



Python 중급 통계분석

중급과정

 금오공과대 수리빅데이터학과
 이 경 준

Pandas

Introduction



Pandas 자료구조

Pandas?

✦ Pandas?

- 시리즈(Series)와 데이터프레임(DataFrame)이라는 구조화된 데이터 형식을 제공
- 서로 다른 종류의 데이터를 한 곳에 담는 그릇의 역할
- 시리즈: 1차원 배열, 데이터프레임: 2차원 배열

✦ 목적

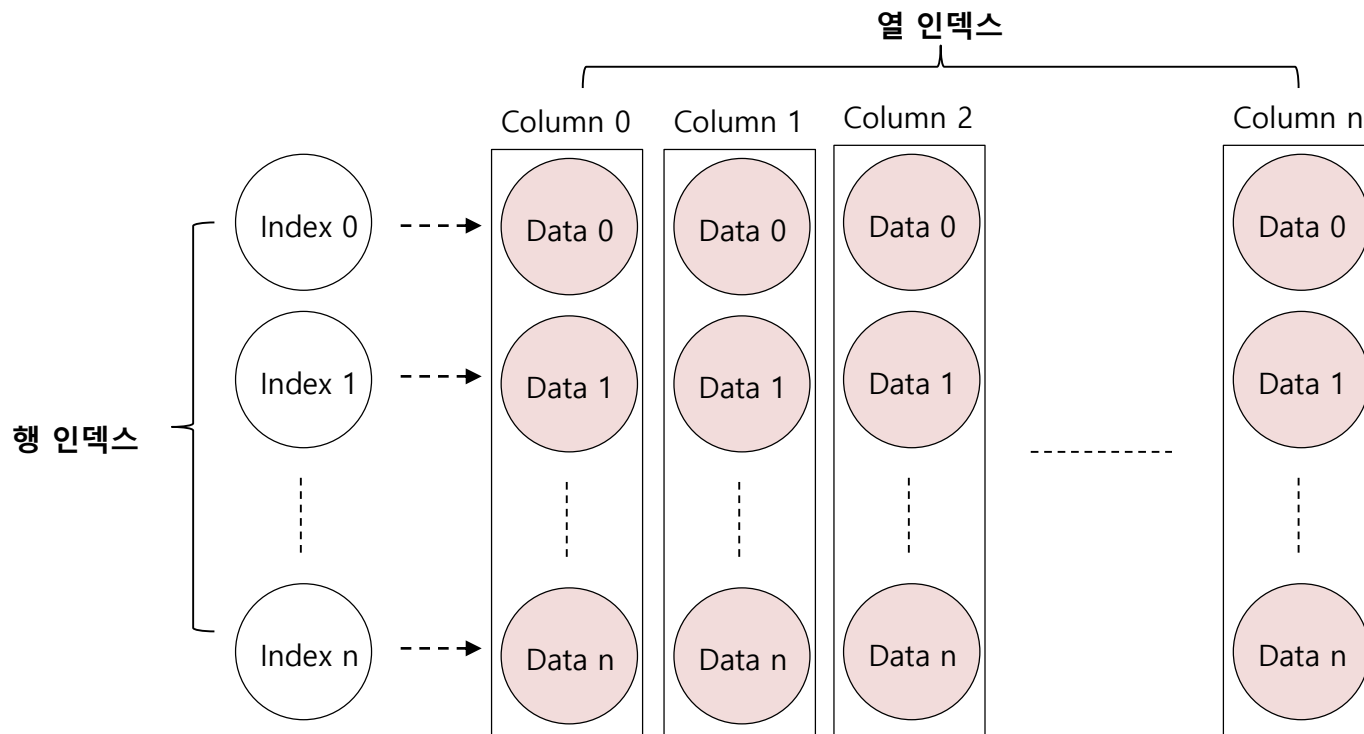
- 서로 다른 여러가지 유형의 데이터를 공통의 포맷으로 정리

Pandas 자료구조

DataFrame

✦ 데이터프레임

- 행과 열로 만들어진 2차원 배열 구조
- 엑셀과 관계형 데이터베이스 등 다양한 분야에서 사용
- Python의 데이터프레임은 R의 데이터프레임에서 유래됨

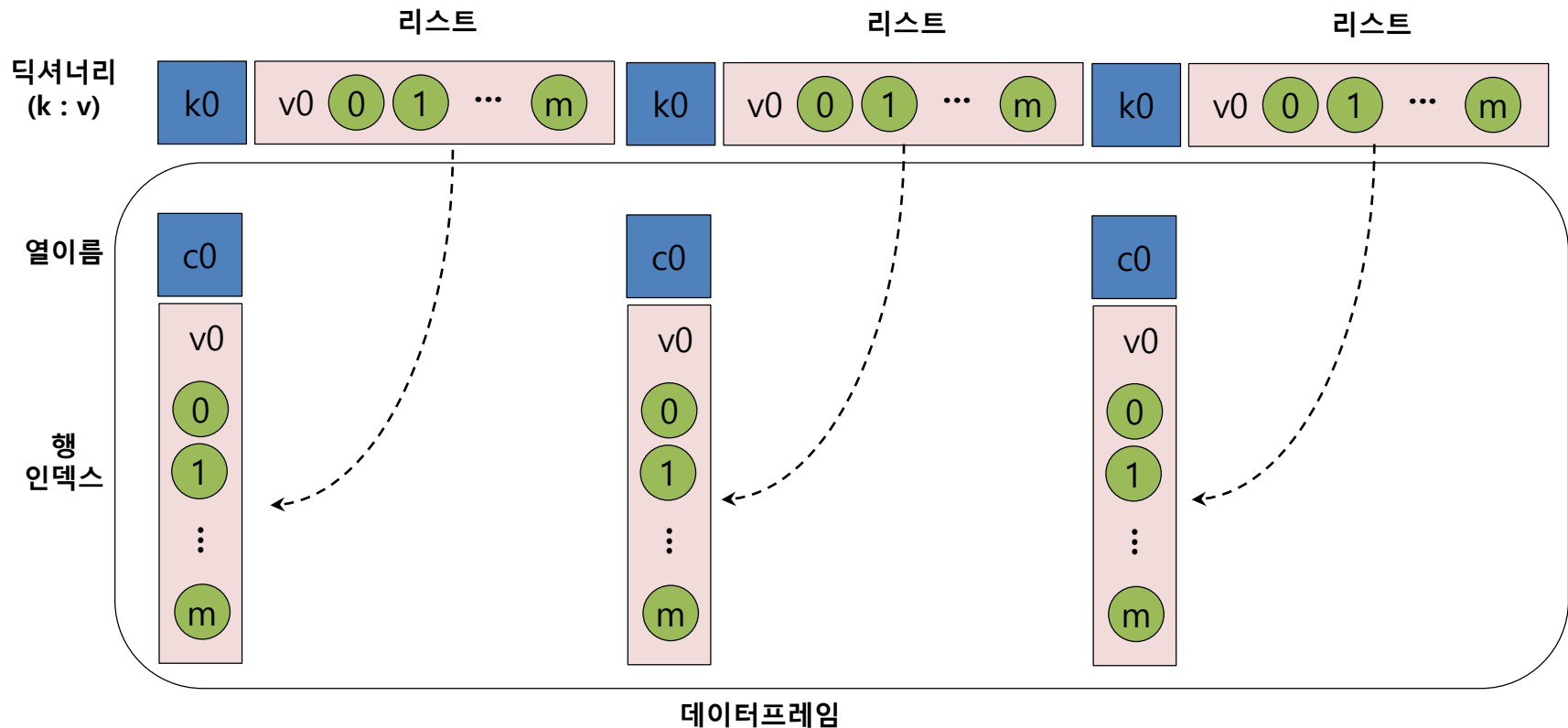


Pandas 자료구조

DataFrame

✦ 데이터프레임 만들기

- 여러 개의 시리즈를 모아 놓은 집합
- 같은 길이의 1차원 배열 여러 개가 필요



Pandas 자료구조

변환

✦ 데이터프레임 변환

딕셔너리 -> 데이터프레임 변환: **pandas.DataFrame**(딕셔너리 객체)

– Ex. 딕셔너리 -> 데이터프레임 변환

```
import pandas as pd

dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
```

Pandas 자료구조

변환

✦ 데이터프레임 변환

딕셔너리 -> 데이터프레임 변환: **pandas.DataFrame**(딕셔너리 객체)

– Ex. 딕셔너리 -> 데이터프레임 변환

```
import pandas as pd

dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}

df = pd.DataFrame(dict_data)

print(type(df))
print('\n')

print(df)
```

Pandas 자료구조

변환

✦ 행 인덱스/열 이름 설정

행 인덱스/열 이름 설정: **pandas.DataFrame**(2차원 배열,
index=행 인덱스 배열,
column=열 이름 배열)

– Ex. 행 인덱스/열 이름 설정

- 행 이름을 준서, 예은으로, 열 이름은 나이, 성별, 학교로 수정

	0	1	2
0	15	남	덕영중
1	17	여	수리중

```
import pandas as pd

df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']])

print(df)
```


Pandas 자료구조

변환

✦ 행 인덱스/열 이름 설정

행 인덱스/열 이름 설정: **pandas.DataFrame**(2차원 배열,
index=행 인덱스 배열,
column=열 이름 배열)

– Ex. 행 인덱스/열 이름 설정

```
import pandas as pd

df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
                  index=['준서', '예은'],
                  columns=['나이', '성별', '학교'])

print(df)
```

Pandas 자료구조

변환

✦ 행 인덱스/열 이름 변경

행 인덱스 변경: DataFrame객체.**index** = 새로운 행 인덱스 배열
 열 이름 변경: DataFrame객체.**columns** = 새로운 열 이름 배열

– rename() 이용

행 인덱스 변경: DataFrame객체.**rename(index={기존인덱스:새인덱스,...})**
 열 이름 변경: DataFrame객체.**rename(columns={기존이름:새이름,...})**

– Ex. 행 인덱스/열 이름 변경

inplace=True: 원본 변경 옵션

- 앞의 자료에서 행 이름을 학생1, 학생2로, 열 이름을 연령, 남녀, 소속으로 변경

```
df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
                  index=['준서', '예은'],
                  columns=['나이', '성별', '학교'])
```

```
print(df)
```

Pandas 자료구조

변환

✦ 행 인덱스/열 이름 변경

행 인덱스 변경: DataFrame객체.**index** = 새로운 행 인덱스 배열
 열 이름 변경: DataFrame객체.**columns** = 새로운 열 이름 배열

– rename() 이용

행 인덱스 변경: DataFrame객체.**rename(index={기존인덱스:새인덱스,...})**
 열 이름 변경: DataFrame객체.**rename(columns={기존이름:새이름,...})**

– Ex. 행 인덱스/열 이름 변경

inplace=True: 원본 변경 옵션

```
df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
                  index=['준서', '예은'],
                  columns=['나이', '성별', '학교'])
df.index=['학생1', '학생2']
df.columns=['연령', '남녀', '소속']

df.rename(columns={'나이':'연령', '성별':'남녀', '학교':'소속'}, inplace=True)
df.rename(index={'학생1':'준서', '학생2':'예은'}, inplace=True)

print(df)
```

Pandas 자료구조

변환

✦ 행/열 삭제

행 삭제: DataFrame객체.**drop**(행 인덱스 또는 배열, **axis=0**)
열 삭제: DataFrame객체.**drop**(열 이름 또는 배열, **axis=1**)

– Ex. 행 삭제

- 아래의 자료에서,
 - 1) 우현의 자료를 삭제
 - 2) 우현과 인아의 자료를 동시에 삭제

```
exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],  
             '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}  
  
df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])  
print(df)
```

Pandas 자료구조

변환

✦ 행/열 삭제

행 삭제: DataFrame객체.**drop**(행 인덱스 또는 배열, **axis=0**)
열 삭제: DataFrame객체.**drop**(열 이름 또는 배열, **axis=1**)

– Ex. 행 삭제

```
exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],  
             '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}  
  
df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])  
print(df)  
print('\n')  
  
df2 = df[:]  
df2.drop('우현', inplace=True)  
print(df2)  
print('\n')  
  
df3 = df[:]  
df3.drop(['우현', '인아'], axis=0, inplace=True)  
print(df3)
```

Pandas 자료구조

변환

✦ 행/열/원소 선택

– 행 선택

구분	loc	iloc
탐색대상	인덱스 이름(index label)	정수형 위치 인덱스(integer position)
범위지정	가능(범위의 끝 포함) Ex. ['a':'c'] -> 'a', 'b', 'c'	가능(범위의 끝 제외) Ex. [1:3]-> 1, 2
비연속 범위 지정	가능(2개의 행 인덱스 사용) Ex. [['a','c']] -> 'a', 'c'	

– 열 선택

열 1개 선택: DataFrame객체["열 이름"] 또는 DataFrame객체.열이름
 열 n개 선택: DataFrame객체[[열1, 열2, ..., 열n]]

– 원소 선택

인덱스 이름: DataFrame객체.loc[행인덱스, 열이름]
 정수 위치 인덱스: DataFrame객체.iloc[행번호, 열번호]

Pandas 자료구조

변환

- Ex. 행/열/원소 선택
 - 아래의 자료에서
 - 1) 서준의 자료 선택
 - 2) 서준과 우현의 자료 선택
 - 3) 수학자료 선택
 - 4) 음악과 체육자료 선택
 - 5) 서준의 음악점수 선택
 - 6) 서준과 우현의 음악과 체육점수 선택

```
exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],  
             '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}  
df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])
```

Pandas 자료구조

변환

- Ex. 행/열/원소 선택

```
exam_data = {'수학' : [ 90, 80, 70], '영어' : [ 98, 89, 95],  
             '음악' : [ 85, 95, 100], '체육' : [ 100, 90, 90]}  
df = pd.DataFrame(exam_data, index=['서준', '우현', '인아'])  
  
#행 선택  
df.loc['서준']; df.iloc[0]  
df.loc[['서준', '우현']; df.iloc[[0, 1]]  
df.loc['서준':'우현']; df.iloc[0:1]  
  
#열 선택  
df['수학']; df.영어  
df[['음악', '체육']; df[['수학']]  
  
#원소 선택  
df.loc['서준', '음악']; df.iloc[0, 2]  
df.loc['서준', ['음악', '체육']; df.iloc[0, [2, 3]]  
df.loc['서준', '음악':'체육']; df.iloc[0, 2:]  
df.loc[['서준', '우현'], ['음악', '체육']; df.iloc[[0, 1], [2, 3]]  
df.loc['서준':'우현', '음악':'체육']; df.iloc[0:2, 2:]
```


Pandas 자료구조

변환

✦ 행/열추가

– 행 추가

행 추가: DataFrame객체.loc["새로운 행 이름"] = 데이터 값 (또는 배열)

– 열 추가

열 추가: DataFrame객체["추가하려는 열 이름"] = 데이터 값

✦ 변경

– 원소 값 변경

원소값 변경: DataFrame객체의 일부분 또는 원소 선택=새로운 값

– 행, 열 위치 변경

행, 열 바꾸기: DataFrame객체.transpose() 또는 DataFrame객체.T

Pandas 자료구조

변환

- Ex. 행/열 추가
 - 아래의 자료에서
 - 1) 동규의 수학(90), 영어(80), 음악(70), 체육(6) 점수를 추가
 - 2) 모든 학생에게 국어점수 80점 동일하게 부여

```
exam_data = {'이름' : ['서준', '우현', '인아'],  
             '수학' : [ 90, 80, 70],  
             '영어' : [ 98, 89, 95],  
             '음악' : [ 85, 95, 100],  
             '체육' : [ 100, 90, 90]}  
df = pd.DataFrame(exam_data)
```

Pandas 자료구조

변환

- Ex. 행/열 추가

```
exam_data = {'이름' : ['서준', '우현', '인아'],
             '수학' : [ 90, 80, 70],
             '영어' : [ 98, 89, 95],
             '음악' : [ 85, 95, 100],
             '체육' : [ 100, 90, 90]}
df = pd.DataFrame(exam_data)

# 행(row)을 추가
df.loc[3] = 0
df.loc[4] = ['동규', 90, 80, 70, 60]
df.loc['행5'] = df.loc[3]

# 열(column)을 추가
df['국어'] = 80
```

Pandas 자료구조

변환

- Ex. 행, 열 바꾸기/원소 추가
 - 아래의 자료에서,
 - 1) 행과 열 바꾸기
 - 2) 서준의 체육점수를 90점으로 바꾸기
 - 3) 서준의 음악과 체육점수를 각각 100점과 50점으로 바꾸기

```
exam_data = {'이름' : ['서준', '우현', '인아'],  
             '수학' : [ 90, 80, 70],  
             '영어' : [ 98, 89, 95],  
             '음악' : [ 85, 95, 100],  
             '체육' : [ 100, 90, 90]}  
df = pd.DataFrame(exam_data)
```

Pandas 자료구조

변환

- Ex. 행, 열 바꾸기/원소 추가

```
exam_data = {'이름' : ['서준', '우현', '인아'],
              '수학' : [ 90, 80, 70],
              '영어' : [ 98, 89, 95],
              '음악' : [ 85, 95, 100],
              '체육' : [ 100, 90, 90]}
df = pd.DataFrame(exam_data)

#행, 열 바꾸기
df.transpose()

#원소 추가
df.set_index( ' 이름 ' , inplace=True)

df.iloc[0][3] = 80
df.loc['서준']['체육'] = 90
df.loc['서준', '체육'] = 100
df.loc['서준', ['음악', '체육']] = 50
df.loc['서준', ['음악', '체육']] = 100, 50
```

Index 활용

Index 설정 및 배열

✦ 행 인덱스를 기준으로 데이터프레임 정렬

행 인덱스 기준 정렬: DataFrame객체.**sort_index()**

✦ 특정 열의 데이터 값을 기준으로 데이터프레임 정렬

- **ascending=False**: 내림차순 정렬

열 기준 정렬: DataFrame객체.**sort_values ()**

Index 활용

Index 설정 및 배열

– Ex. Index 설정 및 배열

- 아래의 자료에서

- 1) new_index를 새로운 index로 변경하고 값이 존재하지 않는 index는 0으로 채우기
- 2) 행 인덱스를 초기화하기
- 3) 행 인덱스 기준으로 정렬하기
- 4) c1 변수의 데이터값 기준으로 내림차순으로 정렬하기

```
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}  
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])  
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
```

```
#행 인덱스 재배열  
df.reindex(new_index, fill_value=0)  
#행 인덱스 초기화  
df.reset_index()  
#행 인덱스 기준 정렬  
df.sort_index(ascending=False)  
#특정 열 데이터값 기준 정렬  
df.sort_values(by='c1', ascending=False)
```

Index 활용

Index 설정 및 배열

– Ex. Index 설정 및 배열

```
exam_data = {'이름' : [ '서준', '우현', '인아'],
              '수학' : [ 90, 80, 70],
              '영어' : [ 98, 89, 95],
              '음악' : [ 85, 95, 100],
              '체육' : [ 100, 90, 90]}
df = pd.DataFrame(exam_data)
#특정 열을 행 인덱스로 설정
df.set_index(['이름'])

dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
new_index = ['r0', 'r1', 'r2', 'r3', 'r4']

#행 인덱스 재배열
df.reindex(new_index, fill_value=0)
#행 인덱스 초기화
df.reset_index()
#행 인덱스 기준 정렬
df.sort_index(ascending=False)
#특정 열 데이터값 기준 정렬
df.sort_values(by='c1', ascending=False)
```


Data Input/Output



외부 파일 읽어오기

Overview

✦ 외부 파일 읽어오기

- Pandas는 다양한 형태의 외부파일을 읽어와서 데이터프레임으로 변화하는 함수 제공
- 어떤 파일이든 데이터프레임으로 변환되고 나면 Pandas의 모든 함수와 기능 사용가능

File Format	Reader	Writer
CSV	read_csv	to_csv
JSON	read_json	to_json
HTML	read_html	to_html
Local clipboard	read_clipboard	to_clipboard
MS Excel	read_excel	to_excel
HDF5 Format	read_hdf	to_hdf
SQL	read_sql	to_sql

외부 파일 읽어오기

CSV 파일

✦ CSV 파일이란?

- 데이터 값을 쉼표(,)로 구분하고 있는 텍스트 파일
- 쉼표로 열을 구분하고 줄바꿈으로 행을 구분

csv파일 -> 데이터프레임: **pandas.read_csv**("파일경로(이름)")

❖ 옵션1:

header=0: 0행을 열 지정

header=1: 1행을 열 지정

header=None: 행을 열 지정하지 않음

❖ 옵션2:

index_col=False: 인덱스 지정하지 않음

index_col="열이름" : 특정 열을 인덱스 지정

외부 파일 읽어오기

CSV 파일

- 옵션 정리

옵션	설명
path	파일의 위치, URL
sep(또는 delimiter)	텍스트 데이터를 필드별로 구분하는 문자
header	열 이름으로 사용될 행의 번호(기본값: 0)
index_col	행 인덱스로 사용할 열의 번호 또는 열 이름
names	열 이름으로 사용할 문자열 리스트
skiprows	처음 몇 줄을 skip 할 것인지 설정(숫자입력) Skip하려는 행의 번호를 담은 리스트도 설정 가능
skip_footer	마지막 몇 줄을 skip 할 것인지를 설정(숫자입력)
encoding	텍스트 인코딩 종류를 지정(예: 'utf-8')

외부 파일 읽어오기

CSV 파일

– Ex. CSV 파일 읽기

```
import pandas as pd

file_path = './read_csv_sample.csv'

df1 = pd.read_csv(file_path)
print(df1)

df2 = pd.read_csv(file_path, header=None)
print(df2)

df3 = pd.read_csv(file_path, index_col=None)
print(df3)

df4 = pd.read_csv(file_path, index_col='c0')
```

	A	B	C	D
1	c0	c1	c2	c3
2	0	1	4	7
3	1	2	5	8
4	2	3	6	9

외부 파일 읽어오기

EXCEL 파일

✦ EXCEL 파일 불러오기

- Excel 파일의 행과 열은 데이터프레임의 행, 열로 일대일 대응
- read_csv() 함수와 거의 비슷하게 사용

Excel파일 -> 데이터프레임: **pandas.read_excel**("파일경로(이름)")

❖ 옵션1:

header=0: 0행을 열 지정

header=1: 1행을 열 지정

header=None: 행을 열 지정하지 않음

❖ 옵션2:

index_col=False: 인덱스 지정하지 않음

index_col="열이름" : 특정 열을 인덱스 지정

외부 파일 읽어오기

EXCEL 파일

– Ex. Excel 파일 읽기

	A	B	C	D	E	F	G	H	I	J	K	L
1	전력량 (억kWh)	발전 전력별	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
2	남한	합계	1,077	1,186	1,310	1,444	1,650	1,847	2,055	2,244	2,153	2,393
3		수력	64	51	49	60	41	55	52	54	61	61
4		화력	484	573	696	803	1,022	1,122	1,264	1,420	1,195	1,302
5		원자력	529	563	565	581	587	670	739	771	897	1,031
6		신재생	-	-	-	-	-	-	-	-	-	-
7	북한	합계	277	263	247	221	231	230	213	193	170	186
8		수력	156	150	142	133	138	142	125	107	102	103
9		화력	121	113	105	88	93	88	88	86	68	83
10		원자력	-	-	-	-	-	-	-	-	-	-

```
import pandas as pd
```

```
df1 = pd.read_excel('./남북한발전전력량.xlsx')
```

```
df2 = pd.read_excel('./남북한발전전력량.xlsx', header=None)
```

```
print(df1)
```

```
print(df2)
```

데이터 저장하기

CSV/EXCEL 파일

✦ CSV 파일로 저장

- 데이터프레임은 2차원 배열 구조 데이터이기 때문에 2차원 구조를 갖는 csv 파일로 변환가능
- `to_csv()` 함수 적용

csv파일로 저장: DataFrame 객체.**to_csv**("파일경로(이름)")

✦ EXCEL 파일로 저장(1)

- 데이터프레임의 행과 열은 Excel파일의 행과 열로 일대일대응
- `to_excel()` 함수 적용 (**openpyxl 라이브러리 설치필요**)

EXCEL파일로 저장: DataFrame 객체.**to_excel**("파일경로(이름)")

데이터 저장하기

CSV/EXCEL 파일

✦ EXCEL 파일로 저장(2)

- 여러 개의 데이터프레임을 하나의 Excel 파일로 저장 가능
- 시트 옵션을 통해 시트 이름 적용하여 저장
- ExcelWriter() 함수 적용

데이터프레임 여러 개를 Excel로 저장: **pandas.ExcelWriter**("파일경로(이름)")

데이터 저장하기

CSV/EXCEL 파일

– Ex. CSV 파일로 저장

```
import pandas as pd

data = {'name' : [ 'Jerry', 'Riah', 'Paul'],
        'algol' : [ "A", "A+", "B"],
        'basic' : [ "C", "B", "B+"],
        'c++' : [ "B+", "C", "C+"],
        }

df = pd.DataFrame(data)
df.set_index('name', inplace=True)
print(df)

df.to_csv("./df_sample.csv")
```

```

           algol basic c++
name
Jerry      A      C  B+
Riah     A+      B   C
Paul      B     B+  C+
```

	A	B	C	D
1	name	algol	basic	c++
2	Jerry	A	C	B+
3	Riah	A+	B	C
4	Paul	B	B+	C+
5				

데이터 저장하기

CSV/EXCEL 파일

- Ex. Excel 파일로 저장

```
import pandas as pd

data = {'name' : [ 'Jerry', 'Riah', 'Paul'],
        'algol' : [ "A", "A+", "B"],
        'basic' : [ "C", "B", "B+"],
        'c++' : [ "B+", "C", "C+"],
        }

df = pd.DataFrame(data)
df.set_index('name', inplace=True)
print(df)

df.to_excel("./df_sample.xlsx")
```

	algol	basic	c++
name			
Jerry	A	C	B+
Riah	A+	B	C
Paul	B	B+	C+

	A	B	C	D
1	name	algol	basic	c++
2	Jerry	A	C	B+
3	Riah	A+	B	C
4	Paul	B	B+	C+
5				

데이터 저장하기

CSV/EXCEL 파일

– Ex. ExcelWriter() 활용

```
import pandas as pd

data1 = {'name' : [ 'Jerry', 'Riah', 'Paul'], 'algol' : [ "A", "A+", "B"],
        'basic' : [ "C", "B", "B+"], 'c++' : [ "B+", "C", "C+"]}
data2 = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}

df1 = pd.DataFrame(data1)
df1.set_index('name', inplace=True)

df2 = pd.DataFrame(data2)
df2.set_index('c0', inplace=True)

writer = pd.ExcelWriter("./df_excelwriter.xlsx")
df1.to_excel(writer, sheet_name="sheet1")
df2.to_excel(writer, sheet_name="sheet2")
writer.save()
```

	A	B	C	D
1	name	algol	basic	c++
2	Jerry	A	C	B+
3	Riah	A+	B	C
4	Paul	B	B+	C+

	A	B	C	D	E
1	c0	c1	c2	c3	c4
2	1	4	7	10	13
3	2	5	8	11	14
4	3	6	9	12	15

Exploratory Data Analysis



데이터프레임의 구조

auto mpg Data Set

✦ 자동차 연비 데이터셋

- 연비, 실린더 수, 배기량, 출력, 차중, 가속능력, 출시년도, 제조국, 모델명
- 398개 데이터로 구성

No.	속성(attributes)		데이터 상세(범위)
1	mpg	연비	연속 값
2	cylinders	실린더 수	이산 값 (예. 3, 4, 6, 8)
3	displacement	배기량	연속 값
4	horsepower	출력	연속 값
5	weight	차중	연속 값
6	acceleration	가속능력	연속 값
7	model_year	출시년도	이산 값 (예. 70, 71, 80, 81)
8	origin	제조국	이산 값 (예. 1(US), 2(EU), 3(JP))
9	name	모델명	문자열

데이터프레임의 구조

데이터 내용 미리보기

✦ 앞/뒷부분 미리보기

- 데이터셋의 내용과 구조를 개략적으로 살펴볼 수 있음
- 이를 통해 분석방향을 정하는데 필요한 정보를 획득
- 데이터 크기가 너무 큰 경우 주로 사용

앞부분 미리보기: DataFrame 객체 **.head**(n)

뒷부분 미리보기: DataFrame 객체 **.tail**(n)

❖ 옵션
n: 행의 개수 지정
디폴트 값: n=5

데이터프레임의 구조

데이터 요약 정보 확인하기

- ✦ 데이터프레임의 크기(행, 열)
 - 데이터프레임의 크기(행, 열)를 tuple의 형태로 출력

데이터프레임의 크기 확인: DataFrame 객체 **.shape**

- ✦ 데이터프레임의 기본 정보
 - 데이터프레임에 관한 기본 정보 출력
 - 클래스 유형, 행 인덱스 구성, 열 이름 종류와 개수, 각 열의 자료형과 개수, 메모리 할당량

데이터프레임의 기본 정보 출력: DataFrame 객체 **.info()**

데이터프레임의 구조

데이터 요약 정보 확인하기

Pandas 자료형	Python 자료형	비고
int64	int	정수형 데이터
float64	float	실수형 데이터
object	string	문자열 데이터
datetime64, timedelta64	없음	시간 데이터

✦ 데이터프레임의 기술 통계 정보 요약

- 산술 데이터를 갖는 열에 대한 주요 기술통계정보(평균, 표준편차, 최대값, 최소값, 중간값)를 출력

데이터프레임의 크기 확인: DataFrame 객체 **.describe()**

데이터프레임의 구조

데이터 내용 미리보기

- Ex. 데이터 미리보기
 - Auth-mpg 자료를 불러온 후,
 - 1) 데이터의 처음과 마지막 5개 자료 불러오기
 - 2) 자료의 기본정보 불러오기
 - 3) 자료의 기술통계량 불러오기
 - 4) 자료의 행과 열의 크기 불러오기

```
import pandas as pd

df = pd.read_csv('./auto-mpg.csv', header=None)
df.columns = ['mpg','cylinders','displacement','horsepower','weight',
              'acceleration','model year','origin','name']
```

데이터프레임의 구조

데이터 내용 미리보기

– Ex. 데이터 미리보기

```
import pandas as pd

df = pd.read_csv('./auto-mpg.csv', header=None)
df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
              'acceleration', 'model year', 'origin', 'name']

print(df.head())
print(df.tail())

print(df.shape)

print(df.info())

print(df.dtypes)

print(df.mpg.dtypes)

print(df.describe())
print(df.describe(include='all'))
```

데이터프레임의 구조

데이터 개수 확인

✦ 각 열의 데이터 개수

- info() 함수는 열에 대한 반환값이 없음
- 유효한 값의 개수만을 계산함

열 데이터 개수 확인: DataFrame 객체.**count**()

✦ 각 열의 고유값 개수

- 각 열의 고유값의 종류와 개수를 확인하여 시리즈 객체를 반환
- 고유값: 행 인덱스, 고유값 개수: 데이터 값

열 데이터 고유값 개수: DataFrame 객체[**"열이름"**].**value_counts**()

❖ 옵션

dropna=False (NaN 포함)

디폴트 값: dropna=True (NaN 미포함)

데이터프레임의 구조

데이터 개수 확인

- Ex. 데이터 개수 확인
 - Auth-mpg 자료를 불러온 후,
 - 1) 유효한 값의 개수 확인하기
 - 2) origin의 고유값 개수 확인하기

```
import pandas as pd

df = pd.read_csv('./auto-mpg.csv', header=None)
df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
              'acceleration', 'model year', 'origin', 'name']
```

데이터프레임의 구조

데이터 개수 확인

– Ex. 데이터 개수 확인

```
import pandas as pd

df = pd.read_csv('./auto-mpg.csv', header=None)
df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
              'acceleration', 'model year', 'origin', 'name']

print(df.count())

print(type(df.count()))

unique_values = df['origin'].value_counts()
print(unique_values)

print(type(unique_values))
```

통계 함수 적용

평균/중간값

✦ 평균값

- 산술 데이터를 갖는 모든 열의 평균 계산
- 계산 결과를 시리즈 객체로 반환

모든 열의 평균값: DataFrame 객체.**mean**()

특정 열의 평균값: DataFrame 객체["열 이름"].**mean**()

✦ 중간값

- 산술 데이터를 갖는 모든 열의 중간값 계산
- 계산 결과를 시리즈 객체로 반환

모든 열의 중간값: DataFrame 객체.**median**()

특정 열의 중간값: DataFrame 객체["열 이름"].**median**()

통계 함수 적용

최대/최소값

✦ 최대값

- 산술 데이터를 갖는 모든 열의 최대값 계산
- 계산 결과를 시리즈 객체로 반환

모든 열의 최대값: DataFrame 객체.**.max()**

특정 열의 최대값: DataFrame 객체**["열 이름"].max()**

✦ 최소값

- 산술 데이터를 갖는 모든 열의 최소값 계산
- 계산 결과를 시리즈 객체로 반환

모든 열의 최소값: DataFrame 객체.**.min()**

특정 열의 최소값: DataFrame 객체**["열 이름"].min()**

통계 함수 적용

표준편차/상관계수

✦ 표준편차

- 산술 데이터를 갖는 모든 열의 표준편차 계산
- 계산 결과를 시리즈 객체로 반환

모든 열의 표준편차: DataFrame 객체.**.std()**

특정 열의 표준편차: DataFrame 객체**[“열 이름”].std()**

✦ 상관계수

- 산술 데이터를 갖는 두 열 간의 상관계수 계산
- 모든 열에 대하여 2개씩 서로 짝을 짓고 각각의 경우에 상관계수 계산

모든 열의 상관계수: DataFrame 객체.**.corr()**

특정 열의 상관계수 : DataFrame 객체**[열 이름 리스트].corr()**

데이터프레임의 구조

평균/중간값

- Ex. 통계함수 적용
 - Auth-mpg 자료를 불러온 후,
 - 1) mpg의 평균, 중앙값 확인하기
 - 2) mpg의 최대, 최소값 확인하기
 - 3) mpg의 표준편차, 상관계수 확인하기

```
import pandas as pd

df = pd.read_csv('./auto-mpg.csv', header=None)
df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
              'acceleration', 'model year', 'origin', 'name']
```

데이터프레임의 구조

평균/중간값

- Ex. 통계함수 적용

```
import pandas as pd

df = pd.read_csv('./auto-mpg.csv', header=None)
df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
              'acceleration', 'model year', 'origin', 'name']

print(df.mean())
print(df['mpg'].mean())
print(df.mpg.mean())
print(df[['mpg', 'weight']].mean())

print(df.median()); print(df['mpg'].median())

print(df.max()); print(df['mpg'].max())

print(df.min()); print(df['mpg'].min())

print(df.std()); print(df['mpg'].std())

print(df.corr()); print(df[['mpg', 'weight']].corr())
```

Data Preprocessing



Overview

Overview

✦ 데이터 품질 향상

- 데이터 분석의 정확도는 분석 데이터의 품질에 의해 좌우
- 누락 데이터, 중복 데이터 등 오류를 수정하고 분석 목적에 맞게 변형하는 과정 필요

✦ 데이터 프레임

- 데이터를 파일로 입력할 때 빠트리거나 파일 형식 변환 중 소실되는 경우 발생
- 유효한 값이 존재하지 않는 누락 데이터를 NaN으로 표시

✦ 데이터 전처리의 필요성

- 누락데이터가 많아지면 데이터의 품질이 떨어지고,
- 분석 결과를 왜곡하는 현상이 발생

누락 데이터 처리

누락 데이터 확인

✦ 누락 데이터 개수 확인

- info()
 - 데이터 프레임의 요약 정보를 출력
 - 각 열에 속하는 데이터 중 유효한 값의 개수를 보여줌
- value_counts()
 - 선택된 열의 범주별 개수를 출력
 - dropna=False를 통해 누락 데이터 개수 확인 가능

✦ 누락 데이터 여부 확인

- 누락 데이터를 직접적으로 찾는 방법
- isnull()
 - 누락 데이터면 True로 반환, 유효 데이터면 False
- notnull()
 - 유효 데이터면 True로 반환, 누락 데이터면 False

누락 데이터 처리

누락 데이터 확인

- 누락 데이터 확인

```
import seaborn as sns
df = sns.load_dataset('titanic')

nan_deck = df['deck'].value_counts(dropna=False)
print(nan_deck)

print(df.head().isnull())

print(df.head().notnull())

print(df.head().isnull().sum(axis=0))
```

NaN	688
C	59
B	47
D	33
E	32
A	15
F	13
G	4

```
survived  pclass  sex    age  ...  deck  embark_town  alive  alone
0      False   False  False  False  ...   True      False  False  False
1      False   False  False  False  ...   False     False  False  False
2      False   False  False  False  ...   True      False  False  False
3      False   False  False  False  ...   False     False  False  False
4      False   False  False  False  ...   True      False  False  False

[5 rows x 15 columns]
survived  pclass  sex    age  ...  deck  embark_town  alive  alone
0      True    True  True  True  ...   False     True  True  True
1      True    True  True  True  ...    True     True  True  True
2      True    True  True  True  ...   False     True  True  True
3      True    True  True  True  ...    True     True  True  True
4      True    True  True  True  ...   False     True  True  True
```

```
survived    0
pclass      0
sex         0
age         0
sibsp       0
parch       0
fare        0
embarked    0
class       0
who         0
adult_male  0
deck        3
embark_town 0
alive       0
alone       0
dtype: int64
```

누락 데이터 처리

누락 데이터 제거

✦ 누락 데이터가 들어있는 행(열) 제거

- `dropna ()`
 - 조건에 맞는 행(열) 제거

객체.**dropna** (axis=행/열 , thresh=유효데이터개수 기준)

: 조건에 맞는 행/열 제거

객체.**dropna** (subset=행/열이름 , how= , axis=행/열)

: 선택한 행/열 제거

❖ dropna 옵션1:

axis=0(행)/1(열)

thresh=제거할 행/열의 유효데이터개수 기준

dropna 옵션2:

subset=제거할 행(열) 선택

how='any'(1의 NaN이 있을 때 삭제),

'all'(모두 NaN일 때 삭제)

누락 데이터 처리

누락 데이터 제거

- 누락 데이터 제거

```
import seaborn as sns
df = sns.load_dataset('titanic')
```

```
missing_df = df.isnull()
for col in missing_df.columns:
    missing_count = missing_df[col].value_counts()
    try:
        print(col, ': ', missing_count[True])
    except:
        print(col, ': ', 0)
```

```
df_thresh = df.dropna(axis=1, thresh=500)
print(df_thresh.columns)
df_age = df.dropna(subset=['age'], how='any', axis=0)
print(len(df_age))
```

```
survived : 0
pclass : 0
sex : 0
age : 177
sibsp : 0
parch : 0
fare : 0
embarked : 2
class : 0
who : 0
adult_male : 0
deck : 688
embark_town : 2
alive : 0
alone : 0
```

```
In [19]: df_thresh = df.dropna(axis=1, thresh=500)
...: print(df_thresh.columns)
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
      'embarked', 'class', 'who', 'adult_male', 'embark_town', 'alive',
      'alone'],
      dtype='object')

In [20]: df_age = df.dropna(subset=['age'], how='any', axis=0)
...: print(len(df_age))
...:
```

714

누락 데이터 처리

누락 데이터 치환

✦ 누락 데이터 치환

- 누락 데이터를 바꿔서 대체할 값 지정
- 데이터의 분포와 특성을 잘 나타낼 수 있는 평균, 최빈값 등을 활용
- `fillna()`
 - 누락 데이터 대체

객체.**fillna** (대체값, `inplace=True/False`)

: 대체값으로 NaN을 대체

객체.**fillna** (`method='ffill'`, `inplace=True/False`)

: NaN 바로 앞의 값으로 NaN을 대체

누락 데이터 처리

누락 데이터 치환

- 누락 데이터 치환

```
import seaborn as sns
df = sns.load_dataset('titanic')
```

```
print(df['age'].head(10))
mean_age = df['age'].mean(axis=0)
df['age'].fillna(mean_age, inplace=True)
print(df['age'].head(10))
```

```
df = sns.load_dataset('titanic')
print(df['embark_town'][825:830])
most_freq = df['embark_town'].value_counts(dropna=True).idxmax()
print(most_freq)
df['embark_town'].fillna(most_freq, inplace=True)
print(df['embark_town'][825:830])
```

```
df = sns.load_dataset('titanic')
print(df['embark_town'][825:830])
df['embark_town'].fillna(method='ffill', inplace=True)
print(df['embark_town'][825:830])
```

```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: age, dtype: float64
```

```
0    22.000000
1    38.000000
2    26.000000
3    35.000000
4    35.000000
5    29.699118
6    54.000000
7     2.000000
8    27.000000
9    14.000000
Name: age, dtype: float64
```

```
825    Queenstown
826    Southampton
827     Cherbourg
828    Queenstown
829             NaN
Name: embark_town, dtype: object
```

```
825    Queenstown
826    Southampton
827     Cherbourg
828    Queenstown
829    Southampton
Name: embark_town, dtype: object
```

```
825    Queenstown
826    Southampton
827     Cherbourg
828    Queenstown
829    Queenstown
Name: embark_town, dtype: object
```

중복 데이터 처리

중복 데이터 확인

- 하나의 데이터 셋에서 동일한 관측값이 2개 이상 중복되는 경우 삭제

★ 중복 데이터 여부 확인

- duplicated()
 - 전에 나온 행들과 비교하여 중복되는 행이면 True 반환
 - 처음 나오는 행에 대해서는 False 반환
- 중복 데이터 확인

```
import pandas as pd

df = pd.DataFrame({'c1':['a', 'a', 'b', 'a', 'b'],
                   'c2':[1, 1, 1, 2, 2],
                   'c3':[1, 1, 2, 2, 2]})

df_dup = df.duplicated()
print(df_dup)
```

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

0	False
1	True
2	False
3	False
4	False
dtype: bool	

중복 데이터 처리

중복 데이터 제거

✦ 중복 데이터 제거

- `drop_duplicates()`
 - 중복되는 행을 제거하고 고유한 관측값을 가진 행들만 남김

객체.**drop_duplicates** ()
: 전체 데이터셋을 기준으로 중복되는 행 제거

객체.**drop_duplicates** (subset=기준열)
: 선택된 열을 기준으로 중복되는 행 제거

- 중복 데이터 확인

```
import pandas as pd

df = pd.DataFrame({'c1':['a', 'a', 'b', 'a', 'b'],
                  'c2':[1, 1, 1, 2, 2],
                  'c3':[1, 1, 2, 2, 2]})

df2 = df.drop_duplicates()
df3 = df.drop_duplicates(subset=['c2', 'c3'])
```

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2

자료형 변환

자료형 변환

- 숫자가 문자열로 저장된 경우 숫자형으로 변환

✦ 자료형 변환

객체.**dtypes** : 자료형 확인
객체.**unique** () : 열의 고유값 확인
객체.**astype** ('변경할 자료형') : 선택한 자료형으로 변환

✦ 지정값 변경

- '?'를 np.nan(누락데이터)로 변경
 - numpy를 통해 NaN 생성

객체.**replace** ('?', np.nan): ?를 누락데이터로 변경
객체.**replace** ({지정값1:변경값1, 지정값2:변경값2, ... }, inplace=)
: 여러 개의 값을 변경

자료형 변환

자료형 변환

– 자료형 변환

```
import pandas as pd
df = pd.read_csv('./auto-mpg.csv', header=None)
df.columns = ['mpg','cylinders','displacement','horsepower','weight',
              'acceleration','model year','origin','name']
print(df.dtypes);print(df['horsepower'].unique())
```

```
import numpy as np
df['horsepower'].replace('?', np.nan, inplace=True)
df.dropna(subset=['horsepower'], axis=0, inplace=True)
df['horsepower'] = df['horsepower'].astype('float')
print(df['horsepower'].dtypes);print(df['origin'].unique())
```

```
df['origin'].replace({1:'USA', 2:'EU', 3:'JAPAN'}, inplace=True)
print(df['origin'].unique());print(df['origin'].dtypes)
```

```
df['origin'] = df['origin'].astype('category')
print(df['origin'].dtypes)
df['origin'] = df['origin'].astype('str')
print(df['origin'].dtypes)
```

```
df['model year'] = df['model year'].astype('category')
print(df['model year'].sample(3))
```

```
mpg          float64
cylinders    int64
displacement float64
horsepower   object
weight       float64
acceleration float64
model year   int64
origin       int64
name         object
dtype: object
```

```
['130.0' '165.0' '150.0' '140.0' '198.0' '220.0' '215.0' '225.0' '190.0'
'170.0' '160.0' '95.00' '97.00' '85.00' '88.00' '46.00' '87.00' '90.00'
'113.0' '200.0' '210.0' '193.0' '?' '100.0' '105.0' '175.0' '153.0'
'180.0' '110.0' '72.00' '86.00' '70.00' '76.00' '65.00' '69.00' '60.00'
'80.00' '54.00' '208.0' '155.0' '112.0' '92.00' '145.0' '137.0' '158.0'
'167.0' '94.00' '107.0' '230.0' '49.00' '75.00' '91.00' '122.0' '67.00'
'83.00' '78.00' '52.00' '61.00' '93.00' '148.0' '129.0' '96.00' '71.00'
'98.00' '115.0' '53.00' '81.00' '79.00' '120.0' '152.0' '102.0' '108.0'
'68.00' '58.00' '149.0' '89.00' '63.00' '48.00' '66.00' '139.0' '103.0'
'125.0' '133.0' '138.0' '135.0' '142.0' '77.00' '62.00' '132.0' '84.00'
'64.00' '74.00' '116.0' '82.00']
```

범주형 데이터 처리

구간 분할

- 연속 데이터를 그대로 사용하기 보다 일정한 구간으로 나눠 분석하는 것이 효율적인 경우도 존재

★ 구간 분할

- 연속 변수를 일정구간으로 나누고 각 구간을 범주형 이산 변수로 변환하는 과정
- `cut()`을 이용하여 연속 데이터를 범주형 데이터로 변환

pd.cut (x=데이터배열, bins=경계값 리스트
labels=bin 이름, include_lowest=True/False)

- 경계값 구하는 방법

- numpy의 `histogram()` 함수 활용
- 구간의 개수 bins 옵션을 통해 구간에 속하는 값(count)과 경계값 리스트 (bin_dividers) 반환

❖ cut 옵션:

`include_lowest=True`(첫 경계값 포함)/
`False`(첫 경계값 미포함)

범주형 데이터 처리

구간 분할

- 구간 분할

```
import pandas as pd
import numpy as np

df = pd.read_csv('./auto-mpg.csv', header=None)

df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
              'acceleration', 'model year', 'origin', 'name']

df['horsepower'].replace('?', np.nan, inplace=True)
df.dropna(subset=['horsepower'], axis=0, inplace=True)
df['horsepower'] = df['horsepower'].astype('float')

count, bin_dividers = np.histogram(df['horsepower'], bins=3)
bin_names = ['저출력', '보통출력', '고출력']

df['hp_bin'] = pd.cut(x=df['horsepower'], bins=bin_dividers,
                     labels=bin_names, include_lowest=True)

print(df[['horsepower', 'hp_bin']].head(15))
```

	horsepower	hp_bin
0	130.0	보통출력
1	165.0	보통출력
2	150.0	보통출력
3	150.0	보통출력
4	140.0	보통출력
5	198.0	고출력
6	220.0	고출력
7	215.0	고출력
8	225.0	고출력
9	190.0	고출력
10	170.0	고출력
11	160.0	보통출력
12	150.0	보통출력
13	225.0	고출력
14	95.0	저출력

범주형 데이터 처리

더미 변수

✦ 더미변수 (dummy variable)

- 범주형 데이터를 회귀분석등에 사용할 경우 컴퓨터가 인식 가능한 입력값
- 0(특성 없음)또는 1(특성 존재)로 표현
- 원핫인코딩(one-hot-encoding)이라고도 함
- `get_dummies()` 함수 사용

pd.get_dummies (범주형 데이터)

범주형 데이터 처리

더미 변수

- 더미 변수

```
import pandas as pd
import numpy as np

df = pd.read_csv('./auto-mpg.csv', header=None)

df.columns = ['mpg','cylinders','displacement','horsepower','weight',
              'acceleration','model year','origin','name']

df['horsepower'].replace('?', np.nan, inplace=True)
df.dropna(subset=['horsepower'], axis=0, inplace=True)
df['horsepower'] = df['horsepower'].astype('float')

count, bin_dividers = np.histogram(df['horsepower'], bins=3)
bin_names = ['저출력', '보통출력', '고출력']

df['hp_bin'] = pd.cut(x=df['horsepower'], bins=bin_dividers,
                     labels=bin_names, include_lowest=True)

horsepower_dummies = pd.get_dummies(df['hp_bin'])
print(horsepower_dummies.head(15))
```

	저출력	보통출력	고출력
0	0	1	0
1	0	1	0
2	0	1	0
3	0	1	0
4	0	1	0
5	0	0	1
6	0	0	1
7	0	0	1
8	0	0	1
9	0	0	1
10	0	0	1
11	0	1	0
12	0	1	0
13	0	0	1
14	1	0	0

열 재구성

열 분리

✦ 열 분리

- 하나의 열이 여러가지 정보를 담고 있을 때 각 정보를 분리
- 시리즈의 문자열 리스트 인덱싱

series객체.**str.split('구분기호').str.get(인덱스)**

- 열 분리

```
import pandas as pd
```

```
df = pd.read_excel('./주가데이터.xlsx')
print(df.head(), '\n')
```

```
df['연월일'] = df['연월일'].astype('str')
dates = df['연월일'].str.split('-')
```

```
df['연'] = dates.str.get(0)
df['월'] = dates.str.get(1)
df['일'] = dates.str.get(2)
print(df.head())
```

	연월일	당일종가	전일종가	시가	고가	저가	거래량
0	2018-07-02	10100	600	10850	10900	10000	137977
1	2018-06-29	10700	300	10550	10900	9990	170253
2	2018-06-28	10400	500	10900	10950	10150	155769
3	2018-06-27	10900	100	10800	11050	10500	133548
4	2018-06-26	10800	350	10900	11000	10700	63039

```
0    [2018, 07, 02]
1    [2018, 06, 29]
2    [2018, 06, 28]
3    [2018, 06, 27]
4    [2018, 06, 26]
Name: 연월일, dtype: object
```

	연월일	당일종가	전일종가	시가	고가	저가	거래량	연	월	일
0	2018-07-02	10100	600	10850	10900	10000	137977	2018	07	02
1	2018-06-29	10700	300	10550	10900	9990	170253	2018	06	29
2	2018-06-28	10400	500	10900	10950	10150	155769	2018	06	28
3	2018-06-27	10900	100	10800	11050	10500	133548	2018	06	27
4	2018-06-26	10800	350	10900	11000	10700	63039	2018	06	26

필터링

불린 인덱싱

- 특정 조건식을 만족하는 원소만 따로 추출

✦ 불린 인덱싱

- 시리즈 객체에 조건식을 적용하여 각 원소에 대해 참/거짓을 판별하여 불린 (참, 거짓) 값으로 구성된 시리즈 반환
- 참에 해당하는 데이터 값을 따로 선택 가능
- 조건식(>, <, ==, ...)을 적용
- &(and) 연산자로 결합하여 두 조건식이 모두 참인 경우 추출
- |(or) 연산자로 결합하여 두 조건 중 하나라도 참인 경우 추출
- 데이터프레임의 불린 인덱싱

객체 **[불린 시리즈]**

필터링

불린 인덱싱

- 불린 인덱싱

```
import seaborn as sns
```

```
titanic = sns.load_dataset('titanic')
```

```
mask1 = (titanic.age >= 10) & (titanic.age < 20)
df_teenage = titanic.loc[mask1, :]
print(df_teenage.head())
```

```
mask2 = (titanic.age < 10) & (titanic.sex == 'female')
df_female_under10 = titanic.loc[mask2, :]
print(df_female_under10.head())
```

```
mask3 = (titanic.age < 10) | (titanic.age >= 60)
df_under10_morethan60 = titanic.loc[mask3, ['age', 'sex', 'alone']]
print(df_under10_morethan60.head())
```

	age	sex	alone
7	2.0	male	False
10	4.0	female	False
16	2.0	male	False
24	8.0	female	False
33	66.0	male	True

	survived	pclass	sex	age	...	deck	embark_town	alive	alone
9	1	2	female	14.0	...	NaN	Cherbourg	yes	False
14	0	3	female	14.0	...	NaN	Southampton	no	True
22	1	3	female	15.0	...	NaN	Queenstown	yes	True
27	0	1	male	19.0	...	C	Southampton	no	False
38	0	3	female	18.0	...	NaN	Southampton	no	False

[5 rows x 15 columns]

	survived	pclass	sex	age	...	deck	embark_town	alive	alone
10	1	3	female	4.0	...	G	Southampton	yes	False
24	0	3	female	8.0	...	NaN	Southampton	no	False
43	1	2	female	3.0	...	NaN	Cherbourg	yes	False
58	1	2	female	5.0	...	NaN	Southampton	yes	False
119	0	3	female	2.0	...	NaN	Southampton	no	False

[5 rows x 15 columns]

필터링

isin() 메소드 활용

- isin()에 데이터프레임의 열에서 추출하려는 값들로 만든 리스트 전달

열 객체.isin(추출값의 리스트)

- isin() 활용

```
import seaborn as sns
import pandas as pd

titanic = sns.load_dataset('titanic')

pd.set_option('display.max_columns', 10)

mask3 = titanic['sibsp'] == 3
mask4 = titanic['sibsp'] == 4
mask5 = titanic['sibsp'] == 5
df_boolean = titanic[mask3 | mask4 | mask5]

isin_filter = titanic['sibsp'].isin([3, 4, 5])
df_isin = titanic[isin_filter]
print(df_isin.head())
```

	survived	pclass	sex	age	sibsp	...	adult_male	deck
7	0	3	male	2.0	3	...	False	NaN
16	0	3	male	2.0	4	...	False	NaN
24	0	3	female	8.0	3	...	False	NaN
27	0	1	male	19.0	3	...	True	C
50	0	3	male	7.0	4	...	False	NaN

	embark_town	alive	alone
7	Southampton	no	False
16	Queenstown	no	False
24	Southampton	no	False
27	Southampton	no	False
50	Southampton	no	False

데이터프레임 합치기

데이터프레임 연결

- 데이터프레임을 합치거나 연결할 때 사용

✦ 데이터프레임 연결: `concat()`

- 서로 다른 데이터프레임들의 구성 형태와 속성이 균일할 때
- 기존 데이터프레임의 형태를 유지하면서 이어 붙이는 개념
- `concat()` 함수에 데이터프레임을 원소로 갖는 리스트를 전달하면 여러 개의 데이터프레임을 서로 연결

pandas.concat(데이터프레임의 리스트, `ignore_index=True/False`,
`axis=0/1`, `join='outer'/'inner'`)

❖ `concat` 옵션:

`ignore_index=True`(기존 행 인덱스 무시)
`axis=0`(행 연결)/`1`(열 연결)
`join='outer'`(합집합으로 연결)/
`'inner'`(교집합으로 연결)

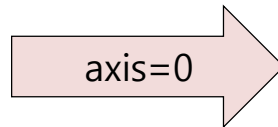
데이터프레임 합치기

데이터프레임 연결

	A	B	C
0			
1			
2			



	B	C	D	E
5				
6				

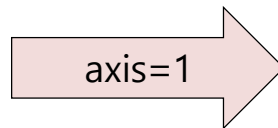


	A	B	C	D	E
0				NaN	NaN
1				NaN	NaN
2				NaN	NaN
5	NaN				
6	NaN				

	A	B	C
0			
1			



	A	B
0		
1		
2		



	A	B	C	A	B
0					
1					
2	NaN	NaN	NaN		

데이터프레임 합치기

데이터프레임 연결

– concat()

```
import pandas as pd
df1 = pd.DataFrame({'a': ['a0', 'a1', 'a2', 'a3'], 'b': ['b0', 'b1', 'b2', 'b3'],
                    'c': ['c0', 'c1', 'c2', 'c3']},
                    index=[0, 1, 2, 3])
df2 = pd.DataFrame({'a': ['a2', 'a3', 'a4', 'a5'], 'b': ['b2', 'b3', 'b4', 'b5'],
                    'c': ['c2', 'c3', 'c4', 'c5'], 'd': ['d2', 'd3', 'd4', 'd5']},
                    index=[2, 3, 4, 5])

result1 = pd.concat([df1, df2])
result2 = pd.concat([df1, df2], ignore_index=True)
result3 = pd.concat([df1, df2], axis=1)
result3_in = pd.concat([df1, df2], axis=1, join='inner')

sr1 = pd.Series(['e0', 'e1', 'e2', 'e3'], name='e')
sr2 = pd.Series(['f0', 'f1', 'f2'], name='f', index=[3, 4, 5])
sr3 = pd.Series(['g0', 'g1', 'g2', 'g3'], name='g')

result4 = pd.concat([df1, sr1], axis=1)
result5 = pd.concat([df2, sr2], axis=1, sort=True)
result6 = pd.concat([sr1, sr3], axis=1)
result7 = pd.concat([sr1, sr3], axis=0)
```

데이터프레임 합치기

데이터프레임 병합

✦ 데이터프레임 병합: merge()

- 어떤 기준(키)에 의해 두 데이터프레임을 병합
- 키가 되는 열/인덱스는 양쪽 데이터프레임에 모두 존재해야 함

```
pandas.merge(df_left, df_right, how='outer'/'inner',  
              on=기준변수, how='left'/'right', left_on= , right_on= )
```

❖ merge 옵션:

how='outer'(합집합으로 연결)/

'inner'(교집합으로 연결)

on=변수명: 기준이 되는 변수

None인 경우 공통의 모든 열을 기준

how='left'(왼쪽 데이터프레임 키에 속하는 데이터값 기준 병합)

'right'(오른쪽 데이터프레임 키에 속하는 데이터값 기준 병합)

left_on=변수명: 좌우 다른 키를 줄 경우 왼쪽 데이터프레임 키

right_on=변수명: 좌우 다른 키를 줄 경우 오른쪽 데이터프레임 키

데이터프레임 합치기

데이터프레임 병합

	A	B	C
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2



	A	B	D
0	0a	0b	0d
1	a1	b1	d1

병합(merge)

(how='outer', on='A')

	A	B_x	C	B_y	D
0	a0	b0	c0	NaN	NaN
1	a1	b1	c1	b1	d1
2	a2	b2	c2	NaN	NaN
3	0a	NaN	NaN	0b	0d

(how='inner', on='A')

	A	B_x	C	B_y	D
0	a1	b1	c1	b1	d1

데이터프레임 합치기

데이터프레임 병합

- merge()

```
import pandas as pd

pd.set_option('display.max_columns', 10)
pd.set_option('display.max_colwidth', 20)
pd.set_option('display.unicode.east_asian_width', True)

df1 = pd.read_excel('./stock price.xlsx')
df2 = pd.read_excel('./stock valuation.xlsx')

merge_inner = pd.merge(df1, df2)
merge_outer = pd.merge(df1, df2, how='outer', on='id')

merge_left = pd.merge(df1, df2, how='left', left_on='stock_name',
right_on='name')
merge_right = pd.merge(df1, df2, how='right', left_on='stock_name',
right_on='name')

price = df1[df1['price'] < 50000]
value = pd.merge(price, df2)
```

Machine Learning



머신러닝 개요

머신러닝이란?

✦ 머신러닝(machine learning)

- 기계 스스로 데이터를 학습하여 서로 다른 변수 간의 관계를 찾아 나가는 과정
- 예측(prediction), 분류(classification), 군집(clustering) 알고리즘 등으로 분류
 - Ex. 주가, 환율 등 경제지표 예측, 은행에서 고객을 분류하여 대출을 승인하거나 거절하는 문제, 비슷한 소비패턴을 가진 고객 유형을 군집으로 묶어내는 문제

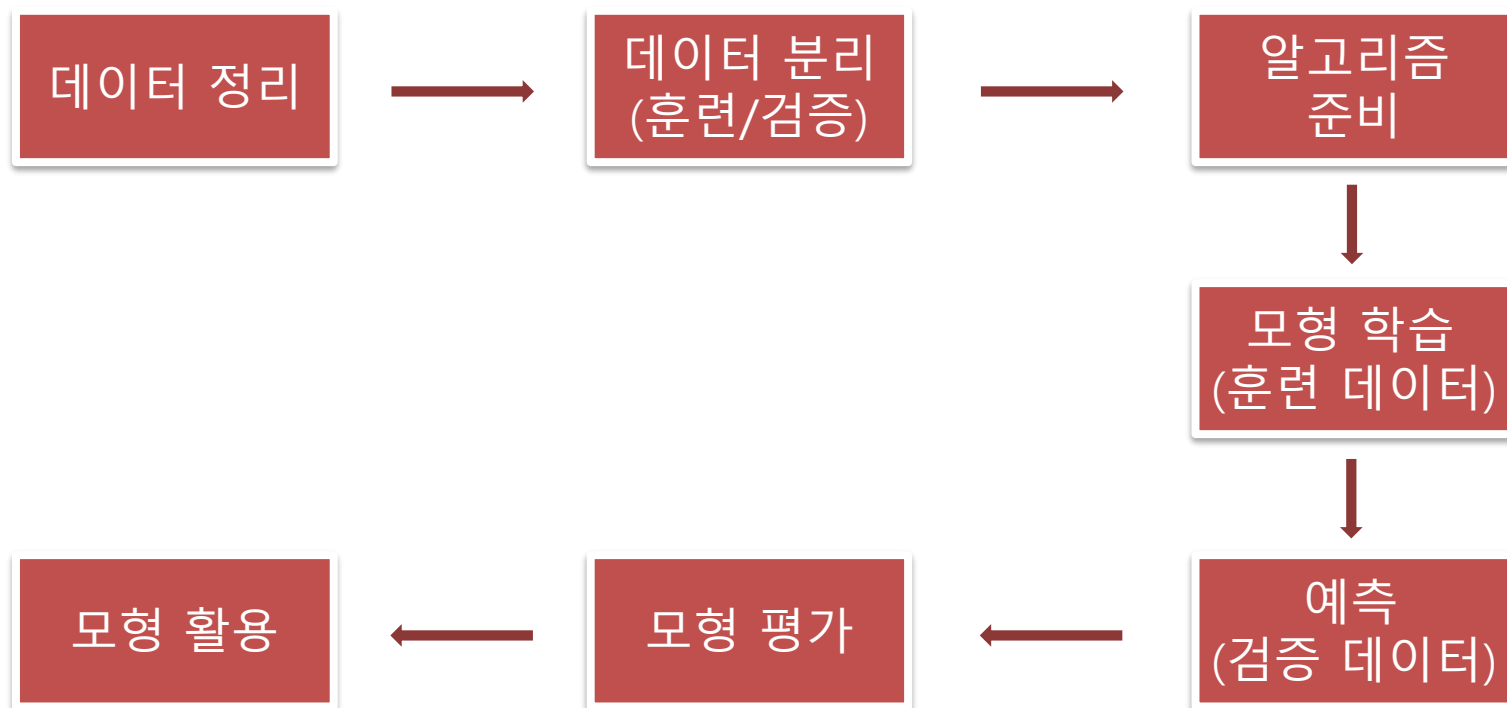
✦ 지도학습 vs 비지도학습

- 지도학습(supervised learning)
 - 정답 데이터를 다른 데이터와 함께 컴퓨터 알고리즘에 입력하는 방식
 - 정답지가 있어서 정답을 맞춰 보면서 문제를 풀어나가는 학습방식
- 비지도학습(unsupervised learning)
 - 정답 데이터 없이 컴퓨터 알고리즘 스스로 데이터로부터 숨은 패턴을 찾아내는 방식
 - 정답지 없이 스스로 답을 찾아내는 학습방식

머신러닝 개요

머신러닝이란?

✦ 머신러닝 프로세스



교차검정

Cross Validation

✦ Train/Test set 분리

– 목적: Overfitting 방지

- Train data를 100% 학습시킨 후 test data에 모델을 적용했을 때 성능이 생각보다 안나오는 경우가 대부분 => overfitting
- 즉, 모델이 train data에 너무 과적합되도록 학습한 나머지, 조금이라도 벗어난 케이스에 대해 예측률이 현저하게 떨어짐
- Train data와 Test data를 일정 비율로 나눈 다음 Test data로 학습한 모델을 평가
- Hold-out 방식, K-Fold 교차 검증 방식

교차검정

Cross Validation

✦ Hold-out 방식

- 전체 데이터셋을 두 개의 데이터 셋(Train, Test)로 나누는 방법



- Train Data Set: 모델 수립
- Test Data Set: 모델의 성능 측정
- 방법: `train_test_split()` 함수 이용

```
pandas.train_test_split(data, target, test_size= , random_state= ,  
                           stratify=target, shuffle=True)
```

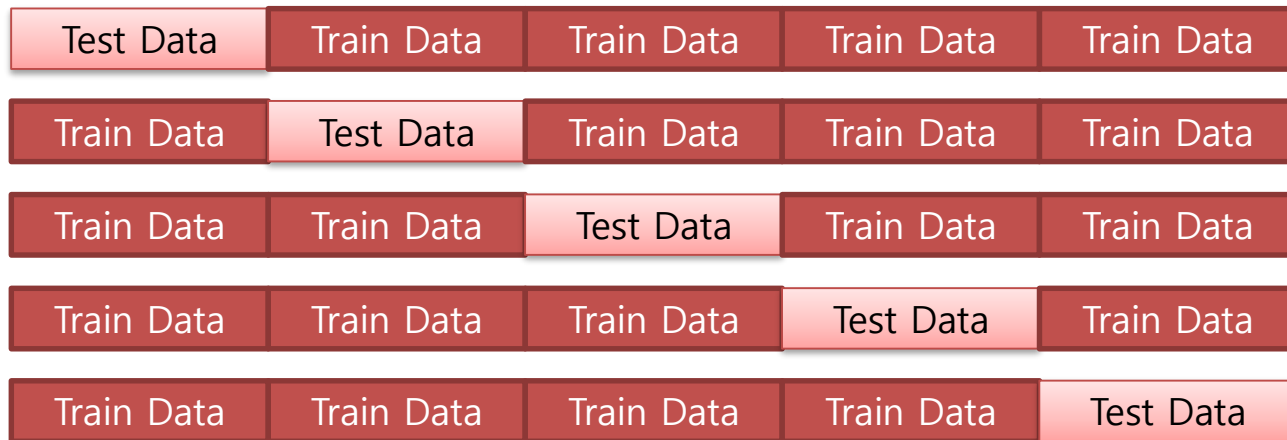
- 단점
 - Train과 Test Data Set이 어떻게 나뉘느냐에 따라 결과가 달라짐
 - Data Set의 양이 적을 수록 학습이 제대로 이루어지지 않음

교차검정

Cross Validation

✦ K-Fold 방식

- Data Set을 K개로 나눈 뒤 하나를 Test Data Set으로 나머지를 Train Data Set으로 나누어 모델을 학습시키고 평가하는 방법
- K번 반복하여 모델을 학습시키고, 평가지표들의 평균을 내는 방식



- 방법: StratifiedKFold (), cross_val_score () 함수 이용

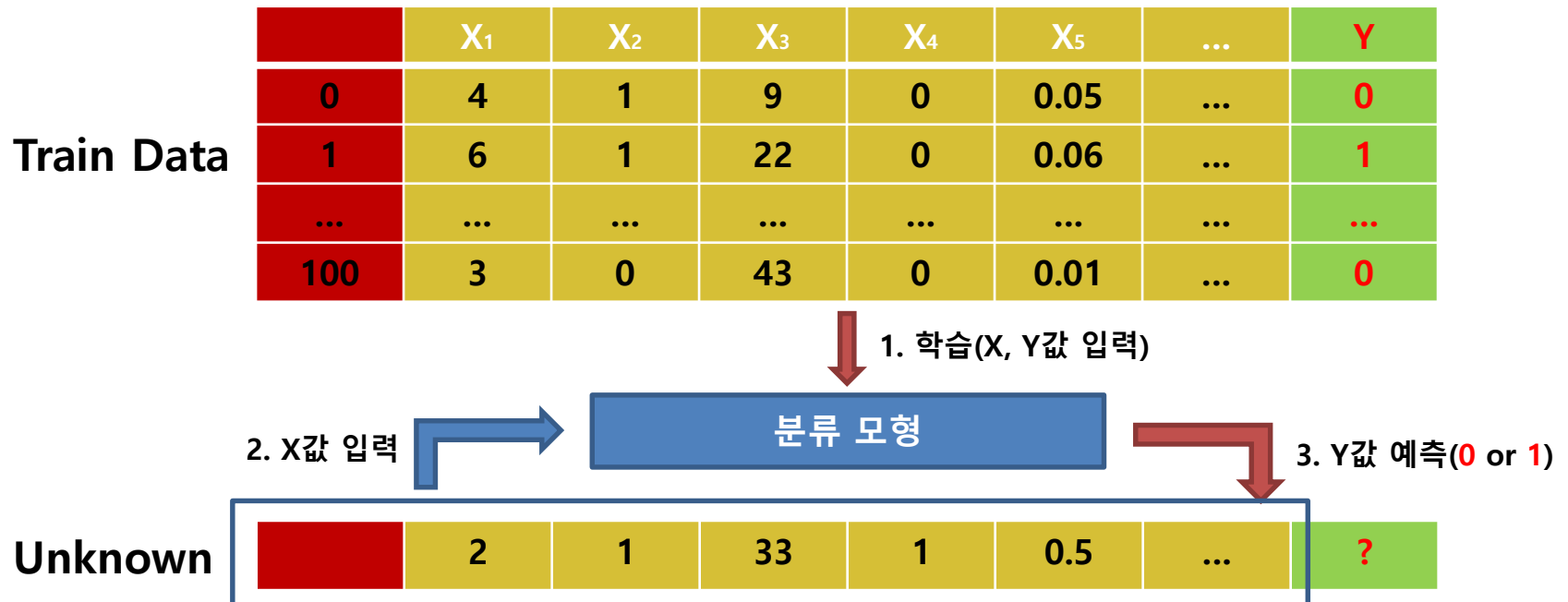
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
kfold = StratifiedKFold(n_splits= )
cross_val_score(estimator= , X, y, scoring='accuracy', cv=kfold)
```

모형평가 - 분류모형

Classification

✦ 분류(classification)

- 예측하려는 대상의 속성을 입력 받고, 목표 변수가 갖고 있는 카테고리 값 중에서 어느 한 값으로 분류하여 예측
 - Ex.고객분류, 질병진단, 스팸 메일 필터링, 음식 인식 등
 - 모형: KNN, SVM, Decision Tree 등



모형평가 - 분류모형

Classification

✦ 분류모형 예측력 평가 지표

		실제값	
		True	False
예측값	True	TP (True Positive)	FP (False Positive)
	False	FN (False Negative)	TN (True Negative)

- Confusion Matrix
 - 모형의 예측값과 실제값을 각각 축으로 하는 2 X 2 Matrix로 표현한 것
- Confusion Matrix 계산 및 평가지표 계산

```
from sklearn.metrics import confusion_matrix, classification_report
accuracy_score(y_test, y_hat)
confusion_matrix(y_test, y_hat)
classification_report(y_test, y_hat)
```

모형평가 - 분류모형

Classification

✦ 분류모형 예측력 평가 지표

– 정확도(Precision)

- True로 예측한 분석대상 중 실제값이 True인 비율
- 모형의 정확성을 나타내는 지표
- $\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$

– 재현율(Recall)

- 실제값이 True인 분석대상 중에서 True로 예측하여 모형이 적중한 비율
- 모형의 완전성을 나타내는 지표
- $\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$

– F1 지표(F1-score)

- 정확도와 재현율의 조화평균을 계산한 값
- 모형의 예측력을 종합적으로 평가하는 지표
- $\text{F1 score} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$

Regression

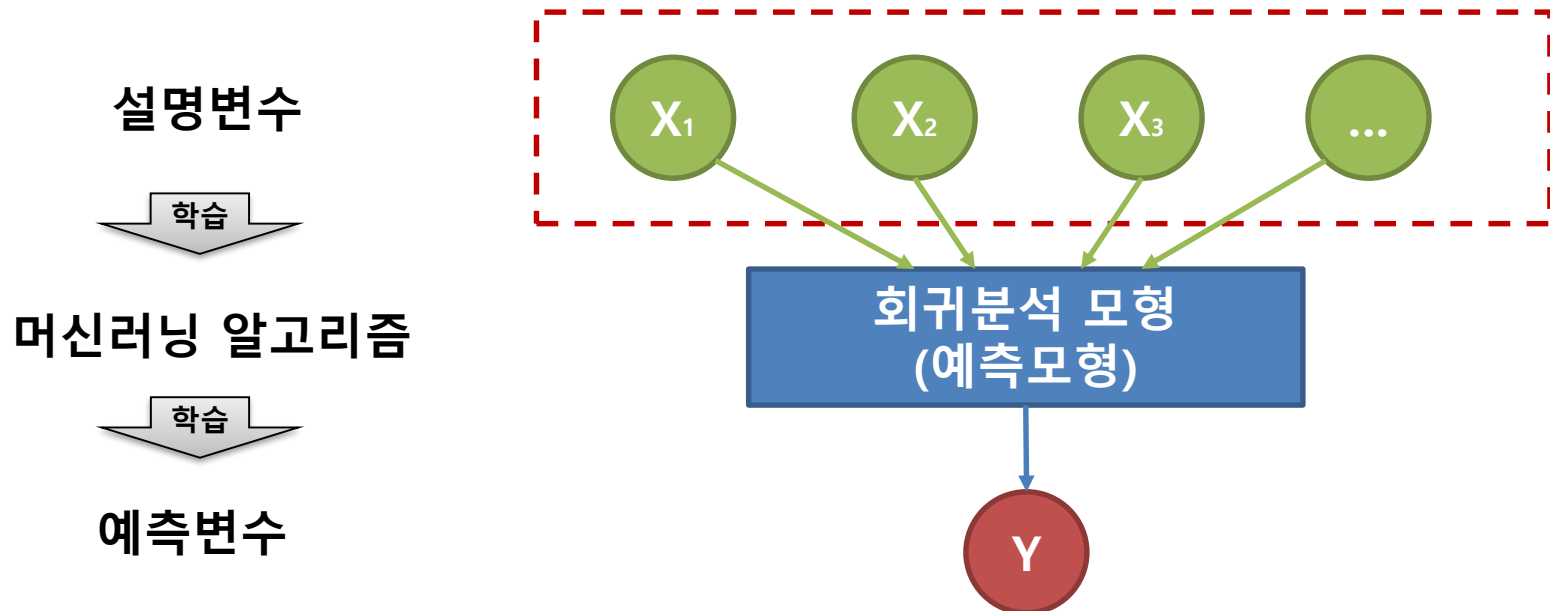


회귀분석

Overview

✦ 회귀분석(regression)

- 비교적 이해하기 쉽고, 널리 활용되고 있는 알고리즘
- 연속적인 값을 갖는 연속 변수를 예측하는데 주로 활용
 - 종속(예측)변수: 분석 모형이 예측하고자 하는 목표
 - 독립(설명)변수: 예측을 위해 모형이 사용하는 속성



선형회귀분석

Linear Regression

✦ 선형회귀분석

- 주어진 독립변수와 종속변수간의 관계를 파악
- 관계를 가장 잘 설명할 수 있는 회귀선을 찾고, 그 선을 통해 새로운 input 변수를 기반으로 output 변수를 예측하는 것
- 종류
 - 단순선형회귀분석: 독립변수가 한 개
 - 다중선형회귀분석: 독립변수가 두 개 이상

$$y = \beta_0 + \sum_{j=1}^p \beta_j x_j + \epsilon, \quad \epsilon \sim N(0, \sigma^2).$$

선형회귀분석

Linear Regression

✦ 회귀선을 찾는 방법

– 최소제곱법

- 편차 제곱의 합을 최소로 함으로써 모집단의 값을 추정하는 방법

$$\arg \min_{\beta_0, \beta} \sum_{i=1}^n (y_i - \beta_0 - \beta' \mathbf{x}_i)^2$$

– 회귀계수 검정

$$t = \frac{\hat{\beta}_j}{s.e(\beta_j)} \stackrel{H_0}{\sim} t(n-p)$$

– 적합도 평가: 결정계수 및 F 검정

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

$$F = \frac{SSR/p}{SSE/(n-p-1)} = \frac{MSR}{MSE} \sim F(p, n-p-1).$$

선형회귀분석

Linear Regression

✦ 가정

– 모형의 선형성

- 예측값과 잔차를 비교
- 모든 예측값에서 가운데 점선에 맞추어 잔차가 비슷하게 분포

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.regplot(fitted, residual, line_kws={'color':'red'})
plt.plot([y_train_pred.min(),y_train_pred.max()], [0,0], '--', color='grey')
```

– 잔차의 정규성

- 잔차가 정규분포를 따른다
- Q-Q plot으로 파악

```
from scipy.stats import zscore, probplot
import seaborn as sns
import matplotlib.pyplot as plt
sr = zscore(residual); (x, y), _ = probplot(sr)
sns.scatterplot(x,y)
plt.plot([-3,3], [-3,3], '--', color='grey')
```

선형회귀분석

Linear Regression

✦ 가정

- 잔차의 등분산성
 - 모든 값들에 대하여 잔차의 분산이 동일하다

```
import seaborn as sns
import numpy as np
sns.regplot(y_train_pred, np.sqrt(np.abs(sr)), line_kws={'color':'red'})
```

- 잔차의 독립성
 - 모형결과(summary())에서 Durbin-Waston값으로 확인

선형회귀분석

Linear Regression

✦ Correlation Analysis

- scipy.stats 모듈을 이용한 correlation analysis

```
Data.corr(method='pearson') #상관계수 출력
```

```
import seaborn as sns
```

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

```
plt.rcParams["figure.figsize"] = (8,5)
```

#상관계수 히트맵 표현

```
sns.heatmap(Data.corr(method='pearson'), annot = True, cmap = '색상', vmin = -1, vmax=1)
```

```
import scipy.stats as stats
```

```
corr1 = stats.pearsonr([데이터 변수명1, 데이터 변수명2]) #피어슨 상관계수 유의성 검정
```

```
corr2 = stats.spearmanr([데이터 변수명1, 데이터 변수명2]) #스피어만 상관계수 유의성 검정
```

선형회귀분석

Linear Regression

- ✦ Linear Regression
 - Sklearn 모듈을 이용한 Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression ( )
model.fit (X_train, Y_train);

model.predict (X_test)    #종속변수 예측값 출력
model.coef_               #회귀계수(기울기) 출력
model.intercept_          #회귀계수(절편) 출력
model.score (X_train, Y_train) #설명력(R2) 출력
```

선형회귀분석

Linear Regression

✦ Linear Regression

- statsmodels 모듈을 이용한 다중공선성(VIF) 확인

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif["variable"] = X_train.columns
print(vif)
```

선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 탐색

```
import pandas as pd
import seaborn as sns

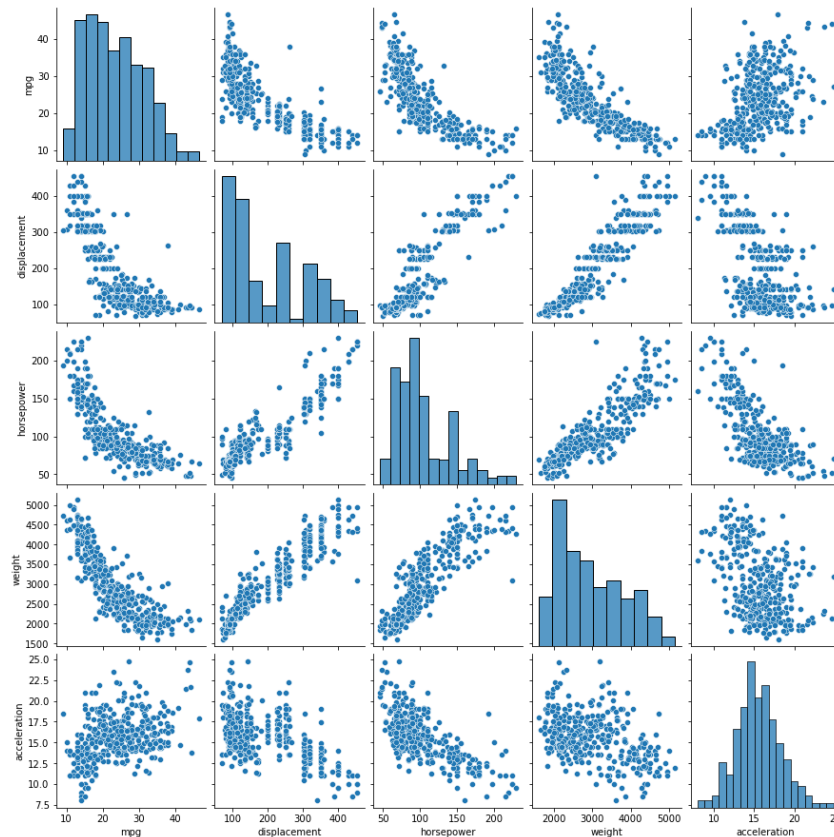
df = sns.load_dataset('mpg')
pd.set_option('display.max_columns', 15)
rdf = df.drop(['name', 'model_year'], axis=1)
rdf = rdf.dropna(subset=['horsepower'], how='any', axis=0)
ndf = rdf[['mpg', 'displacement', 'horsepower', 'weight', 'acceleration']]

import seaborn as sns
sns.pairplot(rdf[['mpg', 'displacement', 'horsepower', 'weight', 'acceleration']])
```


선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 탐색



선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 3: 상관관계 분석

```
ndf.corr(method='pearson')

import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (8,5)
sns.heatmap(ndf.corr(method='pearson'), annot = True, cmap = 'Greens', vmin = -1, vmax=1)

import scipy.stats as stats
corr1 = stats.pearsonr(ndf['mpg'], ndf['displacement'])
corr2 = stats.spearmanr(ndf['mpg'], ndf['displacement'])
print(corr1)
print(corr2)
```

선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 4~5: 속성선택 및 데이터셋 구분

```
X=ndf[['displacement', 'horsepower', 'weight', 'acceleration']]
y=ndf['mpg']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 6: 선형회귀형

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif["variable"] = X_train.columns
print(vif)

r_square = lr.score(X_train, y_train)
print('R^2: ', r_square)
print('y절편: ', lr.intercept_)
print('기울기: ', lr.coef_)

y_train_pred = lr.predict(X_train)
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
ax1 = sns.distplot(y_train, hist=False, label="y")
ax2 = sns.distplot(y_train_pred, hist=False, label="y_hat", ax=ax1)
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 1: 데이터 준비
 - 'bill_length_mm'와 'bill_depth_mm'에서 결측이 존재하는 경우 삭제
 - 종속변수: 'body_mass_g'
 - 독립변수: 'bill_length_mm', 'bill_depth_mm'

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('penguins')
pd.set_option('display.max_columns', 15)
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 1: 데이터 준비
 - 'bill_length_mm'와 'bill_depth_mm'에서 결측이 존재하는 경우 삭제
 - 종속변수: 'body_mass_g'
 - 독립변수: 'bill_length_mm', 'bill_depth_mm'

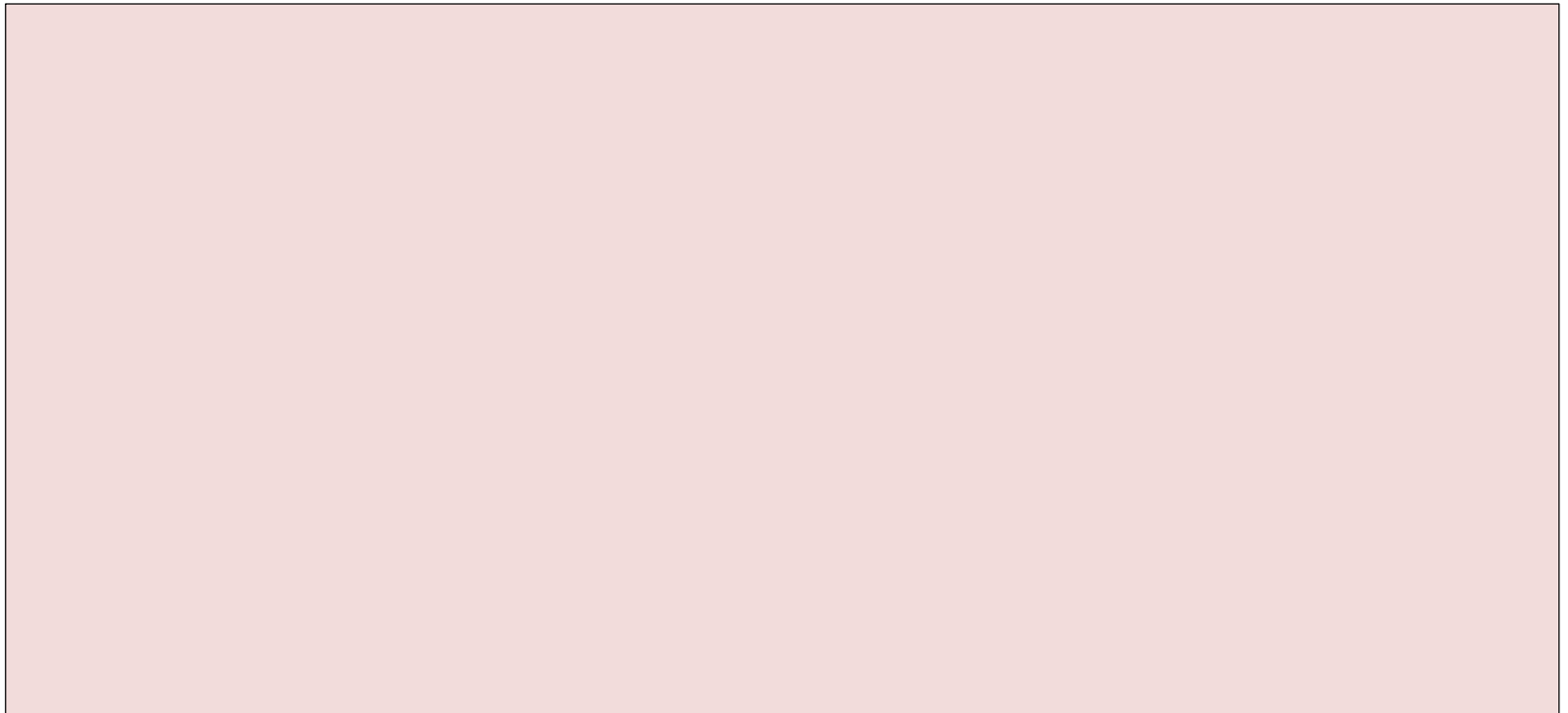
```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('penguins')
pd.set_option('display.max_columns', 15)
rdf = df.dropna(subset=['bill_length_mm', 'bill_depth_mm'], how='any', axis=0)
ndf = rdf[['bill_length_mm', 'bill_depth_mm', 'body_mass_g']]
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 2~3: 데이터 관계 및 상관관계 분석
 - 변수간 산점도 그리기
 - 변수간 상관관계 분석 실시



선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 2~3: 데이터 관계 및 상관관계 분석
 - 변수간 산점도 그리기
 - 변수간 상관관계 분석 실시

```
import seaborn as sns
sns.pairplot(rdf[['bill_length_mm', 'bill_depth_mm', 'body_mass_g']]);

ndf.corr(method='pearson')

import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (8,5)
sns.heatmap(ndf.corr(method='pearson'), annot = True, cmap = 'Greens', vmin = -1, vmax=1)

import scipy.stats as stats
corr1 = stats.pearsonr(ndf['body_mass_g'], ndf['bill_length_mm'])
corr2 = stats.spearmanr(ndf['body_mass_g'], ndf['bill_length_mm'])
print(corr1)
print(corr2)
```


선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 4~5: 속성선택 및 데이터셋 구분
 - 독립변수: 'bill_length_mm', 'bill_depth_mm'
 - 종속변수: 'body_mass_g'
 - Test set 자료의 크기: 전체의 30%

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 4~5: 속성선택 및 데이터셋 구분
 - 독립변수: 'bill_length_mm', 'bill_depth_mm'
 - 종속변수: 'body_mass_g'
 - Test set 자료의 크기: 전체의 30%

```
X=ndf[['bill_length_mm', 'bill_depth_mm']]
y=ndf['body_mass_g']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 6: 선형회귀형
 - Linear Regression 모형 적합
 - 독립변수간 다중공선성 확인
 - 설명력 확인
 - 회귀 계수 확인

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 6: 선형회귀형

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif["variable"] = X_train.columns
print(vif)

r_square = lr.score(X_train, y_train)
print('R^2: ', r_square)
print('y절편: ', lr.intercept_)
print('기울기: ', lr.coef_)

y_train_pred = lr.predict(X_train)
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
ax1 = sns.distplot(y_train, hist=False, label="y")
ax2 = sns.distplot(y_train_pred, hist=False, label="y_hat", ax=ax1)
```

선형회귀분석

Linear Regression

✦ Linear Regression

- statsmodels 모듈을 이용한 Linear Regression

```
import statsmodels.formula.api as smf

model = smf.ols(formula = formula, data = train)
result = model.fit()
result.summary()           #결과 출력
result.predict(X_train)    #종속변수 예측값 출력
```

```
import statsmodels.api as sm

model = sm.OLS(y_train, X_train)
result = model.fit()
result.summary()           #결과 출력
result.predict(X_train)    #종속변수 예측값 출력
```

선형회귀분석

Linear Regression

✦ Linear Regression

- 잔차분석
 - 선형성

```
import matplotlib.pyplot as plt
import seaborn as sns
y_train_pred = result.predict(X_train)
residual = y_train - y_train_pred

sns.regplot(y_train_pred, residual, line_kws={'color':'red'})
plt.plot([y_train_pred.min(), y_train_pred.max()], [0,0], '--', color='grey')
```

- 등분산성

```
import seaborn as sns
import numpy as np
sns.regplot(y_train_pred, np.sqrt(np.abs(sr)), line_kws={'color':'red'})
```

선형회귀분석

Linear Regression

✦ Linear Regression

– 잔차분석

- 정규성

```
from scipy.stats import zscore, probplot
sr = zscore(residual)
(x, y), _ = probplot(sr)
import seaborn as sns
sns.scatterplot(x,y)
import matplotlib.pyplot as plt
plt.plot([-3,3], [-3,3], '--', color='grey')

from scipy.stats import shapiro
shapiro(residual)
```

- 극단값

```
from statsmodels.stats.outliers_influence import OLSInfluence
cd, _ = OLSInfluence(result).cooks_distance
pd.DataFrame(cd,columns=['cook']).sort_values(by='cook',ascending=False).head()
```

선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - 선형회귀모형(1) - OLS이용

```
X=ndf[['displacement', 'horsepower', 'weight', 'acceleration']]
y=ndf['mpg']
import statsmodels.api as sm
X_ad = sm.add_constant(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_ad, y, test_size=0.3, random_state=1)

import statsmodels.api as sm
model = sm.OLS(y_train, X_train)
result = model.fit()
result.summary()
y_train_pred = result.predict(X_train)
```


선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - 선형회귀모형(2) - ols이용

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('mpg')
pd.set_option('display.max_columns', 15)
rdf = df.drop(['name', 'model_year'], axis=1)
rdf = rdf.dropna(subset=['horsepower'], how='any', axis=0)
ndf = rdf[['mpg', 'displacement', 'horsepower', 'weight', 'acceleration']]

X=ndf[['displacement', 'horsepower', 'weight', 'acceleration']]
y=ndf['mpg']
mpg = pd.concat([X,y],axis=1)
```

선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - 선형회귀모형(2) - ols이용

```
import statsmodels.formula.api as smf

formula = 'mpg ~ displacement + horsepower + weight + acceleration'
model = smf.ols(formula = formula, data = train)
result = model.fit()
result.summary()
```

선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - 선형회귀모형 - 잔차분석(1)

```
#선형성
import matplotlib.pyplot as plt
import seaborn as sns
y_train_pred = result.predict(X_train)
residual = y_train - y_train_pred

sns.regplot(y_train_pred, residual, line_kws={'color':'red'})
plt.plot([y_train_pred.min(),y_train_pred.max()], [0,0], '--', color='grey')

#정규성
from scipy.stats import zscore, probplot
import seaborn as sns
import matplotlib.pyplot as plt
sr = zscore(residual)
(x, y), _ = probplot(sr)
sns.scatterplot(x,y)
plt.plot([-3,3], [-3,3], '--', color='grey')

from scipy.stats import shapiro
shapiro(residual)
```

선형회귀분석

Linear Regression

- mpg 데이터셋을 이용한 선형회귀분석 알고리즘
 - 선형회귀모형 - 잔차분석(2)

```
#등분산성
import seaborn as sns
import numpy as np
sns.regplot(y_train_pred, np.sqrt(np.abs(sr)), line_kws={'color':'red'})

#극단값
from statsmodels.stats.outliers_influence import OLSInfluence
cd, _ = OLSInfluence(result).cooks_distance
pd.DataFrame(cd,columns=['cook']).sort_values(by='cook',ascending=False).head()
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 1: 데이터 준비
 - 'bill_length_mm'와 'bill_depth_mm'에서 결측이 존재하는 경우 삭제
 - 종속변수: 'body_mass_g'
 - 독립변수: 'bill_length_mm', 'bill_depth_mm'
 - Test set 자료의 크기: 전체의 30%

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 1: 데이터 준비

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('penguins')
pd.set_option('display.max_columns', 15)
rdf = df.dropna(subset=['bill_length_mm', 'bill_depth_mm'], how='any', axis=0)
ndf = rdf[['bill_length_mm', 'bill_depth_mm', 'body_mass_g']]

X=ndf[['bill_length_mm', 'bill_depth_mm']]
y=ndf['body_mass_g']

import statsmodels.api as sm
X_ad = sm.add_constant(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_ad, y, test_size=0.3, random_state=1)
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 2: 선형회귀모형(1) - OLS이용
 - Linear Regression 모형 적합
 - 독립변수간 다중공선성 확인
 - 설명력 확인
 - 회귀 계수 확인
 - 회귀 계수 유의성 확인

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 2: 선형회귀모형(1) - OLS이용

```
import statsmodels.api as sm
model = sm.OLS(y_train, X_train)
result = model.fit()
result.summary()
y_train_pred = result.predict(X_train)
```


선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 2: 선형회귀모형(2) - ols이용
 - Linear Regression 모형 적합
 - 독립변수간 다중공선성 확인
 - 설명력 확인
 - 회귀 계수 확인
 - 회귀 계수 유의성 확인

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 2: 선형회귀모형(2) - ols이용

```
X=ndf[['bill_length_mm', 'bill_depth_mm']]
y=ndf['body_mass_g']
penguins = pd.concat([X,y],axis=1)

from sklearn.model_selection import train_test_split
train, test = train_test_split(penguins, test_size=0.3, random_state=1)

import statsmodels.api as sm
import statsmodels.formula.api as smf

formula = 'body_mass_g ~ bill_length_mm + bill_depth_mm'
model = smf.ols(formula = formula, data = train)
result = model.fit()
result.summary()
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 3: 선형회귀모형 – 잔차분석
 - 선형성
 - 정규성
 - 등분산성
 - 극단값
 - 독립성

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 3: 선형회귀모형 – 잔차분석(1)

```
#선형성
import matplotlib.pyplot as plt
import seaborn as sns
y_train_pred = result.predict(X_train)
residual = y_train - y_train_pred

sns.regplot(y_train_pred, residual, line_kws={'color':'red'})
plt.plot([y_train_pred.min(),y_train_pred.max()], [0,0], '--', color='grey')

#정규성
from scipy.stats import zscore, probplot
import seaborn as sns
import matplotlib.pyplot as plt
sr = zscore(residual)
(x, y), _ = probplot(sr)
sns.scatterplot(x,y)
plt.plot([-3,3], [-3,3], '--', color='grey')

from scipy.stats import shapiro
shapiro(residual)
```

선형회귀분석

Linear Regression

- penguins 데이터셋을 이용한 선형회귀분석 알고리즘
 - Step 3: 선형회귀모형 – 잔차분석(2)

```
#등분산성
import seaborn as sns
import numpy as np
sns.regplot(y_train_pred, np.sqrt(np.abs(sr)), line_kws={'color':'red'})

#극단값
from statsmodels.stats.outliers_influence import OLSInfluence
cd, _ = OLSInfluence(result).cooks_distance
pd.DataFrame(cd,columns=['cook']).sort_values(by='cook',ascending=False).head()
```

로지스틱회귀분석

Logistic Regression

✦ 로지스틱회귀분석

- 종속변수가 0 또는 1의 값(이분형 자료)을 가지는 경우
- 종속변수가 0 또는 1의 값을 가지는 경우 선형함수(일반적으로 비선형)로 예측할 수 없음

$$\text{logit}(P(y = 1|\mathbf{x})) = \log\left(\frac{P(y = 1|\mathbf{x})}{1 - P(y = 1|\mathbf{x})}\right) = f(x_1, \dots, x_p) = \beta_0 + \sum_{j=1}^p \beta_j x_j.$$

$$P(y = 1|\mathbf{x}) = \frac{\exp(\beta_0 + \sum_{j=1}^p \beta_j x_j)}{1 + \exp(\beta_0 + \sum_{j=1}^p \beta_j x_j)}.$$

- 모형의 해석: 오즈비 이용
 - 독립변수가 1단위 증가함에 따라 오즈는 $\exp(\beta)$ 배 만큼 곱해져서 증가

$$\frac{P(y = 1|x)}{1 - P(y = 1|x)} = e^{\beta_0} (e^{\beta_1})^x$$

로지스틱회귀분석

Logistic Regression

✦ 회귀선을 찾는 방법

– 최대우도추정법

- 우도함수를 최대로 하는 모수값을 추정하는 방법

$$l(\beta) = \sum_{i=1}^n \{y_i \log p_i + (1 - y_i) \log(1 - p_i)\}.$$

- Newton-Raphson 방법 이용

$$\beta^{new} = \beta^{old} - H^{-1} \nabla \beta.$$

로지스틱회귀분석

Logistic Regression

✦ Logistic Regression

- Sklearn 모듈을 이용한 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression( )
lr.fit(X_train, y_train)

lr.intercept_
lr.coef_           #회귀계수 출력

import numpy as np
np.exp(lr.coef_)   #오즈비 계산

lr.score(X_train, y_train)  #정분류를 출력

lr.predict_proba(X_train)  #종속변수 예측 확률값 출력
lr.predict(X_train)        #종속변수 예측값 출력
```


로지스틱회귀분석

Logistic Regression

✦ Logistic Regression

- Sklearn 모듈을 이용한 ROC curve

```
from sklearn.metrics import roc_curve, auc
y_train_pred = lr_result.predict(X_train)
fpr, tpr, threshold = roc_curve(y_train, y_train_pred) #ROC 곡선 구성요소 계산
roc_auc = auc(fpr, tpr) #AUC 계산

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc) #ROC 곡선 구성요소 이용 그림 그리기
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
print(titanic.head())

titanic=titanic[['age','fare', 'sex', 'survived']]
titanic = titanic.dropna(subset=['age'], how='any', axis=0)
y=titanic['survived']
X=titanic[['age','fare', 'sex']]

from sklearn.preprocessing import LabelEncoder
classle=LabelEncoder()
X['sex']=classle.fit_transform(X['sex'].values)
#X['sex']=pd.get_dummies(X['sex'],drop_first=True)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty='none')
lr.fit(X_train, y_train)

print('회귀상수: %n', lr.intercept_)
print('회귀계수: %n', lr.coef_)

import numpy as np
np.exp(lr.coef_)

lr.score(X_train, y_train)
lr.score(X_test, y_test)

lr.predict_proba(X_train)
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_pred, y_train)

from sklearn import metrics
lr_report = metrics.classification_report(y_test, y_test_pred)
print(lr_report)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 6: ROC 곡선 그리기

```
from sklearn.metrics import roc_curve, auc
y_train_pred = lr.predict_proba(X_train)[:,-1]
fpr, tpr, threshold = roc_curve(y_train, y_train_pred)
roc_auc = auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc)
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환
 - breast_cancer 자료 불러오기
 - 0~2번째 변수까지 불러오기 => 독립변수 지정
 - cancer.target 자료를 'target' 변수로 만들어 불러오기 => 종속변수 지정

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])
df = df.iloc[ :,:3]
df['target'] = cancer.target
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
cancer = load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])
df = df.iloc[:, :3]
df['target'] = cancer.target

X=df.iloc[:, :3]
y=df['target']
```


로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분
 - 종속변수와 독립변수 자료를 test set과 train set으로 구분
 - train set은 전체의 70%로 구성

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형
 - LogisticRegression 명령어를 사용하여 로지스틱 회귀모형 적합
 - 독립변수들의 유의성 검정 실시
 - 독립변수들의 오즈비 계산
 - 종속변수의 예측 확률 및 예측값 계산

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty='none')
lr.fit(X_train, y_train)

print('회귀상수: \n', lr.intercept_)
print('회귀계수: \n', lr.coef_)
import numpy as np
np.exp(lr.coef_)

lr.score(X_train, y_train)
lr.score(X_test, y_test)

lr.predict_proba(X_train)
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산
 - train set과 test set의 분류 정확도 계산
 - confusion metrix 계산
 - Step 6: ROC 곡선 그리기

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_pred, y_train)

from sklearn import metrics
lr_report = metrics.classification_report(y_test, y_test_pred)
print(lr_report)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 6: ROC 곡선 그리기

```
from sklearn.metrics import roc_curve, auc
y_train_pred = lr.predict_proba(X_train)[:,-1]
fpr, tpr, threshold = roc_curve(y_train, y_train_pred)
roc_auc = auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc)
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

로지스틱회귀분석

Logistic Regression

✦ Logistic Regression

- statsmodels 모듈을 이용한 Logistic Regression

```
import statsmodels.api as sm
lr = sm.Logit(y_train, X_train)
lr_result = lr.fit()

print(lr_result.summary())    #로지스틱 회귀분석 결과 출력
np.exp(lr_result.params)     #오즈비 계산

y_train_pred = lr_result.predict(X_train)    #종속변수 예측 확률값 출력
y_train_pred = np.around(y_train_pred)      #종속변수 예측값 출력
```


로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
print(titanic.head())

titanic=titanic[['age','fare', 'sex', 'survived']]
titanic = titanic.dropna(subset=['age'], how='any', axis=0)
y=titanic['survived']
X=titanic[['age','fare', 'sex']]

from sklearn.preprocessing import LabelEncoder
import numpy as np
classle=LabelEncoder()
X['sex']=classle.fit_transform(X['sex'].values)

import statsmodels.api as sm
X = sm.add_constant(X)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형

```
import statsmodels.api as sm
lr = sm.Logit(y_train, X_train)
lr_result = lr.fit()
print(lr_result.summary())
np.exp(lr_result.params)

y_test_pred = lr_result.predict(X_test)
y_train_pred = lr_result.predict(X_train)

y_test_pred = np.around(y_test_pred)
y_train_pred = np.around(y_train_pred)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_pred, y_train)

from sklearn import metrics
lr_report = metrics.classification_report(y_test, y_test_pred)
print(lr_report)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 6: ROC 곡선 그리기

```
from sklearn.metrics import roc_curve, auc
y_train_pred = lr_result.predict(X_train)
fpr, tpr, threshold = roc_curve(y_train, y_train_pred)
roc_auc = auc(fpr, tpr)
```

```
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc)
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환
 - breast_cancer 자료 불러오기
 - 0~2번째 변수까지 불러오기 => 독립변수 지정
 - cancer.target 자료를 'target' 변수로 만들어 불러오기 => 종속변수 지정
 - 독립변수에 상수항 추가

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])
df = df.iloc[ :,3]
df['target'] = cancer.target
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])
df = df.iloc[:,3]
df['target'] = cancer.target

X=df.iloc[:,3]
y=df['target']

import statsmodels.api as sm
X = sm.add_constant(X)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분
 - 종속변수와 독립변수 자료를 test set과 train set으로 구분
 - train set은 전체의 70%로 구성

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형
 - Logit 명령어를 사용하여 로지스틱 회귀모형 적합
 - 독립변수들의 유의성 검정 실시
 - 독립변수들의 오즈비 계산
 - 종속변수의 예측 확률 및 예측값 계산

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형

```
import statsmodels.api as sm
lr = sm.Logit(y_train, X_train)
lr_result = lr.fit()
print(lr_result.summary())
np.exp(lr_result.params)

y_test_pred = lr_result.predict(X_test)
y_train_pred = lr_result.predict(X_train)

y_test_pred = np.around(y_test_pred)
y_train_pred = np.around(y_train_pred)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산
 - train set과 test set의 분류 정확도 계산
 - confusion metrix 계산
 - Step 6: ROC 곡선 그리기

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_pred, y_train)

from sklearn import metrics
lr_report = metrics.classification_report(y_test, y_test_pred)
print(lr_report)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 6: ROC 곡선 그리기

```
from sklearn.metrics import roc_curve, auc
y_train_pred = lr_result.predict(X_train)
fpr, tpr, threshold = roc_curve(y_train, y_train_pred)
roc_auc = auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc)
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

로지스틱회귀분석

Logistic Regression

✦ Logistic Regression

- statsmodels 모듈을 이용한 Logistic Regression

```
import statsmodels.formula.api as smf
lr = smf.logit(formula= , data= )
lr_result = lr.fit()

print(lr_result.summary())    #로지스틱 회귀분석 결과 출력
np.exp(lr_result.params)     #오즈비 계산

y_train_pred = lr_result.predict(train)    #종속변수 예측 확률값 출력
y_train_pred = np.around(y_train_pred)    #종속변수 예측값 출력
```

로지스틱회귀분석

Logistic Regression

✦ Logistic Regression

- statsmodels 모듈을 이용한 Logistic Regression

```
import statsmodels.formula.api as smf

lr = smf.glm(formula= , data= , family = sm.families.Binomial())
lr_result = lr.fit()

print(lr_result.summary())    #로지스틱 회귀분석 결과 출력
np.exp(lr_result.params)     #오즈비 계산

y_train_pred = lr_result.predict(train)    #종속변수 예측 확률값 출력
y_train_pred = np.around(y_train_pred)    #종속변수 예측값 출력
```


로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
print(titanic.head())

titanic=titanic[['age','fare', 'sex', 'survived']]
titanic = titanic.dropna(subset=['age'], how='any', axis=0)
y=titanic['survived']
X=titanic[['age','fare', 'sex']]

from sklearn.preprocessing import LabelEncoder
import numpy as np
classle=LabelEncoder()
X['sex']=classle.fit_transform(X['sex'].values)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분

```
import pandas as pd
data_set = pd.concat([X,y],axis=1)
from sklearn.model_selection import train_test_split
train, test = train_test_split(data_set, test_size=0.3, random_state=1,stratify=y)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형

```
import statsmodels.formula.api as smf
lr = smf.logit(formula='survived ~ age + fare + sex',data=train)
lr_result = lr.fit()
print(lr_result.summary())
np.exp(lr_result.params)

y_test_pred = lr_result.predict(test)
y_train_pred = lr_result.predict(train)

y_test_pred = np.around(y_test_pred)
y_train_pred = np.around(y_train_pred)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_pred, y_train)

from sklearn import metrics
lr_report = metrics.classification_report(y_test, y_test_pred)
print(lr_report)
```

로지스틱회귀분석

Logistic Regression

- titanic 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 6: ROC 곡선 그리기

```
from sklearn.metrics import roc_curve, auc
y_train_pred = lr_result.predict(X_train)
fpr, tpr, threshold = roc_curve(y_train, y_train_pred)
roc_auc = auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc)
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환
 - breast_cancer 자료 불러오기
 - 0~2번째 변수까지 불러오기 => 독립변수 지정
 - cancer.target 자료를 'target' 변수로 만들어 불러오기 => 종속변수 지정
 - 독립변수에 상수항 추가

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])
df = df.iloc[ :,3]
df['target'] = cancer.target
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 1~2: 데이터 준비 및 더미변수 변환

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])
df = df.iloc[ :,3]
df['target'] = cancer.target

X=df.iloc[:,3]
y=df['target']
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분
 - 변수 이름을 각각 'radius', 'texture', 'perimeter', 'target ' 로 수정
 - 종속변수와 독립변수 자료를 test set과 train set으로 구분
 - train set은 전체의 70%로 구성

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 3: 데이터셋 구분

```
import pandas as pd
data_set = pd.concat([X,y],axis=1)
data_set.columns = ['radius', 'texture', 'perimeter', 'target']
from sklearn.model_selection import train_test_split
train, test = train_test_split(data_set, test_size=0.3, random_state=1,stratify=y)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형
 - logit 명령어를 사용하여 로지스틱 회귀모형 적합
 - 독립변수들의 유의성 검정 실시
 - 독립변수들의 오즈비 계산
 - 종속변수의 예측 확률 및 예측값 계산

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 4: 로지스틱회귀모형

```
import statsmodels.formula.api as smf
lr = smf.logit(formula='target ~ radius + texture + perimeter',data=train)
lr_result = lr.fit()
print(lr_result.summary())
np.exp(lr_result.params)

y_test_pred = lr_result.predict(test)
y_train_pred = lr_result.predict(train)

y_test_pred = np.around(y_test_pred)
y_train_pred = np.around(y_train_pred)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산
 - train set과 test set의 분류 정확도 계산
 - confusion metrix 계산
 - Step 6: ROC 곡선 그리기

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 5: 분류정확도 계산

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_pred, y_train)

from sklearn import metrics
lr_report = metrics.classification_report(y_test, y_test_pred)
print(lr_report)
```

로지스틱회귀분석

Logistic Regression

- breast_cancer 데이터셋을 이용한 로지스틱회귀분석 알고리즘
 - Step 6: ROC 곡선 그리기

```
from sklearn.metrics import roc_curve, auc
y_train_pred = lr_result.predict(X_train)
fpr, tpr, threshold = roc_curve(y_train, y_train_pred)
roc_auc = auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'b', label='AUC = %.2f' %roc_auc)
plt.plot([0,1], [0,1], '--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.legend(loc='best')
```

Classification



분류

KNN

✦ KNN(K-Near-Neighbors)

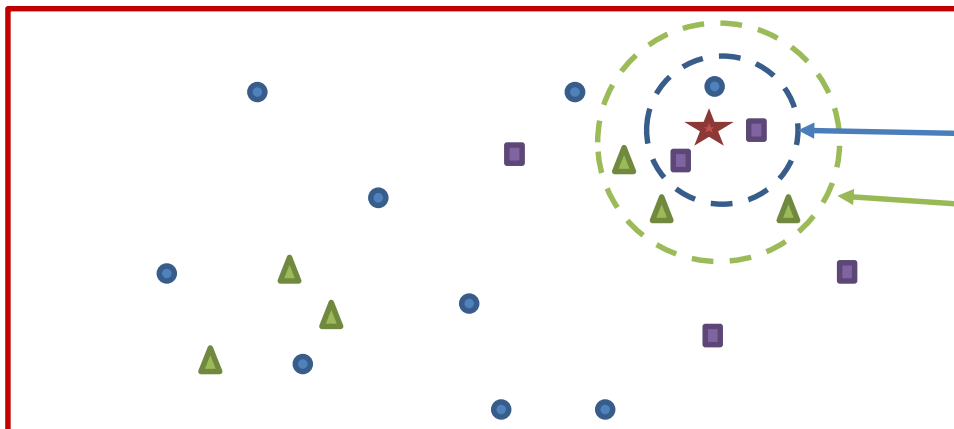
– 알고리즘

- Step1: 새로운 관측 값이 주어지면 기존 데이터 중에서 가장 속성이 비슷한 K개의 이웃을 찾음
- Step2: 가까운 이웃들이 갖고 있는 목표 값과 같은 값으로 분류하여 예측

$$d(x_i, x_j) = \sqrt[p]{\sum_{k=1}^d |x_{ik} - x_{jk}|^p}$$

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=k, p=2)
knn.fit(X_train, Y_train); knn.predict(X_test)
```

- K값에 따라 예측의 정확도가 달라지므로 적절한 K값을 찾는 것이 매우 중요



K	이웃	분류값
3		
6		

분류

KNN

- iris 데이터셋을 이용한 KNN분류 알고리즘
 - Step 1~2: 데이터 준비 및 탐색

```
import seaborn as sns
iris=sns.load_dataset('iris')
print(iris.head())

sns.set()
sns.pairplot(iris,hue='species',size=2.5)

X=iris.drop('species', axis=1)
X.shape
y=iris['species']

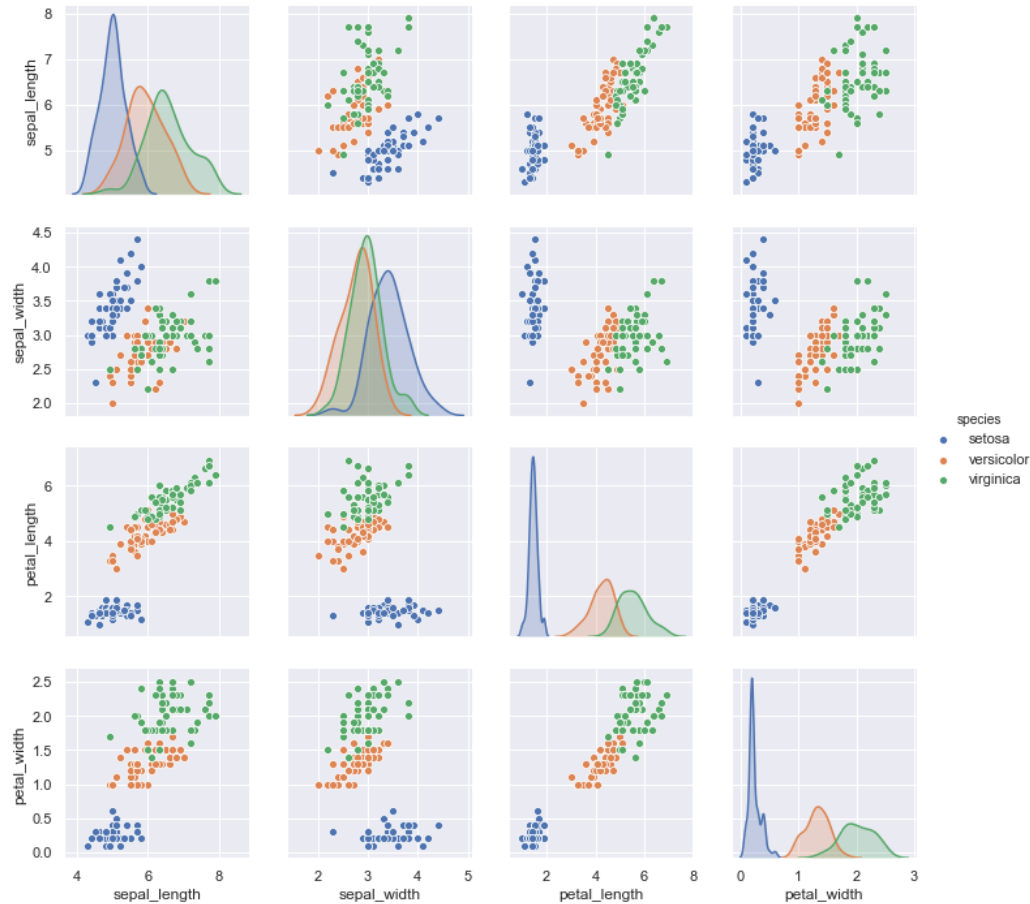
from sklearn.preprocessing import LabelEncoder
import numpy as np
classle=LabelEncoder()
y=classle.fit_transform(iris['species'].values)

y0=classle.inverse_transform(y)
```

KNN

– iris 데이터셋을 이용한 KNN분류 알고리즘

- Step 1~2: 데이터 준비 및 탐색



분류

KNN

- iris 데이터셋을 이용한 KNN분류 알고리즘
 - Step 3~4: 속성선택 및 데이터셋 구분

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

분류

KNN

- iris 데이터셋을 이용한 KNN분류 알고리즘
 - Step 5: KNN 분류모형 (표준화 전)

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5, p=2)
knn.fit(X_train, y_train)

y_train_pred=knn.predict(X_train)
y_test_pred=knn.predict(X_test)
print('Misclassified training samples: %d' %(y_train != y_train_pred).sum())
print('Misclassified test samples: %d' %(y_test != y_test_pred).sum())

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)

knn_report = metrics.classification_report(y_test, y_test_pred)
print(knn_report)
```

분류

KNN

- iris 데이터셋을 이용한 KNN분류 알고리즘
 - Step 5: KNN 분류모형 (표준화 후)

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(X_train)
X_train_std=sc.transform(X_train)
X_test_std=sc.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5, p=2)
knn.fit(X_train_std, y_train)
y_train_pred=knn.predict(X_train_std)
y_test_pred=knn.predict(X_test_std)
print('Misclassified training samples: %d' %(y_train != y_train_pred).sum())
print('Misclassified test samples: %d' %(y_test != y_test_pred).sum())

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)
knn_report = metrics.classification_report(y_test, y_test_pred)
print(knn_report)
```

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 1: 데이터 준비(seaborn의 titanic 자료 불러오기)

```
import pandas as pd  
import seaborn as sns
```

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 1: 데이터 준비

```
import pandas as pd
import seaborn as sns
```

```
df = sns.load_dataset('titanic')
```

```
print(df.head())
```

```
pd.set_option('display.max_columns', 15)
print(df.head())
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class \
0	0	3	male	22.0	1	0	7.2500	S	Third
1	1	1	female	38.0	1	0	71.2833	C	First
2	1	3	female	26.0	0	0	7.9250	S	Third
3	1	1	female	35.0	1	0	53.1000	S	First
4	0	3	male	35.0	0	0	8.0500	S	Third

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 2: 데이터 탐색
 - 결측자료가 많은 deck, embark_town 변수 삭제
 - 'age'에서 결측자료 자료 삭제
 - 'embarked'자료 중 결측자료를 가장 빈도가 많은 자료로 대체

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘

- Step 2: 데이터 탐색

```
print(df.info())
```

```
rdf = df.drop(['deck', 'embark_town'], axis=1)
print(rdf.columns.values)
```

```
rdf = rdf.dropna(subset=['age'], how='any', axis=0)
print(len(rdf))
```

```
most_freq = rdf['embarked'].value_counts(dropna=True).idxmax()
print(most_freq)
```

```
print(rdf.describe(include='all'))
```

```
rdf['embarked'].fillna(most_freq, inplace=True)
```

	survived	pclass	sex	age	sibsp	parch
count	714.000000	714.000000	714	714.000000	714.000000	714.000000
unique	NaN	NaN	2	NaN	NaN	NaN
top	NaN	NaN	male	NaN	NaN	NaN
freq	NaN	NaN	453	NaN	NaN	NaN
mean	0.406162	2.236695	NaN	29.699118	0.512605	0.431373
std	0.491460	0.838250	NaN	14.526497	0.929783	0.853289
min	0.000000	1.000000	NaN	0.420000	0.000000	0.000000
25%	0.000000	1.000000	NaN	20.125000	0.000000	0.000000
50%	0.000000	2.000000	NaN	28.000000	0.000000	0.000000
75%	1.000000	3.000000	NaN	38.000000	1.000000	1.000000
max	1.000000	3.000000	NaN	80.000000	5.000000	6.000000

	fare	embarked	class	who	adult_male	alive	alone
count	714.000000	712	714	714	714	714	714
unique	NaN	3	3	3	2	2	2
top	NaN	S	Third	man	True	no	True
freq	NaN	554	355	413	413	424	404
mean	34.694514	NaN	NaN	NaN	NaN	NaN	NaN
std	52.918930	NaN	NaN	NaN	NaN	NaN	NaN
min	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
25%	8.050000	NaN	NaN	NaN	NaN	NaN	NaN
50%	15.741700	NaN	NaN	NaN	NaN	NaN	NaN
75%	33.375000	NaN	NaN	NaN	NaN	NaN	NaN
max	512.329200	NaN	NaN	NaN	NaN	NaN	NaN

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 3: 속성선택
 - 'survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked' 자료만 분석에 사용
 - 'sex', 'embarked' 변수는 더미변수로 변경 후 기존 변수 삭제

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 3: 속성선택

```
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked']]
print(ndf.head())
```

```
onehot_sex = pd.get_dummies(ndf['sex'])
ndf = pd.concat([ndf, onehot_sex], axis=1)
```

```
onehot_embarked = pd.get_dummies(ndf['embarked'], prefix='town')
ndf = pd.concat([ndf, onehot_embarked], axis=1)
```

```
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)
print(ndf.head())
```

	survived	pclass	sex	age	sibsp	parch	embarked
0	0	3	male	22.0	1	0	S
1	1	1	female	38.0	1	0	C
2	1	3	female	26.0	0	0	S
3	1	1	female	35.0	1	0	S
4	0	3	male	35.0	0	0	S

	survived	pclass	age	sibsp	parch	female	male	town_C	town_Q	town_S
0	0	3	22.0	1	0	0	1	0	0	1
1	1	1	38.0	1	0	1	0	1	0	0
2	1	3	26.0	0	0	1	0	0	0	1
3	1	1	35.0	1	0	1	0	0	0	1
4	0	3	35.0	0	0	0	1	0	0	1

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 4: 데이터셋 구분
 - 독립변수: 'pclass', 'age', 'sibsp', 'parch', 'female', 'male'
 - 종속변수: 'survived'
 - 독립변수 표준화 후 train과 test 데이터로 구분
 - » Random number는 1, train과 test는 7:3의 비율로 분류

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 4: 데이터셋 구분

```
X=ndf[['pclass', 'age', 'sibsp', 'parch', 'female', 'male']]
y=ndf['survived']

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1, stratify=y)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
```

```
train data 개수: (499, 9)
test data 개수: (215, 9)
```

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 5: KNN 분류모형
 - k를 5, p를 2로 하여 KNN분석 수행
 - Confusion matrix 작성 후 분류 정확도 계산
 - » 전체 오분류률, precision, recall, F1-score 계산

분류

KNN

- Titanic 데이터셋을 이용한 KNN분류 알고리즘
 - Step 5: KNN 분류모형

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=5)
```

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

```
y_hat = knn.predict(X_test)
```

```
print(y_hat[0:10])
```

```
print(y_test.values[0:10])
```

```
[0 0 1 0 0 1 1 1 0 0]
[0 0 1 0 0 1 1 1 0 0]
```

```
from sklearn import metrics
```

```
knn_matrix = metrics.confusion_matrix(y_test, y_hat)
```

```
print(knn_matrix)
```

```
knn_report = metrics.classification_report(y_test, y_hat)
```

```
print(knn_report)
```

```
[[109 16]
 [ 25 65]]
```

	precision	recall	f1-score	support
0	0.81	0.87	0.84	125
1	0.80	0.72	0.76	90
micro avg	0.81	0.81	0.81	215
macro avg	0.81	0.80	0.80	215
weighted avg	0.81	0.81	0.81	215

분류

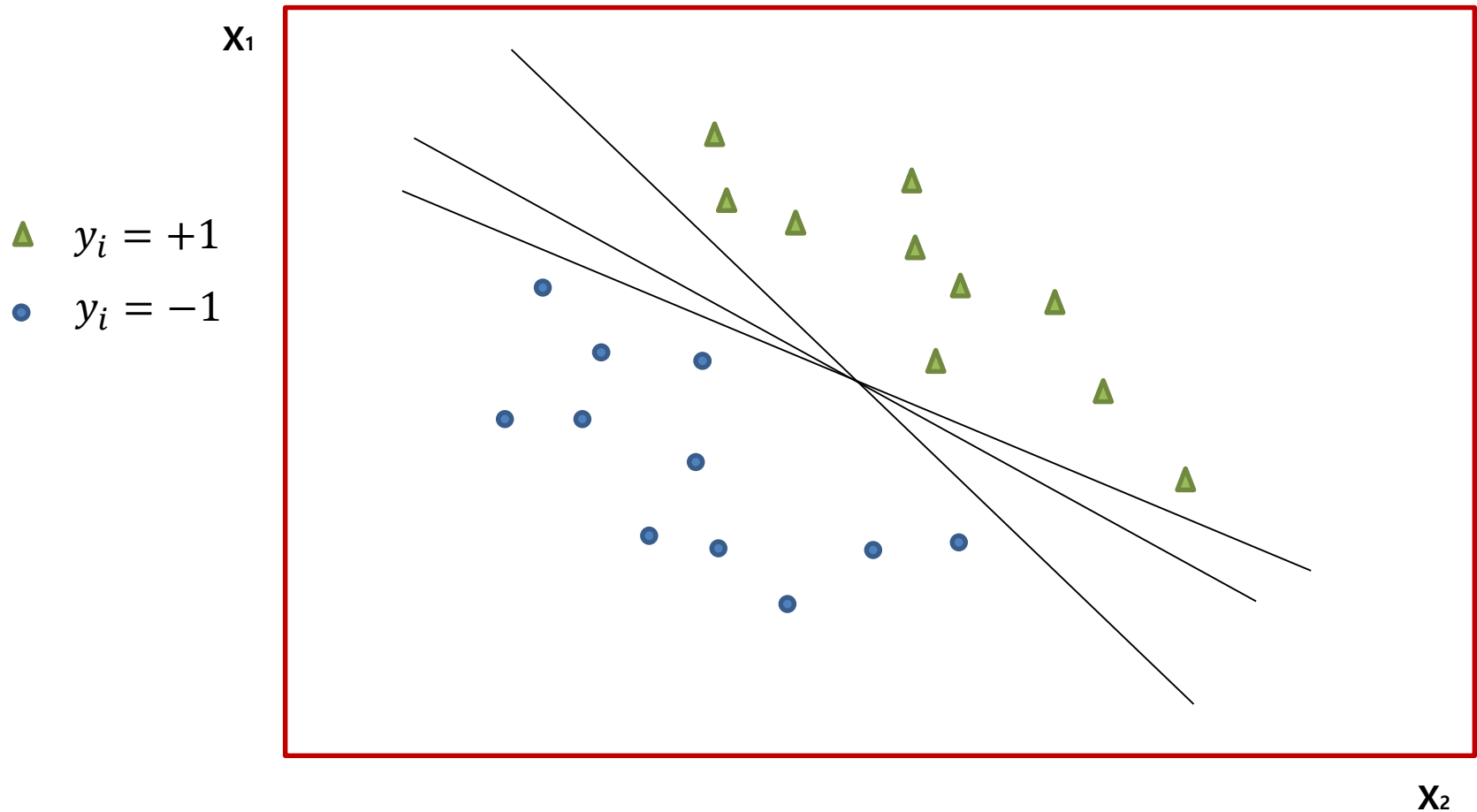
SVM

✦ SVM(Support Vector Machine)

- 벡터 공간에 위치한 훈련 데이터의 좌표와 각 데이터가 어떤 분류 값을 가져야 하는지 정답을 입력 받아서 학습
- 같은 분류 값을 갖는 데이터끼리 같은 공간에 위치하도록 벡터 공간을 나눔
- 분류 및 회귀문제에 적용 가능하며 예측이 정확하고 여러가지 형태의 자료에 적용이 쉬워 여러가지 예측 문제에서 각광받음
- 계산량이 자료의 수에 비례하지만 차원의 크기에는 덜 민감

SVM

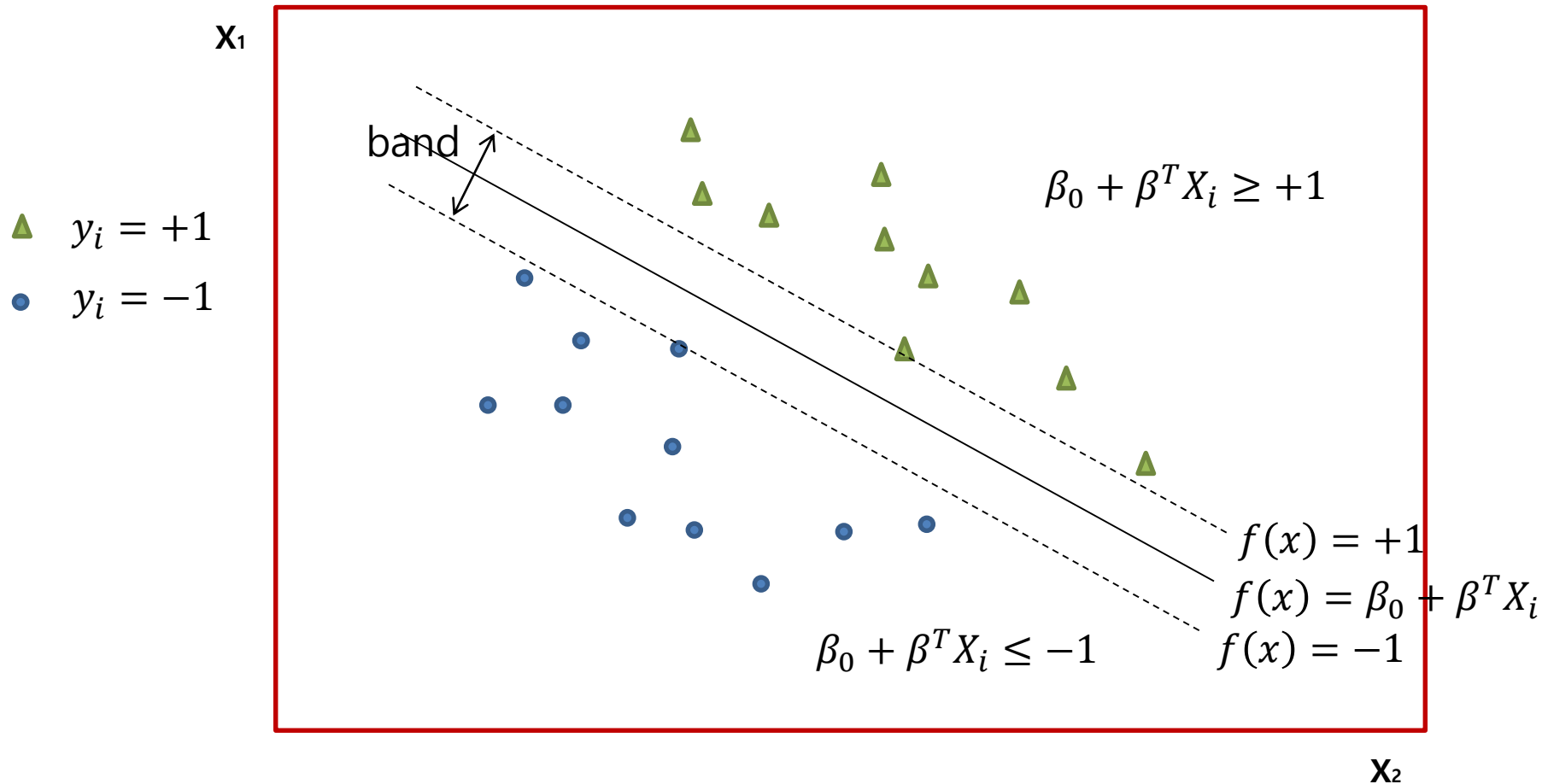
✦ SVM(Support Vector Machine)



분류

SVM

✦ SVM(Support Vector Machine)



분류

SVM

✦ SVM(Support Vector Machine)

$$\begin{aligned} \beta_0 + \beta^T X^+ &= +1 \\ \beta_0 + \beta^T X^- &= -1 \end{aligned} \longrightarrow \beta^T (X^+ - X^-) = 2 : \text{ 두 직선간의 거리}$$

- 표준화: $\frac{\beta^T (X^+ - X^-)}{\|\beta\|} = \frac{2}{\|\beta\|}$

→ $y_i(\beta_0 + \beta^T X_i) \geq +1$ 인 조건하에서 $\|\beta\|$ 를 최소화하는 β

- 자료가 선형적으로 분리가 불가능한 경우 슬랙변수를 사용

→ $y_i(\beta_0 + \beta^T X_i) \geq +1 - \xi_i$ 인 조건하에서 $\|\beta\| + C \sum_{i=1}^n \xi_i$ 를 최소화하는 β

분류

SVM

✦ SVM(Support Vector Machine)

- 자료가 선형적으로 분리가 불가능한 경우 슬랙변수를 사용

→ $y_i(\beta_0 + \beta^T X_i) \geq +1 - \xi_i$ 인 조건하에서 $\|\beta\| + C \sum_{i=1}^n \xi_i$ 를 최소화하는 β

- C 를 크게주면 $\sum_{i=1}^n \xi_i$ 가 약간만 커지더라도 $\|\beta\| + C \sum_{i=1}^n \xi_i$ 가 커지게 되고 오차를 허용하지 않으므로 밴드의 넓이가 좁아짐

→ 과대적합 문제 발생

- C 를 작게주면 $\sum_{i=1}^n \xi_i$ 가 커도되므로 밴드의 넓이가 커지게 되어 편의가 발생

→ 교차검증을 통해 C 값 결정

분류

SVM

✦ SVM(Support Vector Machine)

```
from sklearn.svm import SVC  
svm=SVC(kernel= , C= , gamma= , random_state=1)  
svm.fit(X data, y data)  
svm.predict_proba(X data)  
svm.predict(X data)
```

옵션:

- kernel: 선형('linear')/비선형('rbf')
- C: 선형 SVM 초모수값
- gamma: 비선형 SVM 비선형성 값
- random_state: 초기난수값

분류

SVM

- iris 데이터셋을 이용한 선형 SVM분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분

```
import seaborn as sns
iris=sns.load_dataset('iris')
print(iris.head())

X=iris.drop('species', axis=1)
X.shape
y=iris['species']

from sklearn.preprocessing import LabelEncoder
import numpy as np
classle=LabelEncoder()
y=classle.fit_transform(iris['species'].values)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1,stratify=y)
```

분류

SVM

- iris 데이터셋을 이용한 선형 SVM분류 알고리즘
 - Step 5: 선형 SVM 분류모형

```
from sklearn.svm import SVC
svm=SVC(kernel='linear', C=1.0, random_state=1)
svm.fit(X_train, y_train)
svm.predict_proba(X_train)
y_train_pred=svm.predict(X_train)
y_test_pred=svm.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_train, y_train_pred))
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)

svm_report = metrics.classification_report(y_test, y_test_pred)
print(svm_report)
```

분류

SVM

- Penguins 데이터셋을 이용한 선형 SVM분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분
 - seaborn의 penguins 자료 불러오기
 - island, sex변수 삭제
 - 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'자료 중 결측자료를 평균으로 대체
 - 독립변수: 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'
 - 종속변수: 'species'
 - 독립변수 표준화 후 train과 test 데이터로 구분
 - » random number는 1, train과 test는 7:3의 비율로 분류

분류

SVM

- Penguins 데이터셋을 이용한 선형 SVM분류 알고리즘
 - Step 1~3: 데이터 준비, 전처리 및 분석 사용속성 선택

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('penguins')
pd.set_option('display.max_columns', 15)

rdf = df.drop(['island', 'sex'], axis=1)

rdf =
rdf.dropna(subset=['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
how='any', axis=0)

rdf['bill_length_mm'].fillna(rdf['bill_length_mm'].mean(), inplace=True)
rdf['bill_depth_mm'].fillna(rdf['bill_depth_mm'].mean(), inplace=True)
rdf['flipper_length_mm'].fillna(rdf['flipper_length_mm'].mean(), inplace=True)
rdf['body_mass_g'].fillna(rdf['body_mass_g'].mean(), inplace=True)
```

분류

SVM

- Penguins 데이터셋을 이용한 선형 SVM분류 알고리즘
 - Step 4: 데이터셋 구분

```
X=rdf[['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']]
y=rdf['species']

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
```

분류

SVM

- Penguins 데이터셋을 이용한 선형 SVM분류 알고리즘
 - Step 5: SVM 분류모형
 - 선형 SVM 수행
 - » 초모수 c 값: 1, Random Number: 1
 - Confusion matrix 작성 후 분류 정확도 계산
 - » 전체 오분류률, precision, recall, F1-score 계산

분류

SVM

- Penguins 데이터셋을 이용한 선형 SVM분류 알고리즘
 - Step 5: SVM 분류모형

```
from sklearn.svm import SVC
svm=SVC(kernel='linear', C=1.0, random_state=1, probability=True)
svm.fit(X_train, y_train)
svm.predict_proba(X_train)
y_train_pred=svm.predict(X_train)
y_test_pred=svm.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_train, y_train_pred))
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)

from sklearn import metrics
svm_report = metrics.classification_report(y_test, y_test_pred)
print(svm_report)
```

분류

SVM

- iris 데이터셋을 이용한 비선형 SVM분류 알고리즘
 - Step 5: 선형 SVM 분류모형

```
from sklearn.svm import SVC
svm=SVC(kernel='rbf ' , C=1.0, gamma=0.2, random_state=1)
svm.fit(X_train, y_train)
svm.predict_proba(X_train)
y_train_pred=svm.predict(X_train)
y_test_pred=svm.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_train, y_train_pred))
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)

from sklearn import metrics
svm_report = metrics.classification_report(y_test, y_test_pred)
print(svm_report)
```

분류

SVM

- Penguins 데이터셋을 이용한 비선형 SVM분류 알고리즘
 - Step 5: SVM 분류모형
 - 비선형 SVM 수행
 - » 초모수 c 값: 1, γ 값: 0.2, Random Number: 1
 - Confusion matrix 작성 후 분류 정확도 계산
 - » 전체 오분류률, precision, recall, F1-score 계산

SVM

- Penguins 데이터셋을 이용한 비선형 SVM분류 알고리즘
 - Step 5: SVM 분류모형

```
from sklearn.svm import SVC
svm=SVC(kernel='rbf', C=1.0, gamma=0.2, random_state=1, probability=True)
svm.fit(X_train, y_train)
svm.predict_proba(X_train)
y_train_pred=svm.predict(X_train)
y_test_pred=svm.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_train, y_train_pred))
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)

from sklearn import metrics
svm_report = metrics.classification_report(y_test, y_test_pred)
print(svm_report)
```

분류

SVM

- iris 데이터셋을 이용한 SVM 초모수 선택 알고리즘
 - make_pipeline 및 GridSearchCV를 이용한 초모수 선택

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

pipe_svc = make_pipeline(StandardScaler(), SVC(random_state=1))
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'svc__C': param_range, 'svc__kernel': ['linear']},
               {'svc__C': param_range, 'svc__gamma': param_range,
                'svc__kernel': ['rbf']}]
gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid,
                  scoring='accuracy', cv=10)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```


분류

SVM

- iris 데이터셋을 이용한 SVM 초모수 선택 알고리즘
 - make_pipeline 및 GridSearchCV를 이용한 초모수 선택

```
clf = gs.best_estimator_  
clf.fit(X_train, y_train)  
print(clf.score(X_train, y_train))  
clf.score(X_test, y_test)  
  
gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid,  
                  scoring='accuracy', cv=3)  
from sklearn.model_selection import cross_val_score  
scores = cross_val_score(gs, X, y, scoring='accuracy', cv=5)  
print('CV accuracy: %.8f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

분류

SVM

- Penguins 데이터셋을 이용한 SVM 초모수 선택 알고리즘
 - 선형 및 비선형 중 선택
 - 적절한 C값 선택
 - 적절한 gamma값 선택
 - K-fold CV를 통한 score 확인

분류

SVM

- Penguins 데이터셋을 이용한 SVM 초모수 선택 알고리즘
 - make_pipeline 및 GridSearchCV를 이용한 초모수 선택

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

pipe_svc = make_pipeline(StandardScaler(), SVC(random_state=1))
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'svc__C': param_range, 'svc__kernel': ['linear']},
               {'svc__C': param_range, 'svc__gamma': param_range,
                'svc__kernel': ['rbf']}]
gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid,
                  scoring='accuracy', cv=10)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

분류

SVM

- Penguins 데이터셋을 이용한 SVM 초모수 선택 알고리즘
 - make_pipeline 및 GridSearchCV를 이용한 초모수 선택

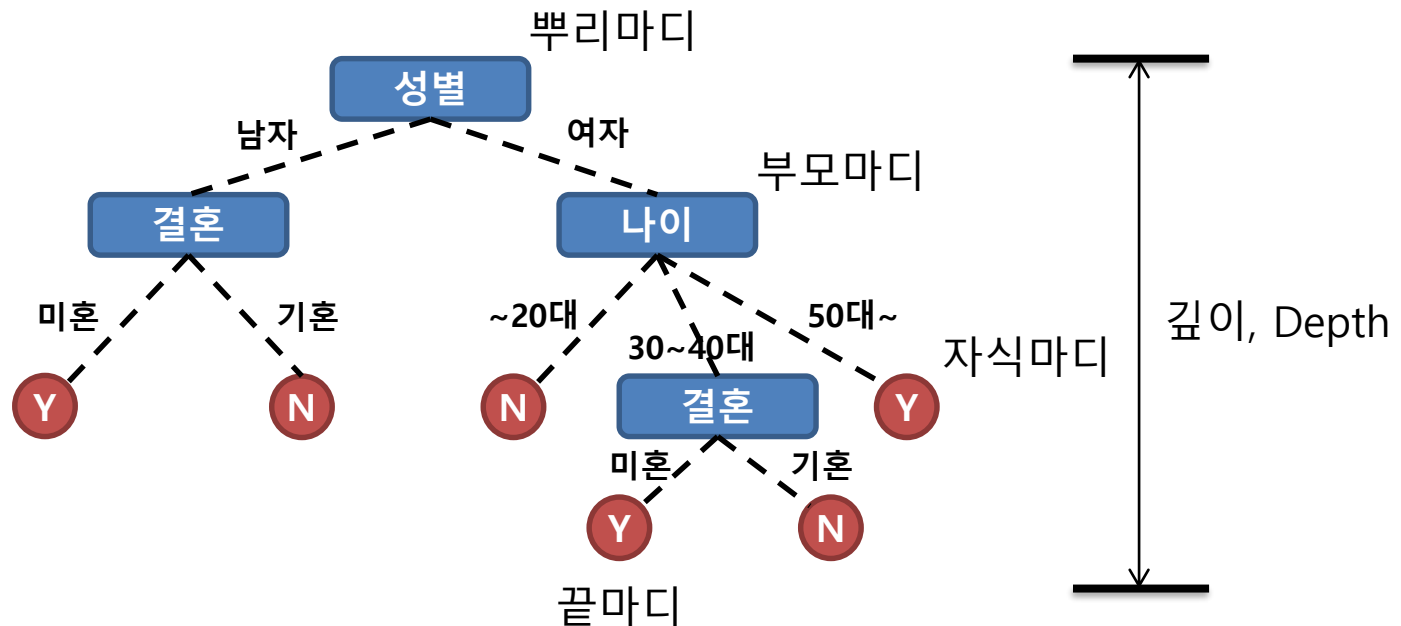
```
clf = gs.best_estimator_  
clf.fit(X_train, y_train)  
print(clf.score(X_train, y_train))  
clf.score(X_test, y_test)  
  
gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid,  
                  scoring='accuracy', cv=3)  
from sklearn.model_selection import cross_val_score  
import numpy as np  
scores = cross_val_score(gs, X, y, scoring='accuracy', cv=5)  
print('CV accuracy: %.8f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

분류

Decision Tree

✦ 의사결정나무(Decision Tree)

- 트리(Tree)구조를 사용하고, 각 분기점(Node)에는 분석 대상의 속성이 위치
- 분기점마다 목표 값을 가장 잘 분류할 수 있는 속성을 찾아서 배치
- 해당 속성이 갖는 값을 이용하여 새로운 가지(Branch)를 생성
- If-then으로 표현되는 규칙으로 이해가 쉽고 해석이 쉬움



분류

Decision Tree

✦ 의사결정나무 장점

- 해석의 용이성
 - 모델을 사용자가 쉽게 이해 가능
 - 새로운 자료에 모델을 적합 시키기가 용이함
 - 목표변수를 설명하기 위한 입력변수의 중요성 파악 용이함
- 상호작용 효과의 해석
 - 두 개 이상의 변수가 결합하여 목표변수에 어떻게 영향을 주는지 쉽게 파악 가능
- 비모수적 모형
 - 선형성, 정규성, 등분산성 등의 가정이 필요하지 않음

✦ 의사결정나무 단점

- 비연속성
 - 연속형 변수를 비연속적인 값으로 취급하기 때문에 분리 경계점 근방에서 예측 오류 가능성이 큼
- 선형성 또는 주효과의 결여
 - 다른 예측변수와 관련시키지 않고 각 변수의 영향력 해석 결과를 얻을 수 없음

분류

Decision Tree

✦ 의사결정나무 형성과정

- 나무의 성장
 - 각 마디에서 적절한 최적의 분리규칙을 찾아서 나무를 성장시킴.
 - 정지규칙을 만족하면 중단
- 가지치기
 - 오분류율을 크게 할 위험이 높거나 부적절한 추론규칙을 가지고 있는 가지 제거
- 타당성 평가
 - 테스트 자료를 사용하여 의사결정나무 평가
- 해석 및 예측
 - 구축된 나무모형을 해석하고 예측모형을 설정

분류

Decision Tree

✦ 의사결정나무 형성과정

– 분리규칙

- 분리규칙: 분리에 사용될 입력변수의 선택과 분리가 이루어질 기준
- 연속변수인 경우 분리기준보다 작으면 왼쪽 자식마디, 크면 오른쪽 자식마디
- 범주형인 경우 분리기준은 전체범주를 두 개의 부분집합으로 나눔

– 순수도와 불순도

- 각 마디에서 분리변수와 분리기준은 목표변수의 분포를 가장 잘 구별해주도록 설정
- 목표변수의 분포를 얼마나 잘 구별하는가에 대한 측정치로 순수도 또는 불순도를 사용

– 불순도의 측도

- 범주형 목표변수: 카이제곱, 지니지수, 엔트로피지수
- 연속형 목표변수: 분산분석에 의한 F통계량, 분산의 감소량

분류

Decision Tree

✦ 의사결정나무 정지규칙

– 정지규칙

- 정지규칙: 현재의 마디가 더 이상 분리가 일어나지 못하게 하는 규칙
- 규칙 종류
 - 모든 자료가 한 그룹에 속할 때
 - 마디에 속하는 자료가 일정 수 이하일 때
 - 불순도의 감소량이 아주 작을 때
 - 뿌리마디로부터의 깊이가 일정 수 이상일 때

분류

Decision Tree

✦ 의사결정나무(Decision Tree)

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(criterion= , max_depth= , random_state=1)
dtc.fit(X_train, y_train)
dtc.predict_proba(X_train)
dtc.predict(X_train)
```

옵션:

- criterion: 지니('gini')/엔트로피('entropy')
- max_depth: 최대깊이
- min_sample_split: 분기 최소 데이터수
- min_sample_leaf: 끝마디 최소 데이터수
- random_state: 초기난수값

분류

Decision Tree

✦ 의사결정나무(Decision Tree)

* pydotplus 모듈 설치
1. Anaconda Prompt 실행
2. conda install -c conda-forge pydotplus 입력
3. y 입력

Anaconda Prompt

done

(base) C:\Users\LeeKJ>conda install -c conda-forge pydotplus_

```
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(result, filled=True, rounded=True,
                           class_names= , feature_names= )
graph = graph_from_dot_data(dot_data)
```

```
from IPython.display import Image
Image(graph.create_png())
```

```
graph.write_png('./tree.png')
```

옵션:

- class_names: 종속변수 요인 이름
- feature_names: 독립변수 이름

분류

Decision Tree

- iris 데이터셋을 이용한 Decision Tree분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분

```
import seaborn as sns
iris=sns.load_dataset('iris')
print(iris.head())

X=iris.drop('species', axis=1)
X.shape
y=iris['species']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1,stratify=y)
```

Decision Tree

- iris 데이터셋을 이용한 Decision Tree 분류 알고리즘
 - Step 5: Decision Tree 분류모형

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=1)
result=dtc.fit(X_train, y_train)
dtc.predict_proba(X_train)
y_train_pred = dtc.predict(X_train)
y_test_pred = dtc.predict(X_test)

print('Misclassified training samples: %d' %(y_train != y_train_pred).sum())
print('Misclassified test samples: %d' %(y_test != y_test_pred).sum())

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)

from sklearn import metrics
dtc_report = metrics.classification_report(y_test, y_test_pred)
print(dtc_report)
```

분류

Decision Tree

- iris 데이터셋을 이용한 Decision Tree 분류 알고리즘
 - Step 6: Decision Tree 분류모형 그리기

```
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(dtc, filled=True, rounded=True,
                           class_names=['Setosa', 'Versicolor', 'Virginica'],
                           feature_names=['sepal length', 'sepal width', 'petal length', 'petal width'])
graph = graph_from_dot_data(dot_data)

from IPython.display import Image
Image(graph.create_png())

graph.write_png('./tree.png')
```

분류

Decision Tree

- mpg 데이터셋을 이용한 선형 Decision Tree 분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분
 - seaborn의 mpg 자료 불러오기
 - 'name' 변수 삭제
 - 'horsepower'에서 결측자료 자료 삭제
 - 'mpg', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', 'origin' 자료만 분석에 사용
 - 독립변수: 'mpg', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', 'origin'
 - 종속변수: 'origin'
 - 독립변수 표준화 후 train과 test 데이터로 구분
 - » Random number는 1, train과 test는 7:3의 비율로 분류

분류

Decision Tree

- mpg 데이터셋을 이용한 Decision Tree 분류 알고리즘
 - Step 1~3: 데이터 준비, 전처리 및 분석 사용속성 선택

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('mpg')
print(df.head())

pd.set_option('display.max_columns', 15)
print(df.head())
print(df.info())

rdf = df.drop(['name'], axis=1)
print(rdf.columns.values)

rdf = rdf.dropna(subset=['horsepower'], how='any', axis=0)

print(rdf.describe(include='all'))

ndf = rdf[['mpg', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', 'origin']]
print(ndf.head())
```


분류

Decision Tree

- mpg 데이터셋을 이용한 Decision Tree 분류 알고리즘
 - Step 4: 데이터셋 구분

```
X=ndf[['mpg', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year']]
y=ndf['origin']

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1,stratify=y)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
```

분류

Decision Tree

- mpg 데이터셋을 이용한 Decision Tree 분류 알고리즘
 - Step 5: Decision Tree 분류모형
 - Decision Tree 수행
 - » criterion: 'gini', max_depth: 3
 - Confusion matrix 작성 후 분류 정확도 계산
 - » 전체 오분류률, precision, recall, F1-score 계산
 - Decision Tree 그리기

Decision Tree

- mpg 데이터셋을 이용한 Decision Tree 분류 알고리즘
 - Step 5: Decision Tree 분류모형

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=1)
result=dtc.fit(X_train, y_train)
dtc.predict_proba(X_train)
y_train_pred = dtc.predict(X_train)
y_test_pred = dtc.predict(X_test)

print('Misclassified training samples: %d' %(y_train != y_train_pred).sum())
print('Misclassified test samples: %d' %(y_test != y_test_pred).sum())

from sklearn.metrics import accuracy_score, confusion_matrix
print(accuracy_score(y_test, y_test_pred))
conf=confusion_matrix(y_true=y_test, y_pred=y_test_pred)
print(conf)

from sklearn import metrics
dtc_report = metrics.classification_report(y_test, y_test_pred)
print(dtc_report)
```

분류

Decision Tree

- mpg 데이터셋을 이용한 Decision Tree 분류 알고리즘
 - Step 6: Decision Tree 그리기

```
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
dot_data = export_graphviz(dtc, filled=True, rounded=True,
                           class_names=['europe','japan','usa'],
                           feature_names=['mpg', 'displacement', 'horsepower', 'weight',
                           'acceleration', 'model_year'],
                           out_file=None)
graph = graph_from_dot_data(dot_data)

from IPython.display import Image
Image(graph.create_png())

graph.write_png('./tree.png')
```

분류

Decision Tree

- iris 데이터셋을 이용한 Decision Tree 초모수 선택 알고리즘
 - GridSearchCV를 이용한 초모수 선택

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=1),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None], 'criterion': ['gini']},
                      {'max_depth': [1, 2, 3, 4, 5, 6, 7, None], 'criterion': ['entropy']}],
                  scoring='accuracy', cv=3)

gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)

clf = gs.best_estimator_
clf.fit(X_train, y_train)
print(clf.score(X_train, y_train))
print(clf.score(X_test, y_test))

from sklearn.model_selection import cross_val_score
scores = cross_val_score(gs, X, y, scoring='accuracy', cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

분류

Decision Tree

- mpg 데이터셋을 이용한 Decision Tree 초모수 선택 알고리즘
 - 'gini' 및 'entropy' 중 선택
 - 적절한 가지 깊이값 선택
 - K-fold CV를 통한 score 확인

분류

Decision Tree

- mpg 데이터셋을 이용한 Decision Tree 초모수 선택 알고리즘
 - GridSearchCV를 이용한 초모수 선택

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=1),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None], 'criterion': ['gini']},
                    {'max_depth': [1, 2, 3, 4, 5, 6, 7, None], 'criterion': ['entropy']}],
                  scoring='accuracy', cv=3)

gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)

clf = gs.best_estimator_
clf.fit(X_train, y_train)
print(clf.score(X_train, y_train))
print(clf.score(X_test, y_test))

from sklearn.model_selection import cross_val_score
scores = cross_val_score(gs, X, y, scoring='accuracy', cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

Clustering



군집

Overview

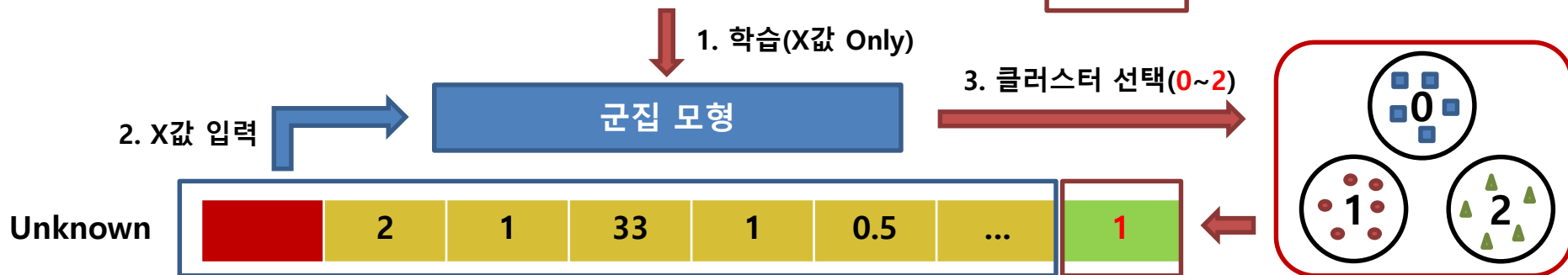
✦ 군집(clustering)

- 데이터셋의 관측 값이 갖고 있는 여러 속성을 분석하여 서로 비슷한 특징을 갖는 관측 값끼리 같은 클러스터로 묶는 알고리즘
- 어느 클러스터에도 속하지 못하는 관측 값이 존재할 수 있음
- 정답이 없는 상태에서 데이터 자체의 유사성만을 기준으로 판단

Train Data		X_1	X_2	X_3	X_4	X_5	...	Y
	0	4	1	9	0	0.05	...	?
	1	6	1	22	0	0.06	...	?

	100	3	0	43	0	0.01	...	?

정답 없음



군집

Overview

✦ 군집분석의 특징

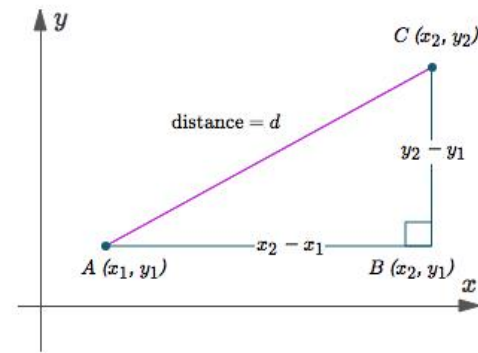
- 종속변수가 없는 데이터 마이닝 기법 (비지도 학습)
- 유클리드 거리 기반 유사 객체 묶음 (유사성 = 유클리드 거리)
- 전체적인 데이터 구조를 파악하는데 이용
- 분석결과에 대한 가설 검정 없음 (타당성 검증 방법 없음)
- 계층적 군집분석(탐색적), 비계층적 군집분석(확인적)
- 고객 DB -> 알고리즘 적용 -> 패턴 추출(rule) -> 근거리 모형으로 군집 형성
- 척도 : 등간, 비율척도 => 명목척도를 만듦

군집

Overview

- ✦ 유클리드 거리 (Euclidean distance)
 - 두 데이터 간의 직선거리
 - 유클리드 거리를 거리 척도로 사용하기 위해 데이터 정규화 필요
 - 데이터의 차원이 증가할수록 유클리드의 유용성이 떨어짐
 - 직관적이고 구현이 간단하며 좋은 결과를 보여주기 때문에 가장 많이 사용

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$



군집

Overview

- ✦ 민코프스키 거리 (Minkowski distance)
 - 유클리드 거리와 맨해튼 거리를 일반화한 거리
 - 유클리드 거리, 맨해튼 거리 등의 거리 척도와 같은 단점이 존재
 - p값을 통해 가장 적합한 거리 측정값을 찾을 수 있음

$$d(x, y) = \left[\sum_{i=1}^n |x_i - y_i|^p \right]^{\frac{1}{p}}$$

- 맨해튼 거리 (Manhattan distance)

$$d(x, y) = \sum_{i=1}^p |x_i - y_i|$$

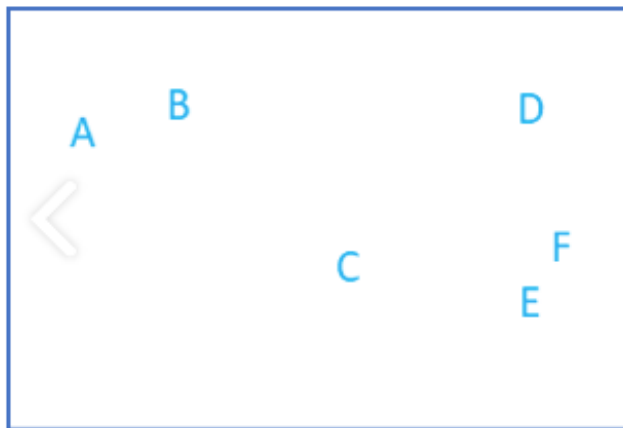
군집

Hierarchical analysis

✦ 계층적 군집분석(Hierarchical analysis)

- 유클리드 거리를 이용한 군집분석 방법
- 계층적으로 군집 결과 도출
- 탐색적 군집분석
- 계층적 군집분석의 결과

- 덴드로그램(Dendrogram): 표본들이 군을 형성하는 과정을 나타내는 나무 형식의 그림



Dendrogram

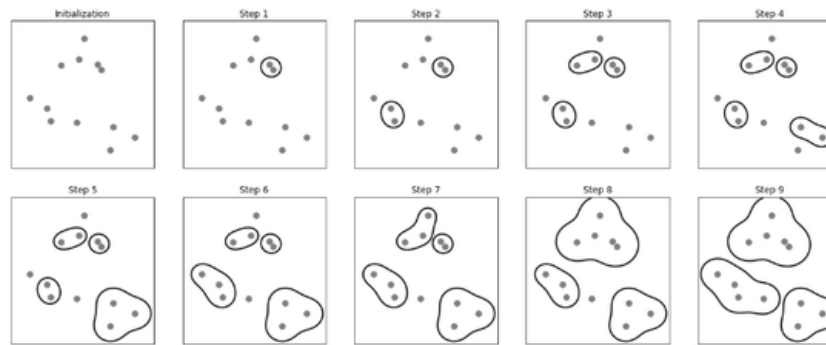


군집

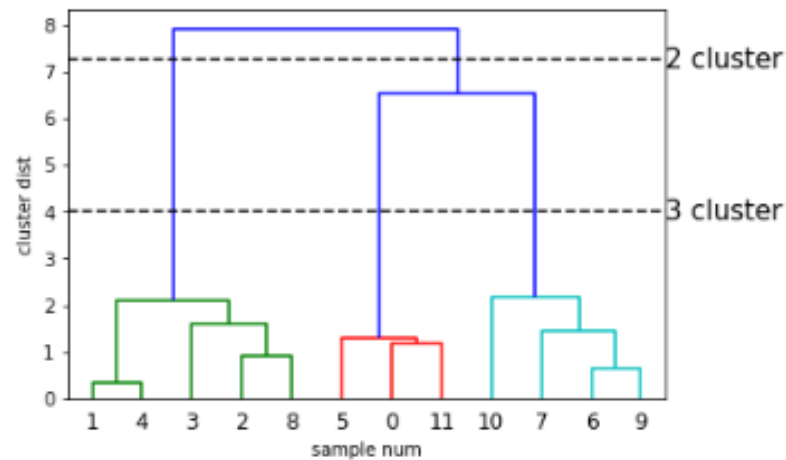
Hierarchical analysis

✦ 계층적 군집분석(Hierarchical analysis)

– 계층적 군집분석의 결과



- 군집수는 사용자가 정할 수 있음



군집

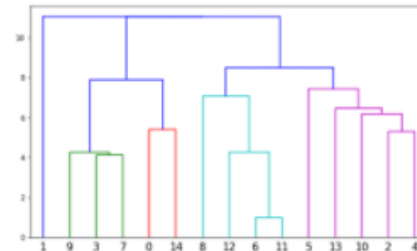
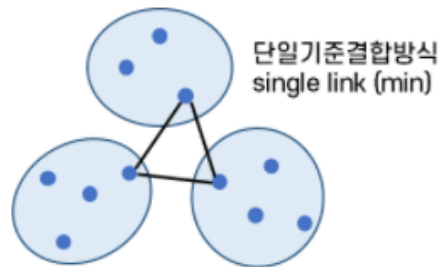
Hierarchical analysis

✦ 계층적 군집분석(Hierarchical analysis)

– 군집화 방식

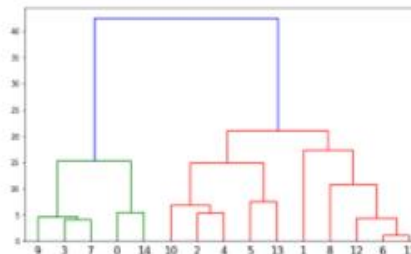
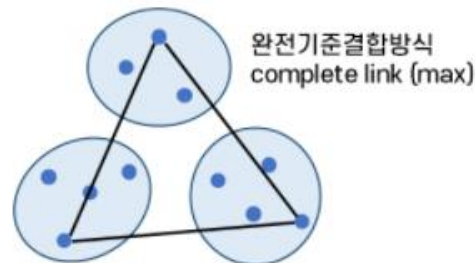
• 단일기준결합방식(single link) :

- 각 군집에서 중심으로부터 거리가 가까운 것 1개씩 비교하여 가장 가까운 것끼리 군집화



• 완전기준결합방식(complete link) :

- 각 군집에서 중심으로부터 가장 먼 대상끼리 비교하여 가장 가까운 것끼리 군집화



군집

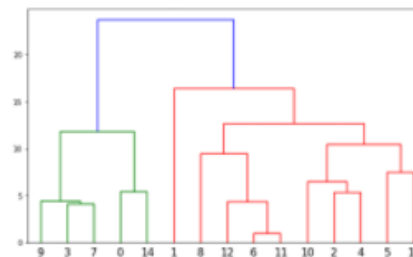
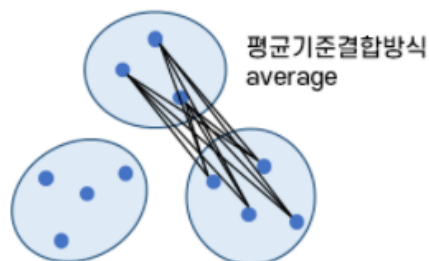
Hierarchical analysis

✦ 계층적 군집분석(Hierarchical analysis)

– 군집화 방식

- 평균기준결합방식(average link) :

- 한 군집 안에 속해 있는 모든 대상과 다른 군집에 속해있는 모든 대상의 쌍 집합에 대한 거리를 평균 계산하여 가장 가까운 것끼리 군집화



군집

Hierarchical analysis

✦ 계층적 군집분석(Hierarchical analysis)

```
from scipy.spatial.distance import pdist, squareform
row_dist = pd.DataFrame(squareform(pdist(DataSet, metric= )))
```

```
from scipy.cluster.hierarchy import linkage
clusters = linkage(y=DataSet, method= , metric= )
```

```
from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt
plt.figure(figsize = (15, 6))
dendrogram(clusters, leaf_rotation=90, leaf_font_size=12)
plt.tight_layout()
plt.ylabel('Euclidean distance')
plt.show()
```

```
from scipy.cluster.hierarchy import fcluster
cut_tree = fcluster(clusters, t= , criterion='distance')
```

옵션:

- metric: 'euclidean'
- 'minkowski'
- method: 'single'
- 'complete'
- 'average'
- 'ward'
- t: cut_off value

군집

Hierarchical analysis

- iris 데이터셋을 이용한 Hierarchical Analysis 분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분

```
import seaborn as sns
iris=sns.load_dataset('iris')

X=iris.drop('species', axis=1)
y=iris['species']

from sklearn.preprocessing import LabelEncoder
import numpy as np
classle=LabelEncoder()
y=classle.fit_transform(iris['species'].values)

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(X)
X_std=sc.transform(X)
```

군집

Hierarchical analysis

- iris 데이터셋을 이용한 Hierarchical Analysis 분류 알고리즘
 - Step 5: Hierarchical Analysis 분류모형

```
from scipy.spatial.distance import pdist, squareform
row_dist = pd.DataFrame(squareform(pdist(X_std, metric='euclidean'))))

from scipy.cluster.hierarchy import linkage
clusters = linkage(y=X_std, method='single', metric='euclidean')
clusters

from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt
plt.figure(figsize = (15, 6))
dendrogram(clusters, leaf_rotation=90, leaf_font_size=12)
plt.tight_layout()
plt.ylabel('Euclidean distance')
plt.show()

from scipy.cluster.hierarchy import fcluster
cut_tree = fcluster(clusters, t=1, criterion='distance')
```

군집

Hierarchical analysis

- Penguins 데이터셋을 이용한 Hierarchical Analysis 분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분
 - seaborn의 penguins 자료 불러오기
 - island, sex 변수 삭제
 - 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g' 자료 중 결측자료를 평균으로 대체
 - 독립변수: 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'
 - 종속변수: 'species'
 - 독립변수 표준화 후 train과 test 데이터로 구분
 - » random number는 1, train과 test는 7:3의 비율로 분류

군집

Hierarchical analysis

- Penguins 데이터셋을 이용한 Hierarchical Analysis 분류 알고리즘
 - Step 1~3: 데이터 준비, 전처리 및 분석 사용속성 선택

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('penguins')
pd.set_option('display.max_columns', 15)

rdf = df.drop(['island', 'sex'], axis=1)

rdf =
rdf.dropna(subset=['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
how='any', axis=0)

rdf['bill_length_mm'].fillna(rdf['bill_length_mm'].mean(), inplace=True)
rdf['bill_depth_mm'].fillna(rdf['bill_depth_mm'].mean(), inplace=True)
rdf['flipper_length_mm'].fillna(rdf['flipper_length_mm'].mean(), inplace=True)
rdf['body_mass_g'].fillna(rdf['body_mass_g'].mean(), inplace=True)
```

군집

Hierarchical analysis

- Penguins 데이터셋을 이용한 Hierarchical Analysis 분류 알고리즘
 - Step 4: 데이터셋 구분

```
X=rdf[['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']]
y=rdf['species']

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
```

군집

Hierarchical analysis

- Penguins 데이터셋을 이용한 Hierarchical Analysis 분류 알고리즘
 - Step 5: Hierarchical Analysis 분류모형
 - Hierarchical Analysis 수행
 - » metric: 'euclidean', method: 'average'
 - Hierarchical Analysis 수행 후 Dendrogram 그리기
 - 2개의 군집이 나타나게 Cut_off value 설정

군집

Hierarchical analysis

- Penguins 데이터셋을 이용한 Hierarchical Analysis 분류 알고리즘
 - Step 5: Hierarchical Analysis 분류모형

```
import pandas as pd
from scipy.spatial.distance import pdist, squareform
row_dist = pd.DataFrame(squareform(pdist(X_std,
metric='euclidean')),columns=X.index,index=X.index)

from scipy.cluster.hierarchy import linkage
clusters = linkage(y=X_std, method='complete', metric='euclidean')

from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt
plt.figure(figsize = (15, 6))
dendrogram(clusters, leaf_rotation=90, leaf_font_size=12)
plt.tight_layout()
plt.ylabel('Euclidean distance')
plt.show()

from scipy.cluster.hierarchy import fcluster
cut_tree = fcluster(clusters, t=5, criterion='distance')
```


군집

K-Means

✦ 비계층적 군집분석(K-Means)

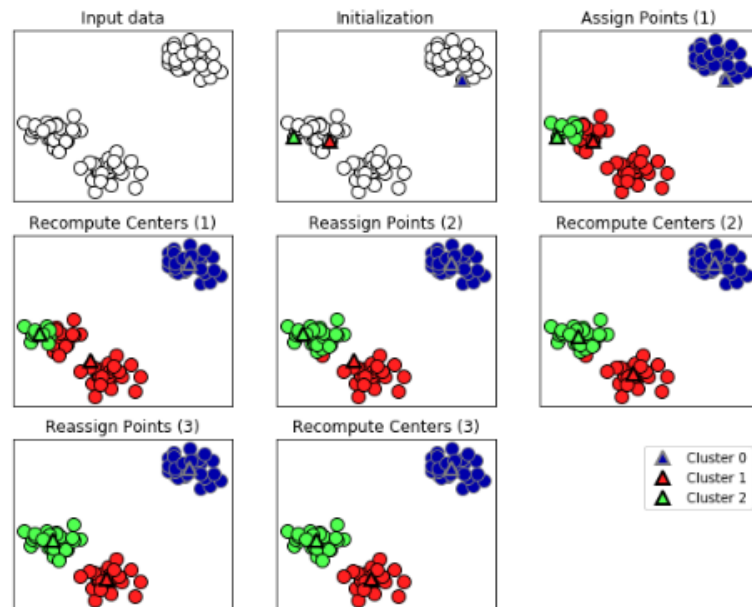
- 데이터의 어떤 영역을 대표하는 클러스터 중심을 찾는 방법
- 군집의 수를 알고 있는 경우 이용
- K는 미리 정하는 군집 수
- 확인적 군집분석
- 계층적 군집화의 결과에 의거하여 군집 수 결정
- 변수보다 관측대상 군집화에 많이 이용
- 군집의 중심은 사용자가 정함
- 계층적 군집분석보다 속도 빠름

군집

K-Means

✦ K-Means 알고리즘

- 클러스터 중심으로 삼을 데이터 포인트를 무작위로 초기화
- 데이터 포인트를 가장 가까운 클러스터 중심에 할당
- 다음 클러스터에 할당된 데이터포인트 평균으로 클러스터 중심 다시 지정
- 두 단계를 반복하고 할당된 데이터 포인트에 변화가 없을 때 종료



군집

K-Means

✦ K-Means

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters= , init= , n_init= , max_iter= , tol= ,
               random_state=1)
model.fit(X_std)
centers = model.cluster_centers_
pred = model.predict(X_std)
```

옵션:

- n_clusters: 군집 개수 지정
- init: 초기중심값 설정 방법
'k-means++'/'random'
- n_init: 초기중심점설정 반복 횟수
- max_iter: 최대반복 수 제한
- tol: 허용오차한계
- random_state: 시드넘버 지정

군집

K-Means

- iris 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분

```
import seaborn as sns
iris=sns.load_dataset('iris')

X=iris.drop('species', axis=1)
y=iris['species']

from sklearn.preprocessing import LabelEncoder
import numpy as np
classle=LabelEncoder()
y=classle.fit_transform(iris['species'].values)

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(X)
X_std=sc.transform(X)
```

군집

K-Means

- iris 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 5: K-means 분류모형

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3, random_state=1, init='k-means++')
model.fit(X_std)
centers = model.cluster_centers_
pred = model.predict(X_std)
```

군집

K-Means

- iris 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 6: 중심점 그래프 표현

```
import pandas as pd
X_std=pd.DataFrame(X_std,columns=X.columns)
irisData = X_std[['sepal_length', 'petal_length']]

from sklearn.cluster import KMeans
model = KMeans(n_clusters=3, random_state=1, init='k-means++')
model.fit(irisData)
centers = model.cluster_centers_
pred = model.predict(irisData)

import matplotlib.pyplot as plt
plt.scatter(x=irisData['sepal_length'], y=irisData['petal_length'], c=pred)
plt.scatter(x=centers[:,0], y=centers[:,1], marker='D', c='red')
```

군집

K-Means

- Penguins 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분
 - seaborn의 penguins 자료 불러오기
 - island, sex 변수 삭제
 - 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g' 자료 중 결측자료를 평균으로 대체
 - 독립변수: 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'
 - 종속변수: 'species'
 - 독립변수 표준화 후 train과 test 데이터로 구분
 - » random number는 1, train과 test는 7:3의 비율로 분류

군집

K-Means

- Penguins 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 1~3: 데이터 준비, 전처리 및 분석 사용속성 선택

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('penguins')
pd.set_option('display.max_columns', 15)

rdf = df.drop(['island', 'sex'], axis=1)

rdf =
rdf.dropna(subset=['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
how='any', axis=0)

rdf['bill_length_mm'].fillna(rdf['bill_length_mm'].mean(), inplace=True)
rdf['bill_depth_mm'].fillna(rdf['bill_depth_mm'].mean(), inplace=True)
rdf['flipper_length_mm'].fillna(rdf['flipper_length_mm'].mean(), inplace=True)
rdf['body_mass_g'].fillna(rdf['body_mass_g'].mean(), inplace=True)
```


군집

K-Means

- Penguins 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 4: 데이터셋 구분

```
X=rdf[['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']]
y=rdf['species']

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
```

군집

K-Means

- Penguins 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 5: K-means 분류모형
 - K-means 수행
 - » 초기중심값 설정방법: 'k-means++'
 - » 군집 수: 3

군집

K-Means

- Titanic 데이터셋을 이용한 K-means 분류 알고리즘
 - Step 5: K-means 분류모형

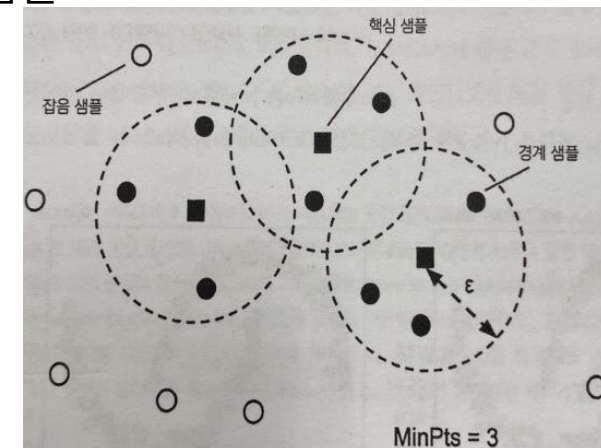
```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3, random_state=1, init='k-means++')
model.fit(X_std)
centers = model.cluster_centers_
pred = model.predict(X_std)
```

군집

DBSCAN

✦ 공간밀집도 군집분석(DBSCAN)

- Density-Based Spatial Clustering of Applications with Noise
 - 샘플이 조밀하게 모인 지역에 클러스터 레이블을 할당. 밀집도는 특정 반경 안에 있는 샘플 개수로 정의
- 레이블
 - 핵심샘플: 어떤 샘플의 특정반경 내에 있는 이웃 샘플이 지정된 개수 이상
 - 경계샘플: 반경 내 지정개수보다 이웃이 적으나, 다른 핵심샘플의 반경안에 있는 경우
 - 잡음샘플: 핵심샘플과 경계샘플이 아닌 다른 모든 샘플



군집

DBSCAN

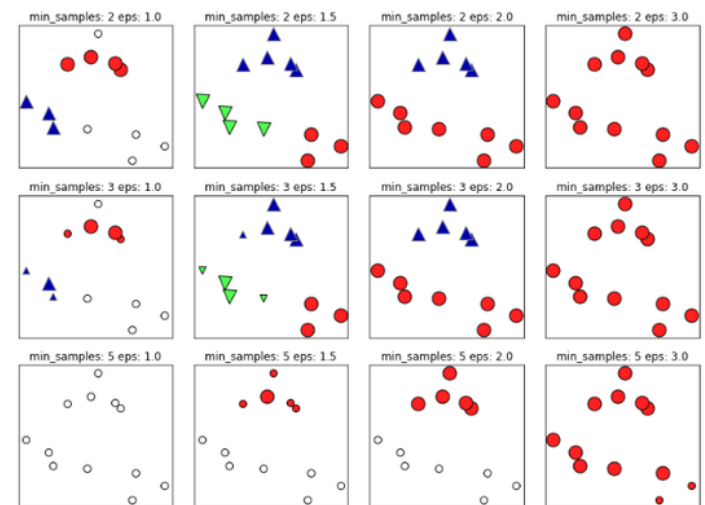
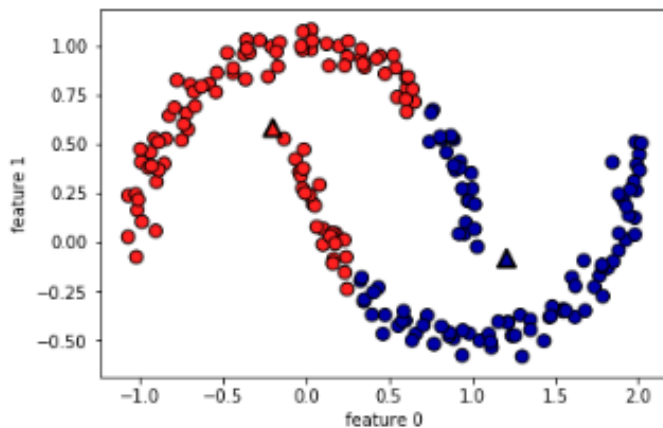
✦ 공간밀집도 군집분석(DBSCAN)

– 장점

- 원형 클러스터를 가정하지 않음
- 모든 샘플들을 반드시 클러스터에 할당하지 않음 (잡음샘플 구분)

– 알고리즘

- 개별 핵심샘플이나 반경 내 핵심샘플들끼리 연결한 핵심샘플의 그룹을 클러스터로 만듦
- 경계샘플들을 해당 핵심샘플의 클러스터에 할당



군집

DBSCAN

✦ DBSCAN

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps= ,min_samples= ,metric= )
pred = dbscan.fit_predict(DataSet)
```

옵션:

- eps: 거리 지정
- min_samples: 최소 포인트 수
- metric: 거리측정 방법

군집

DBSCAN

- iris 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분

```
import seaborn as sns
iris=sns.load_dataset('iris')

X=iris.drop('species', axis=1)
y=iris['species']

from sklearn.preprocessing import LabelEncoder
import numpy as np
classle=LabelEncoder()
y=classle.fit_transform(iris['species'].values)

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(X)
X_std=sc.transform(X)
```

군집

DBSCAN

- iris 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 5: DBSCAN 분류모형

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(min_samples=10,metric='euclidean')
pred = dbscan.fit_predict(X_std)
```


군집

DBSCAN

- iris 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 6: 중심점 그래프 표현

```
import pandas as pd
X_std=pd.DataFrame(X_std,columns=X.columns)
irisData = X_std[['sepal_length', 'petal_length']]

from sklearn.cluster import DBSCAN
dbscan = DBSCAN(min_samples=10,metric='euclidean')
pred = dbscan.fit_predict(irisData)

import matplotlib.pyplot as plt
plt.scatter(x=irisData['sepal_length'], y=irisData['petal_length'], c=pred)
plt.show()
```

군집

DBSCAN

- Penguins 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 1~4: 데이터 준비, 전처리 및 분석 사용속성 선택, 데이터셋 구분
 - seaborn의 penguins 자료 불러오기
 - island, sex 변수 삭제
 - 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g' 자료 중 결측자료를 평균으로 대체
 - 독립변수: 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'
 - 종속변수: 'species'
 - 독립변수 표준화 후 train과 test 데이터로 구분
 - » random number는 1, train과 test는 7:3의 비율로 분류

군집

DBSCAN

- Penguins 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 1~3: 데이터 준비, 전처리 및 분석 사용속성 선택

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('penguins')
pd.set_option('display.max_columns', 15)

rdf = df.drop(['island', 'sex'], axis=1)

rdf =
rdf.dropna(subset=['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
how='any', axis=0)

rdf['bill_length_mm'].fillna(rdf['bill_length_mm'].mean(), inplace=True)
rdf['bill_depth_mm'].fillna(rdf['bill_depth_mm'].mean(), inplace=True)
rdf['flipper_length_mm'].fillna(rdf['flipper_length_mm'].mean(), inplace=True)
rdf['body_mass_g'].fillna(rdf['body_mass_g'].mean(), inplace=True)
```

군집

DBSCAN

- Penguins 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 4: 데이터셋 구분

```
X=rdf[['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']]
y=rdf['species']

from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
```

군집

DBSCAN

- Penguins 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 5: DBSCAN 분류모형
 - DBSCAN 수행

군집

DBSCAN

- Penguins 데이터셋을 이용한 DBSCAN 분류 알고리즘
 - Step 5: DBSCAN 분류모형

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN()
pred = dbscan.fit_predict(X_std)
```