3. 판다스(Pandas) 데이터 분석

- 시리즈와 데이터프레임 클래스
- 시리즈와 데이터프레임의 데이터를 조작
- 데이터프레임의 병합
- Pandas를 이용한 csv 파일 처리
- 데이터를 입출력과 시각화

3.1 시리즈 클래스

- 시리즈 클래스(series)
  - 시리즈 = 값(value) + 인덱스(index)

- 생성
  - 변수= Pandas.Series([값1(value), 값2(value)…],
    index=[“라벨1”, “라벨2”, …]
  - 변수.index
  - 변수.values

- 인덱싱
  - 인덱스 라벨을 이용한 인덱싱과 슬라이싱
  - 자료의 순서변경 및 특정자료 선택
  - 문자열 라벨을 이용한 슬라이싱

- 시리즈와 딕셔너리 자료형
  - In연산
  - item 메서드
    - for k, v in s.items():
      print("%s = %d" % (k, v))

- 인덱스 기반 연산
  - 인덱스가 같은 데이터 연산
  - Series.notnull
    - NaN이 아닌값

- 데이터의 갱신,추가,삭제
  - 딕셔너리와 동일
  - 인덱스를 사용

3.2 데이터 프레임 클래스

- 데이터 프레임 클래스(DataFrame)
  - ➢행 인덱스
  - ➢열 인덱스

- 생성
  - ➢우선 하나의 열이 되는 데이터를 리스트나 일차원 배열로 준비한다.
  - ➢각각의 열에 대한 이름(라벨)을 키로 가지는 딕셔너리를 만든다.
  - ➢데이터를 DataFrame 클래스 생성자에 넣는다.
    열방향 인덱스= columns
    행방향 인덱스= index

- 열 데이터의 갱신, 추가, 삭제
  - ➢열 단위로 데이터를 갱신,추가,삭제

- 열 데이터의 인덱싱
  - ➢열 라벨을 키 값으로 하여 인덱싱
  - ➢하나의 열만 인덱싱: 시리즈
  - ➢여러 개의 열을 인덱싱: 데이터프레임

※열 인덱스가 문자열 라벨을 가지고 있는 경우에는 순서를 나타내는 정수 인덱스를 열 인덱싱에 사용할 수 없다.

- 행 데이터 인덱싱
  - ➢슬라이싱(slicing)

- 개별데이터인덱싱
  - ➢열라벨로 시리즈 인덱싱=>시리즈를 행 라벨로 인덱싱

Ex) [“열 라벨 "][" 행 라벨 "]

❖ 데이터 프레임 고급 인덱싱

- 2차원, 행 열 인덱싱
  - ➢loc: **라벨값** 인덱스 기준
  - ➢iloc: **행 번호** 기준 (반드시 숫자)

- loc 인덱서
  - ➢인덱싱 값이 하나

  loc[행 인덱싱값]
  - ➢인덱싱 값, 행과 열

  loc[행 인덱싱값, 열 인덱싱값]

인덱싱 값
- 인덱스 데이터
- 인덱스 데이터 슬라이스: 또는 리스트
- 같은 행 인덱스를 가지는 불리언 시리즈 또는 함수(행 인덱싱의 경우)

  Ex) df.loc[df.A > 10, ["C", "D"]]

※원래 행 인덱스값이 정수인 경우, 라벨 슬라이싱 방식을 따름(마지막 값 포함)

- iloc 인덱서
  - ➢순서를 나타내는 **정수(integer)** 인덱스 값
  - ➢인덱싱 값이 하나인 경우, 행 선택

❖ 데이터 프레임의 데이터 조작

- 데이터 개수 세기
  - ➢ Count()
- 카테고리 값 세기
  - ➢ value_counts()
  - ➢ 데이터프레임: 각 열마다 별도 적용
- 정렬
  - ➢ sort_index, sort_values
    - ascending=False :내림차순 정렬
  - ➢ 데이터프레임은 기준열을 지정
    - 변수.sort_values(by='기준열인덱스', 정렬순서)
    - 변수[열인덱스].sort_values()
- 행/열 합계
  - ➢ sum(axis인수)
    - axis=0 (생략가능) 열 방향 합계
    - axis=1 행 방향 합계
- apply변환
  - ➢ 행이나 열 단위의 복잡한 처리
  - ➢ 행이나 열을 받는 함수(lambda식)를 apply의 인수로 사용
    - Lambda 인자 : 표현식
    - If문의 경우 반환 값을 먼저 기술
    - If와 else만 지원하고 else if는 지원하지 않음
- fillna메서드
  - ➢ NaN값을 원하는 값으로 변환
- astype메서드
  - ➢ 전체 데이터의 자료형을 변경

❖ 데이터 프레임의 합성

- Merge함수
  - 공통 열 또는 인덱스를 기준으로 병합
  - 양 쪽에 모두 키가 존재하는 데이터 포함. (inner join)
  - 인수how='outer': 한쪽에만 키가 존재하는 데이터 포함. (outer join)
  - 인수how='left': 첫번째 DF의 키 값 기준
  - 인수how='right': 두번째 DF의 키 값 기준
  - 중복된 키 값이 존재하는 경우: 모든 경우의 수를 따져서 조합을 만듦
  - 기준 열 설정 인수 on
  - 기준 열의 이름이 다른 경우 left_on, right_on
  - 인덱스를 기준열로 사용 left_index 또는 right_index를 True로 설정

- Concat함수
  - 기준 열(key column)을 사용하지 않고 단순히 데이터를 연결
  - 위/아래로 데이터 행을 연결
  - 옆으로 데이터 열 연결: axis=1인수

3.3 데이터 입출력

- Pandas 지원 데이터 포맷
  - txt, CSV, Excel, HTML, JSON, SQL 등

- CSV파일 입력
  - pandas.read_csv
    - names인수 : 열 인덱스 정보 추가
    - Index_col: 특정열을 행 index로 지정
    - Sep인수:구분자, default는 쉼표.
      가변공백인 경우 sep='\s+'
      sep='\t', sep=';'
    - delimiter:구분자, default는 None
    - skiprows: 행을 skip
    - na_values: 특정값을 NaN값으로 치환
    - skip_blank_lines: 빈 줄을 skip
    - nrows:읽을 행의 개수

- CSV파일 출력
  - to_csv
    - na_rep: NaN 표시값 변경
    - Index, header:인덱스 및 헤더 출력여부

- 인터넷 상의 CSV파일 입력
  - read_csv(url)
  - pandas_datareader패키지
    - 데이터 제공 사이트 : FRED, World Bank, OECD 등등
    - https://pandas-datareader.readthedocs.io/en/latest/index.html
    - pip install pandas-datareader
    - Ex) FRED데이터 베이스
      미국 국가총생산(GDP)
      소비자 가격 지수(CPIAUCSL)
    - pip install -U finance-datareader: 국내 주식데이타 실습
    - https://financedata.github.io/posts/finance-data-reader-users-guide.html

3.4 데이터 시각화

- 시각화 패키지 맷플롯리브 (Matplotlib)
  - 라인 플롯(line plot) , 바 차트(bar chart),히스토그램(histogram),박스 플롯(box plot)
  - 주피터노트북의 경우 매직 명령
    %matplotlib inline
    import matplotlib as mpl

- Pyplot 서브 패키지
  - 매트랩(matlab)의 시각화 명령을 그대로 사용 가능
  - import matplotlib.pyplot as plt

- 라인플롯
  - 선을 그리는 라인 플롯
  - 시간, 순서 등에 다른 변화를 표시
  - title :제목표시
  - show: 시각화 명령을 차트로 렌더링

- 스타일 지정
  - https://matplotlib.org/stable/api/matplotlib_configuration_api.html
  - 문자열 인수를 사용
  - 색깔, 마커, 선 종류의 순서로 지정
  - 색깔
    색 이름 혹은 약자(blue b, green g 등등), #RGB코드
  - 마커: 데이터 위치를 나타내는 기호
  - 선 종류
    - - solid line style
    - -- dashed line style
    - -. dash-dot line style
    - : dotted line style
  - 기타 스타일

- Color c (선 색깔) , Linewidth lw (선 굵기) , Linestyle ls (선 스타일)
- Marker (마커 종류), markersize Ms(마커 크기)
- Markeredgecolor Mec (마커 선 색깔), markeredgewidth mew(마커 선 굵기)
- Markerfacecolor mfc (마커 내부 색깔)

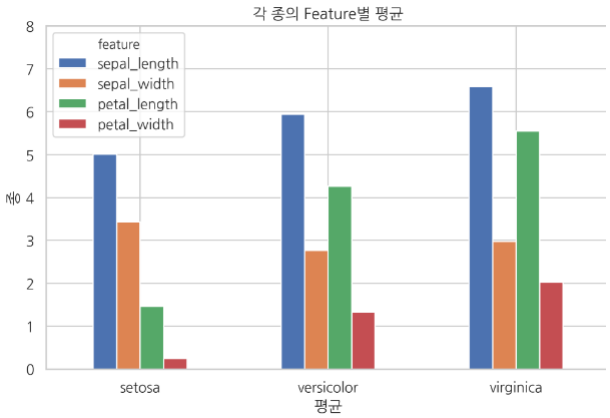
3.5 Pandas의 시각화 기능

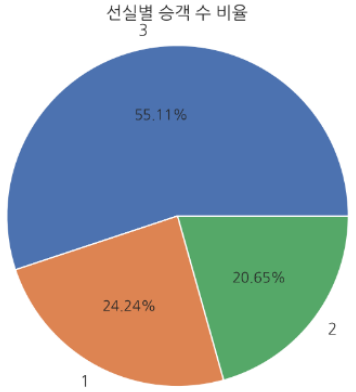
- plot 메서드

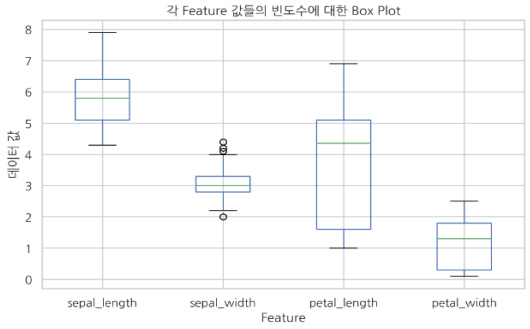
➢ matplotlib를 내부에서 임포트하여 사용

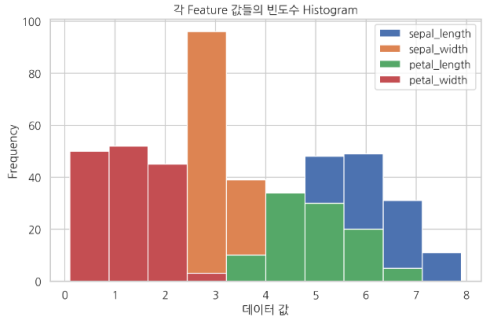
➢ kind인수: 플롯의 종류 설정

Ex) Plot(kind='bar', rot=0) or plot.bar(rot=0)









피봇테이블과 그룹분석

- 피봇테이블

  피봇테이블(pivot table)이란 데이터 열 중에서 두 개의 열을 각각 행 인덱스, 열 인덱스로 사용하여 데이터를 조회하여 펼쳐놓은 것

  pivot 메서드

```
data = {
   "도시": ["서울", "서울", "서울", "부산", "부산", "부산", "인천", "인천"],
   "연도": ["2015", "2010", "2005", "2015", "2010", "2005", "2015", "2010"],
   "인구": [9904312, 9631482, 9762546, 3448737, 3393191, 3512547, 2890451, 263203],
   "지역": ["수도권", "수도권", "수도권", "경상권", "경상권", "경상권", "수도권", "수도권"]
}
columns = ["도시", "연도", "인구", "지역"]
df1 = pd.DataFrame(data, columns=columns)


df1.pivot("도시", "연도", "인구")
df1.set_index(["도시", "연도"])[["인구"]].unstack()
```

다중 인덱스 피봇 테이블

행 인덱스나 열 인덱스를 리스트로 주는 경우

df1.pivot(["지역", "도시"], "연도", "인구")

- 그룹분석

groupby 메서드

열 또는 열의 리스트

행 인덱스

그룹연산 메서드

- size, count: 그룹 데이터의 갯수
- mean, median, min, max: 그룹 데이터의 평균, 중앙값, 최소, 최대
- sum, prod, std, var, quantile : 그룹 데이터의 합계, 곱, 표준편차, 분산, 사분위수
- first, last: 그룹 데이터 중 가장 첫번째 데이터와 가장 나중 데이터

```
np.random.seed(0)

df2 = pd.DataFrame({

   'key1': ['A', 'A', 'B', 'B', 'A'],

   'key2': ['one', 'two', 'one', 'two', 'one'],

   'data1': [1, 2, 3, 4, 5],

   'data2': [10, 20, 30, 40, 50]

})

groups = df2.groupby(df2.key1)

groups.groups

groups.sum()

df2.data1.groupby(df2.key1).sum()
```

```
df2.groupby(df2.key1)["data1"].sum()   # `GroupBy` 클래스 객체에서 data1만 선택하여 분석하는 경우
df2.groupby(df2.key1).sum()["data1"]  # 전체 데이터를 분석한 후 data1만 선택한 경우

 Unstack을 이용한 피봇테이블 형태
df2.data1.groupby([df2["key1"], df2["key2"]]).sum().unstack("key2")

import seaborn as sns
iris = sns.load_dataset("iris")

def peak_to_peak_ratio(x):
    return x.max() / x.min()
iris.groupby(iris.species).agg(peak_to_peak_ratio)
iris.groupby(iris.species).describe().T
```

❖ 데이터 프레임의 인덱스 조작 - 보류

- 데이터프레임 인덱스 설정 및 제거
  - ➢ set_index : 특정한 열을 인덱스로 설정
  - ➢ reset_index : 인덱스를 데이터 열로 추가
    - Drop인수=true로 설정 시 인덱스 열로 사용안함.
- 다중 인덱스(열,행)

```
np.random.seed(0)
df4 = pd.DataFrame(np.round(np.random.randn(6, 4), 2),
                   columns=[["A", "A", "B", "B"],
                            ["C", "D", "C", "D"]],
                   index=[["M", "M", "M", "F", "F", "F"],
                          ["id_" + str(i + 1) for i in range(3)] * 2])
```

df4.columns.names = ["Cidx1", "Cidx2"] 다중 열 인덱스 이름 부여

df4.index.names = ["Ridx1", "Ridx2"] 다중 행 인덱스 이름 부여

- **행 인덱스와 열 인덱스 교환**

Stack(열인덱스 or '열이름') 열 인덱스 -> 행 인덱스로 변환

df4.stack("Cidx1") , df4.stack(1)

Unstack 행 인덱스 -> 열 인덱스로 변환

df4.unstack("Ridx2"), df4.unstack(0)

- **다중 인덱스가 있는 경우의 인덱싱**

df3[("B", "C1")]

df3.loc[0, ("B", "C1")]

iloc 인덱서를 사용하는 경우에는 튜플 형태의 다중인덱스를 사용할 수 없다.

df3.iloc[0, 2]

df4.loc[("M", "id_1"), ("A", "C")]

df4.loc[:, ("A", "C")]

df4.loc[("M", "id_1"), :]

다중 인덱스의 튜플 내에서는 : 슬라이스 기호를 사용할 수 없고 대신 slice(None) 값을 사용

df4.loc[("M", slice(None)), :] == df4.loc["M"]

df4.loc[(slice(None), "id_1"), :]

- **다중 인덱스의 인덱스 순서 교환**

swaplevel(i, j, axis)

- **다중 인덱스가 있는 경우의 정렬**

df5.sort_index(level=0)

df6.sort_index(axis=1, level=0)