기초 인공지능 프로그래밍

11장 Pandas 라이브러리1

pandas

- 데이터 조작 및 분석을 위한 파이썬 라이브러리
- 관계 또는 레이블링 데이터로 쉽고 직관적으로 데이터 분석(data analysis)을 할 수 있도록 설계된 데이터 구조를 제공
- 엑셀의 파이썬 버전
- 대용량 데이터 처리하기 용이하며 머신 러닝/딥러닝 모델에 필요한 데이터 처리
- 데이터 처리에 필요한 다양한 함수들을 지원
- 다양한 외부 리소스 데이터를 읽고 처리하는 기능 지원
    - CSV 파일, 텍스트 파일, 엑셀 파일, web 데이터 등을 읽어 matplotlib를 이용하여 데이터를 시각화 할 수 있음

pandas

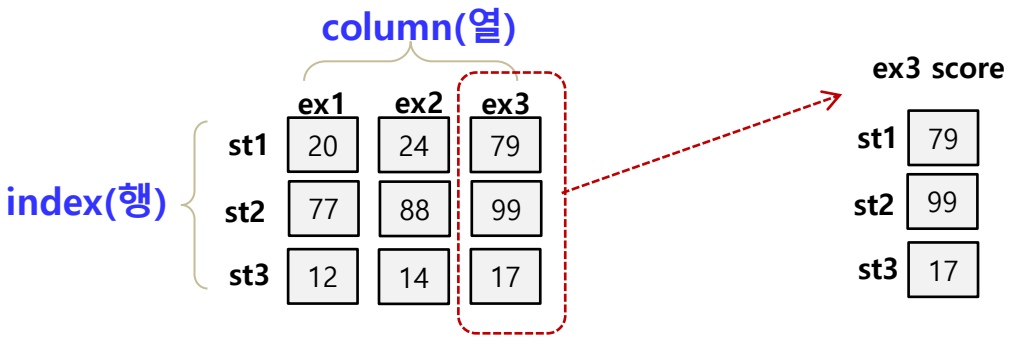
- 대부분의 데이터는 엑셀과 유사한 표 형태로 저장 됨
- 세가지 자료구조를 지원
  - 1차원 자료구조 **Series**
  - 2차원 자료구조 **DataFrame**
  - 3차원 자료구조 **Panel**
- pandas 라이브러리를 사용하기 위해서는 import 해야함

```
import pandas as pd    # 편리를 위해 별칭 pd 지정
```

- numpy 배열은 원소가 모두 같은 자료형이지만 pandas는 원소의 자료형들이 서로 달라도 됨

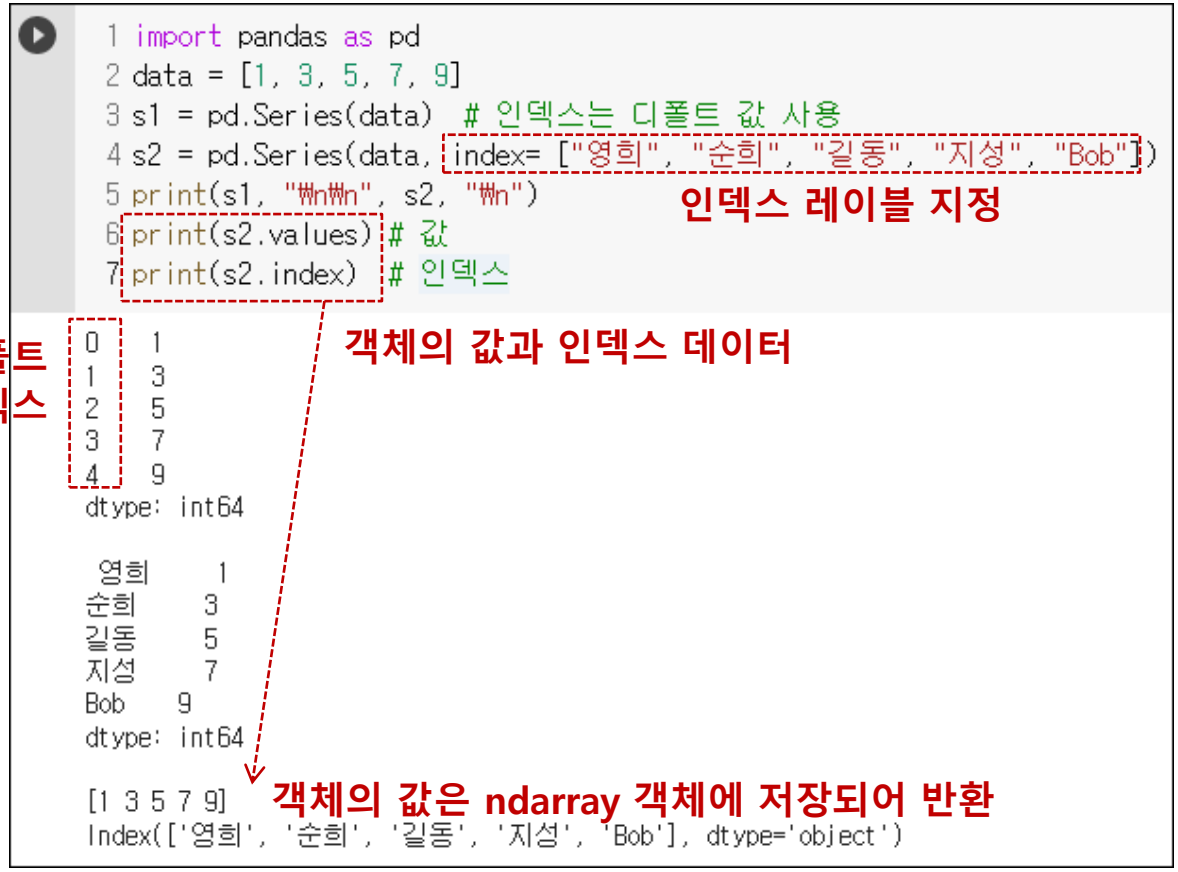
Series & DataFrame

- Series
  - 1차원 자료구조(index가 있는 1차원 배열로 볼 수 있음)
  - 각 원소는 인덱스와 값으로 이루어짐
  - 인덱스는 숫자 또는 문자로 지정할 수 있음
- DataFrame
  - 2차원의 행렬 데이터를 표 형태로 저장
  - 행(row)과 열(column)으로 이루어짐
  - 각 행과 열은 이름을 가지고 있어 데이터를 쉽게 검색, 필터링 할 수 있음
- Series 와 DataFrame
  - DataFrame에서 하나의 열 데이터를 추출하면 Series 객체



Series

- 리스트와 같은 일련의 시퀀스 데이터를 받아들여 인덱스가 있는 객체를 생성 : pd.Series( ) 함수 사용
- 인덱스 레이블 지정이 없으면, 디폴트 정수(0부터) 인덱스를 사용
- Series 객체의 속성인 index, values로 관련 데이터를 얻을 수 있음



Series

- 사전 자료형 데이터로 Series 객체 생성

```
1 import pandas as pd
2 data = pd.Series({"England":"London", "India":"New Delhi", "USA":"Washington",
3         "Belgium":"Brussels"})   # 사전 자료형으로 Series 객체 생성
4 print(data)
5 print(data.index)
6 print(data.values)
```

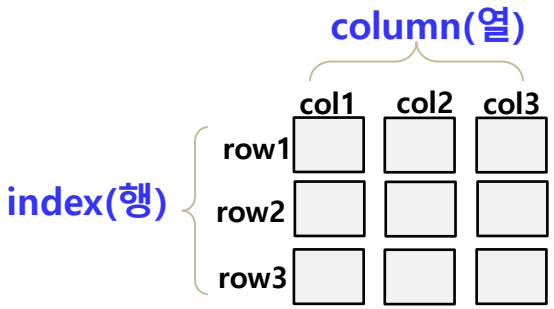
```
England       London
India      New Delhi
USA       Washington
Belgium      Brussels
dtype: object
Index(['England', 'India', 'USA', 'Belgium'], dtype='object')
['London' 'New Delhi' 'Washington' 'Brussels']
```

index

사전의 키를 index로, 사전의 값을 values로 구성

DataFrame

- DataFrame은 행과 열이 있는 테이블 형태
  - 인덱스와 열은 각각의 레이블(이름)을 가짐
  - 열 별로 다른 자료형의 데이터를 가질 수 있음
  - 기본적으로 열 단위로 데이터를 관리
- 다양한 자료형의 데이터를 DataFrame 자료구조로 변환할 수 있음
  - pd.DataFrame( ) 함수 사용해서 사전, 리스트, ndarray 객체, Series 자료구조등을 DataFrame 자료구조로 변환
  - 디폴트 인덱스는 0부터 시작하는 정수로 생성
  - 생성시 index 키워드로 인덱스 이름 지정 가능
  - 또한 columns 키워드로 열 이름도 지정 가능



각 행은 하나의 객체에 대한 정보를 표시
각 열은 서로 다른 속성을 나타냄

DataFrame

- 리스트 자료형으로 DataFrame 생성

```
1 import pandas as pd
2 data = [[2016, 2.8, '1.63M'], [2017, 3.1, '1.73M'],
3         [2018, 3.0, '1.83M'] ]
4 df1 = pd.DataFrame(data)
5 print(df1, "₩n")
6 df2 = pd.DataFrame(data, index= [1, 2, 3],
7                     columns= ['year', 'GDP_rate', 'GDP'])
8 print(df2)
```

디폴트 값(0부터의 정수)으로 인덱스, 열 이름생성

지정한 값으로 인덱스, 열 이름 생성

```
      0    1      2
0  2016  2.8  1.63M
1  2017  3.1  1.73M
2  2018  3.0  1.83M

   year  GDP_rate    GDP
1  2016        2.8  1.63M
2  2017        3.1  1.73M
3  2018        3.0  1.83M
```

DataFrame

- 사전 자료형으로 DataFrame 생성
  - 사전의 key 값은 열 이름, 사전의 value 값은 열의 값이 됨

```
1 import pandas as pd
2 data = {"year" : [2016,2017,2018], "GDP_rate" : [2.8, 3.1, 3.0],
3         "GDP": ["1.63M", "1.73M", "1.83M"]}
4 df = pd.DataFrame(data)  # df = pd.DataFrame(data = data) 와 동일
5 print(df, "₩n")
6 print(df.iloc[1],"₩n")
7 print(df["year"], "₩n") # print(df.year)와 동일
8 print(type(df.iloc[1]))
9 print(type(df["year"]))
```

row 참조 형식 : df.loc[index 이름], df.iloc[index 번호]

column 참조 형식 : df[“year”] 또는 df.year 와 같이 열 이름을 사용

```
   year  GDP_rate   GDP
0  2016       2.8  1.63M
1  2017       3.1  1.73M
2  2018       3.0  1.83M

year         2017
GDP_rate      3.1
GDP          1.73M
Name: 1, dtype: object

0    2016
1    2017
2    2018
Name: year, dtype: int64

<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

3개 열 생성.각 열은 Series 객체

DataFrame

- DataFrame 생성 예제

```
 1 import pandas as pd
 2 s1= pd.Series([2016, 2017, 2018])
 3 s2= pd.Series([2.8, 3.1, 3.0])
 4 s3= pd.Series(["1.63M", "1.73M", "1.83M"])
 5 data = {"year" : s1, "GDP_rate" : s2, "GDP": s3}
 6 df1= pd.DataFrame(data)
 7 print(df1, "₩n")
 8
 9 names = pd.Series(["Bob", "Jessica", "Mary", "John", "Kate"])
10 scores = pd.Series([56, 11, 99, 83, 45])
11 members = {"Name": names, "Score" : scores}
12 df2 = pd.DataFrame(members)
13 print(df2, "₩n")
14 df2["Grade"] = ["C", "F", "A", "B", "D"]    # 열 이름 "Grade"로 열 추가
15 df2.loc[5] = ["Hanna", 90, "A"]           # 행 추가
16 print(df2)
```

```
   year  GDP_rate    GDP
0  2016       2.8  1.63M
1  2017       3.1  1.73M
2  2018       3.0  1.83M

      Name  Score
0      Bob     56
1  Jessica     11
2     Mary     99
3     John     83
4     Kate     45

      Name  Score Grade
0      Bob     56     C
1  Jessica     11     F
2     Mary     99     A
3     John     83     B
4     Kate     45     D
5    Hanna     90     A
```

행 추가 시에 기존 인덱스 이름으로 지정하면 기존의 데이터 값들이 수정됨.

DataFrame

```
1 import numpy as np
2 import pandas as pd
3 data1 = np.array([[1, 2, 3], [4, 5, 6]]) # 2차원 배열(ndarray 객체)
4 df1= pd.DataFrame(data1)
5 print(df1, "₩n")
6
7 data2 = pd.Series({"England" : "London", "India" : "New Delhi",
8                     "USA" : "Washington", "Belgium" : "Brussels"})# Series객체를 사용
9 df2= pd.DataFrame(data2, columns = ["city"])
10 print(df2, "₩n")
11
12  df3  = pd.DataFrame(data = [56, 90,31],index = range(1,4), columns = ["score"] ) # DataFrame객체를 사용
13
14
15 print(df3, "₩n")
```

사전의 key 값이 index, value 값이 데이터 값인 Series 생성

DataFrame 객체로 변환. 열 이름은 “city”

```
   0  1  2
0  1  2  3
1  4  5  6

              city
England     London
India     New Delhi
USA      Washington
Belgium    Brussels

   score
1     56
2     90
3     31
```

DataFrame

- 데이터 참조 (변수 df를 DataFrame 객체라고 가정)
  - 열 값 참조

    df[열이름]
    df.열이름1
    df[[열이름1, 열이름2, ...]] **# 참조할 열이 2개 이상인 경우 리스트로 묶어 지정**
  - 슬라이싱으로 참조

    df[start : end] **# start 부터 end-1 행 까지 참조**
  - 행 참조

    df.loc[행이름]
    df.loc[[행이름1, 행이름2, ...]] # 참조할 행이 2개 이상인 경우 리스트로 묶어 지정
  - 하나의 원소만 참조

    df.loc[행이름, 열이름]
  - 부울린 인덱싱(boolean indexing)으로 조건에 맞는 값 참조

    df[조건식] # 조건식을 만족하는 데이터들만 참조

DataFrame 자료구조

- 데이터 참조 예제

```
import pandas as pd
names = ["Bob", "Jessica", "Mary", "John", "Kate"]
scores = [56, 11, 99, 83, 45]
members = {"Name": names, "Score" : scores}
df = pd.DataFrame(data = members )
cgpa = pd.Series([3.2, 2.5, 4.1, 2.8, 1.9])
df["Cgpa"] = cgpa
print(df, "₩n")
print(df[["Name", "Cgpa"]], "₩n")
print(df[1 : 4], "₩n")
print(df.loc[[2, 4]], "₩n")
print(df.loc[1, "Name"], "₩n")
print(df[df.Score > 60], "₩n")
df["Add"] = df.Score > 60
print(df, "₩n")
df = df.drop("Cgpa", axis = 1)
print(df, "₩n")
```

Series 객체 생성해서 열 추가

Score가 60 초과인 행만 추출

Score가 60 초과 여부를 값으로 하는 열(Add) 추가

Cgpa 열 삭제

```
      Name  Score  Cgpa
0      Bob     56   3.2
1  Jessica     11   2.5
2     Mary     99   4.1
3     John     83   2.8
4     Kate     45   1.9

      Name  Cgpa
0      Bob   3.2
1  Jessica   2.5
2     Mary   4.1
3     John   2.8
4     Kate   1.9

      Name  Score  Cgpa
1  Jessica     11   2.5
2     Mary     99   4.1
3     John     83   2.8

   Name  Score  Cgpa
2  Mary     99   4.1
4  Kate     45   1.9

Jessica

   Name  Score  Cgpa
2  Mary     99   4.1
3  John     83   2.8

      Name  Score  Cgpa    Add
0      Bob     56   3.2  False
1  Jessica     11   2.5  False
2     Mary     99   4.1   True
3     John     83   2.8   True
4     Kate     45   1.9  False

      Name  Score    Add
0      Bob     56  False
1  Jessica     11  False
2     Mary     99   True
3     John     83   True
4     Kate     45  False
```

DataFrame 자료구조

- 데이터 참조 예제

```
1 import pandas as pd
2 names = ["Bob", "Jessica", "Mary", "John", "Kate"]
3 scores = [56, 11, 99, 83, 45]
4 members = {"Name": names, "Score" : scores}
5 df = pd.DataFrame(data = members, index=range(1, 6) )
6 print(df, "₩n")
7 print(df.loc[2], "₩n")
8 print(df.iloc[2], "₩n")
9 print(df.Name)
```

```
      Name  Score
1      Bob     56
2  Jessica     11
3     Mary     99
4     John     83
5     Kate     45

Name     Jessica
Score         11
Name: 2, dtype: object

Name     Mary
Score      99
Name: 3, dtype: object

1        Bob
2    Jessica
3       Mary
4       John
5       Kate
Name: Name, dtype: object
```

df[“Name”] 같은 명령어

loc[n] : 실질적인 정해져 있는 index
iloc[n] : 0부터 시작하는 index

DataFrame

- 특정 컬럼의 값을 기준으로 전체 데이터를 정렬
  - df.sort_values(by = "column name") 함수 사용
  - by 인자로 지정한 열의 값 기준으로 정렬(디폴트는 오름차순 정렬이며 인자 ascending 값을 False로 하면 내림차순 정렬함)
  - 구한 DataFrame의 인덱스를 초기화한 새로운 객체를 얻기 위해서는 reset_index() 함수 사용(인자 drop 값을 True 으로 설정하면 오름 순 인덱스로 초기화)

```
1 import pandas as pd
2 names = ["Bob", "Jessica", "Mary", "John", "Kate"]
3 scores = [56, 11, 99, 83, 45]
4 members = {"Name": names, "Score" : scores}
5 df = pd.DataFrame(members)
6 df["Grade"] = ["C", "F", "A", "B", "D"] # 열 이름 "Grade"로 열 추가
7 df.loc[5] = ["Hanna", 90, "A"]   # 행 추가
8 print(df, "₩n")
9 df1 = df.sort_values(by = "Score", ascending=False)
10 print(df1, "₩n")
11 df2 = df1.reset_index(drop = True)
12 print(df2)
```

```
     Name  Score Grade
0     Bob     56     C
1 Jessica     11     F
2    Mary     99     A
3    John     83     B
4    Kate     45     D
5   Hanna     90     A
```

```
     Name  Score Grade
2    Mary     99     A
5   Hanna     90     A
3    John     83     B
0     Bob     56     C
4    Kate     45     D
1 Jessica     11     F
```

"Score" 열 기준으로 내림 차순으로 정렬한 객체 생성

```
     Name  Score Grade
0    Mary     99     A
1   Hanna     90     A
2    John     83     B
3     Bob     56     C
4    Kate     45     D
5 Jessica     11     F
```

df1 객체의 인덱스를 초기화한 새로운 객체 생성

DataFrame

- 객체에 새로운 열 추가
- 누락 값 처리
  - 대부분의 원본(raw) 데이터들은 부정확하거나 누락 값 (결측값)이 존재
  - 데이터를 제거하거나 대체하는 등의 정제 과정이 필요
  - 결측 데이터들은 null, NaN, NA등으로 표기

```
 1 import pandas as pd
 2 import numpy as np
 3 names = ["Bob", "Jessica", "Mary", "John", "Kate"]
 4 scores = [56, 11, 99, 83, 45]
 5 members = {"Name": names, "Score" : scores}
 6 df = pd.DataFrame(members)
 7 df["Grade"] = np.nan     # 데이터 값 지정없이 열만 추가
 8 df.loc[2,"Grade"] = "A"  # 기존 존재 행의 추가한 열에 데이터 저장
 9 df.loc[5,"Grade"] = "F"  # 새로운 행이 추가되면서 지정 열에 데이터 저장, 나머지 데이터는 Nan으로 초기화
10 print(df, "₩n")
```

import numpy as np 해야함 (np.nan을 사용하기 위해서)

```
      Name  Score Grade
0      Bob   56.0   NaN
1  Jessica   11.0   NaN
2     Mary   99.0     A
3     John   83.0   NaN
4     Kate   45.0   NaN
5      NaN    NaN     F
```

값이 정해지지 않은 열 "Grade"을 추가

DataFrame

- 누락 값 처리 예시

```
import pandas as pd
import numpy as np
a = np.array([[32, 77, np.nan, 41, 50],
              [11, np.nan, 9, np.nan,np.nan],
              [99, 88, 70, np.nan,100]])
df = pd.DataFrame(a)
print(df, "₩n")
df = df.dropna(axis = 1)
print(df)
```

```
      0     1     2     3      4
0  32.0  77.0   NaN  41.0   50.0
1  11.0   NaN   9.0   NaN    NaN
2  99.0  88.0  70.0   NaN  100.0

      0
0  32.0
1  11.0
2  99.0
```

결측치(Nan)가 존재하는 열 삭제

DataFrame

- 데이터 병합
  - 데이터가 여러 개로 분산되어 있거나 추가로 데이터를 합칠 경우
  - 데이터 병합으로 데이터 일관성 유지, 분석 결과에 대한 신뢰도를 얻을 수 있음
  - concat() 함수

    pd.concat([df1, df2], axis = 0(or 1), join = "outer"(or "inner"),
    ignore_index = False(or True))

    - 인자 axis가 0이면 행방향(세로방향)으로 병합
    - 인자 join의 값이 "outer"이면 열을 합집합으로, "inner" 이면 교집합으로 구함
    - 인자 ignore_index가 False 이면 원 DataFrame의 인덱스를 그대로 유지하면서 병합, 아니면 새로운 인덱스 부여하면서 병합

DataFrame

- 데이터 병합 (concat() 함수)

```
import pandas as pd
name1 = ["Bob", "Jessica", "Mary", "John", "Kate"]
score1 = [56, 11, 99, 83, 45]
score2 = [34, 60, 99, 78, 54]
class1 = {"Name": name1, "Quiz1" : score1, "Quiz2" : score2}
df1 = pd.DataFrame(class1)
print(df1, "₩n")
name2 = ["Jisu", "Elli", "June"]
score3 = [100, 99, 89]
score4 = [30, 60, 50]
score5 = [0, 10, 20]
class2 = {"Name": name2, "Quiz1" : score3, "Quiz2" : score4, "Quiz3" : score5}
df2 = pd.DataFrame(class2)
print(df2, "₩n")
df_inner = pd.concat([df1, df2], axis = 0, join = "inner")
print(df_inner, "₩n")
df_outer = pd.concat([df1, df2], axis = 0, ignore_index = True)
print(df_outer)
```

두개 DataFrame 모두에 포함된 열만 행방향으로 결합

두개 DataFrame에 있는 열들 전부, 행 방향, 인덱스를 다시 부여해서 결합

```
    Name  Quiz1  Quiz2
0    Bob     56     34
1  Jessica     11     60
2   Mary     99     99
3   John     83     78
4   Kate     45     54

   Name  Quiz1  Quiz2  Quiz3
0  Jisu    100     30      0
1  Elli     99     60     10
2  June     89     50     20

    Name  Quiz1  Quiz2
0    Bob     56     34
1  Jessica     11     60
2   Mary     99     99
3   John     83     78
4   Kate     45     54
0   Jisu    100     30
1   Elli     99     60
2   June     89     50

    Name  Quiz1  Quiz2  Quiz3
0    Bob     56     34    NaN
1  Jessica     11     60    NaN
2   Mary     99     99    NaN
3   John     83     78    NaN
4   Kate     45     54    NaN
5   Jisu    100     30    0.0
6   Elli     99     60   10.0
7   June     89     50   20.0
```

DataFrame

- 데이터 병합
  - merge() 함수 : 특정 열을 기준으로 합칠 경우

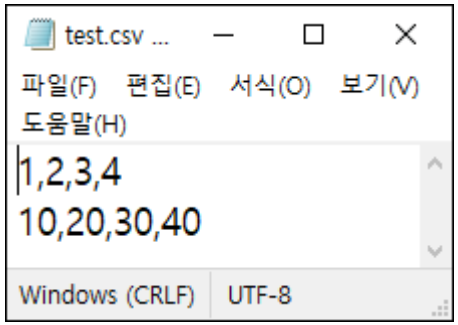
    pd.merge(df1, df2, axis = 0(or 1), on = "column name",
    how = "outer"(or "inner", or "left", or "right"))

    - 인자 on은 합칠 때 기준이 되는 열의 이름
    - 인자 how의 값이 "outer"이면 두 DataFrame의 기준 열에 있는 모든 데이터 사용, "inner"이면 기준 열에 공통으로 있는 데이터만 사용, "left"이면 df1의 기준 열에 있는 데이터만 사용, "right" 이면 df2의 기준 열에 있는 데이터만 사용

외부 데이터 파일 읽기(CSV 파일)

- pandas는 CSV 파일, 엑셀 파일, 텍스트 파일 등 다양한 외부 데이터를 읽고 쓸 수 있는 기능을 제공
- CSV 파일(확장자 csv)을 읽어 DataFrame 객체를 만들어 반환하는 함수
  - **read_csv("파일이름")**
  - CSV 파일은 쉼표(,)로 데이터를 구분하며, 레코드를 나타내는 행과 필드를 나타내는 열로 구성

```
import pandas as pd
pf = pd.read_csv("test.csv")
print(pf, "₩n")
pf1 = pd.read_csv("test.csv", header = None)
print(pf1)
```



출력

```
   1   2   3   4
0  10  20  30  40

   0   1   2   3
0  1   2   3   4
1  10  20  30  40
```

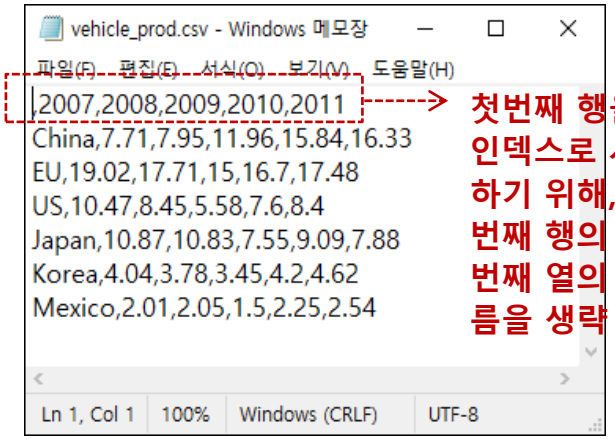
외부 데이터 파일 읽기 (CSV 파일)

- "vehicle_prod.csv" 파일 읽어 오기

```
import pandas as pd
import matplotlib.pyplot as plt
pf_f1 = pd.read_csv("vehicle_prod.csv")
print(pf_f1, "₩n")
pf_f2 = pd.read_csv("vehicle_prod.csv", index_col = 0)
print(pf_f2)
pf_f2["2007"].plot(kind = "bar")
plt.show()
```

0번째 열을 인덱스로 설정

0번째 열 "2007" 데이터 시각화



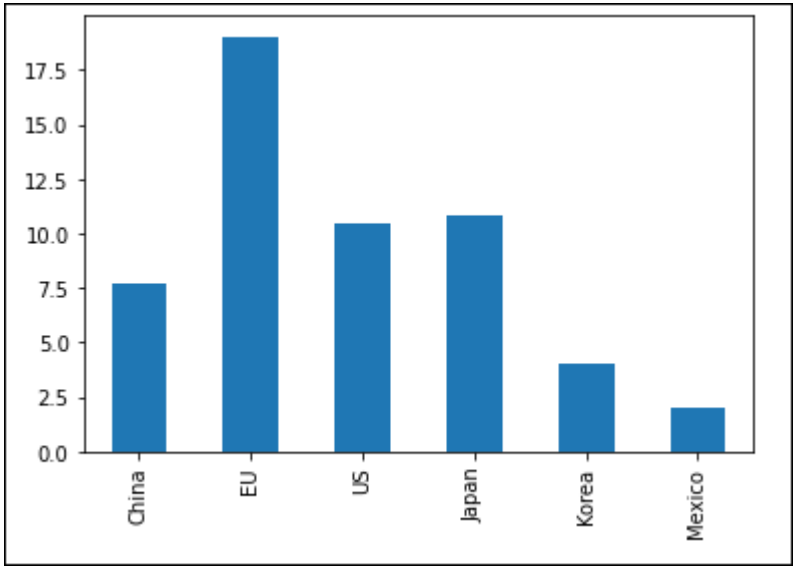
첫번째 행을 인덱스로 사용하기 위해, 첫 번째 행의 첫 번째 열의 이름을 생략

파일에서 비워 두었던 첫 줄의 첫 열

행 번호가 자동으로 부여

Unnamed: 0	2007	2008	2009	2010	2011
0	China	7.71	7.95	11.96	15.84	16.33
1	EU	19.02	17.71	15.00	16.70	17.48
2	US	10.47	8.45	5.58	7.60	8.40
3	Japan	10.87	10.83	7.55	9.09	7.88
4	Korea	4.04	3.78	3.45	4.20	4.62
5	Mexico	2.01	2.05	1.50	2.25	2.54

	2007	2008	2009	2010	2011
China	7.71	7.95	11.96	15.84	16.33
EU	19.02	17.71	15.00	16.70	17.48
US	10.47	8.45	5.58	7.60	8.40
Japan	10.87	10.83	7.55	9.09	7.88
Korea	4.04	3.78	3.45	4.20	4.62
Mexico	2.01	2.05	1.50	2.25	2.54

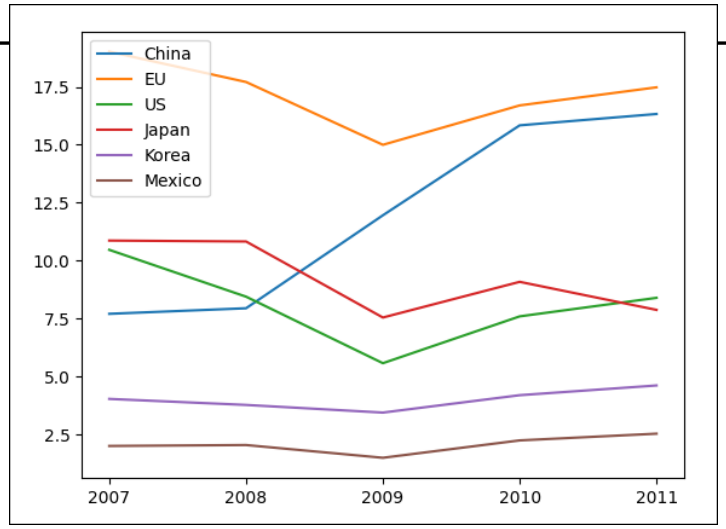


국가 이름이 있는 첫번째 열을 인덱스로 설정

외부 데이터 파일 읽기 (CSV 파일)

- "vehicle_prod.csv" 파일 읽어 국가 별로 년도 별 생산량 그래프 그리기

```
import pandas as pd
import matplotlib.pyplot as plt
pf_f = pd.read_csv("vehicle_prod.csv", index_col = 0)
plt.plot(pf_f.loc["China"])
plt.plot(pf_f.loc["EU"])
plt.plot(pf_f.loc["US"])
plt.plot(pf_f.loc["Japan"])
plt.plot(pf_f.loc["Korea"])
plt.plot(pf_f.loc["Mexico"])
plt.legend(["China", "EU", "US", "Japan", "Korea", "Mexico"])
plt.show()
```



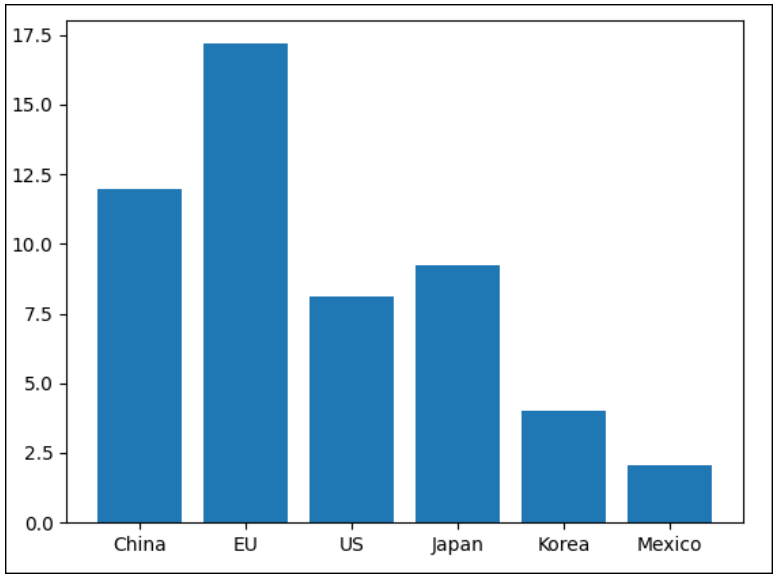
외부 데이터 파일 읽기 (CSV 파일)

- "vehicle_prod.csv" 파일 읽어 통계(합계, 평균)

```
import pandas as pd
import matplotlib.pyplot as plt
pf_f = pd.read_csv("vehicle_prod.csv", index_col = 0)
pf_f["Total"] = pf_f.sum(axis = 1)  axis=1은 열 방향, axis=0 행 방향을 의미
pf_f["Average"] = pf_f[["2007", "2008", "2009", "2010", "2011"]].mean(axis = 1)
print(pf_f, "₩n")
plt.bar(pf_f.index, pf_f["Average"])
plt.show()
```

	2007	2008	2009	2010	2011	Total	Average
China	7.71	7.95	11.96	15.84	16.33	59.79	11.958
EU	19.02	17.71	15.00	16.70	17.48	85.91	17.182
US	10.47	8.45	5.58	7.60	8.40	40.50	8.100
Japan	10.87	10.83	7.55	9.09	7.88	46.22	9.244
Korea	4.04	3.78	3.45	4.20	4.62	20.09	4.018
Mexico	2.01	2.05	1.50	2.25	2.54	10.35	2.070

열 추가



x축 인덱스(국가 코드), y축 평균