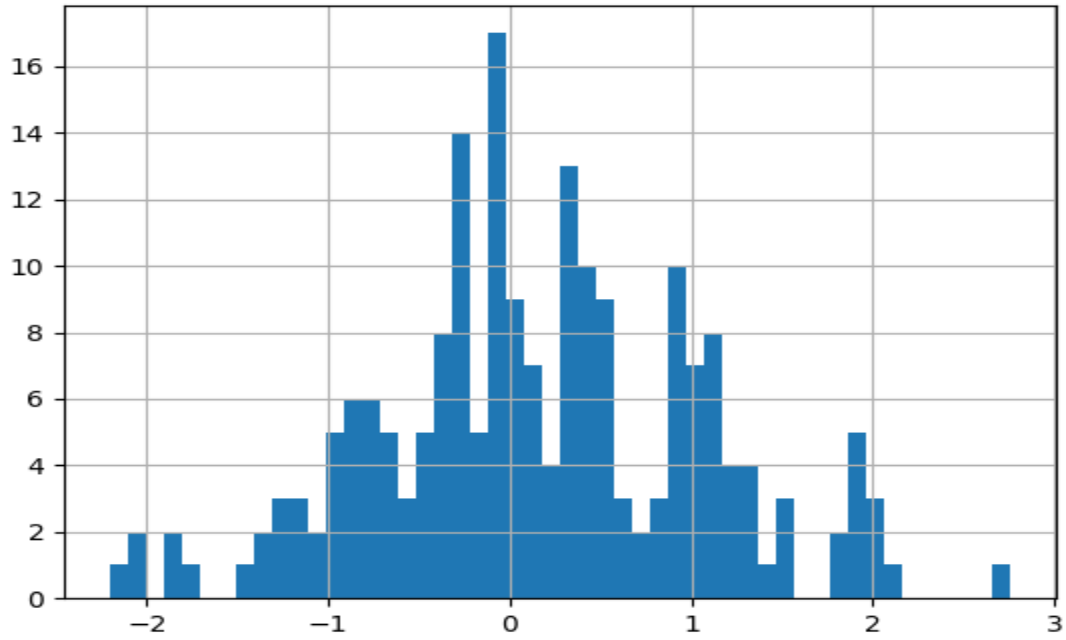
Length: 200, dtype: float64

히스토그램(histogram)

히스토그램 (histogram)

각 구간에 속하는 값의 갯수를 카운팅 할 때, 구간의 개수는 자동으로 10개로 설정되는데, 이 구간을 '**bin(빈)**'이라고 부릅니다. bin의 갯수를 직접 설정할 수 있음.

```
s3.hist(bins=50)
```

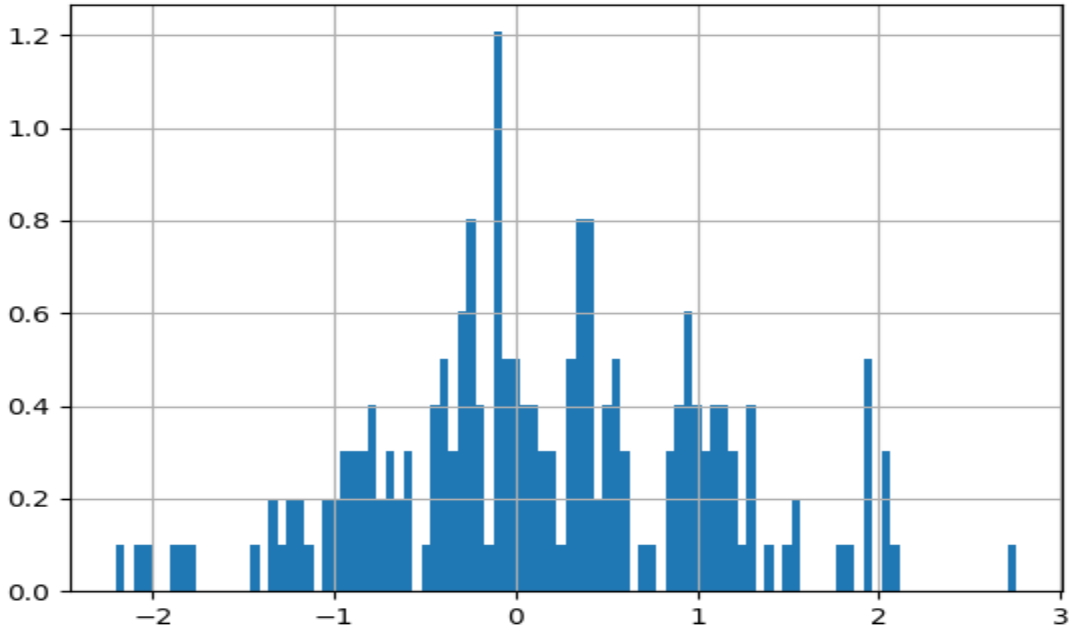


히스토그램(histogram)

히스토그램 (histogram)

만약 **normed=True** 인자를 넣어주면, 각 bin에 속하는 갯수를 전체 갯수로 나눈 비율, 즉 정규화한 (normalized) 값을 사용하여 히스토그램을 그립니다.

```
s3.hist(bins=100, normed=True)
```

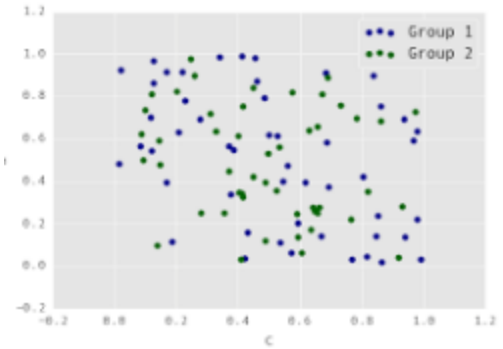


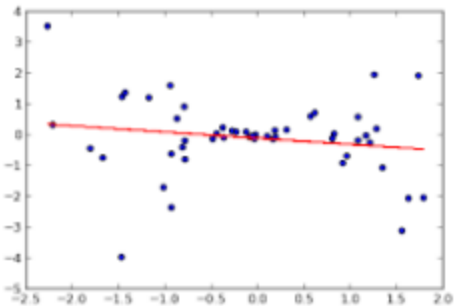
산점도(scatter plot)

산점도(scatter plot)

라인 플롯이나 바 플롯의 경우 어떤 독립변수 X가 변화함에 따라 종속변수 Y가 어떻게 변화하는지 나타내는 것이 목적이었다면,

산점도의 경우에는 서로 다른 두 개의 독립변수 X1, X2 간에 어떠한 관계가 있는지 알아보고자 할 때 일반적으로 많이 사용합니다. 즉 산점도는, 두 독립변수 X1과 X2의 값을 각각의 축으로 하여 2차원 평면 상에 점으로 나타낸 플롯입니다.





산점도(scatter plot)

산점도(scatter plot)

랜덤한 값들로 구성된 두 개의 array를 생성한 뒤, np.concatenate() 함수를 사용하여 열 방향으로 연결

```
x1 = np.random.normal(1, 1, size=(100, 1))
x2 = np.random.normal(-2, 4, size=(100, 1))
X = np.concatenate((x1, x2), axis=1)
```

생성된 **X** array를 사용하여 DataFrame **df3**를 생성하면, **df3**에는 'x1'과 'x2'의 두 개의 열이 포함됨.

plt.scatter(df3["x1"], df3["x2"])를 실행하면, 두 열 간의 값을 기준으로 산점도를 그립니다.

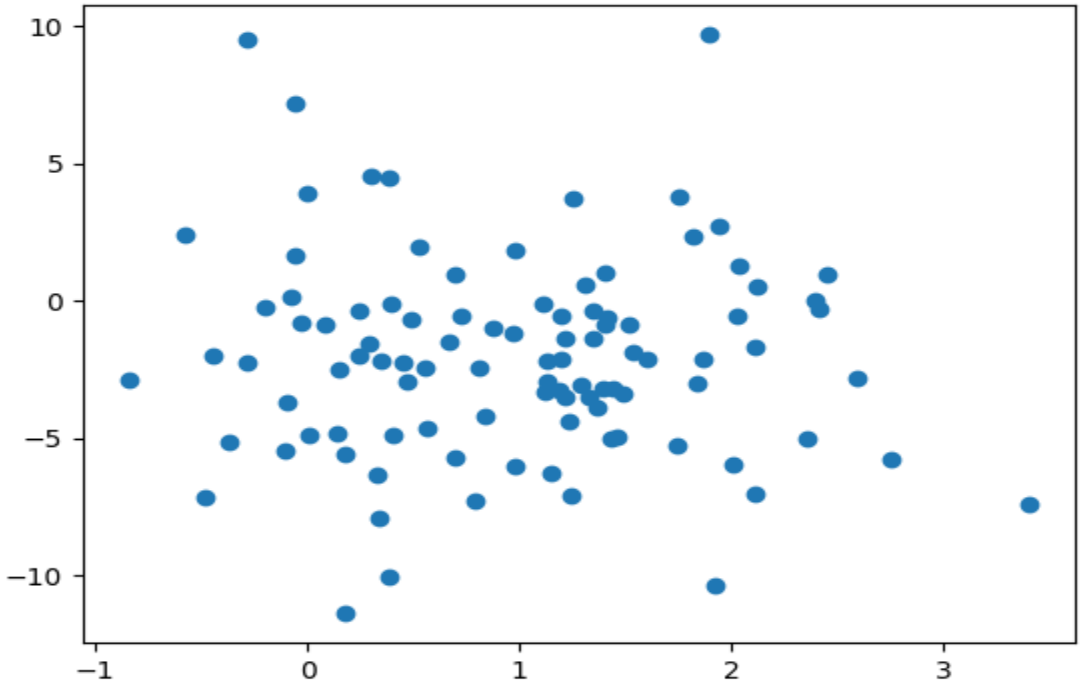
```
df3 = pd.DataFrame(X, columns=["x1", "x2"])
plt.scatter(df3["x1"], df3["x2"])
```

얻어진 산점도의 수평축에는 'x1'의 값, 수직축에는 'x2'의 값을 사용하여 해당하는 위치에 점을 찍어서 데이터를 표현합니다.

산점도(scatter plot)

산점도(scatter plot)

```
df3 = pd.DataFrame(X, columns=["x1", "x2"])
plt.scatter(df3["x1"], df3["x2"])
```



X

```
array([[  2.11025954e+00,  -7.00286993e+00],
       [  9.83568403e-01,   1.83583949e+00],
       [  2.94542805e-01,  -1.53845373e+00],
       [ -4.42345234e-01,  -1.99034546e+00],
          ...
       [  3.93703957e-01,  -1.37469693e-01],
       [  1.29143663e+00,  -3.05784419e+00],
       [ -5.75918694e-01,   2.38967116e+00],
       [  1.83437956e+00,  -3.00800515e+00],
       [  1.92380220e+00,  -1.03416595e+01],
       [  4.88665755e-01,  -6.68200007e-01]])
```

df3

	x1	x2
0	2.110260	-7.002870
1	0.983568	1.835839
...		
98	1.923802	-10.341659
99	0.488666	-0.668200

100 rows × 2 columns

플롯 모양 변형하기 – Figure,subplot 및 axes

- **Figure와 subplots 이란?**

matplotlib에서는 'figure(피겨)'라는 그림 단위를 사용하여, 이 안에 한 개 혹은 복수개의 플롯을 관리할 수 있도록 하는 기능을 지원함. figure 안에 들어가는 플롯 공간 하나를 '**subplot(서브플롯)**' 이라고 함.

- **Figure 생성과 빈 좌표 평면(axes)**

새로운 figure를 직접 생성하려면, **plt.figure()** 함수를 사용함.

figure에 subplot을 하나 추가하고 싶으면, **fig.add_subplot()** 함수를 사용함.

fig.add_subplot() 함수의 반환값(ax1)은 해당 subplot에 그려진 빈 좌표 평면 변수입니다.

matplotlib에서는 이 빈 좌표평면을 '**axes(액시스)**'라고 부릅니다. Figure안의 subplot에 axes를 생성한 순간부터, 플롯을 그릴 수 있는 상태가 됩니다.

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
```

플롯 모양 변형하기 – Figure,subplot 및 axes

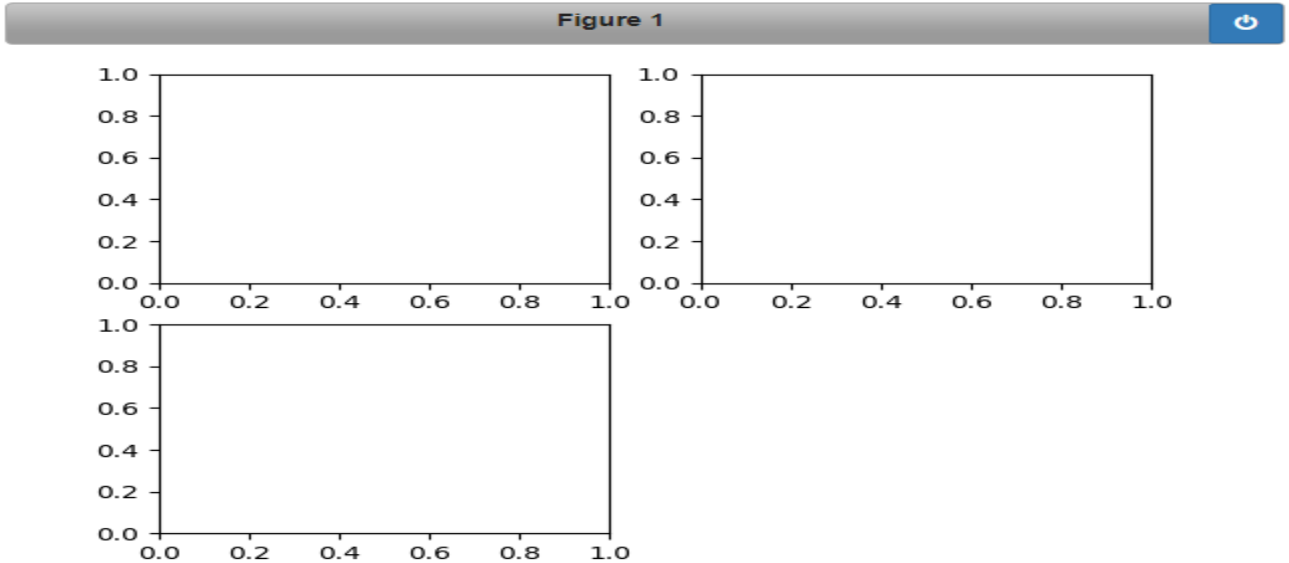
- **각 subplot 위치별 새로운 axes 생성**

fig.add_subplot() 함수에는 총 3개의 인자

첫번째,두번째 인자 : 해당 figure 안에서 subplot들을 몇 개의 행, 몇 개의 열로 할 것인지 설정함

세번째 인자는 : 해당 subplot을 실제로 어느 위치에 배치 할지를 나타내는 번호

```
ax2 = fig.add_subplot(2, 2, 2) ax3 = fig.add_subplot(2, 2, 3)
```

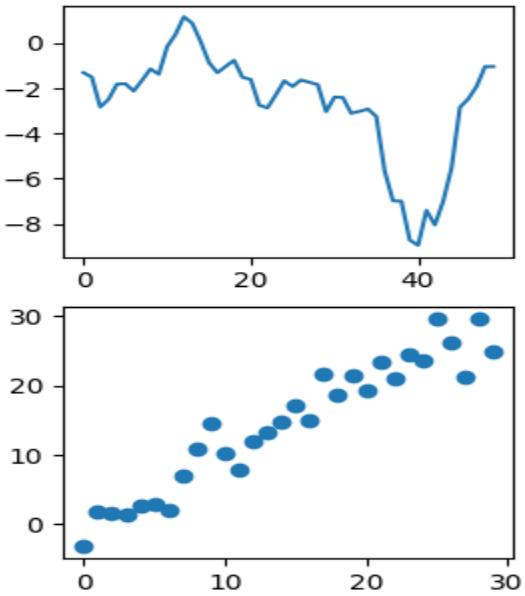


플롯 모양 변형하기 – Figure,subplot 및 axes

- **해당 axes에 plot 그리기**

axes를 직접 지정하여 플롯을 그리는 경우

```
ax1.plot(np.random.randn(50).cumsum())
ax2.hist(np.random.randn(100), bins=20)
ax3.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

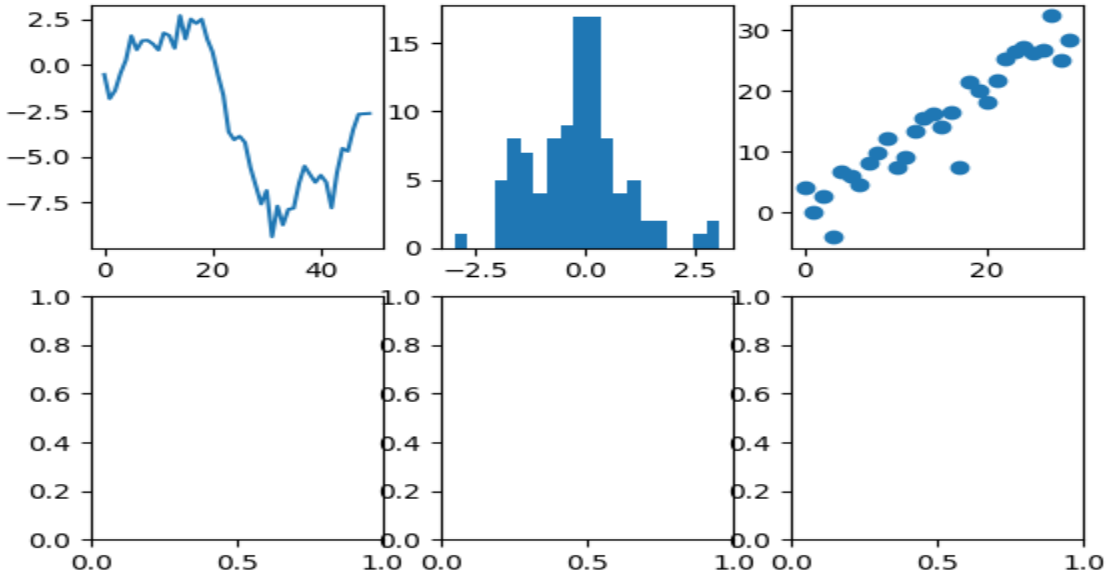


플롯 모양 변형하기 – Figure,subplot 및 axes

- **subplots() 함수의 사용**

figure 안에 총 6개의 subplot들을 2x3으로 배치하며, 각각의 내부에 axes를 생성합니다.

```
fig, axes = plt.subplots(2, 3)
axes[0,0].plot(np.random.randn(50).cumsum())
axes[0,1].hist(np.random.randn(100), bins=20)
axes[0,2].scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

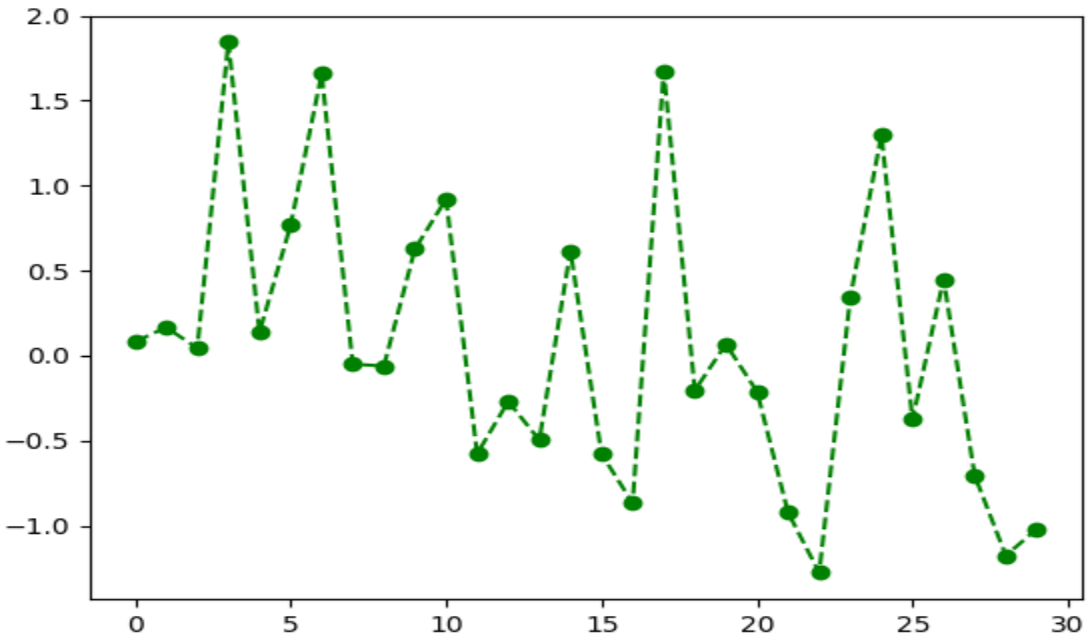


플롯 모양 변형하기 – 색상, 마킹 및 라인스타일 수정

- **라인플롯의 스타일 변경**

plt.plot() 함수에 **color(라인색상)**, **marker(점을 마킹하는 기호)**, **linestyle(라인스타일)** 인자의 값으로 스타일을 지정. color="g " 녹색, marker='o'면 O 모양의 마킹 기호, linestyle="--"이면 점선 스타일

```
plt.plot(np.random.randn(30), color="g", marker='o', linestyle="--")
```

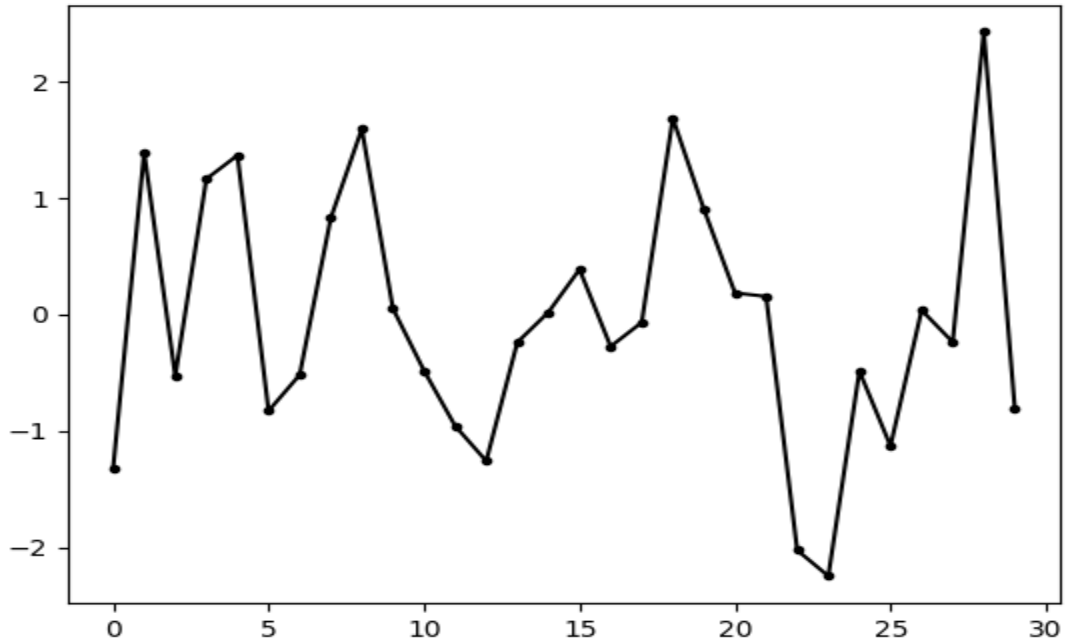


플롯 모양 변형하기 – 색상, 마킹 및 라인스타일 수정

라인플롯의 스타일 변경

라인 플롯의 색상, 마킹 및 라인 스타일을 나타내는 값들을 하나의 문자열로 붙여서 입력할 수도 있음.

```
plt.plot(np.random.randn(30), "k.-")
```

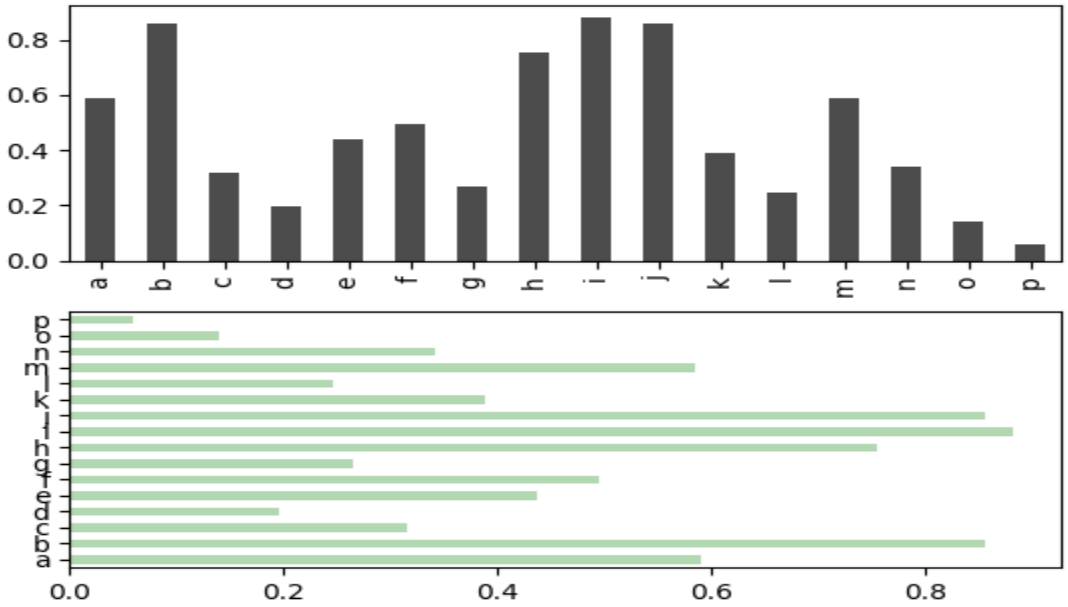


플롯 모양 변형하기 – 색상, 마킹 및 라인스타일 수정

바플롯의 스타일 변경

바 플롯이나 히스토그램, 산점도 등에는 색상과 알파값 등을 지정할 수 있습니다.

```
fig, axes = plt.subplots(2, 1)
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))
data.plot(kind="bar", ax=axes[0], color='k', alpha=0.7)
data.plot(kind="barh", ax=axes[1], color='g', alpha=0.3)
```



플롯 모양 변형하기 – 눈금,레이블 및 범례 수정

- **라인플롯의 눈금, 레이블, 범례 수정**

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(np.random.randn(1000).cumsum())
```

플롯의 수평축 혹은 수직축에 나타난 눈금을 matplotlib에서는 '**틱(tick)**'이라고 부릅니다.

수평축의 눈금은 'xtick', 수직축의 눈금은 'ytick'

수평축의 눈금변경은 **ax.set_xticks()** 함수를 사용함.

수평축의 눈금을 숫자가 아닌 문자열 레이블로 **ax.set_xticklabels()** 함수를 사용함.

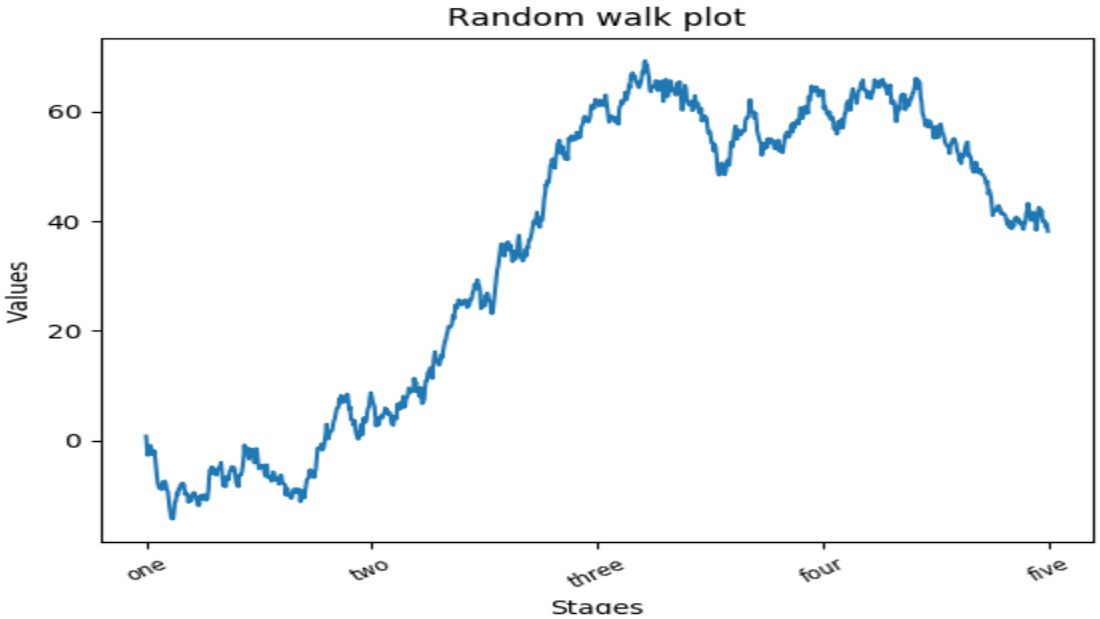
```
ticks = ax.set_xticks([0, 250, 500, 750, 1000])
labels = ax.set_xticklabels(["one", "two", "three", "four", "five"],
                            rotation=30, fontsize="small")
```

플롯 모양 변형하기 – 눈금,레이블 및 범례 수정

- **axes에 제목(title) 입력**

axes에 제목 입력은 **ax.set_title()**, 수평축과 수직축에 이름 설정은 **ax.set_xlabel()**, **ax.set_ylabel()**

```
ax.set_title("Random walk plot")
ax.set_xlabel("Stages")
ax.set_ylabel("Values")
```

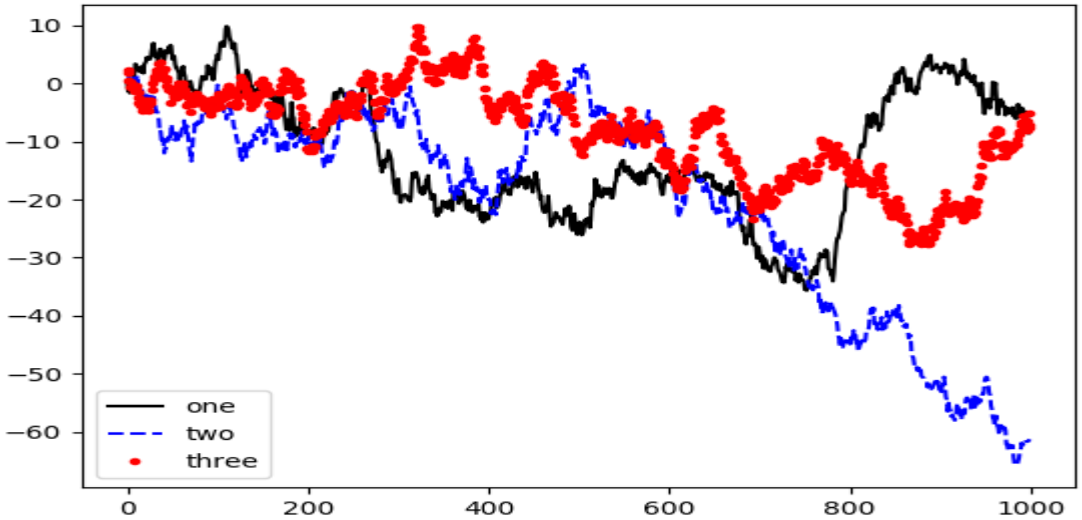


플롯 모양 변형하기 – 눈금,레이블 및 범례 수정

- **axes에 범례(legend) 입력**

ax.legend() 함수를 실행하면, axes 상에 범례가 표시됨. **loc="best"** 는 최적의 위치에 범례를 자동배치

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(np.random.randn(1000).cumsum(), 'k', label="one")
ax.plot(np.random.randn(1000).cumsum(), "b--", label="two")
ax.plot(np.random.randn(1000).cumsum(), "r.", label="three")
ax.legend(loc="best")
```



matplotlib에서 사용 가능한 주요 color, marker, linestyle 값

- matplotlib에서 사용 가능한 주요 color,marker,linestyle 값

사용 가능한 주요 color 값

값	색상
"b"	blue
"g"	green
"r"	red
"c"	cyan
"m"	magenta
"y"	yellow
"k"	black
"w"	white

사용 가능한 주요 marker 값

값	마킹
"."	point
","	pixel
"o"	circle
"v"	triangle_down
"^"	triangle_up
"<"	triangle_left
">"	triangle_right
"8"	octagon
"s"	square
"p"	pentagon
"*"	star
"h"	hexagon
"+"	plus
"x"	x
"D"	diamond

사용 가능한 주요 linestyle 값

값	라인 스타일
"-"	solid line
"--"	dashed line
"-."	dash-dotted line
":"	dotted line
"None"	draw nothing

matplotlib를 사용한 데이터 시각화 : Game of Thrones 데이터셋 분석

Game of Thrones 데이터셋의 주요 컬럼 요약 (battles.csv)

1) name: String variable. The name of the battle.
2) year: Numeric variable. The year of the battle.
3) battle_number: Numeric variable. A unique ID number for the battle.
4) attacker_king: Categorical. The attacker's king. A slash indicators that the king charges over the course of the war. For example, "Joffrey/Tommen Baratheon"
5) defender_king: Categorical variable. The defender's king.
6) attacker_1: String variable. Major house attacking.
7) attacker_2: String variable. Major house attacking.
8) attacker_3: String variable. Major house attacking.
9) attacker_4: String variable. Major house attacking.
10) defender_1: String variable. Major house defending.

Game of Thrones 데이터셋의 주요 컬럼 요약 (battles.csv)

11) defender_2: String variable. Major house defending.
12) defender_3: String variable. Major house defending.
13) defender_4: String variable. Major house defending.
14) attacker_outcome: Categorical variable. The outcome from the perspective of the attacker. Categories: win, loss, draw.
15) battle_type: Categorical variable. A classification of the battle's primary type. Categories: pitched_battle, ambush, siege, razing.
16) major_death: Binary variable. If there was a death of a major figure during the battle.
17) major_capture: Binary variable. If there was the capture of the major figure during the battle.
18) attacker_size: Numeric variable. The size of the attacker's force.
19) defender_size: Numeric variable. The size of the defenders's force.

Game of Thrones 데이터셋의 주요 컬럼 요약 (battles.csv)

20) attacker_commander: String variable. Major commanders of the attackers.

21) defender_commander: String variable. Major commanders of the defener.

22) summer: Binary variable. Was it summer?

23) location: String variable. The location of the battle.

24) region: Categorical variable. The region where the battle takes place.

Categories: Beyond the Wall, The North, The Iron Islands, The Riverlands, The Vale of Arryn

25) note: String variable. Coding notes regarding individual observations.

matplotlib를 사용한 데이터 시각화 : Game of Thrones 데이터셋 분석

Game of Thrones 데이터셋의 주요 컬럼 요약 (character-deaths.csv)

1) Name: character name.
2) Allegiances: character house
3) Death Year: year character died
4) Book of Death: book character died in
5) Death Chapter: chapter character died in
6) Book Intro Chapter: chapter character was introduced in
7) Nobility: 1 is nobel, 0 is a commoner(평민)
8) GoT: Appeared in first book
9) CoK: Appeared in second book
10) SoS: Appeared in third book
11) FfC: Appeared in fourth book
12) DwD: Appeared in fifth book

참고: https://www.kaggle.com/mylesoneill/game-of-thrones

matplotlib를 사용한 데이터 시각화 : Game of Thrones 데이터셋 분석

- 필요한 library import

```
%matplotlib nbagg
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

- 파일 읽기

```
battles = pd.read_csv("data/game-of-thrones/battles.csv",sep=",")
deaths = pd.read_csv("data/game-of-thrones/character-deaths.csv",sep=",")
```

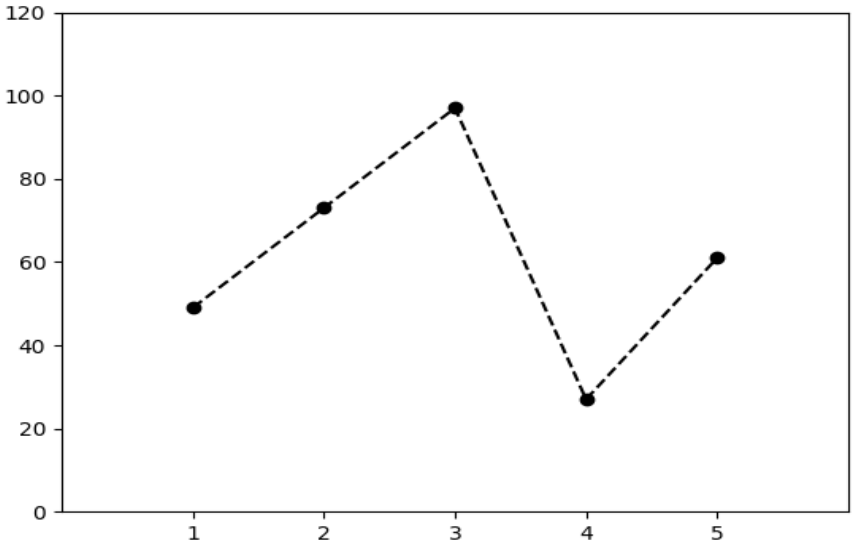
- 파일의 내용 확인

```
battles.shape        #라인수, 컬럼수 확인
battles.columns      #컬럼명 확인
battles.head()       #상위 5개 행의 데이터 확인
```

matplotlib를 사용한 데이터 시각화 : Game of Thrones 데이터셋 분석

- 라인 플롯 : 작품번호에 따른 인물들의 죽음 횟수 시각화하기

```
book_nums_to_death_count = deaths["Book of Death"].value_counts().sort_index()
ax1 = book_nums_to_death_count.plot(color="k", marker="o", linestyle="--")
ax1.set_xticks(np.arange(1, 6))
ax1.set_xlim([0, 6])
ax1.set_ylim([0, 120])
```



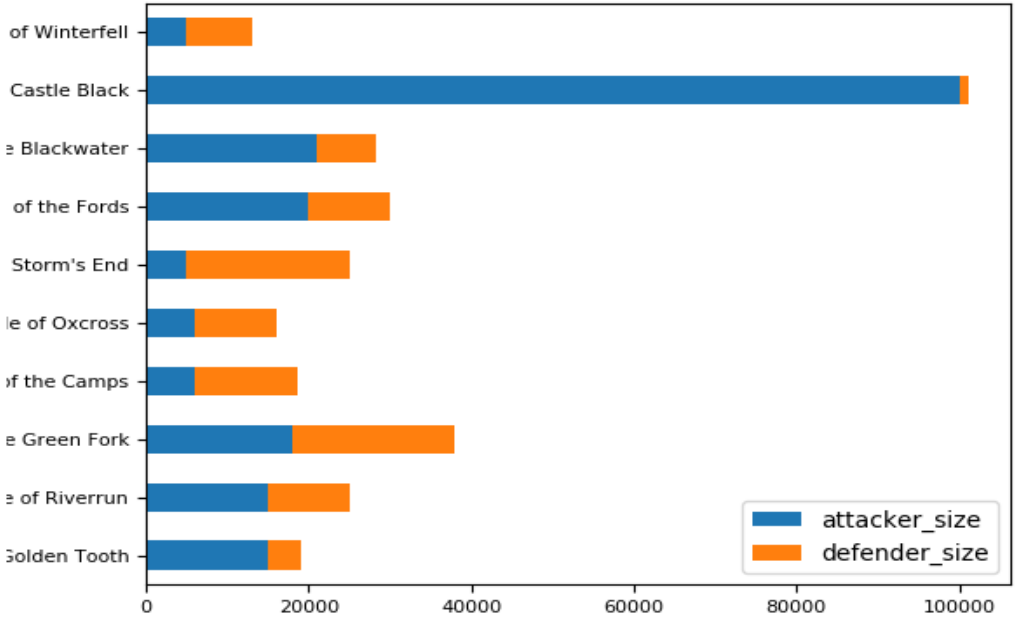
book_nums_to_death_count

```
1.0    49
2.0    73
3.0    97
4.0    27
5.0    61
Name: Book of Death, dtype: int64
```

matplotlib를 사용한 데이터 시각화 : Game of Thrones 데이터셋 분석

- 박스 플롯 : 대규모 전투 상에서 공격군과 수비군 간의 병력 차이 시각화하기

```
battles = battles.set_index(["name"])
large_battles_mask = battles["attacker_size"] + battles["defender_size"] > 10000
large_battles = battles.loc[large_battles_mask,["attacker_size", "defender_size"]]
ax2 = large_battles.plot(kind="barh", stacked=True, fontsize=8)
```



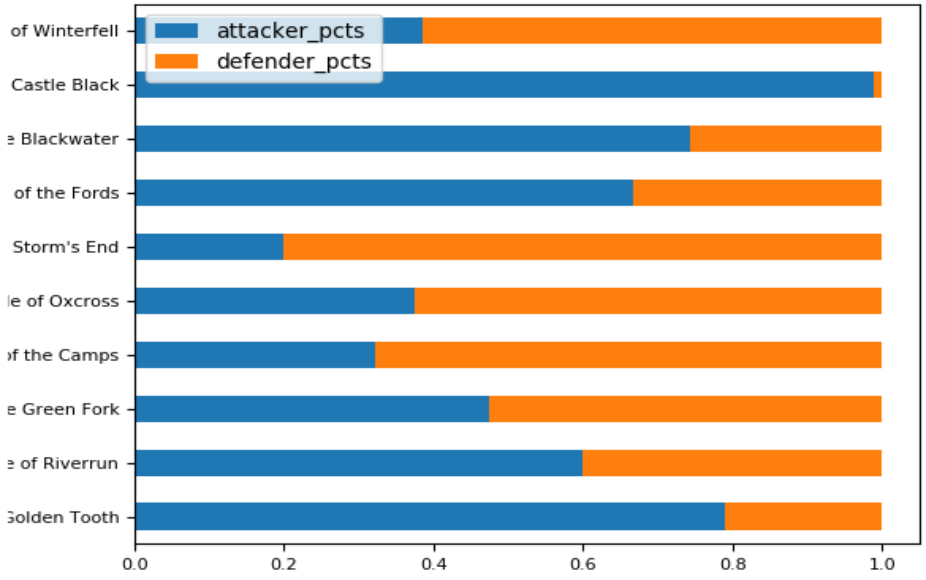
large_battles

name	attacker_size	defender_size
Battle of the Golden Tooth	15000.0	4000.0
Battle of Riverrun	15000.0	10000.0
Battle of the Green Fork	18000.0	20000.0
Battle of the Camps	6000.0	12625.0
Battle of Oxcross	6000.0	10000.0
Siege of Storm's End	5000.0	20000.0
Battle of the Fords	20000.0	10000.0
Battle of the Blackwater	21000.0	7250.0
Battle of Castle Black	100000.0	1240.0
Siege of Winterfell	5000.0	8000.0

matplotlib를 사용한 데이터 시각화 : Game of Thrones 데이터셋 분석

박스 플롯 : 대규모 전투 상에서 공격군과 수비군 간의 병력 차이 시각화하기

```
large_battles["attacker_pcts"] = large_battles["attacker_size"] /
(large_battles["attacker_size"] + large_battles["defender_size"])
large_battles["defender_pcts"] = large_battles["defender_size"] /
(large_battles["attacker_size"] + large_battles["defender_size"])
ax3 = large_battles[["attacker_pcts", "defender_pcts"]].plot(kind="barh",
stacked=True, fontsize=8)
```



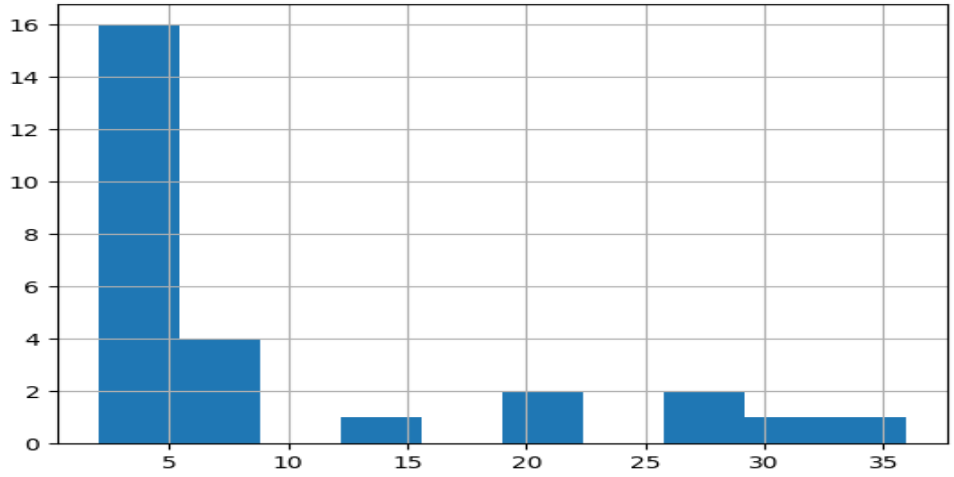

```
large_battles[["attacker_pcts", "defender_pcts"]]
```

name	attacker_pcts	defender_pcts
Battle of the Golden Tooth	0.789474	0.210526
Battle of Riverrun	0.600000	0.400000
Battle of the Green Fork	0.473684	0.526316
Battle of the Camps	0.322148	0.677852
Battle of Oxcross	0.375000	0.625000
Siege of Storm's End	0.200000	0.800000
Battle of the Fords	0.666667	0.333333
Battle of the Blackwater	0.743363	0.256637
Battle of Castle Black	0.987752	0.012248
Siege of Winterfell	0.384615	0.615385

matplotlib를 사용한 데이터 시각화 : Game of Thrones 데이터셋 분석

- 히스토그램 : 전체 전투 중 각 가문의 개입 빈도 시각화하기

```
col_names = battles.columns[4:12]
house_names = battles[col_names].fillna("None").values
house_names = np.unique(house_names)
house_names = house_names[house_names != "None"]
houses_to_battle_counts = pd.Series(0, index=house_names)
for col in col_names:
    houses_to_battle_counts =
           houses_to_battle_counts.add(battles[col].value_counts(), fill_value=0)
ax5 = houses_to_battle_counts.hist(bins=10)
```




```
Baratheon                        22.0
Blackwood                         2.0
Bolton                            8.0
Bracken                           2.0
Brave Companions                  6.0
Brotherhood without Banners       2.0
Darry                             4.0
Free folk                         2.0
Frey                              8.0
Giants                            2.0
Glover                            4.0
Greyjoy                          22.0
Karstark                          4.0
Lannister                        36.0
Mallister                         2.0
Mormont                           4.0
Night's Watch                     2.0
Stark                            32.0
Thenns                            2.0
Tully                            14.0
Tyrell                            4.0
dtype: float64
```

6. 쉽고 강력한 데이터 분석 라이브러리
- pandas의 merge, groupby 기능 사용하기

여러 개의 DataFrame 합치기

- **merging(병합)**

  두 DataFrame에 공통적으로 포함되어 있는 하나의 열을 기준으로 삼아, 해당 열의 값이 동일한 두 개의 행들을 하나의 행으로 합치는 방식

- **concatenating(연결)**

  단순히 하나의 DataFrame에 다른 DataFrame을 연속적으로 붙이는 방식

Merging DataFrame – 열 기준 merging

열(column) 기준 merging

df1과 df2는 각각 "key"라는 열을 공통적으로 포함하고 있습니다. pd.merge() 함수를 사용하여, "key" 열을 기준으로 하여 두 DataFrame에 대한 merging을 수행합니다.

```
df1 = pd.DataFrame({"key": list("bbacaab"),
                    "data1": range(7)})
df2 = pd.DataFrame({"key": list("abd"),
                    "data2": range(3)})
```

```
pd.merge(df1, df2, on="key")
```

SQL구문의 'inner join' 연산과 유사함

df1

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

df2

	data2	key
0	0	a
1	1	b
2	2	d

pd.merge(df1, df2, on="key")

	data1	key	data2
0	0	b	1
1	1	b	1
2	6	b	1
3	2	a	0
4	4	a	0
5	5	a	0

Merging DataFrame – 열 기준 merging

- **열(column) 기준 merging**

pd.merge() 함수의 how 속성값을 입력하면, 다른 조합 방식에 따라 새로운 행들을 생성합니다.

df1

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

df2

	data2	key
0	0	a
1	1	b
2	2	d

pd.merge(df1, df2, on="key", how="outer")

SQL구문의 'full outer join' 연산과 유사함

	data1	key	data2
0	0.0	b	1.0
1	1.0	b	1.0
2	6.0	b	1.0
3	2.0	a	0.0
4	4.0	a	0.0
5	5.0	a	0.0
6	3.0	c	NaN
7	NaN	d	2.0

pd.merge(df1, df2, on="key", how="left")

SQL구문의 'left outer join' 연산과 유사함

	data1	key	data2
0	0	b	1.0
1	1	b	1.0
2	2	a	0.0
3	3	c	NaN
4	4	a	0.0
5	5	a	0.0
6	6	b	1.0

Merging DataFrame – 인덱스 기준 merging

■ 인덱스(index) 기준 merging

인덱스를 기준으로 merging 하고자 할 때는 left1에서는 "key" 열을, right1에서는 index를 기준으로 merging을 수행합니다.

```
left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
'value': range(6)})
right1 = pd.DataFrame({'group_val': [3.5, 7]},
index=['a', 'b'])
```

```
pd.merge(left1, right1,
   left_on="key", right_index=True)
```

left1

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

right1

	group_val
a	3.5
b	7.0

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

Merging DataFrame – 인덱스 기준 merging

인덱스(index) 기준 merging

pd.merge() 함수의 how= 'outer'로 입력하면, 다른 조합 방식에 따라 새로운 행들을 생성합니다.

```
left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
    index=['a', 'c', 'e'], columns=['Seoul', 'Incheon'])
right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13,
14]], index=['b', 'c', 'd', 'e'], columns=['Daegu', 'Ulsan'])
```

```
pd.merge(left2, right2, how="outer",
  left_index=True, right_index=True)
```

left2

	Seoul	Incheon
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

right2

	Daegu	Ulsan
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

	Seoul	Incheon	Daegu	Ulsan
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

Concatenating DataFrame

Concatenating (연결)

Concatenating은 하나의 DataFrame에 다른 DataFrame을 행 방향 또는 열 방향으로 단순 연결합니다.

```
s1 = pd.Series([0, 1], index=["a", "b"])
s2 = pd.Series([2, 3, 4], index=["c", "d", "e"])
s3 = pd.Series([5, 6], index=["f", "g"])
```

1차원적으로 연결

```
pd.concat([s1, s2, s3])
```

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

axis=1 인자를 명시하면, 2차원

```
pd.concat([s1, s2, s3], axis=1)
```

	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

axis=1 인자, 컬럼명 설정

```
pd.concat([s1, s2, s3], axis=1,
 keys=["one", "two", "three"])
```

	one	two	three
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

계층적 인덱싱

Hierarchical Indexing Dataframe 정의

DataFrame의 인덱스와 컬럼에 2차원 리스트 혹은 2차원 numpy array 등을 입력하게 되면, 두 층을 가지는 인덱스 혹은 컬럼이 생성됩니다. 바깥쪽의 인덱스와 컬럼 부터 순서대로 붙이면 됩니다.

```
df = pd.DataFrame(np.arange(12).reshape((4, 3)),
                  index=[["a", "a", "b", "b"], [1, 2, 1, 2]],
                  columns=[["Seoul", "Seoul", "Busan"],
                           ["Green", "Red", "Green"]])
```

```
df.index.names = ["key1", "key2"]
df.columns.names = ["city","color"]
```

df

		Seoul		Busan
		Green	Red	Green
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

df

city		Seoul		Busan
color		Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

계층적 인덱싱

Hierarchical Indexing Dataframe 조회

df["Seoul"]을 실행하면 첫 번째 층의 컬럼이 생성되고, df["Seoul","Green"]을 실행하면 "Seoul" 컬럼 하위 층의 "Green" 컬럼에 해당하는 열을 Series 형태로 얻습니다.

```
df
```

	city	Seoul		Busan
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

```
df["Seoul"]
```

	color	Green	Red
key1	key2		
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10

```
df[("Seoul","Green")]
```

```
key1  key2
a     1     0
      2     3
b     1     6
      2     9
Name: (Seoul, Green), dtype: int32
```

```
df.loc["a"]
```

city	Seoul		Busan
color	Green	Red	Green
key2			
1	0	1	2
2	3	4	5

DataFrame의 컬럼-인덱스 간 변환

■ 컬럼을 인덱스로 변환

기존에 DataFrame에서 특정 열에 포함되어 있던 값을 계층적 인덱스로 변환 할 수 있으며, df2 DataFrame에서 'c'와 'd' 열의 값을 계층적 인덱스로 변환하고자 할 경우, set_index() 함수를 사용합니다.

```
df2 = pd.DataFrame({'a': range(7),
                    'b': range(7, 0, -1),
                    'c': ['one', 'one', 'one', 'two',
                          'two', 'two', 'two'],
                    'd': [0, 1, 2, 0, 1, 2, 3]})
```

```
df3 = df2.set_index(["c", "d"])
```

```
df4 = df2.set_index(["c", "d"], drop=False)
```

df2

	a	b	c	d
0	0	7	one	0
1	1	6	one	1
2	2	5	one	2
3	3	4	two	0
4	4	3	two	1
5	5	2	two	2
6	6	1	two	3

df3

		a	b
c	d		
one	0	0	7
	1	1	6
	2	2	5
two	0	3	4
	1	4	3
	2	5	2
	3	6	1

df4

		a	b	c	d
c	d				
one	0	0	7	one	0
	1	1	6	one	1
	2	2	5	one	2
two	0	3	4	two	0
	1	4	3	two	1
	2	5	2	two	2
	3	6	1	two	3

Reshaping DataFrame

Reshaping DataFrame

df.stack() 함수를 사용하면, DataFrame의 최하위 컬럼이 현재 DataFrame의 최하위 인덱스로 붙습니다.

만약 컬럼이 한 층인 경우, 이를 최하위 인덱스로 붙이면서 Series 형태로 변화합니다.

df.unstack() 함수는 df.stack() 함수와 반대의 작용을 하며, DataFrame상의 최하위 인덱스를 최하위 컬럼으로 올려 보냅니다.

```
df4 = pd.DataFrame(np.arange(6).reshape((2, 3)),
                   index=['Seoul', 'Busan'],
                   columns=['one', 'two', 'three'])
df4.index.name = "city"
df4.columns.name = "number"
```

```
df5 = df4.stack()
```

```
df6 = df5.unstack()
```

df4

number	one	two	three
city			
Seoul	0	1	2
Busan	3	4	5

df6

number	one	two	three
city			
Seoul	0	1	2
Busan	3	4	5

df5

```
city   number
Seoul  one       0
       two       1
       three     2
Busan  one       3
       two       4
       three     5
dtype: int32
```

데이터 그룹화 이해하기

- **그룹화의 흐름 : split-apply-combine**

  pandas에서의 모든 그룹화의 흐름은 "**split-apply-combine**""이라는 키워드로 표현하기도 합니다.
  그룹화하고자 하는 열의 값을 기준으로 데이터를 나누고(split), 각 그룹에 대한 통계 함수를 적용하여 (apply), 최종적인 통계량을 산출한 결과를 하나로 통합하여 표시하는(combine) 과정을 거칩니다.

- **그룹화 함수 : groupby()**

  데이터 그룹화를 하기 위해서 Series 혹은 DataFrame에 대하여 .groupby() 함수를 사용합니다.
  groupby()를 수행하면 결과로 SeriesGroupBy 객체가 생성되어 진다.
  SeriesGroupBy 객체에 mean()과 같은 함수를 적용하면 그때 그룹화된 그룹에 대하여 평균을 산출한 결과를 얻을 수 있습니다.

데이터 그룹화 이해하기

- **그룹화 함수 : Series 객체의 groupby() 사용**

df의 "key1" 열의 값을 기준으로 데이터를 먼저 그룹화를 진행한 뒤에, grouped 변수에 .mean() 함수를 적용하면, "key1" 열의 값을 기준으로 그룹화 된 각각의 그룹에 대하여 "data1" 열 값의 평균을 산출한 결과를 얻을 수 있습니다.

```
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                   'key2' : ['one', 'two', 'one', 'two', 'one'],
                   'data1': np.random.randn(5),
                   'data2': np.random.randn(5)})
grouped = df["data1"].groupby(df["key1"])
grouped.mean()
```

df

	data1	data2	key1	key2
0	-0.912105	1.596654	a	one
1	0.081429	0.828674	a	two
2	0.191187	-0.474726	b	one
3	0.994953	1.035298	b	two
4	1.494394	-0.323296	a	one

grouped

```
<pandas.core.groupby.SeriesGroupBy object
```

grouped.mean()

```
key1
a    0.221239
b    0.593070
Name: data1, dtype: float64
```

데이터 그룹화 이해하기

그룹화 함수 : DataFrame 객체의 groupby() 사용

Series 객체 뿐만 아니라 DataFrame 객체에서도 groupby() 함수를 제공한다.

```
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                   'key2' : ['one', 'two', 'one', 'two', 'one'],
                   'data1': np.random.randn(5),
                   'data2': np.random.randn(5)})
df.groupby("key1").mean()
df.groupby("key1").count()
df.groupby(["key1", "key2"]).mean()
df.groupby(["key1", "key2"]).count()
```

`df.groupby("key1").mean()`

key1	data1	data2
a	0.221239	0.700677
b	0.593070	0.280286

`df.groupby("key1").count()`

key1	data1	data2	key2
a	3	3	3
b	2	2	2

`df.groupby(["key1", "key2"]).mean()`

key1	key2	data1	data2
a	one	0.291144	0.636679
	two	0.081429	0.828674
b	one	0.191187	-0.474726
	two	0.994953	1.035298

`df.groupby(["key1", "key2"]).count()`

key1	key2	data1	data2
a	one	2	2
	two	1	1
b	one	1	1
	two	1	1

데이터 그룹화 이해하기

그룹에 대해 반복문 사용하기

그룹에 대해 반복문을 수행하면 그룹화된 결과물에 대한 확인이 가능합니다.

df.groupby("key1")을 실행한 결과를 반복문에서 순회할 대상으로 명시한 뒤,

name과 **group** 변수에서 값을 받아서 출력하면, 그룹화 기준 열의 값과 그룹화된 결과물을 확인 가능

```
for name, group in df.groupby("key1"):
    print(name)
    print(group)
```

```
a
      data1     data2 key1 key2
0 -0.912105  1.596654    a  one
1  0.081429  0.828674    a  two
4  1.494394 -0.323296    a  one
b
      data1     data2 key1 key2
2  0.191187 -0.474726    b  one
3  0.994953  1.035298    b  two
```

```
for (k1, k2), group in df.groupby(["key1","key2"]):
    print(k1, k2)
    print(group)
```

```
a one
      data1     data2 key1 key2
0 -0.912105  1.596654    a  one
4  1.494394 -0.323296    a  one
a two
      data1     data2 key1 key2
1  0.081429  0.828674    a  two
b one
      data1     data2 key1 key2
2  0.191187 -0.474726    b  one
b two
      data1     data2 key1 key2
3  0.994953  1.035298    b  two
```

데이터 그룹화 이해하기

Dictionary 혹은 Series 기준으로 그룹화하기

그룹화를 수행하기 위한 기준으로, 별도로 정의된 딕셔너리 혹은 Series를 사용할 수 있습니다.

df2를 생성할 때 **map_dict**라는 딕셔너리도 함께 생성하였는데, 이는 컬럼을 다른 문자열로 매핑하는 역할을 합니다. 'a'부터 'e'까지의 열을 **map_dict**에 정의된 'red'와 'blue'로 그룹화한 뒤, 합계를 산출함

```
df2 = pd.DataFrame(np.random.randn(5, 5),
                   columns=['a', 'b', 'c', 'd', 'e'],
                   index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
map_dict = {'a': 'red', 'b': 'red', 'c': 'blue',
            'd': 'blue', 'e': 'red', 'f' : 'orange'}
df2.groupby(map_dict, axis=1).sum()
```

	blue	red
Joe	-0.985825	-0.862905
Steve	0.955064	0.898309
Wes	-0.686721	0.572196
Jim	0.533874	-1.294048
Travis	-0.536213	-1.791445

데이터 그룹화 이해하기

- 그룹에 적용할 수 있는 주요 통계함수

함수	설명
count	각 그룹 내의 (NaN이 아닌) 값들의 갯수를 계산
sum	각 그룹 내의 (NaN이 아닌) 값들의 합을 계산
mean	각 그룹 내의 (NaN이 아닌) 값들의 평균을 계산
median	각 그룹 내의 (NaN이 아닌) 값들 중 중간값을 반환
std, var	각 그룹 내의 값들의 표준편차, 분산을 계산
min, max	각 그룹 내의 값들 중 최솟값, 최댓값을 반환
prod	각 그룹 내의 (NaN이 아닌) 값들 전체의 곱을 계산
first, last	각 그룹 내의 (NaN이 아닌) 값들 중 맨 첫번째, 맨 마지막 값을 반환

데이터 그룹화 이해하기

■ 그룹에 사용자 정의 함수 사용하기

그룹화 결과 얻어진 그룹에 대하여 사용자가 정의한 집계 함수를 사용할 수도 있습니다.

"key1" 열의 값을 기준으로 그룹화한 뒤, 이를 **grouped** 변수에 저장합니다.

peak_to_peak() 사용자 정의 함수는 그룹 내 각 열의 최댓값에서 최솟값을 뺀 값을 계산합니다

.agg() 함수를 사용하면, 그룹에 대하여 사용자가 정의한 집계 함수를 적용할 수 있습니다.

```
grouped = df.groupby("key1")
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

df

	data1	data2	key1	key2
0	-0.912105	1.596654	a	one
1	0.081429	0.828674	a	two
2	0.191187	-0.474726	b	one
3	0.994953	1.035298	b	two
4	1.494394	-0.323296	a	one

grouped.agg(peak_to_peak)

key1	data1	data2
a	2.406499	1.919949
b	0.803766	1.510024

2016 US Election 데이터셋의 주요 컬럼 요약 (county_facts.csv)

1) "RHI125214: White alone, percent, 2014
2) "RHI225214": Black or African American alone, percent, 2014
3) "RHI325214": "American Indian and Alaska Native alone, percent, 2014
4) "RHI425214": Asian alone, percent, 2014
5) "RHI525214": Native Hawaiian and Other Pacific Islander alone, percent, 2014
6) "RHI625214": Two or More Races, percent, 2014
7) "RHI725214": Hispanic or Latino, percent, 2014
8) "RHI825214": White alone, not Hispanic or Latino, percent, 2014

pandas의 그룹화 기능을 사용한 데이터 분석 : 2016 US Election 데이터셋 분석

2016 US Election 데이터셋의 주요 컬럼 요약 (primary_results.csv)

1) state: state where the primary or caucus was held
2) state_abbreviation: two letter state abbreviation
3) county: county where the results come from
4) fips: FIPS county code
5) party: Democrat or Republican
6) candidate: name of the candidate
7) votes: number of votes the candidate received in the corresponding state and county (may be missing)
8) fraction_votes: fraction of votes the president received in the corresponding state, county, and primary

* 참고: https://www.kaggle.com/benhamner/2016-us-election

pandas의 그룹화 기능을 사용한 데이터 분석 : 2016 US Election 데이터셋 분석

- 필요한 library import

```
%matplotlib nbagg
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

- 파일 읽기

```
primary = pd.read_csv("data/2016_presidential_election/primary_results.csv",sep=",")
counties = pd.read_csv("data/2016_presidential_election/county_facts.csv",sep=",")
```

- 파일의 내용 확인

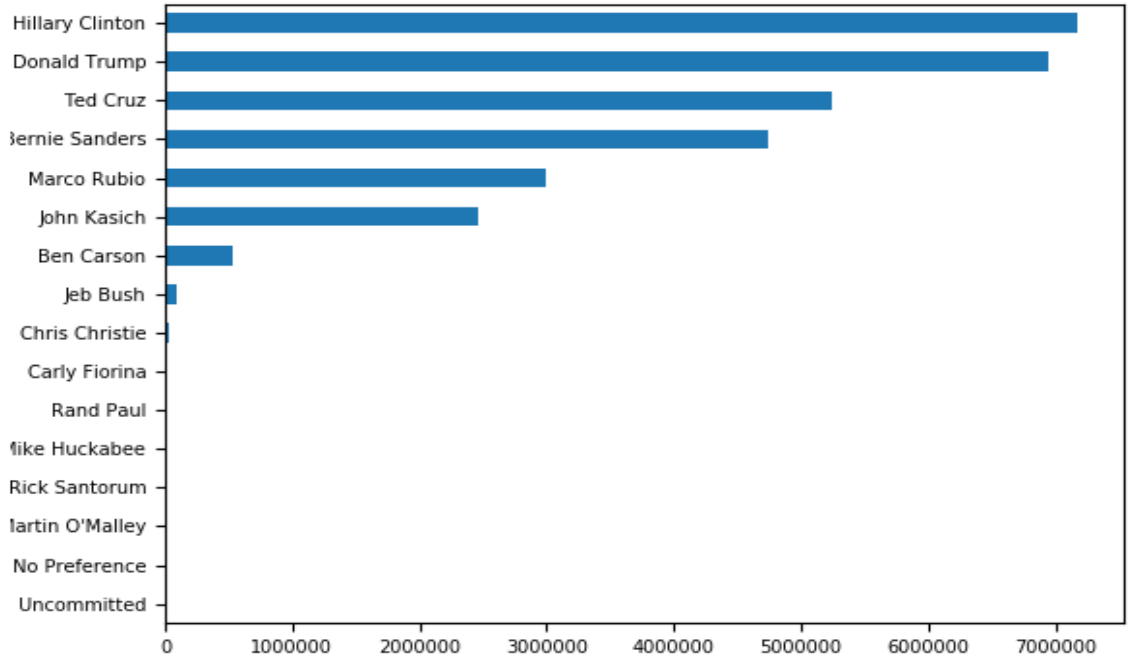
```
primary.shape        #라인수, 컬럼수 확인
primary.columns      #컬럼명 확인
primary.head()       #상위 5개 행의 데이터 확인
```

pandas의 그룹화 기능을 사용한 데이터 분석 : 2016 US Election 데이터셋 분석

- 각 후보 별 전체 득표수 계산하기

```
candidate_to_votes_s = primary.groupby("candidate")["votes"].sum().sort_values()
candidate_to_votes_s.plot(kind="barh", fontsize=7)
```

```
candidate
 Uncommitted          43
 No Preference       313
Martin O'Malley      747
Rick Santorum       1773
Mike Huckabee       3300
Rand Paul           8460
Carly Fiorina      15181
Chris Christie     24347
Jeb Bush           94394
Ben Carson        528463
John Kasich      2456406
Marco Rubio      2998335
Bernie Sanders   4740278
Ted Cruz         5248807
Donald Trump     6944654
Hillary Clinton  7178257
Name: votes, dtype: int64
```

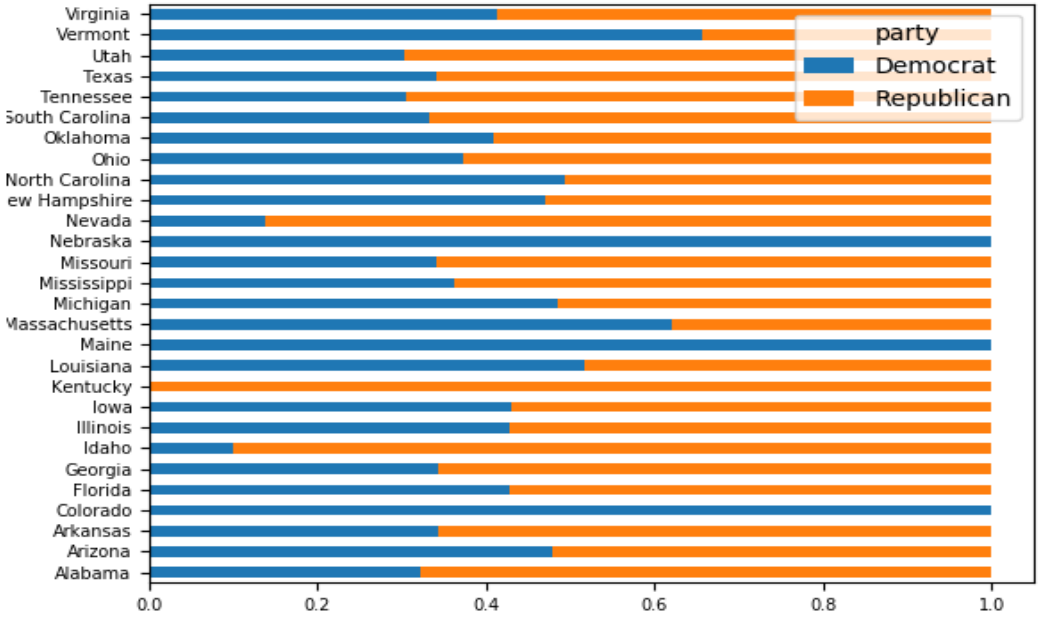


pandas의 그룹화 기능을 사용한 데이터 분석 : 2016 US Election 데이터셋 분석

- 각 주별, 각 정당의 득표 비율 계산하기

```
state_party_to_votes_s = primary.groupby(["state", "party"])["votes"].sum()
state_to_votes_s = primary.groupby("state")["votes"].sum()
state_party_to_vote_pcts_s = state_party_to_votes_s / state_to_votes_s
state_party_to_vote_pcts_s.unstack().plot(kind="barh", stacked=True, fontsize=8)
```

state	party	
Alabama	Democrat	0.321491
	Republican	0.678509
Arizona	Democrat	0.478419
	Republican	0.521581
Arkansas	Democrat	0.343990
	Republican	0.656010
Colorado	Democrat	1.000000
Florida	Democrat	0.427721
	Republican	0.572279
Georgia	Democrat	0.342714
	Republican	0.657286
Idaho	Democrat	0.100263
	Republican	0.899737
Illinois	Democrat	0.427667
	Republican	0.572333
Iowa	Democrat	0.429540
	Republican	0.570460



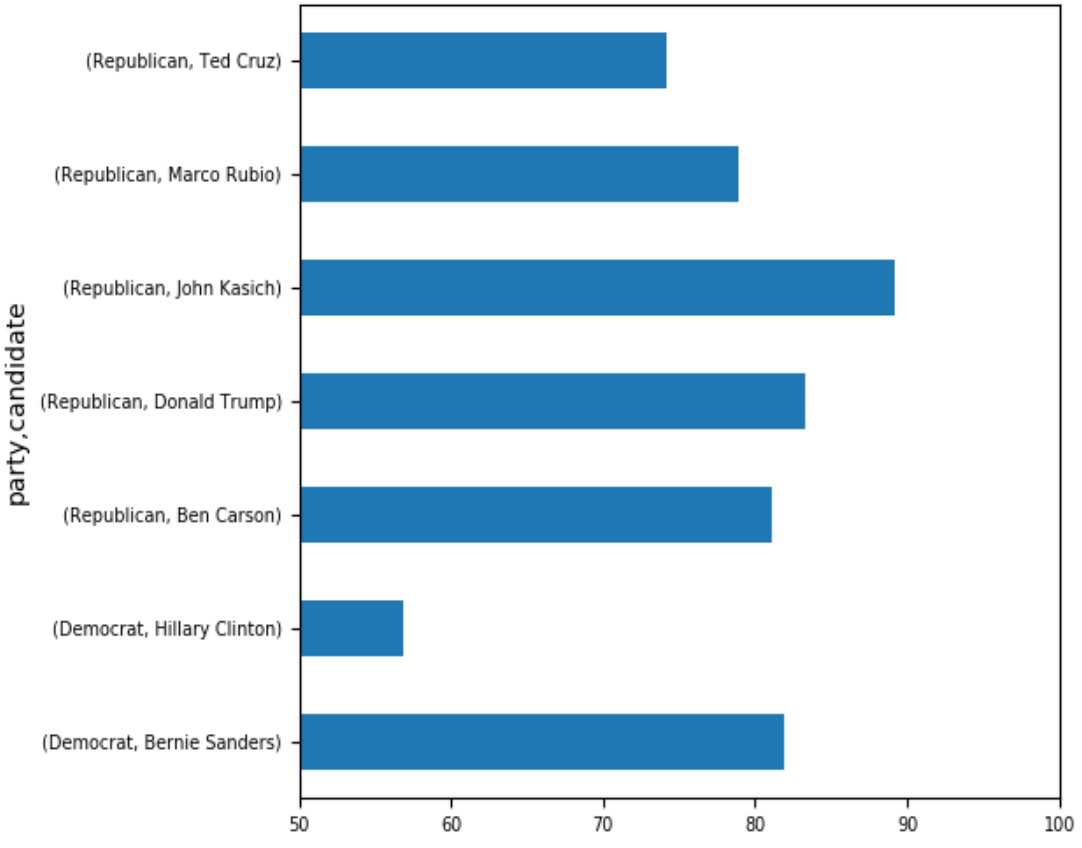
pandas의 그룹화 기능을 사용한 데이터 분석 : 2016 US Election 데이터셋 분석

- 각 후보가 당선된 county의 평균 백인 유권자 비율 조사하기

```
#fips별 최대 득표수 가져오기
func = lambda agg_df: agg_df.sort_values("votes", ascending=False).iloc[0]
winners = primary.groupby("fips").agg(func)
#primary와 couties 데이터 프레임을 merge 함
winners_county_races = pd.merge(winners, counties[["fips", "RHI825214"]],
left_index=True, right_on="fips", how="left")
# RHI825214 컬럼명을 white_pcts로 변경함
winners_county_races =
winners_county_races.rename(columns={"RHI825214":"white_pcts"})
#party별, candidate별 득표수의 평균값
winners_county_white_pcts = winners_county_races.groupby(["party",
"candidate"])[“votes"].mean()
#barh plot 그리기
ax = winners_county_white_pcts.plot(kind="barh", fontsize=7)
ax.set_xlim([50, 100])
plt.tight_layout()
```

pandas의 그룹화 기능을 사용한 데이터 분석 : 2016 US Election 데이터셋 분석

- 각 후보가 당선된 county의 평균 백인 유권자 비율 조사하기




```
party       candidate
Democrat    Bernie Sanders      81.944030
            Hillary Clinton     56.856920
Republican  Ben Carson          81.100000
            Donald Trump        83.235638
            John Kasich         89.226415
            Marco Rubio         78.883333
            Ted Cruz            74.164665
Name: white_pcts, dtype: float64
```

Pivot Table이란?

'**피벗 테이블(pivot table)**'을 사용하면, 데이터셋에 대하여 그룹화에 기반한 통계량을 계산하는 작업이 더 간편해집니다.

pivot_table() 함수

피벗 테이블은 df.pivot_table() 함수를 사용하여 생성합니다. values 인자로는 통계량을 계산할 열을, index와 columns 인자는 각각 그 값을 피벗 테이블의 인덱스와 컬럼으로 사용할 열을, aggfunc 인자로는 적용할 통계 함수를 명시합니다.

```
total_votes = primary.pivot_table(values="votes", index="state",
                                  columns="candidate", aggfunc="sum",
                                  fill_value=0)
```

total_votes라는 피벗 테이블은 "state"와 "candidate" 열의 값을 그룹화 기준으로 하여, "votes" 열의 값의 합계를 산출한 결과입니다. 이 때 fill_value=0 인자를 넣어주면, NaN이 나올 부분이 0으로 대체됩니다. 피벗 테이블도 DataFrame 형태를 지닙니다.

pandas의 그룹화 기능을 사용한 데이터 분석 : 2016 US Election 데이터셋 분석

- pivot_table() 함수

"state_abbreviation"과 "party" 열의 값을 그룹화 기준으로 하여, "fraction_votes" 열의 값의 평균을 나타내는 피벗 테이블을 만들 수 있습니다.

```
mean_frac_voates=primary.pivot_table(values="fraction_votes",
                                     index="state_abbreviation",
                                     columns="party", aggfunc="mean")
```

party	Democrat	Republican
state_abbreviation		
AL	0.476823	0.195277
AR	0.464784	0.191924
AZ	0.478433	0.283867
CO	0.481016	NaN
FL	0.469349	0.242413
GA	0.493525	0.196939
IA	0.250003	0.090857
ID	0.494733	0.240773
IL	0.489632	0.242013

7.머신러닝 이란?

머신러닝이란?

- 머신러닝은 무엇일까? 어떤 때는 통계학 같고, 어떤 때는 알고리즘 같고, 어떤 때는 프로그래밍 같이 들립니다. 이는 머신러닝 이라는 말 자체가 굉장히 넓은 의미를 가지고 있기 때문입니다.
- 머신러닝은 한 마디로

  " 데이터를 이용해서 명시적으로 정의되지 않은 패턴을 컴퓨터로 학습하여 결과를 만들어 내는 학문분야 "
- 이 정의는 1959년 아서 사무엘(Arthur Lee Samuel)이 정의와 비슷

  ' 직접적으로 프로그래밍 하지 않아도 컴퓨터가 스스로 학습할 수 있는 능력을 주는 학문 분야'
- 1959년의 정의에서는 통계학적 머신러닝(statistical machine learning)과 딥러닝(deep learning)에서 강조하는 데이터의 중요성은 미처 부각되지 않았음

머신러닝의 3가지 요소

1. 데이터

머신러닝은 항상 데이터를 기반으로 합니다.

머신러닝은 알고리즘이 아닌 데이터 학습을 통해 실행 동작이 바뀝니다. 데이터를 기반으로 한다는 점에서 통계학과 가깝다고 할 수 있음

2. 패턴인식

머신러닝은 통계학을 비롯해 딥러닝을 이용하여 데이터의 패턴을 유추하는 방법이 주축이 됩니다.

사용자가 일일 정해 놓은 패턴으로 데이터를 분석하는 것이 아니라 데이터를

보고 패턴을 추리하는 것이 머신러닝의 핵심입니다.

3. 컴퓨터를 이용한 계산

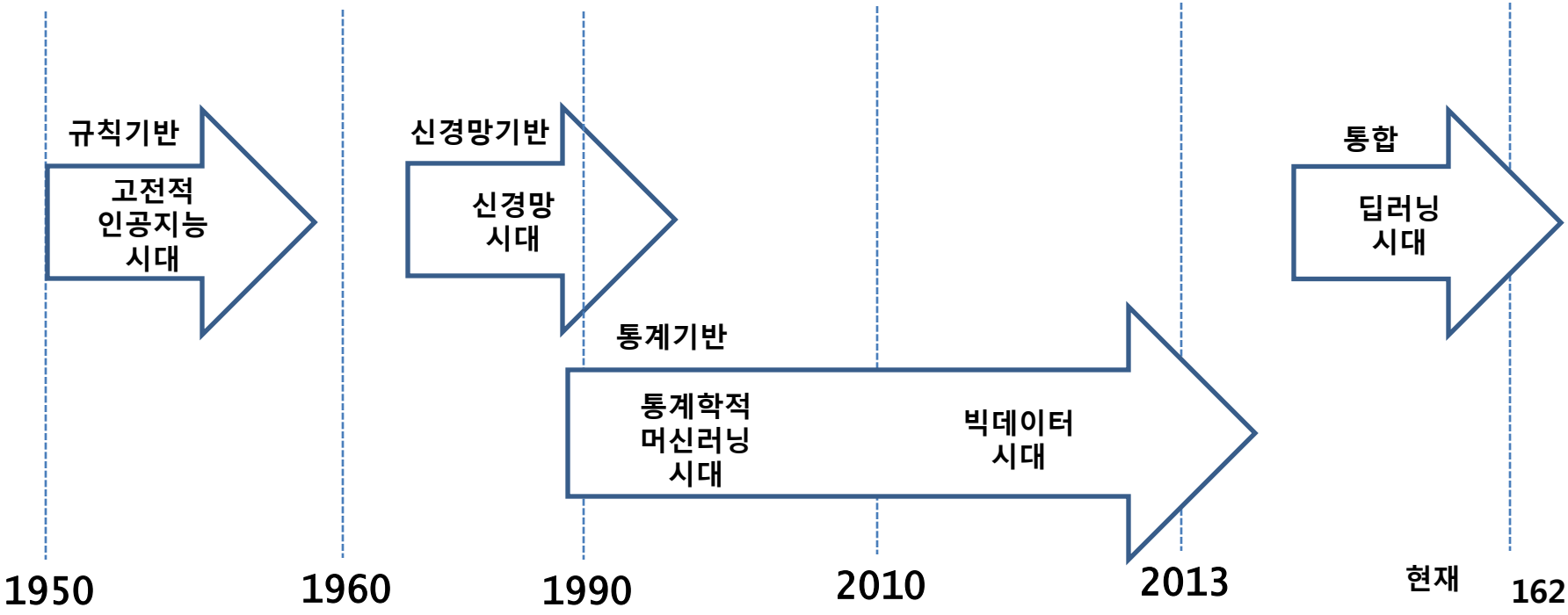
머신러닝은 데이터를 처리하고 패턴을 학습하고 계산하는데 컴퓨터를 사용합니다. 계산

그 자체도 머신러닝에서는 아주 중요합니다.

머신러닝의 발전사

■ 머신러닝의 역사와 현재 트렌드

1950년대 인공지능이라는 개념으로 태동했고, 신경망 시대를 거쳐 통계학적 머신러닝과 빅데이터 시대를 지나 지금의 딥러닝 시대에 다다랐음



머신러닝의 발전사

고전적 인공지능 시대

1950년대에는 컴퓨터의 가능성에 대한 다양한 논의가 있었음

신경망 시대

1957년에 퍼셉트론(perceptron)이라는 기초적인 신경망이 개발되어 이를 여러 개로 묶어 복잡한 신경망을 구성하면 입력과 출력을 유연하게 연결할 수 있었지만, 당시 신경망에는 여러가지 문제점이 있었음

통계학적 머신러닝 시대

1990년대 들어 통계학을 전산학과 접목시켜 대규모의 데이터에서 패턴을 찾는 시도가 기존에 비해 진일보된 성과를 내었으며, 기존의 방법과 가장 큰 차이점은 데이터에 훨씬 더 중요한 비중을 두었다는 점이다. 이 시기에 머신러닝이라는 용어가 등장했으며, 차후에 딥러닝이 나온 이후 이런 통계학에 중심을 둔 기법들을 통계학적 머신러닝이라고 함

머신러닝의 발전사

빅데이터 시대

빅데이터라는 용어가 2010년대부터 유행했는데,통계학적 머신러닝은 웹에서 쏟아지는 데이터, 대용량 저장장치, 분산처리 기술과 결합하여 시너지를 만들었음

딥러닝 시대

기존의 신경망 시대보다 훨씬 더 많은 데이터와 새로 개발된 이론을 합치자 단순히 통계학적 머신러닝만 사용하는 모델을 넘어서는 결과를 얻을 수 있게 되었음, 기존의 신경망보다 훨씬 더 복잡한 깊이가 있는 신경망을 사용하게 되었기 때문에 딥러닝이라고 부름

현재 트렌드

현재 머신러닝은 대량의 데이터를 바탕으로 하는 딥러닝 기법을 주로 사용합니다. 기존에 해결하기 힘들었던 음성인식, 번역, 이미지 인식에서 좋은 성과를 보이고 있음

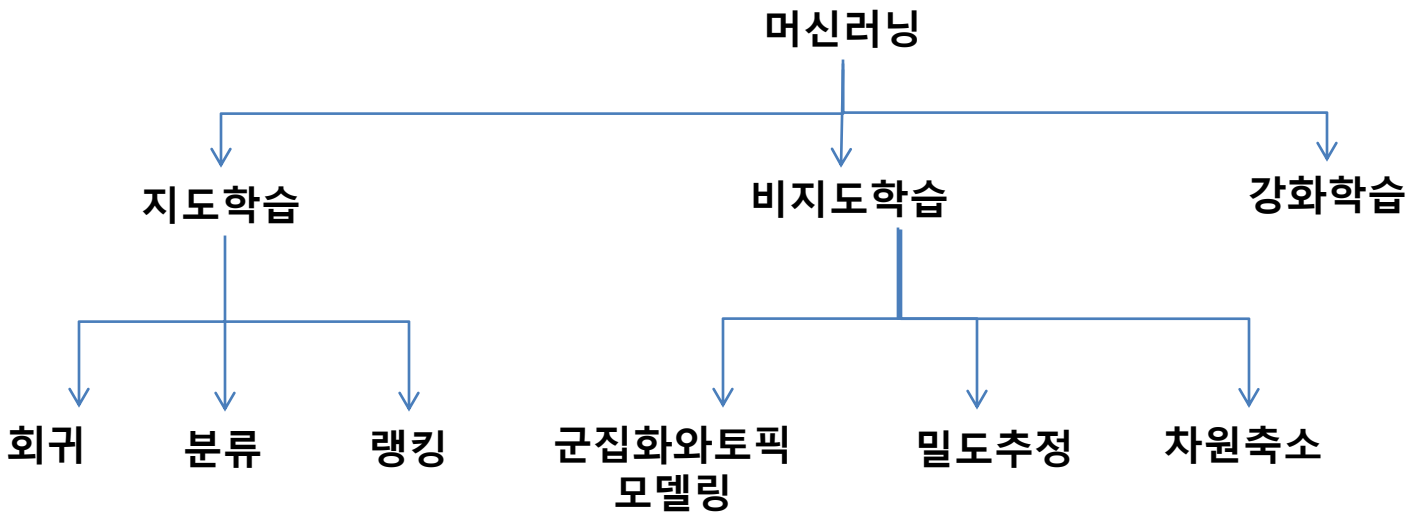
머신러닝의 분류

- **풀고자 하는 목표에 따른 분류**

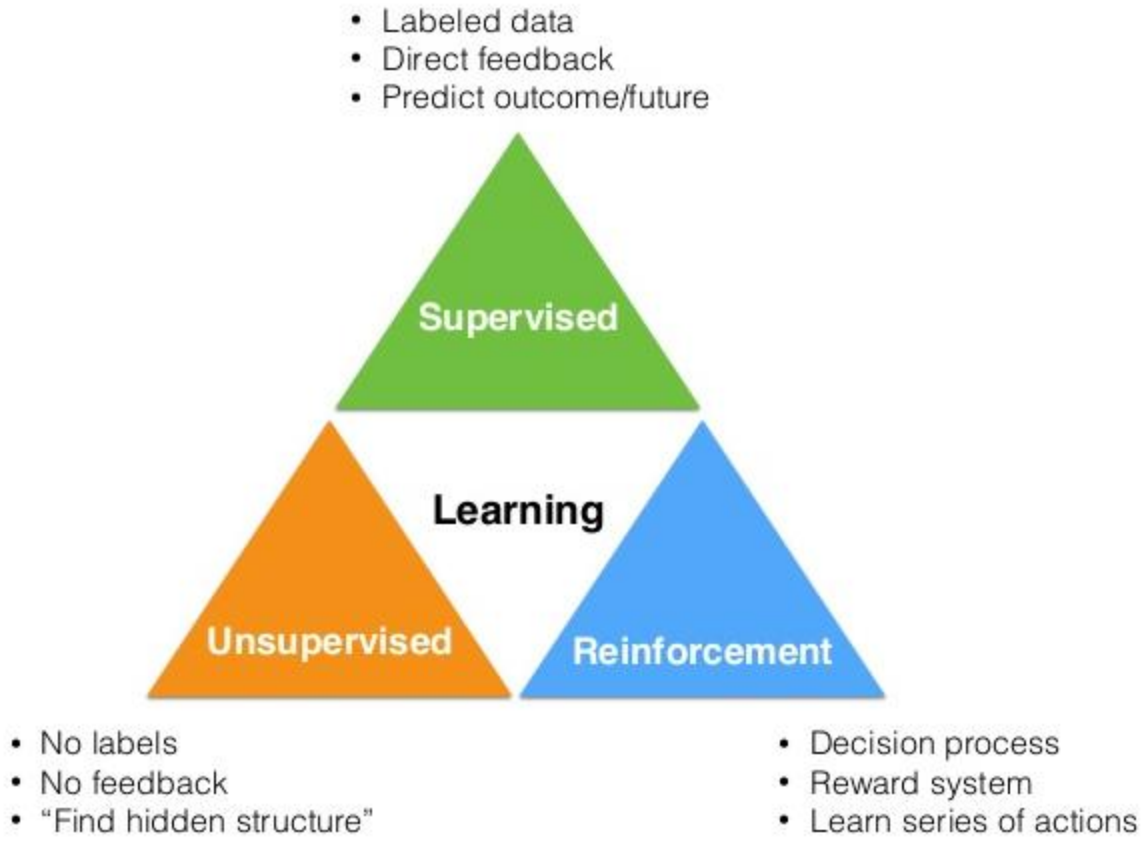
지도학습(supervised learning) : 값/레이블을 예측하는 시스템 구축

비지도학습(unsupervised learning) : 패턴 추출

강화학습(reinforcement learning) : 상호작용 가능한 시스템 구축



머신러닝의 분류


• Labeled data
• Direct feedback
• Predict outcome/future
Supervised
Learning
Unsupervised
Reinforcement
• No labels
• No feedback
• "Find hidden structure"
• Decision process
• Reward system
• Learn series of actions

머신러닝의 분류

- **1. 지도학습(**supervised learning), **지도러닝, 교사 학습**

지도학습은 주어진 데이터와 레이블(정답)을 이용해서 **미지의 상태나 값을 예측하는** 학습 방법이다.

대부분의 머신러닝은 지도학습에 해당함.

예를 들어 문서에 사용된 단어를 보고 해당문서의 카테고리 분류하기, 사용자가 구매한 상품을 토대로 다음에 구입할 상품 예측하기

- **2. 비지도 학습(**unsupervised learning), **자율러닝, 비교사 학습**

비지도학습은 데이터와 주어진 레이블 간의 관계를 구하는 것이 아니라 **데이터 자체에서 유용한 패턴을 찾아내는** 학습방법이다.

예를 들어 비슷한 데이터 끼리 묶는 군집화, 데이터에서 이상한 점을 찾아내는 이상검출, 데이터 분포 추측이 있다. 지도학습과 가장 다른 점은 데이터가 주어졌을 때 특정값을 계산하는 함수를 만드는 대신 데이터의 성질을 직접적으로 추측한다는 것이다.

머신러닝의 분류 - 지도학습

1. 지도학습(supervised learning)의 세부 분류와 예시

- 1.1 회귀(값 예측)와 분류(항목 선택)

  회귀(Regression)의 경우에는 숫자값을 예측함. 연속된 숫자를 예측하는데 예를 들어 기존 온도 추이를 보고 내일 온도를 예측하는 경우.

  분류(Classification)는 입력 데이터를 주어진 항목으로 나누는 방법임. 예를 들어 어떤 문서가 도서관 어떤 분류에 해당하는지 고르는 경우

- 1.2 추천 시스템과 랭킹학습 (순서배열)

  추천 시스템은 상품에 대한 사용자의 선호도를 예측하는 시스템이다. 예를 들어 영화추천 시스템에서 회귀와 다른 점은 입력과 출력이 아니라 관객과 영화, 관객과 점수 등 다양한 관계를 고려한다는 점이 다르다.

  랭킹학습(learning to rank)은 회귀에서 처럼 각 입력 데이터의 출력 값을 예측하는 것이 아니라 데이터의 순위를 예측합니다. 예를 들어 영화 평점을 가지고 특정 관객이 몇점을 줄지 예측하는 것은 회귀이고, 반면 좋아할 만한 영화 10편을 추천한다면 랭킹학습에 해당 합니다.

머신러닝의 분류 - 지도학습

- **1. 지도학습(**supervised learning)의 세부 분류와 예시
  - 1.3 지도학습의 예

    ① 편지봉투에 손으로 쓴 우편번호 숫자판별

    : 입력은 손글씨를 스캔한 이미지이고 기대하는 출력은 우편번호 숫자입니다.

    ② 의료 영상 이미지에 기반한 종양 판단

    : 입력은 이미지이고 출력은 종양이 양성인지의 여부입니다. 모델 구축에 사용할 데이터셋을 만들려면 의료영상 데이터베이스가 필요합니다. 전문가의 의견이 필요합니다.

머신러닝의 분류 - 비지도학습

2. 비지도학습(unsupervised learning)의 세부 분류와 예시

- 2.1 군집화와 토픽모델링 (비슷한 데이터를 묶음)

군집화(클러스터링 clustering)는 비슷한 데이터를 묶어서 여러 그룹으로 만드는 기법으로 예를 들어, 사용자의 취향을 그룹으로 묶어서 사용자 취향에 맞는 광고를 제공하는 경우

토픽모델링(topic modeling)은 군집화와 매우 유사하지만 주로 텍스트 데이터에 대해 사용합니다.

- 2.2 밀도 추정(데이터 분포를 예측)

밀도 추정(density estimation)은 관측한 데이터로부터 데이터를 생성한 원래의 분포를 추측하는 방법

- 2.3 차원 축소(데이터 차원을 간추림)

차원 축소(dimensionality reduction)는 데이터의 차원을 낮추는 기법으로 데이터가 복잡하고 높은 차원을 가져서 시각화하기 어려울 때 2차원이나 3차원으로 표현하기 위해 사용합니다.

머신러닝의 분류 - 비지도학습

2. 비지도학습(unsupervised learning)의 세부 분류와 예시

- 2.4 비지도학습의 예

① 고객들을 취향이 비슷한 그룹으로 묶기

: 고객 데이터를 이용해서 어떤 고객들의 취향이 비슷한지 알고 싶거나 비슷한 취향의 고객을 그룹으로 묶고 싶을 것입니다. 어떤 그룹이 있는지 미리 알 수 없고 얼마나 많은 그룹이 있는지도 모르니 출력을 가지고 있지 않습니다.

② 블로그 글의 주제 구분

: 많은 양의 텍스트 데이터를 요약하고 그 안에 담긴 핵심 주제를 찾고자 할 수 있습니다. 사전에 어떤 주제인지 알지 못하고 얼마나 많은 주제가 있는지 모릅니다. 그러므로 출력값을 준비할 수 있습니다.

머신러닝의 데이터

■ 머신 러닝 데이터 관련 용어들

- 머신러닝에서는 샘플(sample) 또는 데이터 포인트(data point)라고 부릅니다.
- **특성(Feature)** : 샘플의 속성,즉 열의 특성을 특성(feature)라고 합니다.

✓ 입력 데이터(ex: 사용자 로그, 음성, 이미지 등)를 구별해낼 수 있는 특징들을 정량화한 것

✓ 통계학에서는 '설명변수', '독립변수', '예측변수'로 표현한다.

✓ 적절한 특성의 선정이 효과적인 머신러닝을 하는데 중요한 요소임

- 좋은 입력 데이터를 만들어 내는 과정을 특성 추출(feature extraction) 혹은 특성 공학(feature engineering)이라고 한다.
- **레이블(Label)**

: 학습 데이터의 속성을 우리가 분석하고자 하는 관점에서 정의하는 것

- **지도(Supervised) 학습 & 비지도(Unsupervised) 학습**

✓ 레이블은 사람이 사진을 보고 정의한 것이기 때문에 그러한 레이블된 사진을 읽어서 학습하는 컴퓨터 입장에서는 사람으로부터 지도를 받는 것이라 하여 지도 학습

✓ 입력 데이터에 레이블이 없다면 컴퓨터가 사람으로부터 지도를 받은 것이 없기 때문에 비지도 학습이라 한다.

✓ 지도학습의 종류: 분류 모델(Classification), 예측 모델(Regression)

✓ 비지도 학습의 종류: 군집 모델(Clustering)

머신러닝의 데이터

■ 머신 러닝 데이터 관련 용어들

▪ 분류 모델 (Classification)

- ✓ kNN (k nearest neighbor)
- ✓ 서포트 벡터 머신 (Support Vector Machine)
- ✓ 의사결정 트리 (Decision Tree)
- ✓ 분류 모델의 예를 들면, A,B,C 레이블로 구성된 데이터세트가 있다고 하면 분류모델의 결과값은 A,B,C 셋 중에 하나가 나온다.

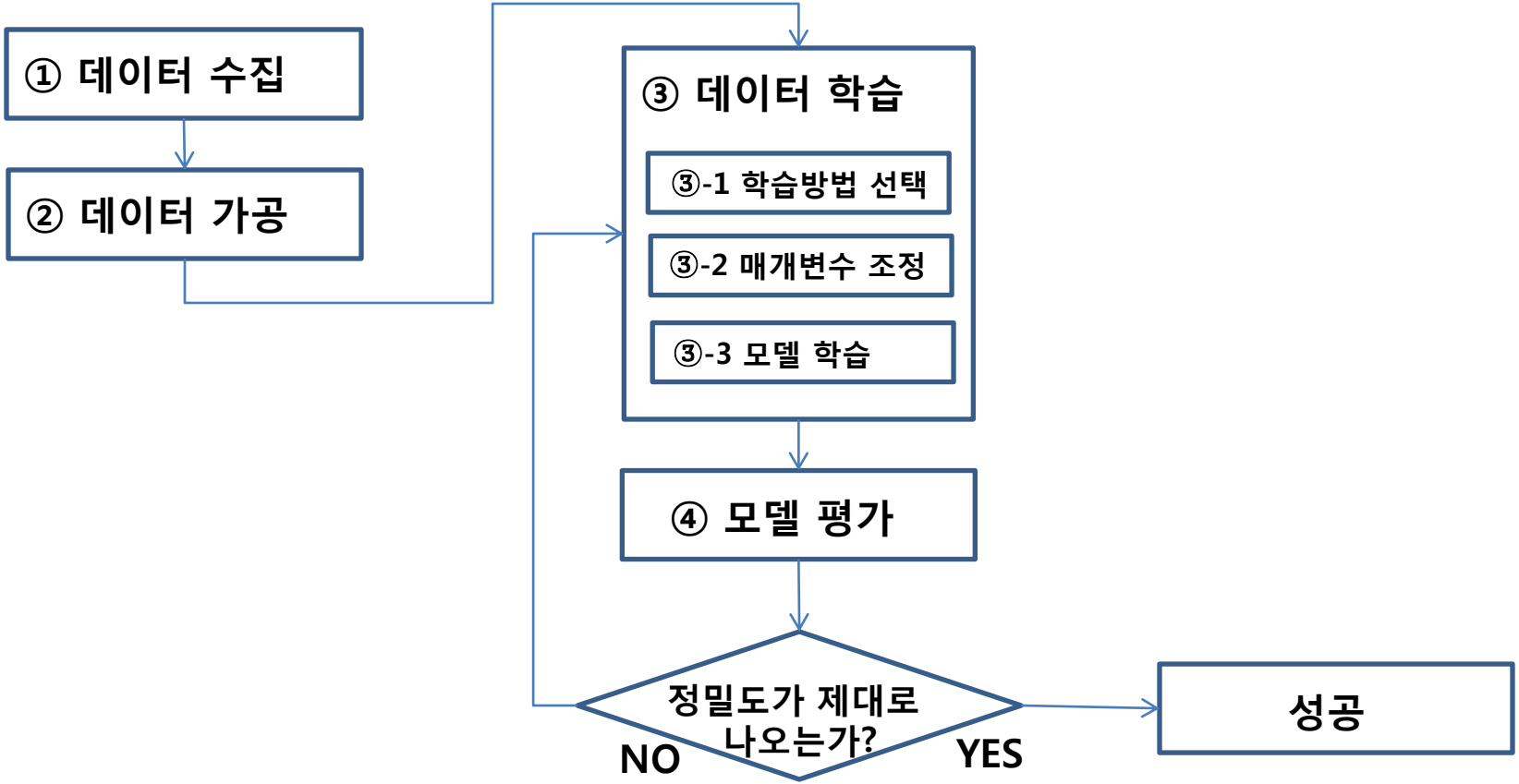
▪ 예측 모델 (Regression)

- ✓ 회귀(Regression)가 주된 방식이라 예측 모델은 회귀 모델이라고 하기도 함
- ✓ 회귀 모델은 레이블 된 학습 데이터를 가지고 특성(Feature)과 레이블의 관계를 함수식으로 표현하는 것이 목적
- ✓ 회귀 모델은 A,B,C와 같이 유한 개의 결값이 나오지 않고 어떤 값이 나올지 예상하지 못하기 때문에 예측 모델이라고 한다.
- ✓ 주가 분석과 같이 연속적인 범위 내에 결과값을 예측하는 문제에는 선형 회귀 모델(Linear Regression)을 사용

머신러닝 알고리즘 종류

지도학습(supervised)	비지도학습(unsupervised)
회귀(Regression) - Linear regression - Decision Tree - Random Forests - Neural Networks(딥러닝)	– Clustering – K means – PCA(Principal component analysis) – Density estimation – Expection maximization – Pazen window – DBSCAN
분류(classification) - Naïve-Bayes - K-NearestNeighbors(KNN) - Support Vector Machine(SVM)	

머신러닝의 흐름


① 데이터 수집
② 데이터 가공
③ 데이터 학습
③-1 학습방법 선택
③-2 매개변수 조정
③-3 모델 학습
④ 모델 평가
정밀도가 제대로 나오는가?
NO
YES
성공

머신러닝의 흐름

■ 흐름에 대한 설명

- 과정 ①에서는 데이터를 수집함. 머신러닝에서 제일 어려운 것은 바로 데이터를 수집하는 것이다. 이때 어느 정도의 양의 데이터를 확보해야 합니다.
- 과정 ②에서는 프로그램이 다루기 쉬운 형태로 데이터를 가공해야 합니다. 이때 어떤 형식으로 가공해야 할지를 생각해야 합니다.
- 과정 ③에서는 실제로 데이터를 학습시킵니다.

③-1에서는 어떤 방법(알고리즘)을 사용해 학습을 시킬지 생각해야 합니다.

③-2에서는 데이터와 알고리즘에 맞게 매개변수를 지정합니다.

- 과정 ④에서는 테스트 데이터를 활용해 어느 정도의 정밀도가 나오는지 확인합니다. 만약 원하는 결과가 나오지 않는다면 매개변수를 수정하거나 알고리즘을 변경하는 방법 등을 검토하며 반복해야 합니다.

머신러닝의 국내 도입현황

개요

- 인공지능(AI)을 구현하는 머신러닝 기술이 활발히 연구되면서 의료, 금융, 교통, 제조 등의 분야에서 도입할 것으로 전망됨.
- 금융 산업은 의료 산업에 이어 인공지능 기술 활용도가 두 번째로 높을 것으로 기대되며, 대표적인 금융서비스로 로보어드바이저, 시장분석, 금융보안, 신용평가 등이 예상됨.

금융권 머신러닝 도입 분야

- (업무자동화) 복잡한 현황의 통계자료 산출, 자연어 분석을 통한 업무문서의 주요 특징 추출, 임직원 및 고객이 입력한 데이터의 오류 탐지, 사고 손해액 예측 시스템 등에 머신러닝 활용
- 국민카드 : `17년 하반기 딥러닝 기반 이상거래탐지시스템 (FDS, Fraud Detection System)을 도입 예정
- 신한은행 : 13년 거래 패턴을 이용한 FDS를 도입한 이후 딥러닝 솔루션 'GruDEEP'을 개발한 핀테크 업체 인피니그루'와 협력하여 딥러닝 기반의 FDS를 도입
- BC 카드 : 상권분석 시스템을 통해 별도의 인력이 수행하던 데이터 분석, 시각 자료 작성 등을 머신러닝기반으로 자동화

7-1. 첫번째 머신러닝 모델

– 붓꽃의 품종 분류
– k-최근접 이웃 알고리즘

Scikit-learn

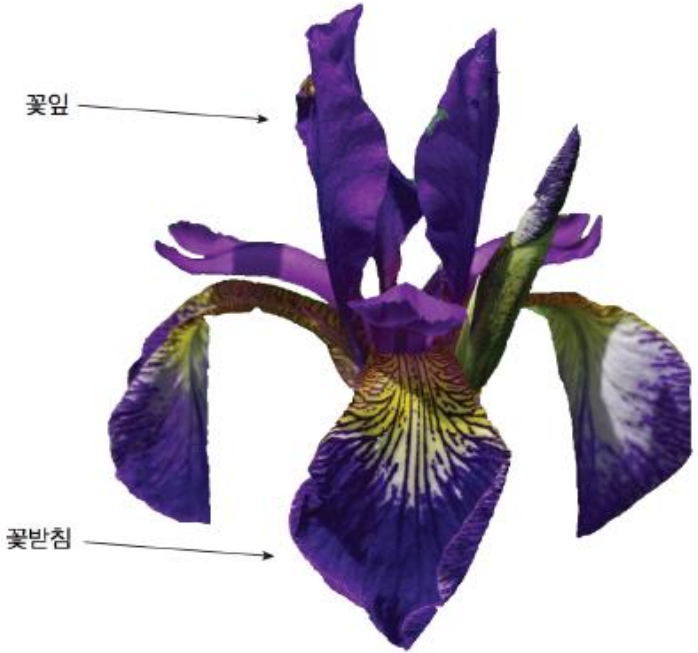
- 오픈소스

  https://github.com/scikit-learn/scikit-learn
- 회귀, 분류, 군집, 차원축소, 특성공학, 전처리, 교차검증, 파이프라인 등 머신러닝에 필요한 도구를 두루 갖춤
- 풍부한 문서 (영문):

  http://scikit-learn.org/stable/documentation
- 학교, 산업 현장에서 널리 사용됨
- 폭 넓은 커뮤니티

붓꽃의 품종 분류하기

- Iris-setosa, Iris-versicolor, Iris-virginica 종 분류
- SepalLength(꽃받침의길이), SepalWidth(꽃받침의폭)
  PetalLength(꽃잎의길이), PetalWidth(꽃잎의폭)
- 사전에 준비한 데이터를 이용하므로 지도 학습
- 3개의 붓꽃 품종에서 고르는 분류classification
- 클래스class: 가능한 출력값. 즉 세개의 붓꽃 품종
- 레이블label: 데이터 포인트 하나에 대한 출력
- 목표는 어떤 품종인지 구분해놓은 측정 데이터를 이용
  측하는 머신러닝 모델을 만드는 것



머신러닝으로 붓꽃의 품종 분류하기

1. 데이터 적재

- 붓꽃iris 데이터셋은 scikit-learn의 datasets에 포함되어 있어 load_iris 함수를 사용해서 데이터를 적재

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

- load_iris가 반환한 iris 객체는 파이썬의 딕셔너리Dictionary와 유사한 Bunch 클래스의 객체입니다.

```
print("iris_dataset의 키: ₩n{}".format(iris_dataset.keys()))
```

- target_names의 값은 우리가 예측하려는 붓꽃 품종의 이름을 문자열 배열로 가지고 있음

```
print("타깃의 이름: {}".format(iris_dataset['target_names']))
```

- feature_names의 값은 각 특성을 설명하는 문자열 리스트

```
print("특성의 이름: ₩n{}".format(iris_dataset['feature_names']))
```

- data의 값은 꽃잎의 길이와 폭, 꽃받침의 길이와 폭을 수치 값으로 가지고 있는 NumPy 배열

```
print("data의 타입: {}".format(type(iris_dataset['data'])))
```

머신러닝으로 붓꽃의 품종 분류하기

- data 배열의 행은 개개의 꽃이 되며 열은 각 꽃에서 구한 네 개의 측정치

```
print("data의 크기: {}".format(iris_dataset['data'].shape))
```

- data의 값 다섯 샘플의 값 확인

```
print("data의 처음 다섯 행:₩n{}".format(iris_dataset['data'][:5]))
```

- target은 각 원소가 붓꽃 하나에 해당하는 1차원 배열입니다.

```
print("target의 크기: {}".format(iris_dataset['target'].shape))
```

- 붓꽃의 종류는 0에서 2까지의 정수로 기록되어 있음 (0은 setosa, 1은 versicolor, 2는 virginica)

```
print("타깃:₩n{}".format(iris_dataset['target']))
```

머신러닝으로 붓꽃의 품종 분류하기

2. 성과 측정: 훈련 데이터와 테스트 데이터

- 레이블된 데이터(150개의 붓꽃 데이터)를 두 그룹으로 나누어, 하나는 머신러닝 모델을 만들 때 사용하는 **훈련 데이터** 나머지는 모델이 잘 작동하는지 측정하는 데 사용하는 **테스트 데이터로** 나눔
- scikit-learn은 데이터셋을 섞어서 나눠주는 train_test_split 함수를 제공합니다.

  이 함수는 75%를 레이블 데이터와 함께 훈련 세트로 뽑고, 나머지 25%는 레이블 데이터와 함께 테스트 세트가 됩니다.
- scikit-learn에서 데이터는 대문자 X로 표시하고 레이블(답)은 소문자 y로 표기함.

  data는 2차원 배열(행렬)이므로 대문자 X를, target은 1차원 배열(벡터)이므로 소문자 y를 사용함

  train_test_split 함수로 데이터를 나눌 때 유사 난수 생성기를 사용해 데이터셋을 무작위로 섞어야 함

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
iris_dataset['data'], iris_dataset['target'], random_state=0)
```

- X_train은 전체 데이터셋의 75%를, X_test는 나머지 25%를 담고 있음.

머신러닝으로 붓꽃의 품종 분류하기

- 훈련 데이터 확인

```
print("X_train 크기: {}".format(X_train.shape))
print("y_train 크기: {}".format(y_train.shape))
```

- 테스트 데이터 확인

```
print("X_test 크기: {}".format(X_test.shape))
print("y_test 크기: {}".format(y_test.shape))
```

k-최근접 이웃(KNN) 알고리즘 사용하기

k-최근접 이웃 알고리즘

- k-최근접 이웃 알고리즘에서 k는 훈련 데이터에서 새로운 데이터 포인트에 가장 가까운 'k개'의 이웃을 찾는다는 뜻 그런 다음 이 이웃들의 클래스 중 빈도가 가장 높은 클래스를 예측값으로 사용함
- k-최근접 이웃 분류 알고리즘은 neighbors 모듈 아래 **KNeighborsClassifier** 클래스에 구현되어 있음.
- 모델을 사용하려면 클래스로부터 객체를 만들어야 하며 가장 중요한 매개변수는 이웃의 개수입니다. 1로 지정함

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

- knn 객체는 훈련 데이터로 모델을 만들고 새로운 데이터 포인트에 대해 예측하는 알고리즘을 캡슐화한 것으로 KNeighborsClassifier의 경우는 훈련 데이터 자체를 저장하고 있음
- 훈련 데이터셋으로부터 모델을 만들려면 **knn 객체의 fit 메서드**를 사용함.

```
knn.fit(X_train, y_train)
```

- fit 메서드는 knn 객체 자체를 반환함

k-최근접 이웃(KNN) 알고리즘 사용하기

예측하기

- 꽃받침의 길이가 5cm, 폭이 2.9cm이고 꽃잎의 길이가 1cm, 폭이 0.2cm인 붓꽃의 품종은 무엇일까요?

```
X_new = np.array([[5, 2.9, 1, 0.2]])
prediction = knn.predict(X_new)
print("예측: {}".format(prediction))
print("예측한 타깃의 이름: {}".format( iris_dataset['target_names'][prediction]))
```

모델 평가하기

- 테스트 데이터는 모델을 만들 때 사용하지 않았으며 각 붓꽃의 품종을 정확히 알고 있습니다.

```
y_pred = knn.predict(X_test)
print("테스트 세트에 대한 예측값:₩n {}".format(y_pred))
```

- 테스트 데이터에 있는 붓꽃의 품종을 예측하고 실제 레이블(품종)과 비교할 수 있습니다.

```
print("테스트 세트의 정확도: {:.2f}".format(np.mean(y_pred == y_test)))
```

- knn 객체의 score 메서드로 테스트 세트의 정확도를 계산할 수 있습니다.

```
print("테스트 세트의 정확도: {:.2f}".format(knn.score(X_test, y_test)))
```

scikit-learn에 들어 있는 위스콘신 유방암 데이터셋

- 유방암 종양의 임상 데이터를 기록해놓은 위스콘신 유방암Wisconsin Breast Cancer 데이터셋입니다(줄여서 cancer라고)
- 각 종양은 양성benign(해롭지 않은 종양)과 악성malignant(암 종양)으로 레이블 되어 있고, 조직 데이터를 기반으로 종양이 악성인지를 예측할 수 있도록 학습하는 것이 과제입니다.

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): ₩n{}".format(cancer.keys()))
```

- 이 데이터셋은 569개의 데이터 포인트를 가지고 있고 특성은 30개입니다.

```
print("유방암 데이터의 형태: {}".format(cancer.data.shape))
```

- 569개 데이터 포인트 중 212개는 악성이고 357개는 양성입니다.

```
print("클래스별 샘플 개수:₩n{}".format( {n: v for n, v in zip(cancer.target_names,
np.bincount(cancer.target))}))
```

- feature_names 속성을 확인하면 각 특성의 의미를 알 수 있습니다.

```
print("특성 이름:₩n{}".format(cancer.feature_names))
```

8. 데이터 분석/시각화/모델링(Regression)

: Forecast use of a city bike share system

https://www.kaggle.com/c/bike-sharing-demand

Bike Sharing Demand

Forecast use of a city bikeshare system

3,251 teams · 3 years ago

Forecast use of a city bike share demand system

- **1.Defining the problem statement**

: training set (train.csv) / test set (test.csv)

Data Fields

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

판교 | 경기도 성남시 분당구 삼평동 대왕판교로 670길 유스페이스2 B동 8층 T. 070-5039-5805
가산 | 서울시 금천구 가산동 371-47 이노플렉스 1차 2층 T. 070-5039-5815
웹사이트 | http://edu.kosta.or.kr **팩스 |** 070-7614-3450