머신러닝 이란?

이론적으로.

1.소개

- 머신러닝은 데이터에서 지식을 추출하는 작업.
- 최근 몇 년 동안 머신러닝을 사용한 애플리케이션들이 우리 일상에 널리 퍼지고 있습니다.
- 상업적인 애플리케이션 이외에도 머신러닝은 오늘날 데이터에 기반을 둔 연구에 커다란 영향을 끼쳐 왔습니다.

1.1 왜 머신러닝인가 ?

- 초창기 지능형 애플리케이션들은 데이터를 처리하고 사용자의 입력을 다루는 데 하드 코딩된 “if”와 “else” 명령을 사용하는 시스템.

- 하지만 직접 규칙을 만드는 것은 두 가지 커다란 단점이 있습니다.

  - 결정에 필요한 로직은 한 분야나 작업에 국한됩니다. 작업이 조금만 변경되더라도 전체 시스템을 다시 개발해야 할 수 있습니다.

  -규칙을 설계하려면 그 분야 전문가들이 내리는 결정 방식에 대해 잘 알아야 합니다.

1.1.1 머신러닝으로 풀 수 있는 문제.

- 가장 많이 사용되는 머신러닝 알고리즘들은 이미 알려진 사례를 바탕으로 일반화된 모델을 만들어 의사 결정 프로세스를 자동화하는 것들입니다. 이 방식을 **지도 학습**이라고 하며 사용자는 알고리즘에 입력과 기대되는 출력을 제공하고 알고리즘은 주어진 입력에서 원하는 출력을 만드는 방법을 찾습니다.

1.1.1 머신러닝으로 풀 수 있는 문제

- **편지 봉투에 손으로 쓴 우편번호 숫자 판별**
- **의료 영상 이미지에 기반한 종양 판단**
- **의심되는 신용카드 거래 감지**
- 이런 사례들에서 주목할 점은 입력과 출력이 상당히 직관적으로 보이지만, 데이터를 모으는 과정은 세 경우가 많이 다르다는 것입니다.

1.1.2 문제와 데이터 이해하기

1. 어떤 질문에 대한 답을 원하는가? 가지고 있는 데이터가 원하는 답을 줄 수 있는가?
2. 내 질문을 머신러닝의 문제로 가장 잘 기술하는 방법은 무엇인가?
3. 문제를 풀기에 충분한 데이터를 모았는가?
4. 내가 추출한 데이터의 특성은 무엇이며 좋은 예측을 만들어낼 수 있을 것인가?
5. 머신러닝 애플리케이션의 성과를 어떻게 측정할 수 있는가?
6. 머신러닝 솔루션이 다른 연구나 제품과 어떻게 협력할 수 있는가?

1.3 Scikit-learn

- 오픈 소스인 **scikit-learn**은 자유롭게 사용하거나 배포할 수 있고, 누구나 소스 코드를 보고 실제로 어떻게 동작하는지 쉽게 확인할 수 있습니다. scikit-learn 프로젝트는 꾸준히 개발, 향상되고 있고 커뮤니티도 매우 활발합니다

1.3 Scikit-learn

- Jupyter notebook 에서 설치하는 방법

- !pip install numpy, scipy, matplotlib, scikit-learn, pandas, pillow

- ! pip install graphviz

1.4.2 Numpy

- Numpy는 파이썬으로 과학 계산을 하려면 꼭 필요한 패키지
- 다차원 배열을 위한 기능과 선형대수 연산과 푸리에 변환 같은 고수준 수학 함수와 유사 난수 생성기를 포함합니다.
- scikit-learn에서 NumPy 배열은 기본 데이터 구조입니다. scikit-learn은 NumPy 배열 형태의 데이터를 입력으로 받습니다.

1.4.2 Numpy

```
import numpy as np

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

```
x:
[[1 2 3]
 [4 5 6]]
```

1.4.2 Numpy

- numpy 배열 (1/2)
  - numpy 배열을 만들 때는 np.array() 메소드 이용
  - np.array()는 파이썬의 리스트를 numpy,ndarray 형태로 변환
  - numpy는 다차원 행렬로 표현하고, 계산하기 쉬움

```
arr = [[1,2,3,4], [5,6,7,8]]
arr2= np.array(arr)

arr2.ndim
Out[]:2

arr2.shape
Out[]:(2, 4)

arr2 * 10
Out[]: array([[10, 20, 30, 40], [50, 60, 70, 80]])

arr2 + 10
Out[]: array([[11, 12, 13, 14], [15, 16, 17, 18]])
```

1.4.2 Numpy

- numpy 배열 (2/2)
  - zeros : 0으로 초기화된 배열, ones : 1로 초기화된 배열
  - linespace : 선형 구간에서 지정 구간의 수만큼 분할
  - logspace : 로그 구간에서 지정 구간의 수만큼 분할
  - empty : 배열을 메모리에 생성만하고 특정한 값을 초기화하지 않는 배열 (초기화 시간 단축)

```
np.zeros(5)
Out[]: array([0., 0., 0., 0., 0.])
np.ones(5)
Out[]: array([ 0. , 2.5, 5. , 7.5, 10. ])
np.logspace(0, 100, 5)
Out[]: array([1.e+000, 1.e+025, 1.e+050, 1.e+075, 1.e+100])
np.empty(5)
Out[]: array([0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000
                    0000e+000, 1.70704615e-312])
```

1.4.3 Numpy

- numpy 산술 연산
  - numpy는 배열과 스칼라 값을 조합으로 산술 연산이 가능
  - numpy 배열은 스칼라 값에 의해 원소별(element-wise)로 한 번씩 수행 (브로드캐스트)

```
arr =  np.array([[1,2,3,4], [5,6,7,8]] )


arr * 10
Out[]: array([[10, 20, 30, 40], [50, 60, 70, 80]])

arr + 10
Out[]: array([[11, 12, 13, 14], [15, 16, 17, 18]])


arr / 2
Out[]: array([[0.5, 1. , 1.5, 2. ],
                                [2.5, 3. , 3.5, 4. ]])
arr % 2
Out[]: array([[1, 0, 1, 0],
       [1, 0, 1, 0]], dtype=int32)
```

1.4.3 Numpy

기초 사용법

- 다차원으로 변환
  - numpy는 배열은 reshape로 N차원 매트릭스로 변환 가능
  - reshape 명령어 중 원소는 -1로 대체할 수 있다.
    - arr.reshape(4,-1) : 4개 ROW를 가지는 행렬
    - arr.reshape(-1,5) : 5개 COLUMN을 가지는 행렬

```
arr = np.array(range(0,20))        # 0~19까지 생성
arr_2d = arr.reshape(4,5)
arr_2d
Out[]: array([[ 0, 1, 2, 3, 4],
              [ 5, 6, 7, 8, 9],
             [10, 11, 12, 13, 14],
             [15, 16, 17, 18, 19]])


arr.reshape(4,-1)
arr.reshape(-1, 5)
Out[]: array([[ 0, 1, 2, 3, 4],
             [ 5, 6, 7, 8, 9],
             [10, 11, 12, 13, 14],
             [15, 16, 17, 18, 19]])
```

1.4.3 Numpy

기초 사용법

- 다차원으로 변환
  - flatten 메소드 : numpy 다차원 배열을 1차원 리스트로 변환
  - np.newaxis :차원을 증가시킬 때 사용

```
arr_3d =  arr.reshape(5, 2, 2)
arr_2d.reshape(1,20)
Out[]: array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  19]])

arr_1d = arr_3d.flatten()
arr_1d
Out[]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,17, 18,  19])

arr_1d[np.newaxis]
Out[]: array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  19]])

arr_1d[np.newaxis, np.newaxis]
Out[]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  19]]])
```

1.4.3 Numpy

배열 붙이기

- 행(COLUMN)의 수나 열(ROW)의 수가 같은 두 개 이상의 배열을 잇는(concatenate) 방법
  - hstack : ROW의 수가 같은 경우, COLUMN으로 붙임 (좌측에서 옆으로 붙임)
  - vstack : COLUMN의 수가 같은 경우, ROW으로 붙임 (위에세서 아래로 붙임)
  - stack : 배열을 별개 차원으로 구성한다. (2차원 배열 두 개를 합치면 3차원 배열로.)

```
a1 = np.ones((3,4)) #1로 채워진 배열생성
a2 = np.zeros((3,4)) #0으로 채워진 배열 생성

np.hstack([a1, a2])
Out[]: array([[1., 1., 1., 1., 0., 0., 0., 0.],
              [1., 1., 1., 1., 0., 0., 0., 0.],
              [1., 1., 1., 1., 0., 0., 0., 0.]])
np.vstack([a1,a2])
Out[]: array([[1., 1., 1., 1.],
              [1., 1., 1., 1.],
              [1., 1., 1., 1.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.],
              [0., 0., 0., 0.]])
```

```
np.stack([a1,a2])
Out[]: array([[[1., 1., 1., 1.],
               [1., 1., 1., 1.],
               [1., 1., 1., 1.]],
              [[0., 0., 0., 0.],
               [0., 0., 0., 0.],
               [0., 0., 0., 0.]]])
```

```
#axis=1 지정 시각 배열의 row별 묶음 np.stack([a1,a2], axis=1) array([[[1., 1., 1., 1.],
        [0., 0., 0., 0.]],
       [[1., 1., 1., 1.],
        [0., 0., 0., 0.]],
       [[1., 1., 1., 1.],
        [0., 0., 0., 0.]]])
```

1.4.3 Numpy

기술통계

- 집계 합수(aggregate function)
  - max (배열의 최대값) min (배열의 최소값) mean (배열의 평균 값)
  - numpy의 모든 집계 함수는 AXIS 기준으로 계산이 이루어짐

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

```
arr.max()
Out[]: 9
```

```
arr.max(axis=1)
Out[]: array([9, 4])
```

```
arr.max(axis=0)
Out[]: array([5, 6, 7, 8, 9])
```

```
arr.min()
Out[]: 0
```

```
arr.min(axis=1)  Out[
]: array([5, 0])
```

```
arr.min(axis=0)
Out[]: array([0, 1, 2, 3, 4])
```

```
arr.mean()
Out[]: 4.5
```

```
arr.mean(axis=1)  Out
[]: array([7., 2.])
```

```
arr.mean(axis=0)
Out[]: array([2.5, 3.5, 4.5, 5.5, 6.5])
```

1.4.3 Numpy

랜덤 함수

- 난수(random number)

```
np.random.seed(0)  np.random.ra          # 랜덤 seed 지정
ndint(0, 1000)                           # 0~1000사이 난수 생성
Out[]: 629

np.random.rand(5)                        #5개의 난수 생성
Out[]: array([0.22232139, 0.38648898, 0.90259848, 0.44994999, 0.61306346])

np.random.rand(3,5)                      #3x5 행렬 형태의 난수 생성
Out[]: array([[0.888609 , 0.90658127, 0.55368029, 0.83453863],
              [0.29628516, 0.2548365 , 0.59444402, 0.14268807],
              [0.13459794, 0.81022365, 0.69110336, 0.37632546]])

x = np.arange(10)                        # 0~9까지 배열 생성
Out[]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.random.shuffle(x)                     # 배열 x의 순서를 무작위로 변경 ()
```

1.4.3 Numpy

```
x = np.random.choice(10, 10, replace=False)   # shuffle 기능과 동일
Out[]: array([0, 7, 1, 5, 2, 3, 9, 6, 8, 4])

np.random.choice(x, 5, replace=True)          # 5개 샘플링, 중복 허용
Out[]: array([2, 4, 6, 3, 6])

np.random.choice(4, 3, replace=False, p=[0.4, 0.2, 0, 0.4]) # 선택 확률 별도 지정
Out[]: array([0, 3, 1])
```

1.4.3 Numpy

비교 연산

- 비교 연산 처리
  - 인덱싱 영역에 조건 값을 입력하면 해당 조건에 맞는 데이터 추출 가능
  - 인덱싱 영역에 조건 값을 입력하고 해당 조건에 맞는 영역에 새로운 값을 대입 가능

```
f = np.random.rand(3,4)
Out[]: array([[0.888609 , 0.90658127, 0.55368029, 0.83453863],
              [0.29628516, 0.2548365 , 0.59444402, 0.14268807],
              [0.13459794, 0.81022365, 0.69110336, 0.37632546]])

tf = f > 0.3
Out[]: array([[ True, True, True, True],
              [False, False, True, False],  [False, True, True, True]])

f[ f > 0.3 ] = 1
Out[]: array(     [[1., 1., 1., 1.],
             [0.29628516, 0.2548365 , 1., 0.14268807],
             [0.13459794, 1., 1., 1.]])
```

1.4.3 Numpy

유니버설 함수

- ufunc라고 불리는 유니버설 함수.
- ndarray 안에 있는 데이터 원소별로 연산을 수행
- 고속으로 수행할 수 있는 벡터화된 Wrapper 함수

1.4.3 Numpy

```
x = np. linspace(-5, 5, 20)

np.sqrt(x)
np.exp(x)
np.abs(x) np.ceil(x) np.floor(x) np.round
(x) np.cos(x), np.sin(x) np.power(x, n)
np.log(x) np.log10(x) np.log2(x)
np.mod(x, n)
```

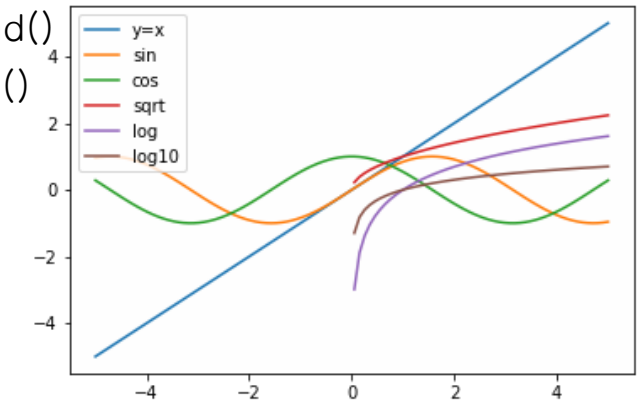
```
# 선형공간 0~10에서 20개 데이터 고르게 추출

# 제곱근 연산 √x
# 지수함수 연산 e^x (e=2.718281…)
# 절대값 연산
# 소수점 올림 연산
# 소수점 내림 연산
# 소수점 반올림 연산
# sin함수, cos함수 처리 (그 외 다양한 삼각함수들 존재)
# n 제곱 처리
# 자연로그 연산 (log_e x)
# 로그10 연산 (log_10 x)
# 로그2 연산 (log_2 x)
# 각 요소별로 n으로 나눈 뒤의 나머지 추출
```

1.4.3 Numpy

유니버설 함수

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5, 5, 100)

y = x
y_sin = np.sin(x)
y_cos = np.cos(x) y_
sqrt = np.sqrt(x) y_lo
g = np.log(x) y
_log10 = np.log10(x)
```

```
fig= plt.figure()
ax = fig.add_subplot(111
)
ax.plot(x, y, label='y=x') ax.plot
(x, y_sin, label='sin') ax.plot(x, y
_cos, label='cos') ax.plot(x, y_s
qrt, label='sqrt') ax.plot(x, y_log,
label='log') ax.plot(x, y_log10, l
abel='log10′)

ax.legend()
plt.show()
```



- matplotlib는 이후 데이터시각화에서 다룰 예정
- plot 그래프에 x 좌표, y 좌표에 어떤 데이터가 입력되는지 확인하고 눈으로 보는 데 의의두자!

1.4.3 Scipy

- SciPy는 과학 계산용 함수를 모아놓은 파이썬 패키지입니다.
  SciPy는 고성능 선형대수, 함수 최적화, 신호 처리, 특수한 수학 함수와 통계 분포 등을 포함한 많은 기능을 제공합니다.

1.4.3 Scipy

```
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.]]
```

```
# NumPy 배열을 CSR 포맷의 SciPy 희소 행렬로 변환합니다.
# 0이 아닌 원소만 저장됩니다.
sparse_matrix = sparse.csr_matrix(eye)
print("SciPy의 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
```

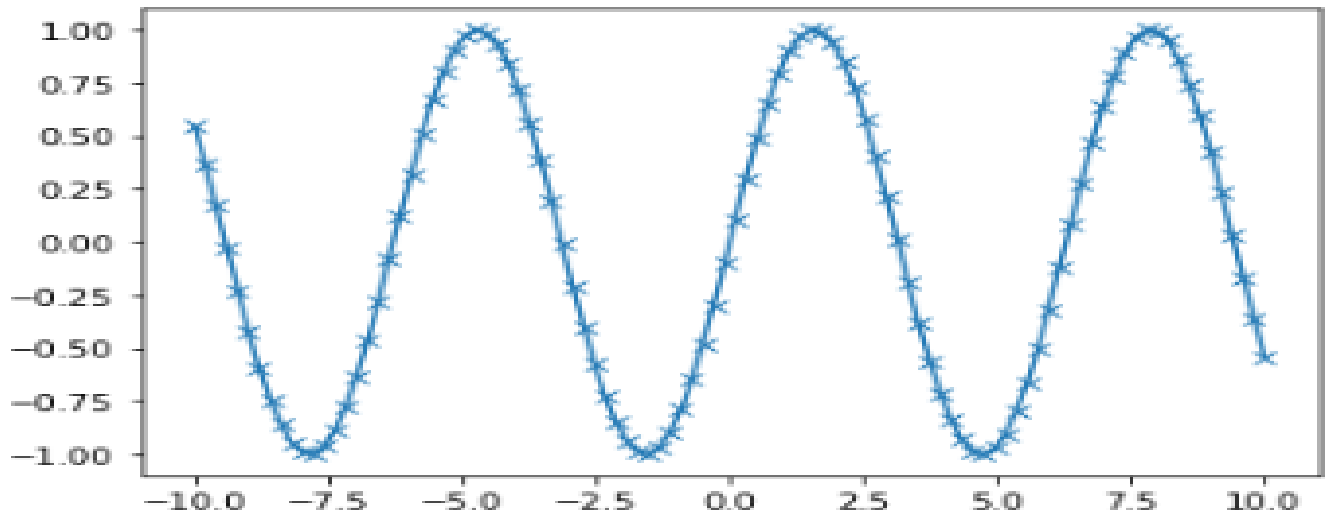
1.4.4 matplotlib

- matplotlib은 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 시각화해보면 매우 중요한 통찰을 얻을 수 있습니다
- 주의 : jupyter notebook 에서는
  %matplotlib notebook, %matplotlib inline 명령어를 쳐야 보임

1.4.4 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")
```

```
[<matplotlib.lines.Line2D at 0x292c38d4cf8>]
```

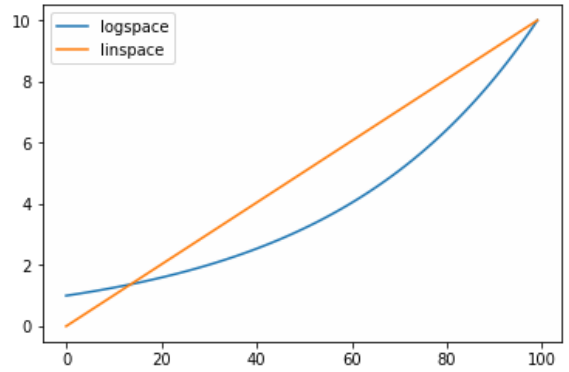


1.4.5Pandas

기초 개념

- Pandas
  - Numpy 기반으로 개발되어 고성능 데이터 분석 가능
  - R언어에서 제공하는 DataFrame 자료형 제공
  - 명시적으로 축의 이름에 따라 데이터 정렬 가능한 자료구조
  - 통합된 Time Series 분석 기능
  - 누락된 데이터를 유연하게 처리할 수 있는 기능

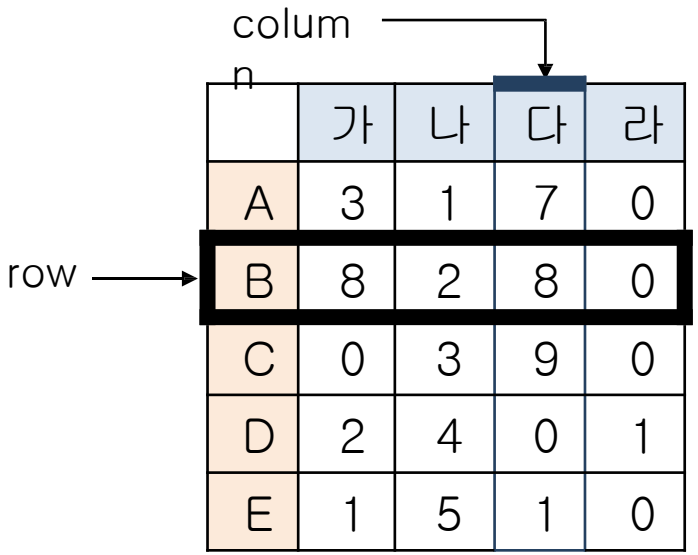
```
import numpy as np
import  pandas as pd              # Pandas 라이브러리 선언
logx =  np.logspace(0,  1,  100)
linx   =  np.linspace(0, 10, 100)

df  =  pd.DataFrame()             #Pandas의 DataFrame
생성  df['logspace'] = logx      #컬럼명 logspace의
Series 데이터  df['linspace']  =  linx  #컬럼명 linespace
의 Series 데이터

df.head()         # DataFrame 상위 5개 보기
df.plot()         # 그래프로 출력
```



Pandas

기초 개념

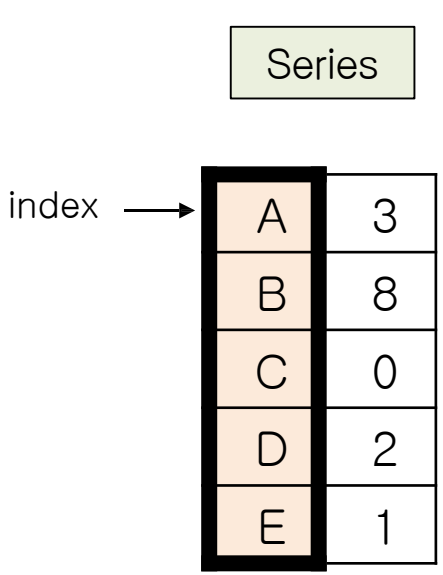
- DataFrame
  - 레이블(Labeled)된 행(Column)과 열(Row)을 가진 2차원 데이터구조
  - 칼럼 마다 데이터 형식이 다를 수 있음
  - 각 행(Column)과 열(Row)들을 산술연산이 가능한 구조
  - DataFrame 크기는 유동적으로 변경 가능
  - DataFrame끼리 여러 가지 조건을 사용한 결합처리 가능

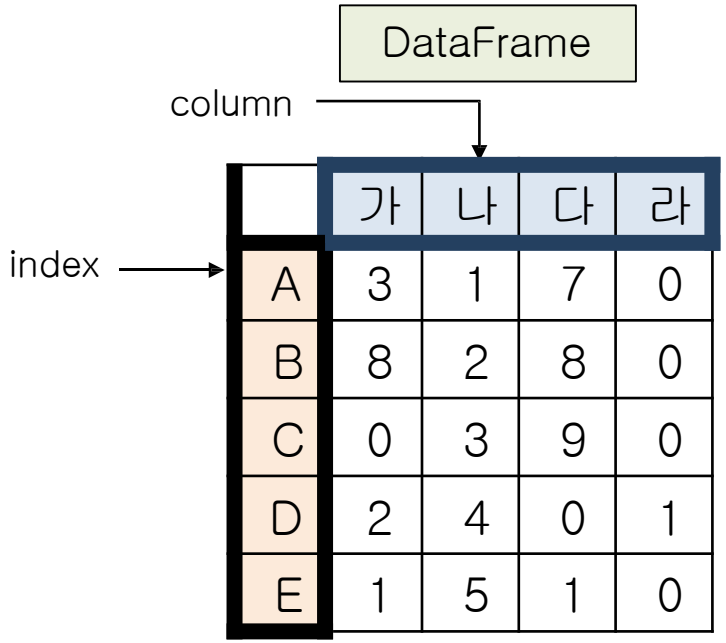


Pandas

기초 개념

- Series와 DataFrame
  - Series : 인덱스 라벨이 붙은 1차원 리스트 데이터 구조
  - DataFrame : Series가 모인 2차원 테이블 데이터 구조 (인덱스와 컬럼 라벨이 붙음)
  - DataFrame이 핵심이고, Series는 개념으로 알아두자!





Pandas

기초 개념

- Series와 DataFrame
  - Series 생성 : pd.Series(values, index=index, names=names)
  - DataFrame 생성 : pd.DataFrame(values, index=index, columns=cols)

```
pd.Series([1,2,3,4], index=['one','two','three','four'])
Out[]: one      1
two      2
three    3
four     4  dtype: int64
```

```
pd.DataFrame([1,2,3,4], index=['one','two','three','four'], columns=['number'])
```

Out[]:

	number
one	1
two	2
three	3
four	4

Pandas_DataFrame 생성

Dictionary

List

Row oriented

```
ds = [ {'고객':'A', '금액':1000, '가맹점':'0001', '카드':'9440'},
       {'고객':'B', '금액':2000, '가맹점':'0002', '카드':'4805'},
       {'고객':'C', '금액':5000, '가맹점':'0003', '카드':'4602'} ]
df = pd.DataFrame(ds)
```

```
ds = [['A', 1000, '0001', '9440'],
      ['B', 2000, '0002', '4805'],
      ['C', 5000, '0003', '4602']]
df = pd.DataFrame.from_records(ds, columns=lb)
#df = pd.DataFrame(ds, columns=lb) 가능
lb = ['고객', '금액', '가맹점', '카드']
```

default

	고객	금액	가맹점	카드
0	A	1000	0001	944020
1	B	2000	0002	480520
2	C	5000	0003	460220

from_records

Column oriented

```
ds = { '고객'   :['A','B','C'],
       '금액'   :[1000, 2000, 5000],
       '가맹점' :['0001', '0002', '0003'],
       '카드'   :['9440', '4805', '4602']
df = pd.DataFrame.from_dict(ds)
#df = pd.DataFrame(ds) 가능
```

```
ds = [['고객'   ,['A', 'B', 'C']],
      ['금액'   ,[1000, 2000, 5000]],
      ['가맹점' ,['0001', '0002', '0003']],
      [ '카드' ,['9440', '4805', '4602' ]]
df = pd.DataFrame.from_items(ds)
```

Pandas

DataFrame 생성

- CSV 형식 파일을 읽어 들여 DataFrame 생성
  - pd.read_csv(path, sep='구분자', names=['컬럼명',..]) : 파일로부터 DataFrame 생성
  - github.com/sh2orc/datascience에서 데이터셋 다운로드

```
df = pd.read_csv('titanic.csv', sep='	')       #데이터가 TAB으로 구분되어 있는 경우
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S

```
#컬럼 헤더가 없는 파일의 경우는 name= 옵션을 통해 지정해줄 수 있다.
f = pd.read_csv('msg_code', names=['MSG_CODE', 'MSG_NAME'], dtype=str)
```

Pandas

DataFrame 생성

- dictionary 객체로 DataFrame 생성
  - dictionary 객체가 단순 {key:value}로 구성되어있을 경우
  - pd.from_dict(ds, oriented='index', columns=['value에 대한 칼럼명'])

```
firm_name={"03":"IBK기업은행", "11":"NH농협은행", "20":"우리카드",
           "23":"SC제일은행", "48":"새마을금고", "72":"우체국" }

df = pd.DataFrame.from_dict(firm_name, orient='index', columns=['FIRM_NAME']
) df
```

	FIRM_NAME
03	IBK기업은행
11	NH농협은행
20	우리카드
23	SC제일은행
48	새마을금고
72	우체국

Panda_Column명 변경.

- DataFrame의 rename
  - pd.DataFrame.columns = ['칼럼1', '칼럼2', … ,] 이용
  - pd.DataFrame.rename({ 'OLD칼럼명' : 'NEW칼럼명', …, }, axis=1) 이용
  - pd.DataFrame.rename(columns={ 'OLD칼럼명' : 'NEW칼럼명', …, }) 이용

```
df = pd.read_csv('msg_code', names=['MSG_CODE', 'MSG_NAME'], dtype=str)
df.head()
```

	MSG_CODE	MSG_NAME
0	0100	MS승인요청
1	0110	MS승인응답
2	0300	IC승인 요청
3	0310	IC승인 응답
4	0420	MS승인취소 요청

```
df.rename({'MSG_CODE' : 'COD                axis=1                    #둘 다 같은 역
df.rename(columns  {'MSG_CODE' : 'CODE'}                              할
```

	CODE	MSG_NAME
0	0100	MS승인요청
1	0110	MS승인응답
2	0300	IC승인 요청
3	0310	IC승인 응답
4	0420	MS승인취소 요청

Pandas

DataFrame Index

- index 지정 방법
  - 생성 시점 : pd.read_csv(path, names=[], index_col=칼럼인덱스 번호)
  - 생성 이후 : pd.DataFrame.set_index('칼럼명')

```
df = pd.read_csv('msg_code', names=['MSG_CODE', 'MSG_NAME'], dtype=str)
df = df.set_index('MSG_CODE')

df['COUNT'] = [ np.random.randint(5,10) for i in range(len(df.index))
] df.plot(kind='bar')
```

MSG_CODE	MSG_NAME
0100	MS승인요청
0110	
0300	
0310	IC승인 응답
0420	MS승인취소 요청

Index로 지정 시 해당 데이터 컬럼이 데이터의 축이 된다.

Pandas_DataFrame index

DataFrame Index

- index 초기화
  - reset_index() : 기존 index를 컬럼으로 위치시키고, 새로운 index를 생성한다
    .

```
df = pd.read_csv('msg_code', names=['MSG_CODE', 'MSG_NAME'], dtype=
str)  df = df.set_index('MSG_CODE')

df['COUNT'] = [ np.random.randint(5,10) for i in range(len(df.index)) ]
df = df.reset_index()
```

MSG_CODE	MSG_NAME
0100	MS출입요청
0110	MS출입불가
0300	IC출입 요청
0310	IC출입 불가
0420	MS출입취소 요청

set_index()

	MSG_CODE	MSG_NAME	COUNT
0	0100	MS출입요청	9
1	0110	MS출입불가	7
2	0300	IC출입 요청	5
3	0310	IC출입 불가	7
4	0420	MS출입취소 요청	6

reset_index()

Pandas

행(Column)과 열(Row) 삭제

- pd.DataFrame.drop([label, …,], axis=0 또는 1)
  - 열(Row) 삭제 – label에 삭제할 row의 index들을 입력
  - 행(Colum) 삭제 – label에 삭제할 column들을 입력. 이 때 axis=1로 지정

```
df = pd.read_csv('msg_code', names=['MSG_CODE', 'MSG_NAME'], dtype=str)
df['COUNT'] = [ np.random.randint(5,10) for i in range(len(df.index)) ]
```

	MSG_CODE	MSG_NAME	COUNT
0	0100	MS승인요청	5
1	0110	MS승인응답	7
2	0300	IC승인 요청	7
3	0310	IC승인 응답	8
4	0420	MS승인취소 요청	7

df.drop([0,1,3])

	MSG_CODE	MSG_NAME
2	0300	IC승인 요청
4	0420	MS승인취소 요청
5	0430	MS승인취소 응답
6	9300	기타승인 요청
7	9310	기타승인 응답

df.drop('COUNT', axis=1)

	MSG_CODE	MSG_NAME
0	0100	MS승인요청
1	0110	MS승인응답
2	0300	IC승인 요청
3	0310	IC승인 응답
4	0420	MS승인취소 요청

Pandas_

행(Column)과 열(Row) 삭제

- 특정 컬럼 기준으로 삭제 방법
  - 칼럼을 index로 지정하고, 삭제할 대상을 각각 하드코딩
  - 칼럼을 index를 리스트로 추출하여, 삭제할 대상에 입력

```
df = pd.read_csv('msg_code', names=['MSG_CODE', 'MSG_NAME'], dtype=
str)  tf = df.set_index('MSG_CODE')


tf.drop(['0100','0110'])                    #index 0100, 0110 기준 ROW 삭제


tf.index.str[:2] == '01'
Out[]: array([ True, True, False, False, False, False, False, False, False, False, False, Fa
lse])


tf.index[tf.index.str[:2] == '01']
Out[]: Index(['0100', '0110'], dtype='object', name='MSG_CODE')
```

Pandas_

선택과 변경

- DataFrame의 배열[ ] 내 값을 지정한 선택
  - 숫자 인덱스 선택 : 열(ROW)
  - KEY값으로 선택 : 행(COLUMN)

```
df = pd.DataFrame( np.arange(9).reshape(3,3), index=['r1','r2','r3'], columns=['c1','c2','c3'])
```

	c1	c2	c3
r1	0	1	2
r2	3	4	5
r3	6	7	8

```
df[:2]        #슬라이싱 인덱스 기준은 ROW
```

	c1	c2	c3
r1	0	1	2
r2	3	4	5

```
df[['c1','c2']] # [ ] 내에는 KEY 조회는 칼럼만 가능
```

	c1	c2
r1	0	1
r2	3	4
r3	6	7

```
df['r1']      # KEY 값을 인덱스 KEY로 지정 불가
KeyError: 'r1'
```

Pandas_

선택과 변경

- DataFrame의 배열[ ] 내 (칼럼) 조건 지정
  - 해당 지정된 칼럼의 조건에 의한 Boolean 선택
  - 2개 이상의 조건은 각 조건별로 ( ) 묶음을 통해 AND/OR 조건 지정

```
df = pd.DataFrame( np.arange(9).reshape(3,3), index=['r1','r2','r3'], columns=['c1','c2','c3'])
```

	c1	c2	c3
r1	0	1	2
r2	3	4	5
r3	6	7	8

```
df[df.c1 > 4]                    #DataFrame 객체의 멤버 변수로 칼럼명 지정 접근
```

	c1	c2	c3
r3	6	7	8

```
df[df[ ' c1 ' ] > 4]             #DataFrame의 배열 Key로 칼럼명 지정 접근
```

	c1	c2	c3
r3	6	7	8

```
df[(df.c1 > 1) & (df.c2 < 5)] #2개 이상의 조건 지정 때는 각 조건별로 ( )로 묶어줘야 함
```

Pandas_

선택과 변경

- DataFrame의 배열[ ] 내 (칼럼) 조건 지정
  - df[[칼럼명$_1$, 칼럼명$_2$, …,]] 형식으로 정의시 DataFrame 재정의 가능
  - 재정의된 DataFrame은 마치 가상 뷰(View) 테이블처럼 구성

```
df = pd.DataFrame( np.arange(9).reshape(3,3), index=['r1','r2','r3'], columns=['c1','c2','c3'])
```

	c1	c2	c3
r1	0	1	2
r2	3	4	5
r3	6	7	8

```
df
['c1', 'c2']
```

#c1, c2 칼럼으로만 DataFrame 재구성

	c1	c2
r1	0	1
r2	3	4
r3	6	7

```
df ['c1'
[ ]
```

#c1 칼럼으로만 DataFrame 재구성

	c1
r1	0
r2	3
r3	6

Pandas_행 / 열 합계

- df.sum(axis=?)
  - axis는 1과 0 값 이며 입력하지 않으면 0로 자동 처리 (칼럼 기준 합)
  - axis = 1로 입력시 각 열 기준으로 합산

df

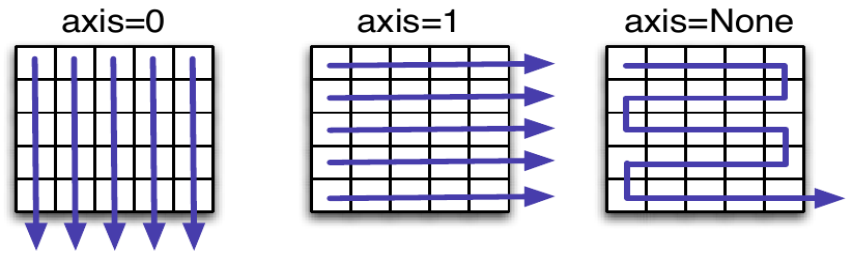
	성능	가격
BC1	70	20000
BC2	70	50000
BC3	90	50000

```
df.sum()      # 각 칼럼별 합계
성능     230
가격    120000
dtype: int64

df.sum(axis=1) # 각 로우별 합계
BC1    20070
BC2    50070
BC3    50090
dtype: int64
```

```
df['ROW_SUM'] = df.sum(axis=1)     # 신규 칼럼 추가
df.loc['COL_SUM', :] = df.sum()    # 신규 로우 추가
```

df

	성능	가격	ROW_SUM
BC1	70	20000	20070
BC2	70	50000	50070
BC3	90	50000	50090
COL_SUM	230	120000	120230



Pandas_행 / 열 합계

- df.sum(axis=?)
  - axis는 1과 0 값 이며 입력하지 않으면 0로 자동 처리 (칼럼 기준 합)
  - axis = 1로 입력시 각 열 기준으로 합산

df

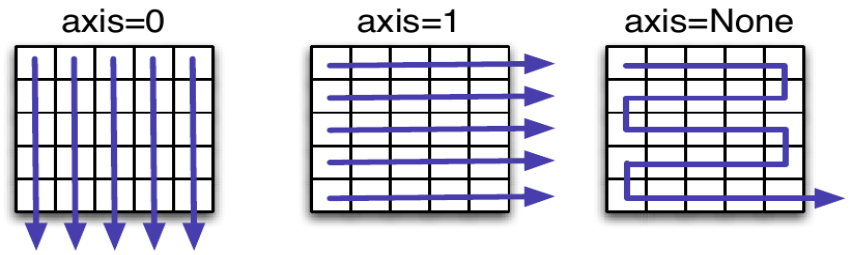
	성능	가격
BC1	70	20000
BC2	70	50000
BC3	90	50000

```
df.sum()      # 각 칼럼별 합계
성능     230
가격    120000
dtype: int64

df.sum(axis=1) # 각 로우별 합계
BC1    20070
BC2    50070
BC3    50090
dtype: int64
```

```
df['ROW_SUM'] = df.sum(axis=1)     # 신규 칼럼 추가
df.loc['COL_SUM', :] = df.sum()    # 신규 로우 추가
```

df

	성능	가격	ROW_SUM
BC1	70	20000	20070
BC2	70	50000	50070
BC3	90	50000	50090
COL_SUM	230	120000	120230



Pandas_행 / 시계열 시간 관리

- pd.to_datetime('문자열', format='입력된 문자열의 시간포멧')
  - 문자열을 입력받아 Timestamp 객체로 변환. 파이썬 datetime 객체 변환 용이
  - 시간포멧

기호	의미	자리수	출력
%Y	년	4	2018
%y	년	2	18
%m	월	2	09
%d	일	2	07
%H	시 (24시간)	2	17
%M	분	2	32
%S	초	2	58
%f	마이크로초 (1/10^{-6}초)	6	004321

```
#문자열 입력을 받아서 Timestamp 객체로 변환
beg_time = pd.to_datetime('20180907 12:30:05', format='%Y%m%d %H:%M:%S'
)  end_time = pd.to_datetime('20180907 12:31:05', format='%Y%m%d %H:%M:%
S')  Out[]: Timestamp('2018-09-07 12:30:05')

end_time - beg_time
Out[]: Timedelta('0 days 00:01:00')
```

Pandas_행

시계열 시간 관리

- pd.date_range('시작 일시 ', '종료 일시' , freq='빈도 포멧')
  - 시작 일시, 종료 일시 변수는 문자열, datetime 객체
  - datetime 객체의 차를 구하려면 datetime.timedelta 함수를 사용

```
#현재 시점의 시간으로 부터 앞으로 5분 사이의 10초당 간격 리스트 추출
import datetime  import pandas as pd


beg_time =  datetime.datetime.now()                                    #현재시간 datetime 객체
end_time =  datetime.datetime.now() +  datetime.timedelta(minutes=5)   #5분 이후 datetime 객체

                    pd.date_range(beg_time, end_time, freq='10s')                    #10초 간격으로 시간 리스트 구함
Out[]: DatetimeIndex(['2018-09-07 09:38:19.593836', '2018-09-07 09:38:29.593836',
                      '2018-09-07 09:38:39.593836', '2018-09-07 09:38:49.593836',
                      '2018-09-07 09:38:59.593836', '2018-09-07 09:39:09.593836',
                          .........
                      '2018-09-07 09:42:59.593836', '2018-09-07 09:43:09.593836',
                      '2018-09-07 09:43:19.593836'],
                     dtype='datetime64[ns]', freq='10S')
```

Pandas_행 / 열 합계

- df.sum(axis=?)
  - axis는 1과 0 값 이며 입력하지 않으면 0로 자동 처리 (칼럼 기준 합)
  - axis = 1로 입력시 각 열 기준으로 합산

df

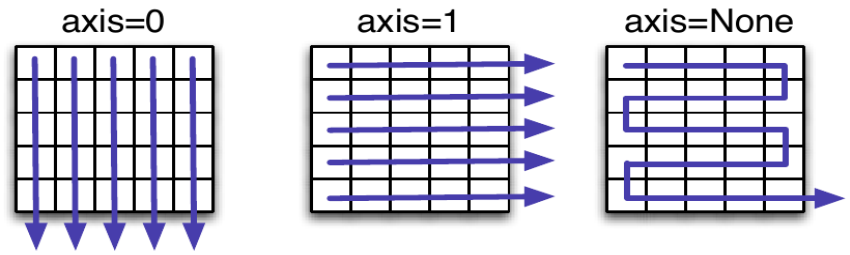
	성능	가격
BC1	70	20000
BC2	70	50000
BC3	90	50000

```
df.sum()      # 각 칼럼별 합계
성능     230
가격    120000
dtype: int64

df.sum(axis=1) # 각 로우별 합계
BC1    20070
BC2    50070
BC3    50090
dtype: int64
```

```
df['ROW_SUM'] = df.sum(axis=1)     # 신규 칼럼 추가
df.loc['COL_SUM', :] = df.sum()    # 신규 로우 추가
```

df

	성능	가격	ROW_SUM
BC1	70	20000	20070
BC2	70	50000	50070
BC3	90	50000	50090
COL_SUM	230	120000	120230



Pandas

```
from IPython.display import display
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 미려하게 출력해줍니다.
display(data_pandas)
```

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

Pandas

```
# Age 열의 값이 30 이상인 모든 행을 선택합니다.
display(data_pandas[data_pandas.Age > 30])
```

	Name	Location	Age
2	Peter	Berlin	53
3	Linda	London	33

1.4.6 **mglearn**

- 이 라이브러리는 그래프나 데이터 적재와 관련한 세세한 코드를 일일이 쓰지 않아도 되게끔 하게 만든 유틸리티 함수

- !pip install mglearn

실습하기 전 라이브러리를 사용

```
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
```

1.7 Application : 붓꽃의 품종 종류

- 한 아마추어 식물학자가 들에서 발견한 붓꽃의 품종을 알고 싶다고 가정.
- 이 식물학자는 붓꽃의 꽃잎과 꽃받침의 폭과 길이를 센티미터 단위로 측정하였습니다

- 또 전문 식물학자가 setosa, versicolor, virginica 종으로 분류한 붓꽃의 측정 데이터도 가지고 있습니다.
- 이 측정값을 이용해서 앞에서 채집한 붓꽃이 어떤 품종인지 구분하려고 합니다. 이 아마추어 식물학자가 야생에서 채집한 붓꽃은 이 세 종류뿐이라고 가정하겠습니다.

Iris

1.7 Application

- 붓꽃의 품종을 정확하게 분류한 데이터를 가지고 있으므로 이 문제는 지도 학습에 속함.
- 이 경우에는 몇 가지 선택사항(붓꽃의 품종) 중 하나를 선택하는 문제.
- 그러므로 이 예는 **분류** 문제에 해당.
- 출력될 수 있는 값(붓꽃의 종류)들을 **클래스라고 함**.

- 데이터 포인트 하나(붓꽃 하나)에 대한 기대 출력은 꽃의 품종이 됩니다. 이런 특정 데이터 포인트에 대한 출력, 즉 품종을 **레이블**.

1.7.1 데이터 적재

- 우리가 사용할 데이터셋은 머신러닝과 통계 분야에서 오래전부터 사용해온 붓꽃 데이터셋입니다.
- 이 데이터는 scikit-learn의 datasets 모듈에 포함되어 있습니다.

1.7.1 데이터 적재

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

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

```
iris_dataset의 키:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

1.7.1 데이터 적재

```
print(iris_dataset['DESCR'][:193] + "₩n...")
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, pre
...
```

1.7.1 데이터 적재

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

```
타깃의 이름: ['setosa' 'versicolor' 'virginica']
```

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

```
특성의 이름:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

1.7.1 데이터 적재

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

```
data의 타입: <class 'numpy.ndarray'>
```

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

```
data의 크기: (150, 4)
```

1.7.1 데이터 적재

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

```
data의 처음 다섯 행:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

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

```
target의 타입: <class 'numpy.ndarray'>
```

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

```
target의 크기: (150,)
```

1.7.1 데이터 적재

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

```
타깃:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

1.7.2 성과 측정 : 훈련 데이터와 테스트 데이터

- 이 데이터로 머신러닝 모델을 만들고 새로운 데이터의 품종을 예측
- 모델을 만들 때 쓴 데이터는 평가 목적으로 사용할 수 없습니다.
- 모델의 성능을 측정하려면 레이블을 알고 있는 (이전에 본 적 없는) 새 데이터를 모델에 적용해봐야 합니다.
- 이 중 하나는 머신러닝 모델을 만들 때 사용하며, **훈련 데이터** 혹은 **훈련 세트**라고 합니다. 나머지는 모델이 얼마나 잘 작동하는지 측정하는 데 사용하며, 이를 **테스트 데이터**, **테스트 세트** 혹은 **홀드아웃 세트**라고 부릅니다.

1.7.2 성과 측정 : 훈련 데이터와 테스트 데이터

```
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)

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

```
X_train 크기: (112, 4)
y_train 크기: (112,)
```

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

```
X_test 크기: (38, 4)
y_test 크기: (38,)
```

1.7.3 가장 먼저 할일 : 데이터 살펴보기

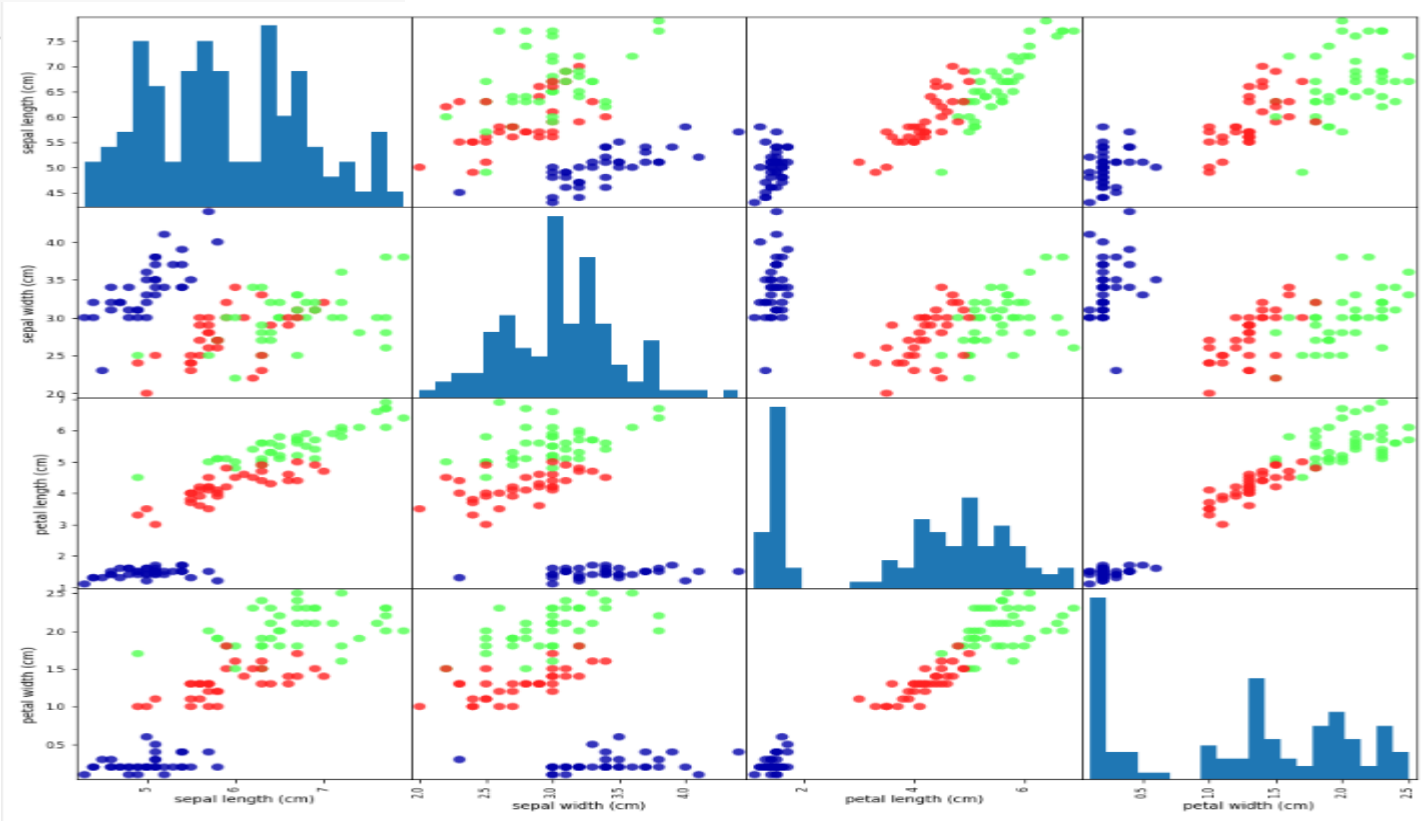
- 머신러닝 모델을 만들기 전에 머신러닝이 없이도 풀 수 있는 문제는 아닌지, 혹은 필요한 정보가 누락되지는 않았는지 데이터를 조사해보는 것이 좋습니다

- 데이터를 탐색하면서 비정상적인 값이나 특이한 값들을 찾을 수도 있습니다

- 시각화는 데이터를 조사하는 아주 좋은 방법

1.7.3 가장 먼저 할일 : 데이터 살펴보기

```
# X_train 데이터를 사용해서 데이터프레임을 만듭니다.
# 열의 이름은 iris_dataset.feature_names에 있는 문자열을 사용합니다.
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# 데이터프레임을 사용해 y_train에 따라 색으로 구분된 산점도 행렬을 만듭니다.
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
```



1.7.4 첫번째 머신러닝 모델 : k-최근접 이웃 알고리즘

정의

가장 근접해 있는 데이터들을 이용하여 예측하는 알고리즘

특징

1. 간단함 : 트레이닝 데이터셋이 곧 모델(근방의 데이터들을 기준으로 예측하므로)
2. Lazy model : 예측을 위해서 항상 계산을 해줘야 함(트레이닝 데이터가 많을 경우 예측 수행 시 오래걸릴 가능성이 높음)
3. 노이즈에 강함 : 학습 데이터 내에 포함된 이상치에 크게 영향을 받지 않음 (참조 : https://ratsgo.github.io/machine%20learning/2017/04/17/KNN/)
4. Hyper Parameter : 근방에 탐색할 데이터 수(k), 데이터 간의 거리(d) (k가 작으면 overfitting 가능성 높고 k가 크면 underfitting 될 가능성이 높음)

k-최근접 이웃 알고리즘

- **관련 sklearn 패키지**
- KNeighborsClassifier : http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- KNeighborsRegressor : http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
- train_test_split :http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

k-최근접 이웃 알고리즘

KNN은 분류 및 회귀 예측 문제 모두에 사용가능하다. KNN은 지도학습 알고리즘에 속한다. 우리의 목표는 x를 활용하여 알지 못하는 y를 자신있게 예측할 수 있는 $h(x)$ 함수를 학습하는 것이다.

1.2 Distance measure

분류 문제에서 k-최근접 이웃(K-nearest neighbor algorithm)은 새로 들어온 x값과 가장 가까운 k개의 점들 중에서 과반수를 가지는 y값을 선택한다. 유사성은 두 데이터 포인트 간의 거리로 정의된다. knn은 일반적으로 새로 들어온 x값과 지정된 트레이닝 x값들 간의 유클리드 거리를 기반으로 한다. x_i는 p개의 변수 $(x_{i1}, x_{i2}, \ldots, x_{ip})$를 가지는 하나의 입력값이고, n은 입력값$(i = 1, 2, \ldots, n)$의 총 개수이다. 샘플 x_i와 x_l 사이의 유클리드 거리는 다음과 같이 정의된다.

$$d(x_i, x_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \ldots + (x_{ip} - x_{lp})^2}$$

때로는 다른 측정 값이 주어진 설정에 더 적합 할 수 있으며 the Manhattan, Chebyshev and Hamming distance를 포함 할 수 있다.

k-최근접 이웃 알고리즘

- Algorithm Steps

STEP 1: 이웃 수 k를 선택하라

STEP 2: 새로운 x값에 대하여 트레이닝 셋과의 유클리디언 거리가 가장 가까운 k개의 이웃 점을 찾아라

STEP 3: k개의 이웃들이 가지는 각 y값 별 나타난 횟수를 세어라

STEP 4: k개의 이웃 중에 가장 많이 등장한 y값을 새로운 x값에 대한 예측 y값으로 할당하라

실습

- https://github.com/Youngpyoryu/Python_basic