

데이터마이닝

- 데이터 시각화 -



2023.03.20 (MON)

동덕여자대학교 HCI사이언스

데이터 시각화

- 예측 분석 관련 데이터 시각화
- 본격적인 분석 전에 수행하는 데이터 탐색에서 시각적 탐색에는 데이터 구조 이해, 데이터 수정(예상치 못한 격차 또는 오류 값 식별), 이상치 식별, 기본 패턴 발견(변수 간의 상관관계, 클러스터), 흥미로운 질문을 도출하기 위한 자유형 탐색 방법 있음.
- 매우 기초적인 차트를 만드는 것부터 필터링하거나 확대하는 작업을 사용하여 서로 연결된 차트들을 탐색하는 데 이르기까지 아주 폭넓음

파이썬 패키지 불러오기

필요한 파이썬 라이브러리 불러오기

```
import os
import calendar
import numpy as np
import networkx as nx
import pandas as pd
from pandas.plotting import scatter_matrix, parallel_coordinates
import seaborn as sns
from sklearn import preprocessing
import matplotlib.pyplot as plt
```

예제 데이터와 파이썬 코드

■ 보스턴 주택 데이터(BostonHousing.csv)

- <https://github.com/reisanar/datasets/blob/master/BostonHousing.csv>

■ 앰트랙 기차 이용 승객(Amtrak.csv)

- <https://github.com/reisanar/datasets/blob/master/Amtrak.csv>

예제 1: 보스턴 주택 데이터

보스턴 주택 데이터셋의 변수 설명

CRIM	범죄율
ZN	25,000제곱피트 이상의 부지에 대해 계획된 주거용 토지의 비율
INDUS	비소매업이 차지하는 토지 비율
CHAS	찰스강 인접 여부(1=인접, 0=비인접)
nox	10ppm당 일산화질소
RM	주택의 평균 방 개수
AGE	1940년 이전에 건축된 주택에 사는 비율
DIS	보스턴 5대 상업 지구와의 거리
RAD	고속도로 진입 용이성 정도
TAX	재산세율(10,000달러당)
PTRATIO	시 ^{town} 별 학생 대 교사 비율
LSTAT	저소득층 비율
MEDV	주택 가격의 중앙값(단위: 1,000달러)
CAT.MEDV	주택 가격의 중앙값이 3만 달러 이상인지 여부(1=이상, 0=미만)

예제 1: 보스턴 주택 데이터

```
import os
import calendar
from pathlib import Path
import numpy as np
import networkx as nx
import pandas as pd
from pandas.plotting import scatter_matrix, parallel_coordinates
import seaborn as sns
from sklearn import preprocessing
import matplotlib.pyplot as plt
import dmba
```

```
%matplotlib inline
```

```
#보스턴 주택 데이터 불러오기
housing_df = pd.read_csv('BostonHousing.csv')
#housing_df = dmba.load_data('BostonHousing.csv')
print(housing_df.columns)
housing_df.head(10)
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
      'PTRATIO', 'LSTAT', 'MEDV', 'CAT_MEDV'],
      dtype='object')
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV	CAT_MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	0
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	1
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	5.21	28.7	0
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	12.43	22.9	0
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	19.15	27.1	0
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311	15.2	29.93	16.5	0
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	17.10	18.9	0

기본 차트: 막대그래프, 선그래프, 산점도

■ 선그래프

- 주로 시계열을 보여주는 데 사용
- 그래프를 그리기 위한 시간 프레임의 크기는 시간 척도와 마찬가지로 예측 작업의 규모와 데이터의 속성에 따라 달라짐

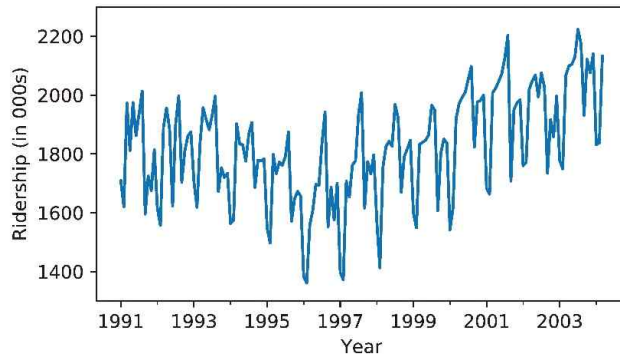
■ 막대그래프

- 평균, 개수, 비율과 같은 단일 통계값의 그룹별 비교에 유용
- 막대 높이는 통계값을 나타내고, 각 막대는 그룹을 표시하며, 각 막대의 높이(수평 막대의 경우에는 길이)는 변수 값을 나타냄

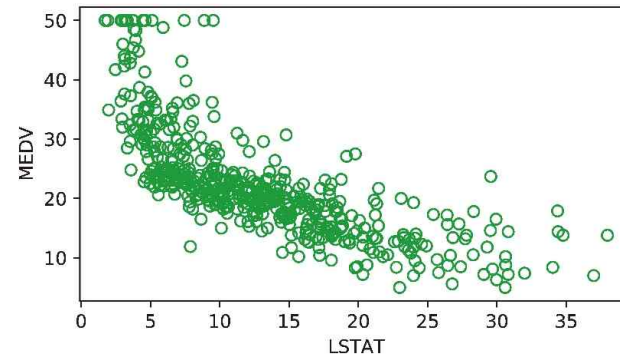
■ 산점도

- 수치형 변수 간의 관계를 보여주는 데 사용
- 비지도 학습에서 두 가지 수치형 변수 간의 정보 중복이나 군집 발견과 같은 연관성을 밝히는 데 도움이 됨

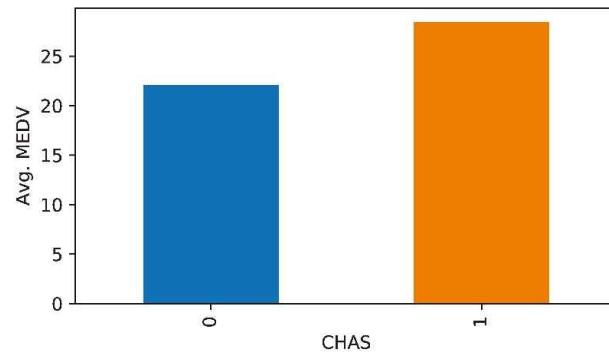
기본 차트: 막대그래프, 선그래프, 산점도



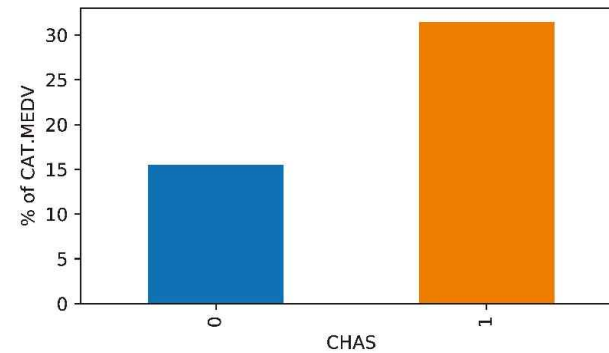
(a) 선그래프



(b) 산점도



(c) 수치형 변수에 대한 막대그래프



(d) 범주형 변수에 대한 막대그래프

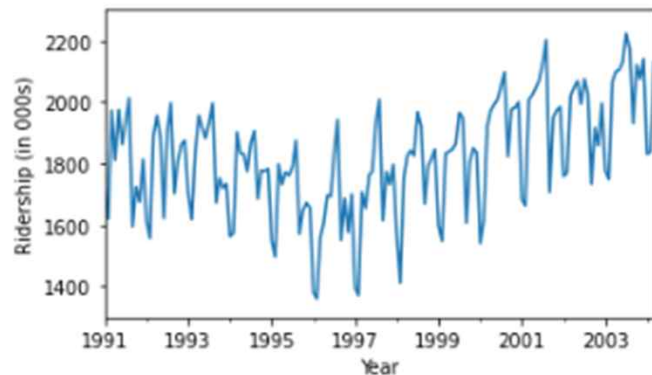
기본 차트: 막대그래프, 선그래프, 산점도

```
# Amtrak 데이터를 로드하고 시계열 분석에 적합하도록 변환
Amtrak_df = dmba.load_data('Amtrak.csv')
Amtrak_df['Date'] = pd.to_datetime(Amtrak_df.Month, format='%d/%m/%Y')
ridership_ts = pd.Series(Amtrak_df.Ridership.values, index=Amtrak_df.Date)
```

판다스 코드

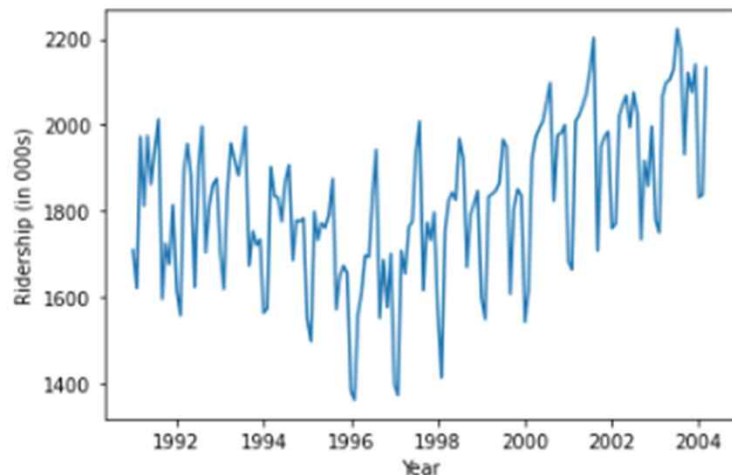
```
ridership_ts.plot(ylim=[1300, 2300], legend=False, figsize=[5, 3])
plt.xlabel('Year') # x-axis label 설정
plt.ylabel('Ridership (in 000s)') # y-axis label 설정

plt.tight_layout()
plt.show()
```



맷플롯립 코드

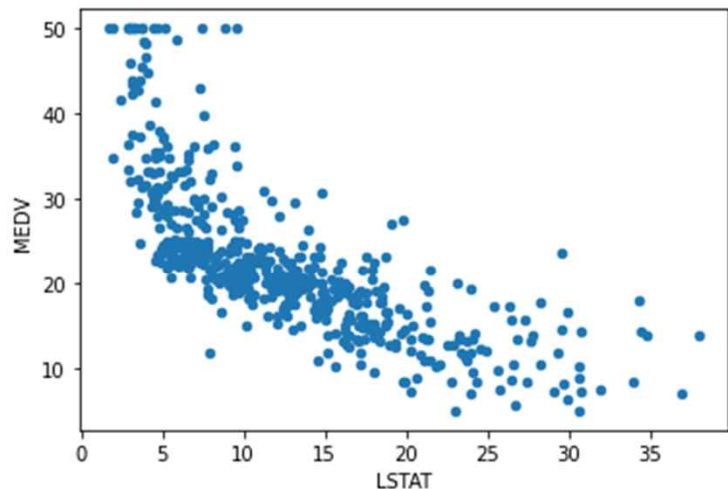
```
plt.plot(ridership_ts.index, ridership_ts)
plt.xlabel('Year') # x-axis label 설정
plt.ylabel('Ridership (in 000s)') # y-axis label 설정
plt.show()
```



기본 차트: 막대그래프, 선그래프, 산점도

```
#보스턴 주택 데이터 불러오기
housing_df = pd.read_csv('BostonHousing.csv')
#housing_df = dmba.load_data('BostonHousing.csv')
print(housing_df.columns)
housing_df.head(10)
```

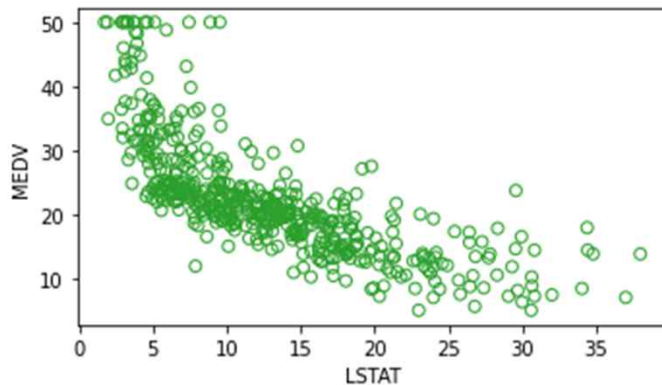
```
housing_df.plot.scatter(x='LSTAT', y='MEDV', legend=False)
plt.show()
```



```
# 산점도에서 점의 색상을 설정하고 열린 원으로 그리고자 할
fig, ax = plt.subplots()
fig.set_size_inches(5, 3)
```

```
ax.scatter(housing_df.LSTAT, housing_df.MEDV, color='C2', facecolor='none')
plt.xlabel('LSTAT')
plt.ylabel('MEDV')

plt.tight_layout()
plt.show()
```

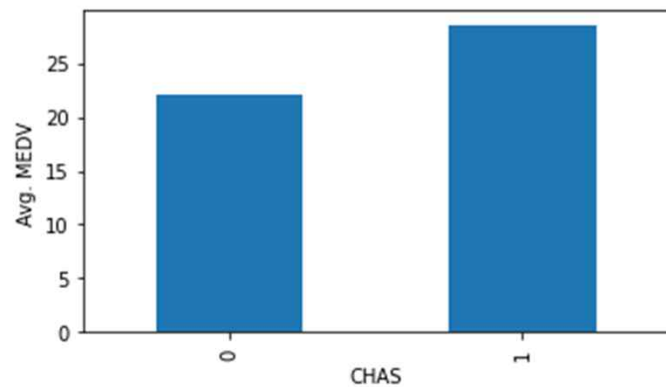


기본 차트: 막대그래프, 선그래프, 산점도

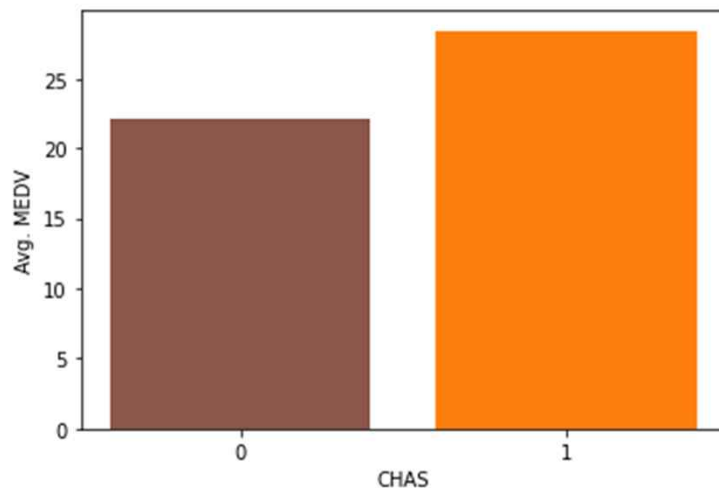
```
#보스턴 주택 데이터 불러오기
housing_df = pd.read_csv('BostonHousing.csv')
#housing_df = dmdb.load_data('BostonHousing.csv')
print(housing_df.columns)
housing_df.head(10)
```

```
ax = housing_df.groupby('CHAS').mean().MEDV.plot(kind='bar', figsize=[5, 3])
ax.set_ylabel('Avg. MEDV')

plt.tight_layout()
plt.show()
```



```
dataForPlot = housing_df.groupby('CHAS').mean().MEDV
fig, ax = plt.subplots()
ax.bar(dataForPlot.index, dataForPlot, color=['C5', 'C1'])
ax.set_xticks([0, 1])
ax.set_xlabel('CHAS')
ax.set_ylabel('Avg. MEDV')
plt.show()
```



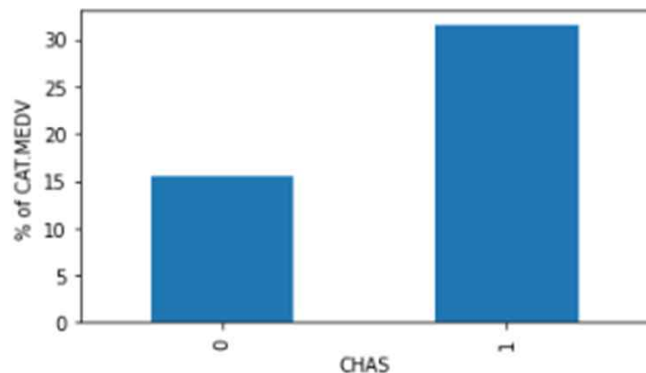
기본 차트: 막대그래프, 선그래프, 산점도

```
#보스턴 주택 데이터 불러오기
housing_df = pd.read_csv('BostonHousing.csv')
#housing_df = dmdb.load_data('BostonHousing.csv')
print(housing_df.columns)
housing_df.head(10)
```

```
# barchart CHAS vs. CAT_MEDV
```

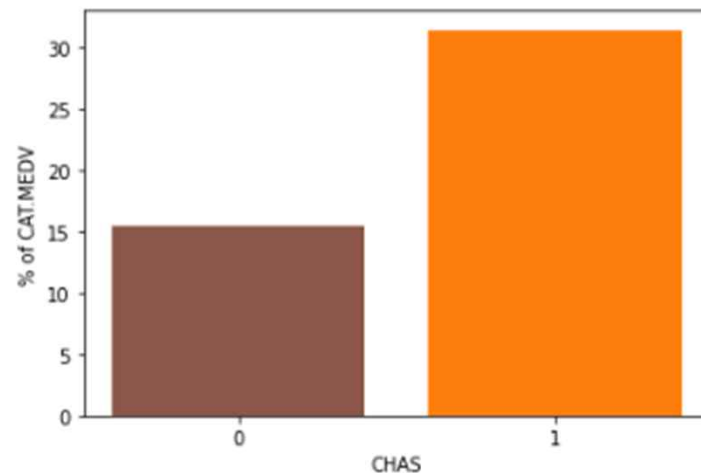
```
dataForPlot = housing_df.groupby('CHAS').mean()['CAT_MEDV'] * 100
ax = dataForPlot.plot(kind='bar', figsize=[5, 3])
ax.set_ylabel('% of CAT_MEDV')

plt.tight_layout()
plt.show()
```



```
fig, ax = plt.subplots()
ax.bar(dataForPlot.index, dataForPlot, color=['C5', 'C1'])
ax.set_xticks([0, 1])
ax.set_xlabel('CHAS')
ax.set_ylabel('% of CAT_MEDV')
```

```
Text(0, 0.5, '% of CAT_MEDV')
```



분포도: 박스 플롯과 히스토그램

■ 분포도

- 수치형 변수의 전반적인 분포를 표시
- 데이터 마이닝 방법과 변수 변환을 결정하기 위한 지도 학습에 유용

■ 박스 플롯

- 나란히 생성해 하위 그룹끼리 비교하거나, 여러 개의 박스 플롯을 시간 별로 생성함으로써 시간 변화에 따른 분포를 관찰할 수 있음

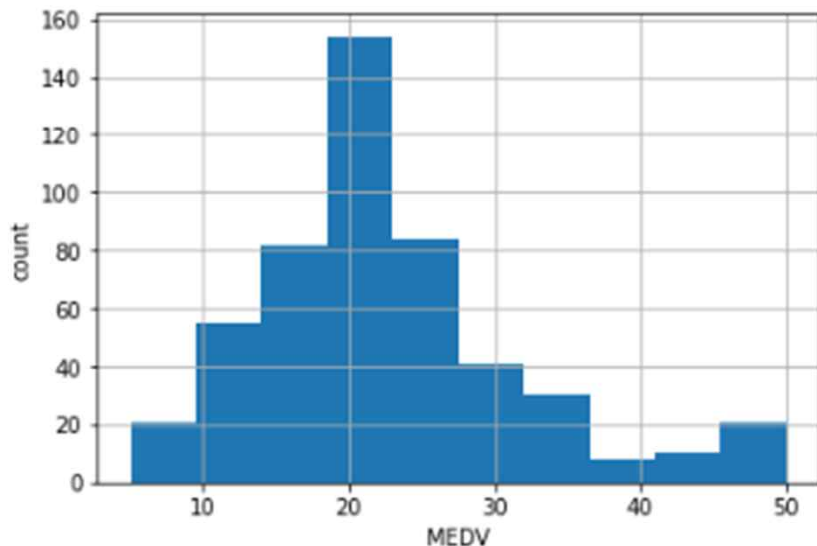
■ 히스토그램

- 일련의 수직 연결된 막대로 모든 x값의 빈도를 나타냄

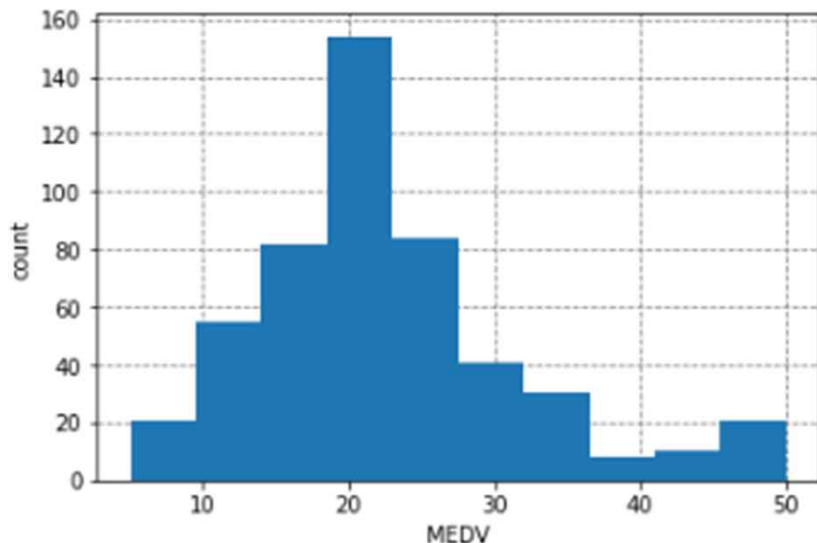
분포도: 박스 플롯과 히스토그램

MEDV에 대한 히스토그램

```
ax = housing_df.MEDV.hist()  
ax.set_xlabel('MEDV')  
ax.set_ylabel('count')  
  
plt.show()
```



```
fig, ax = plt.subplots()  
ax.hist(housing_df.MEDV)  
ax.set_axisbelow(True) # 히스토그램 뒤에 격자선 표시  
ax.grid(which='major', color='grey', linestyle='--')  
ax.set_xlabel('MEDV')  
ax.set_ylabel('count')  
plt.show()
```

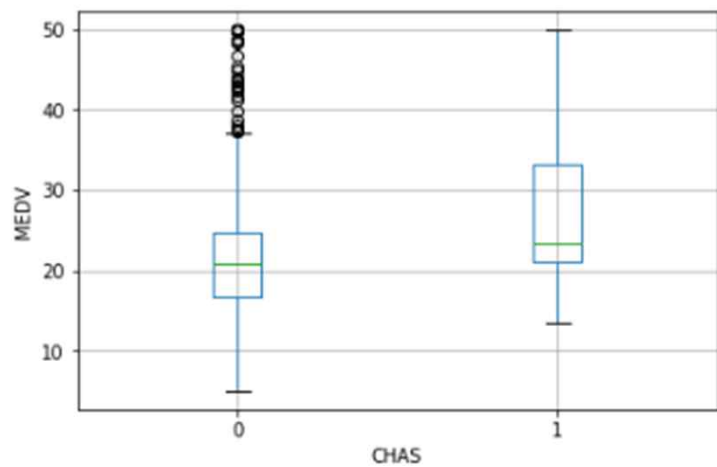


분포도: 박스 플롯과 히스토그램

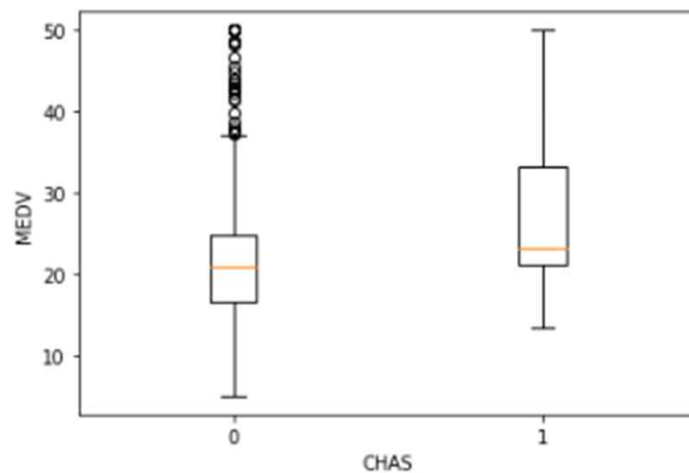
```
# CHAS 별 MEDV의 박스 플롯
```

```
ax = housing_df.boxplot(column='MEDV', by='CHAS')
ax.set_ylabel('MEDV')
plt.suptitle('') # 제목 없음
plt.title('')

plt.show()
```



```
dataForPlot = [list(housing_df[housing_df.CHAS==0].MEDV),
               list(housing_df[housing_df.CHAS==1].MEDV)]
fig, ax = plt.subplots()
ax.boxplot(dataForPlot)
ax.set_xticks([1, 2])
ax.set_xticklabels([0, 1])
ax.set_xlabel('CHAS')
ax.set_ylabel('MEDV')
plt.show()
```



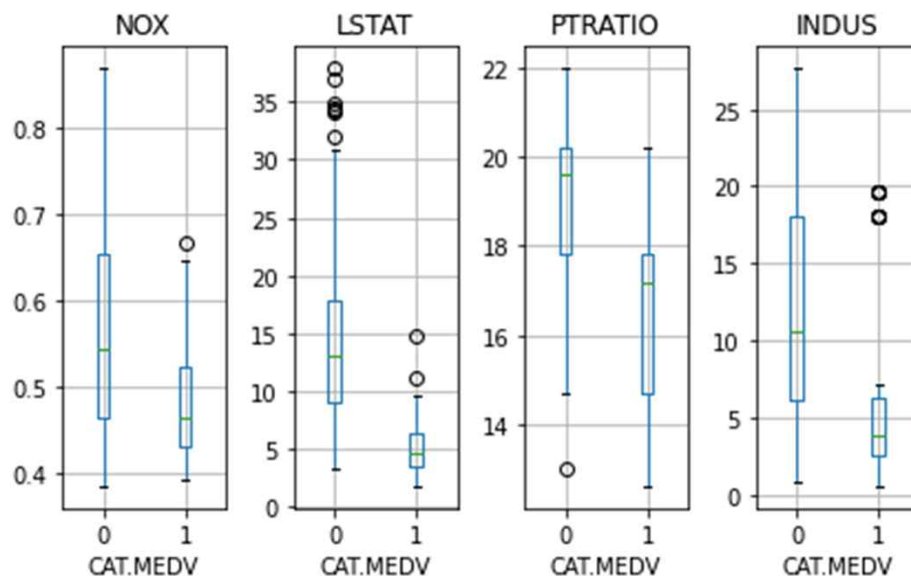
분포도: 박스 플롯과 히스토그램

■ 다양한 수치형 예측 변수로 결과 변수 CAT.MEDV를 탐색하기 위한 병렬 박스 플롯

- 각 병렬 박스 플롯에서 한 축은 범주형 변수 사용. 다른 축은 수치형 변수를 사용
- 범주형 결과 변수와 수치형 예측 변수를 함께 그리면 예측 변수의 분포가 각 결과 범주마다 비교
- 수치형 결과 변수와 범주형 예측 변수를 함께 그리면 결과 변수의 분포가 예측 변수의 다양한 수준에 걸쳐 나타남

CAT.MEDV 별 박스 플롯

```
fig, axes = plt.subplots(nrows=1, ncols=4)
housing_df.boxplot(column='NOX', by='CAT_MEDV', ax=axes[0])
housing_df.boxplot(column='LSTAT', by='CAT_MEDV', ax=axes[1])
housing_df.boxplot(column='PTRATIO', by='CAT_MEDV', ax=axes[2])
housing_df.boxplot(column='INDUS', by='CAT_MEDV', ax=axes[3])
for ax in axes:
    ax.set_xlabel('CAT.MEDV')
plt.suptitle('') # 제목 숨기기
plt.tight_layout() # 그래프 사이 간격 늘리기
plt.show()
```



히트맵: 상관관계와 결측치 시각화

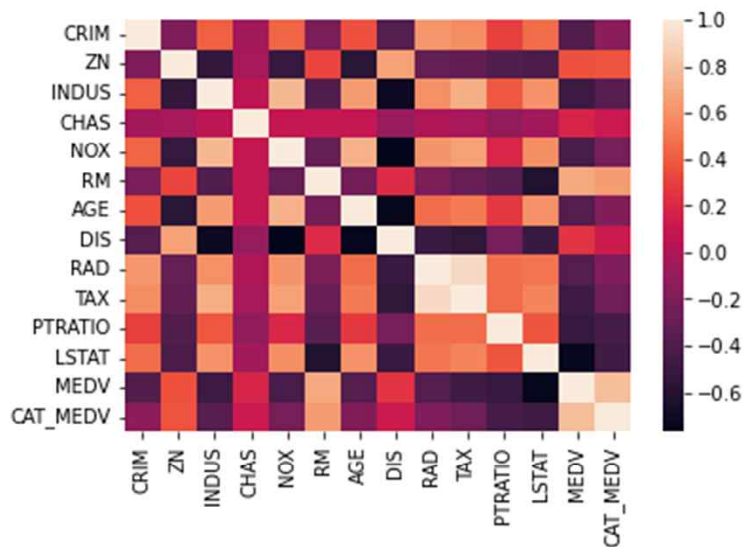
■ 히트맵

- 수치형 데이터를 그래픽으로 나타내는 차트이며, 값을 색상으로 표시
- 상관관계표의 시각화와 결측치의 시각화에 특히 유용
- p 개의 변수에 대한 상관 관계표는 p 행 \times p 열이고 값보다는 색상을 살펴보는 것이 더 쉽고 빠름
- 큰 숫자 값을 검토하는 데 유용하지만 색상 차이를 정확하게 인지하기 어려워 막대그래프와 같은 정확한 그래픽 시각화를 대체하진 못함

히트맵: 상관관계와 결측치 시각화

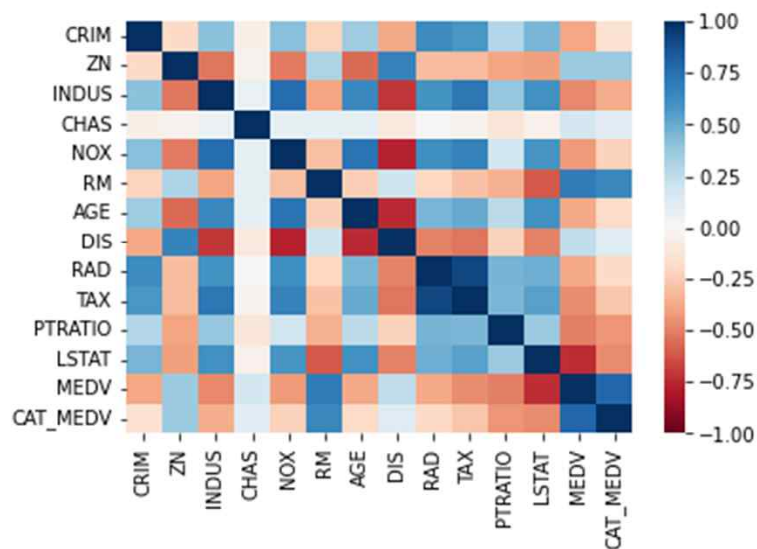
상관 관계에 대한 단순 히트맵

```
corr = housing_df.corr()  
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns)  
plt.show()
```



컬러맵을 *divergent scale*로 변경하고 컬러맵의 범위를 고정

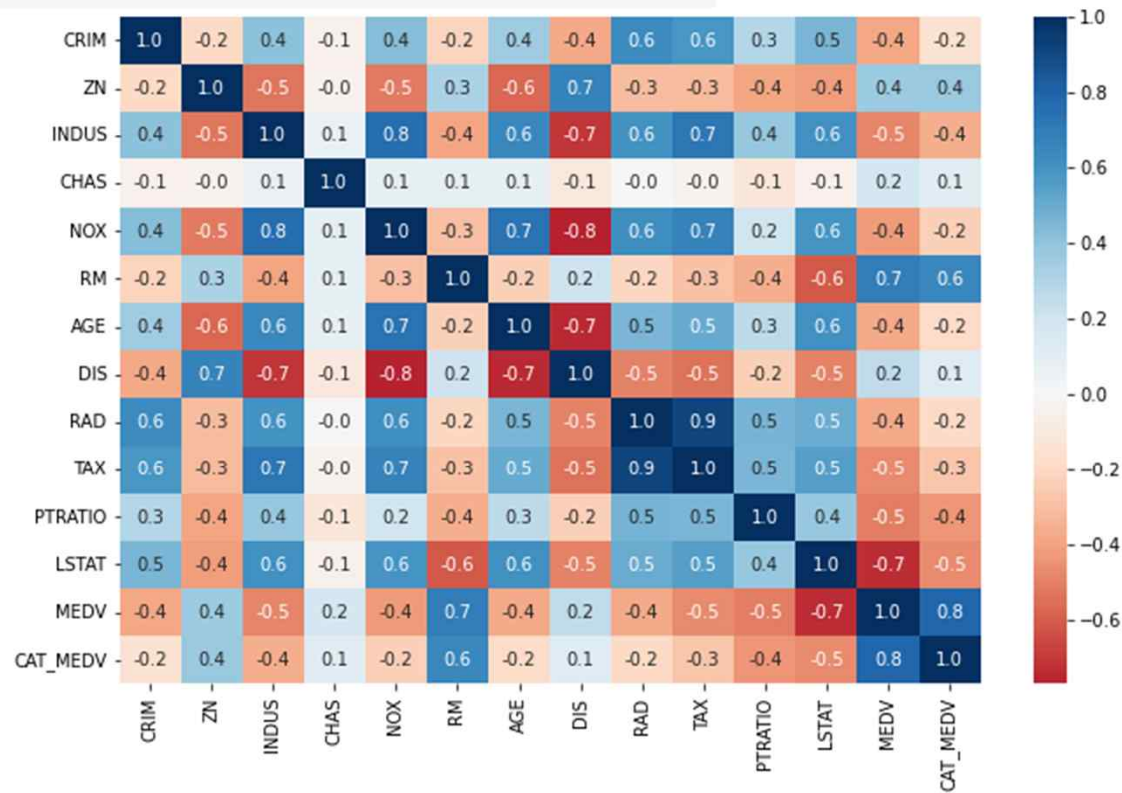
```
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, vmin=-1, vmax=1, cmap="RdBu")  
plt.show()
```



히트맵: 상관관계와 결측치 시각화

```
# 상관계수 값을 반영
fig, ax = plt.subplots()
fig.set_size_inches(11, 7)
sns.heatmap(corr, annot=True, fmt=".1f", cmap="RdBu", center=0, ax=ax)

plt.show()
```



히트맵: 상관관계와 결측치 시각화

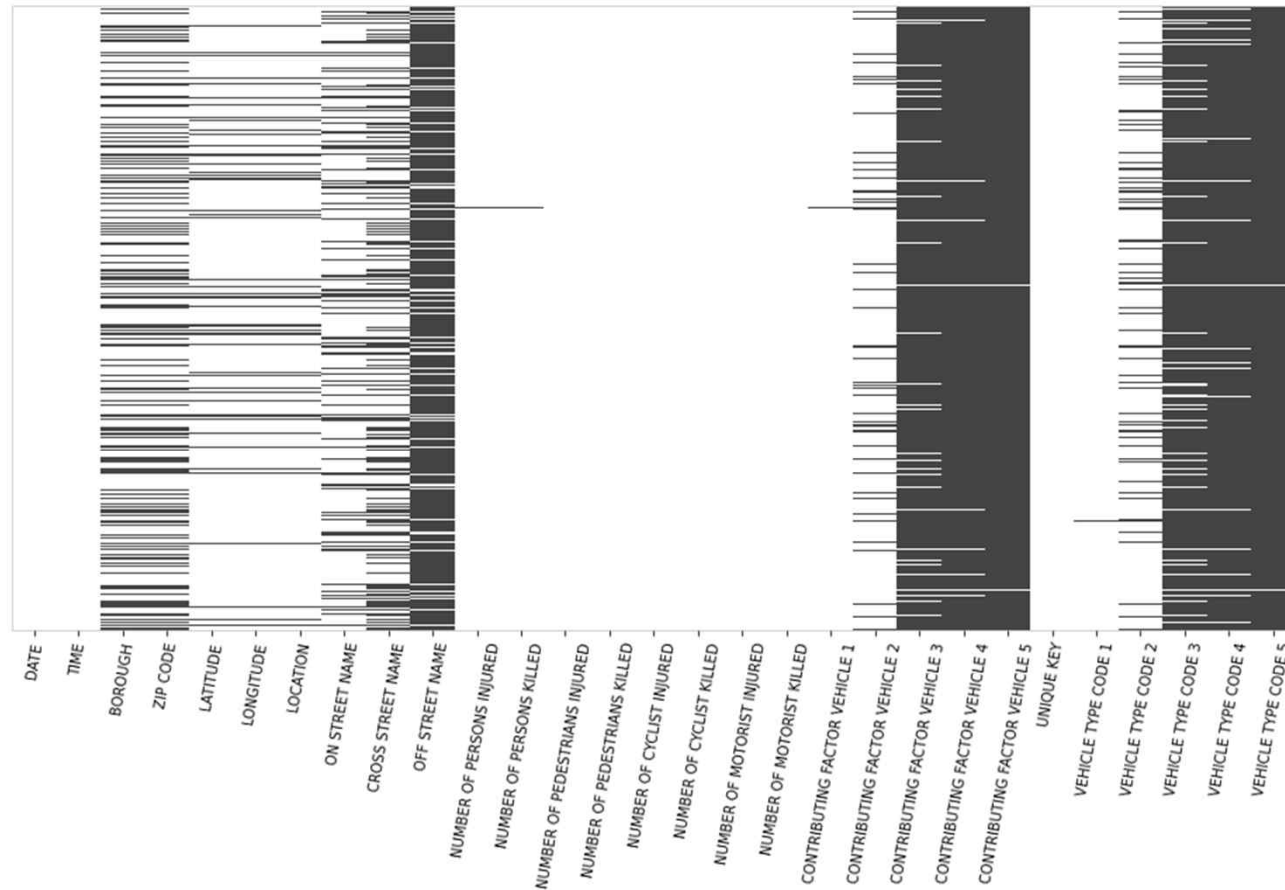
■ EX

- 히트맵을 사용하여 누락된 값을 시각화 해보자.
- Data set (NYPD Motor Vehicle Collisions)



히트맵: 상관관계와 결측치 시각화

- 자동차 충돌에 관한 데이터셋의 결측치 히트맵. 회색은 결측치 표시



색상, 크기, 모양, 다중 패널, 애니메이션

■ 차트 속성 변수 추가

- 차트에 더 많은 변수를 포함하려면 먼저 변수 형태를 고려해야 함
- 표시가 너무 많으면 어수선해 인지하기가 어려우므로 변수를 추가할 때는 신중해야 함
- 범주형 정보를 추가하려면 색조, 모양, 다중 패널을 사용하는 것이 좋음
- 수치 정보를 추가하려면 색상 강도를 조절하거나 크기 변화
- 시간 정보가 어떻게 변화하는지는 차트에 시간 차원을 추가해 애니메이션을 통해 보여주는 것이 효과적(예> www.gapminder.org).

색상, 크기, 모양, 다중 패널, 애니메이션

■ 색상

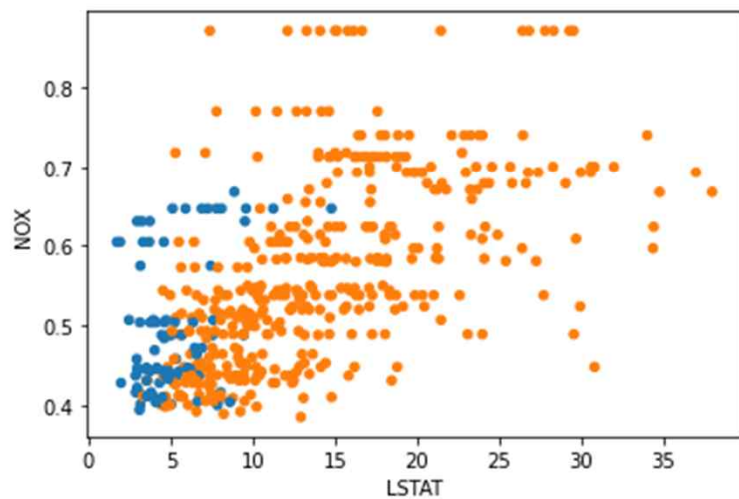
- 색상 코드는 수치 결과 변수(y축)와 수치 예측 변수 사이의 관계를 탐색하는 데 도움을 줌
- 색상-코드화된 산점도는 상호작용하는 조건을 만들 필요가 있는지 평가하는 데 도움이 됨
- 범주 수가 적은 경우 색상을 사용해 범주형 변수를 막대그래프에 추가. 범주 수가 많을 때는 다중 패널 사용. 다중 패널에서는 범주형 변수에 따라 관측 데이터를 분리하고, 별도의 차트로 작성

■ 산점도 매트릭스

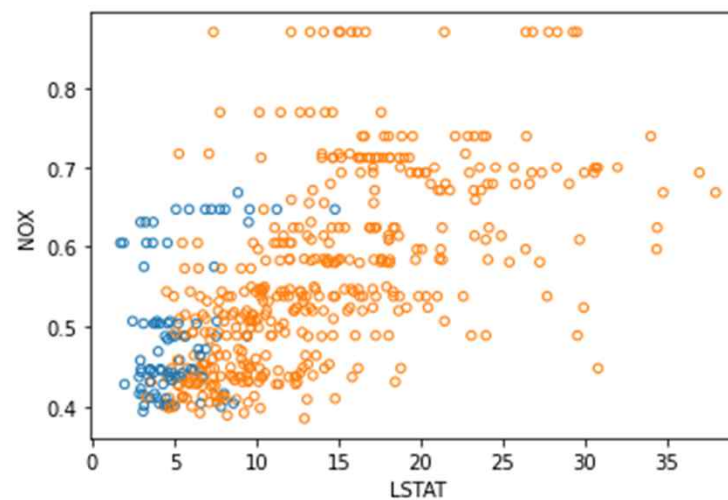
- 다중 패널 산점도를 이용하는 특별한 차트
- 비지도 학습에서는 수치형 변수들 간의 연관성 분석, 아웃라이어 탐지, 군집 식별 등에 유용
- 지도 학습에서는 예측 변수들 간의 쌍별 관련성을 평가함으로써 변수 변환과 변수 선택 도움
- 예측 분야에서는 수치형 예측 변수와 결과 변수 간 관계를 서술하는 데 쓰임

색상, 크기, 모양, 다중 패널, 애니메이션

```
# CAT, MEDV의 값으로 포인트 색상을 정함
housing_df.plot.scatter(x='LSTAT', y='NOX',
                       c=['CO' if c == 1 else 'C1' for c in housing_df.CAT_MEDV])
plt.show()
```



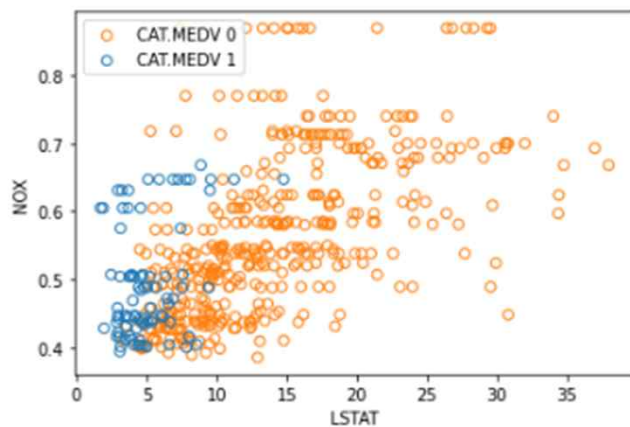
```
# 점의 렌더링을 열린 원으로 변경
housing_df.plot.scatter(x='LSTAT', y='NOX', color='none',
                       edgecolor=['CO' if c == 1 else 'C1' for c in housing_df.CAT_MEDV])
plt.show()
```



색상, 크기, 모양, 다중 패널, 애니메이션

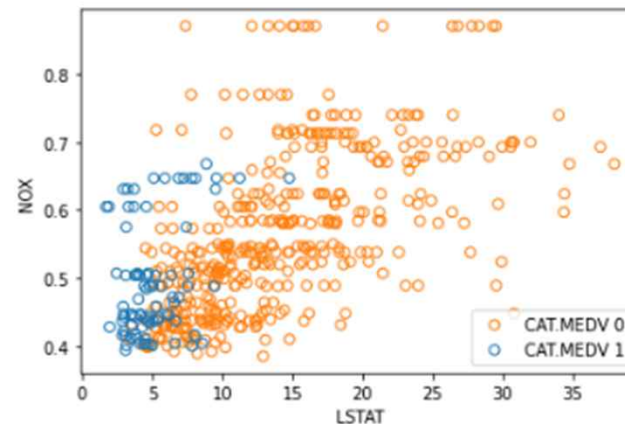
```
# CAT.MEDV의 데이터 포인트 0을 플로팅한 다음 1을 플로팅
fig, ax = plt.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
    subset_df = housing_df[housing_df.CAT_MEDV == catValue]
    ax.scatter(subset_df.LSTAT, subset_df.NOX, color='none', edgecolor=color)
ax.set_xlabel('LSTAT')
ax.set_ylabel('NOX')
ax.legend(["CAT.MEDV 0", "CAT.MEDV 1"])

plt.show()
```



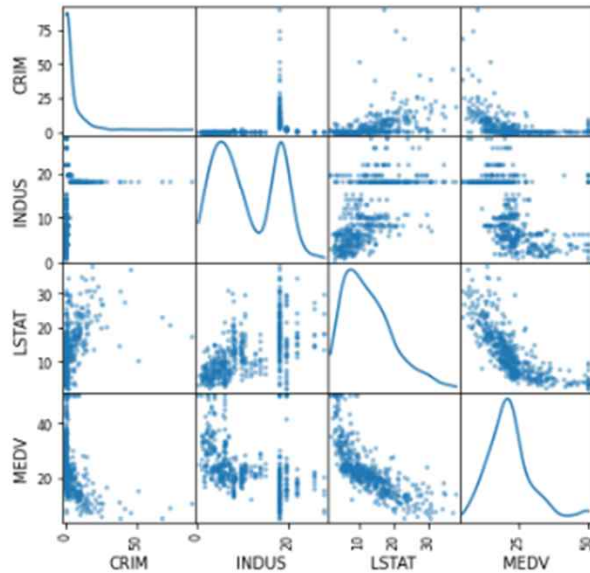
```
fig, ax = plt.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
    subset_df = housing_df[housing_df.CAT_MEDV == catValue]
    ax.scatter(subset_df.LSTAT, subset_df.NOX, color='none', edgecolor=color)
ax.set_xlabel('LSTAT')
ax.set_ylabel('NOX')
ax.legend(["CAT.MEDV 0", "CAT.MEDV 1"], loc=4)

plt.show()
```

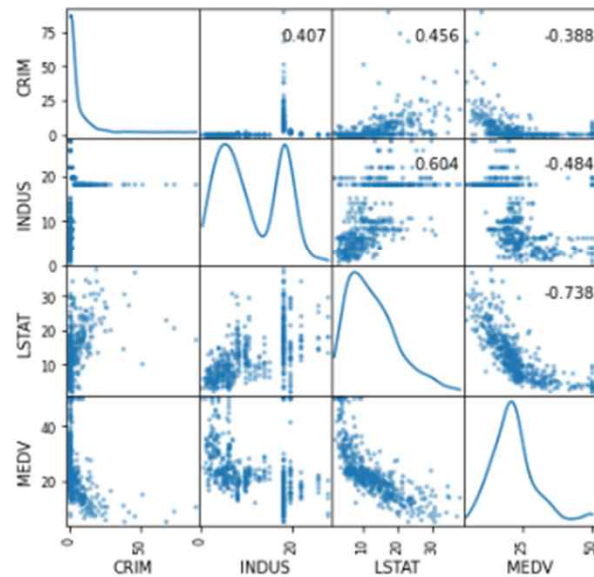


색상, 크기, 모양, 다중 패널, 애니메이션

```
# 각 변수 별 scatter plot 작성
# diag_kind='kde' 를 사용하여 각 변수별 커널밀도추정곡선 보여줌
df=housing_df[['CRIM', 'INDUS', 'LSTAT', 'MEDV']]
axes=scatter_matrix(df, alpha=0.5, figsize=(6,6), diagonal='kde')
```



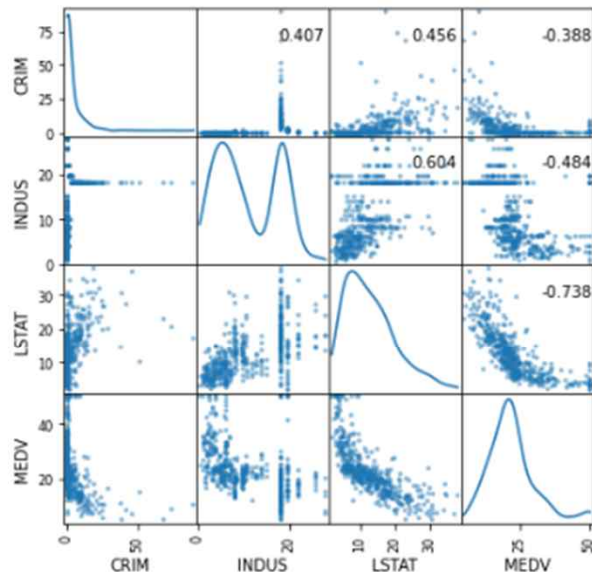
```
# 대각 행렬 위쪽에 상관 계수 추가
df = housing_df[['CRIM', 'INDUS', 'LSTAT', 'MEDV']]
axes = scatter_matrix(df, alpha=0.5, figsize=(6, 6), diagonal='kde')
corr = df.corr().values
for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
    axes[i, j].annotate("%.3f" % corr[i,j], (0.8, 0.8), xycoords='axes fraction', ha='center', va='center')
plt.show()
```



색상, 크기, 모양, 다중 패널, 애니메이션

```
# 대각 행렬 위쪽에 상관 계수 추가
df = housing_df[['CRIM', 'INDUS', 'LSTAT', 'MEDV']]
axes = scatter_matrix(df, alpha=0.5, figsize=(6, 6), diagonal='kde')
corr = df.corr().values
for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
    axes[i, j].annotate("%.3f" % corr[i, j], (0.8, 0.8), xycoords='axes fraction', ha='center', va='center')

plt.show()
```



```
#주석 달기
#matplotlib.pyplot.annotate(s, xy, *args, **kwargs)
# s: str
# xy: (float, float)
#선택
# xytext: (float, float), defaults:xy
# xycoords: str
#     'figure points' Points from the lower left of the figure
#     'figure pixels' Pixels from the lower left of the figure
#     'figure fraction' Fraction of figure from lower left
#     'axes points' Points from lower left corner of axes
#     'axes pixels' Pixels from lower left corner of axes
#     'axes fraction' Fraction of axes from lower left
#     'data' Use the coordinate system of the object being annotated (default)
#     'polar' (theta,r) if not native 'data' coordinates
# textcoords: str
#     'offset points' Offset (in points) from the xy value
#     'offset pixels' Offset (in pixels) from the xy value
# arrowprops: dict
# annotation_clip: bool
```

스케일링, 집계와 계층 구조

■ 스케일링, 집계와 계층 구조

- 스케일, 집계, 계층을 달리하며 분석하면 다양한 수준의 패턴과 관계가 드러나고, 집중 분석해야 하는 새로운 변수군 발견 가능

■ 스케일링

- 디스플레이의 축척을 변경하면 변수 간의 관계를 부각시킬 수 있음
- 밀집 현상을 해소하고, 2개의 로그 스케일 변수 간 선형 관계(로그-로그 관계)를 드러내줌

■ 집계와 계층 구조

- 집계의 수준을 변경하는 스케일링
- 시계열 데이터에서 흔히 사용하는 집계 방법은 이동 평균 (moving ave.)
- 비시간성 변수는 지리적 위치(보스턴 주택 예제의 우편번호 별 주택), 조직도(부서 또는 부문의 인력) 등 의미 있는 계층이 있다면 집계될 수 있음

스케일링, 집계와 계층 구조

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))
```

```
# Regular scale
```

```
housing_df.plot.scatter(x='CRIM', y='MEDV', ax=axes[0])
```

```
# log scale
```

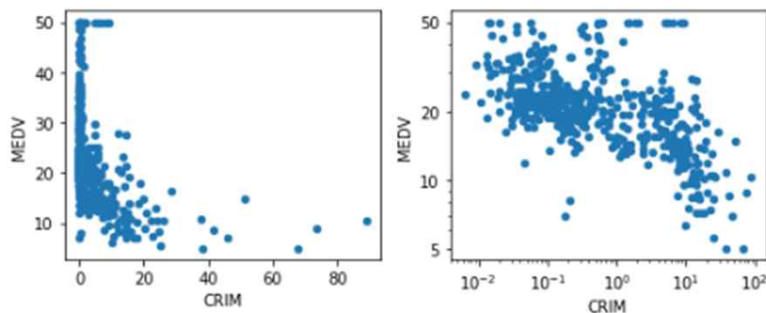
```
ax = housing_df.plot.scatter(x='CRIM', y='MEDV', logx=True, logy=True, ax=axes[1])
```

```
ax.set_yticks([5, 10, 20, 50])
```

```
ax.set_yticklabels([5, 10, 20, 50])
```

```
plt.tight_layout()
```

```
plt.show()
```



```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))
```

```
# Regular scale
```

```
housing_df.plot.scatter(x='CRIM', y='MEDV', ax=axes[0])
```

```
# log scale
```

```
ax = housing_df.plot.scatter(x='CRIM', y='MEDV', logx=True, logy=True, ax=axes[1])
```

```
# 로그 인덱스를 반영한 단위 표시
```

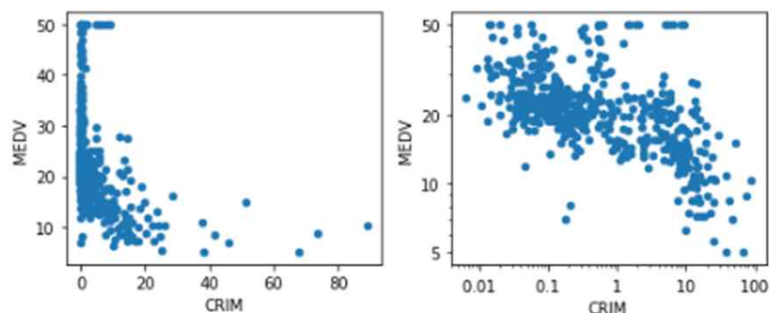
```
plt.rcParams['axes.formatter.min_exponent'] = 4
```

```
ax.set_yticks([5, 10, 20, 50])
```

```
ax.set_yticklabels([5, 10, 20, 50])
```

```
plt.tight_layout()
```

```
plt.show()
```



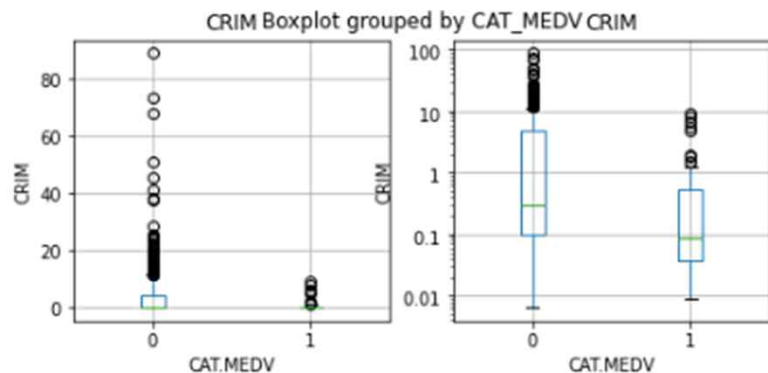
스케일링, 집계와 계층 구조

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))

# regular scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[0])
ax.set_xlabel('CAT.MEDV')
ax.set_ylabel('CRIM')

# log scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[1])
ax.set_xlabel('CAT.MEDV')
ax.set_ylabel('CRIM')
ax.set_yscale('log')

plt.show()
```



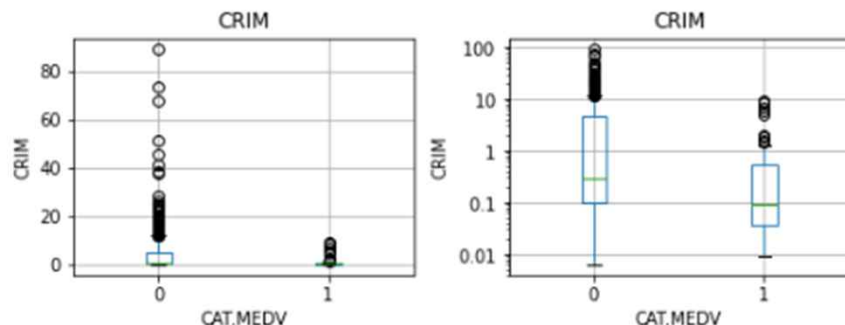
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))

# regular scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[0])
ax.set_xlabel('CAT.MEDV')
ax.set_ylabel('CRIM')

# log scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[1])
ax.set_xlabel('CAT.MEDV')
ax.set_ylabel('CRIM')
ax.set_yscale('log')

# suppress the title
axes[0].get_figure().suptitle('')
plt.tight_layout()

plt.show()
```



확대/축소와 패닝, 필터링

■ 확대/축소와 패닝

- 차트의 특정 영역을 확대 또는 축소, 확대/축소 윈도를 다른 영역으로 이동(패닝)
- 확대/축소와 패닝을 통해 새로운 상호작용 조건, 새로운 변수, 데이터 서브셋에 관한 별도 모델을 만들어낼 수 있음

■ 필터링

- 노이즈를 제거함으로써 특정 데이터에 관심을 집중하는 것

확대/축소와 패닝, 필터링

```
fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(10,10))

Amtrak_df = dmba.load_data('Amtrak.csv')
Amtrak_df['Month'] = pd.to_datetime(Amtrak_df.Month, format='%d/%m/%Y')
Amtrak_df.set_index('Month', inplace=True)

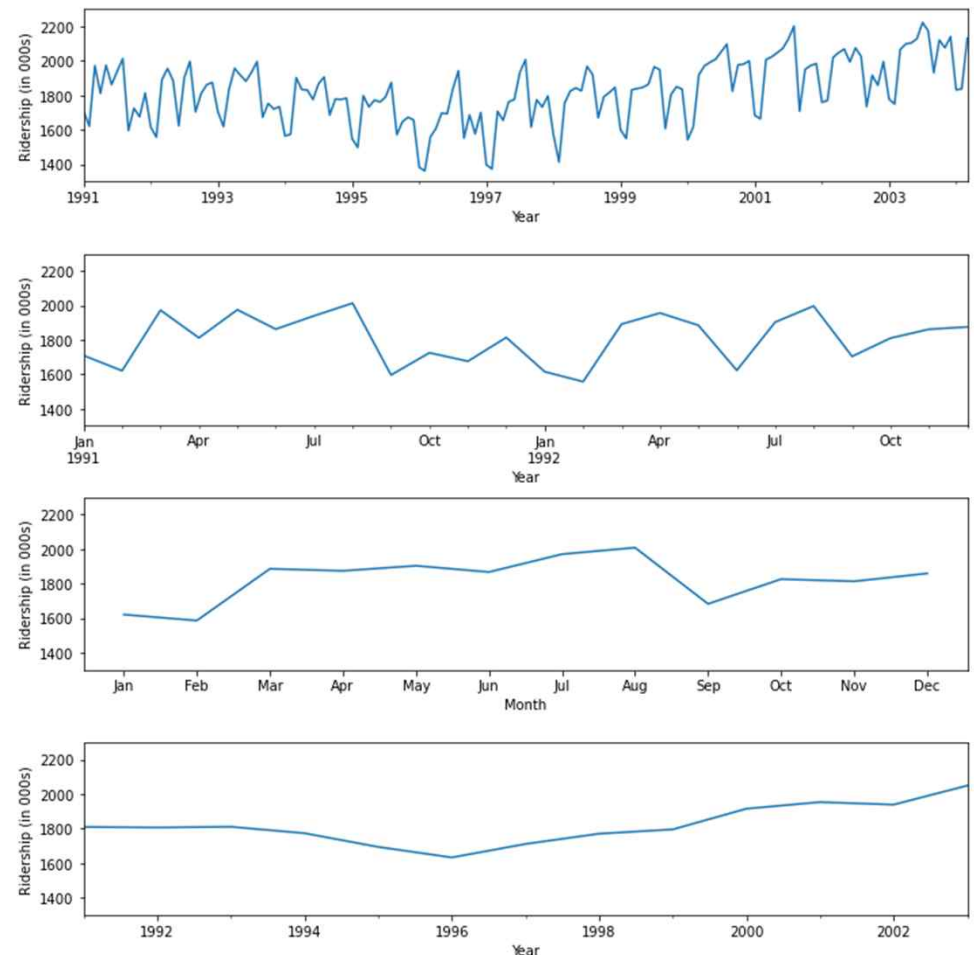
# Raw Data
ax = Amtrak_df.plot(ylim=[1300, 2300], legend=False, ax=axes[0])
ax.set_xlabel('Year') # set x-axis label
ax.set_ylabel('Ridership (in 000s)') # set y-axis label

# 1991년 1월 1일부터 1992년 12월 31일까지 확대
ridership_2yrs = Amtrak_df.loc['1991-01-01':'1992-12-31']
ax = ridership_2yrs.plot(ylim=[1300, 2300], legend=False, ax=axes[1])
ax.set_xlabel('Year') # set x-axis label
ax.set_ylabel('Ridership (in 000s)') # set y-axis label

# 월별 평균
byMonth = Amtrak_df.groupby(by=[Amtrak_df.index.month]).mean()
ax = byMonth.plot(ylim=[1300, 2300], legend=False, ax=axes[2])
ax.set_xlabel('Month') # set x-axis label
ax.set_ylabel('Ridership (in 000s)') # set y-axis label
ax.set_xticks(range(1, 13))
ax.set_xticklabels([calendar.month_abbr[i] for i in range(1, 13)]);

# 연도별 평균 (2004년 데이터 제외)
byYear = Amtrak_df.loc['1991-01-01':'2003-12-31'].groupby(pd.Grouper(freq='A')).mean()
ax = byYear.plot(ylim=[1300, 2300], legend=False, ax=axes[3])
ax.set_xlabel('Year') # set x-axis label
ax.set_ylabel('Ridership (in 000s)') # set y-axis label

plt.tight_layout()
plt.show()
```



대용량 데이터셋으로 스케일 업하기

- 관측(행) 수가 많아 개별적인 관측치를 표시하는 플롯(산점도)이 비효과적일 때 대안
 - ① 샘플링: 표본을 랜덤하게 추출해 차트를 그린다.
 - ② 표시 크기를 줄인다.
 - ③ 표시 색의 투명도를 사용한다.
 - ④ 데이터를 구분해 서브셋을 만든다(다중 패널 사용).
 - ⑤ 집계를 사용한다(거품 차트에서 거품 크기는 관측 건수에 비례).
 - ⑥ 지터링을 사용한다(적은 양의 노이즈를 추가해 개별 표시를 짧게 이동)

대용량 데이터셋으로 스케일 업하기

```
universal_df = dmba.load_data('UniversalBank.csv')
universal_df.head(5)
```

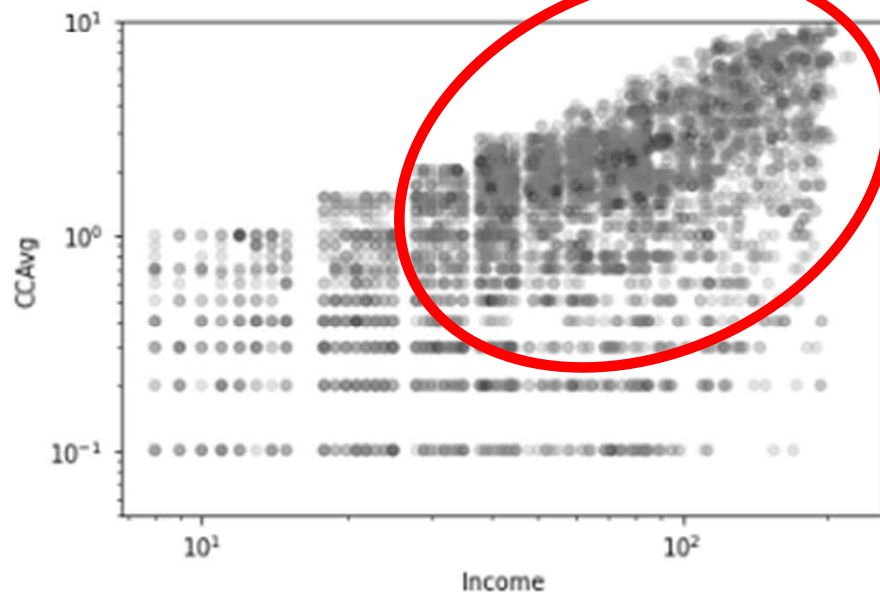
	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

```
universal_df.describe()
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937938	1.881000	56.498800	0.096000	0.104400	
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747659	0.839869	101.713802	0.294621	0.305809	
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	

대용량 데이터셋으로 스케일 업하기

```
universal_df.plot.scatter(x='Income', y='CCAvg',  
                          c=['black' if c == 1 else 'grey' for c in universal_df['Securities Account']],  
                          ylim = (0.05, 10), alpha=0.2,  
                          logx=True, logy=True)  
plt.show()
```



- 오버플롯으로 인해 증권 계좌의 분포가 명확하지 않음. 두 개의 account 별로 plot
- 중첩 포인트: 데이터 포인트에 랜덤 지터 추가

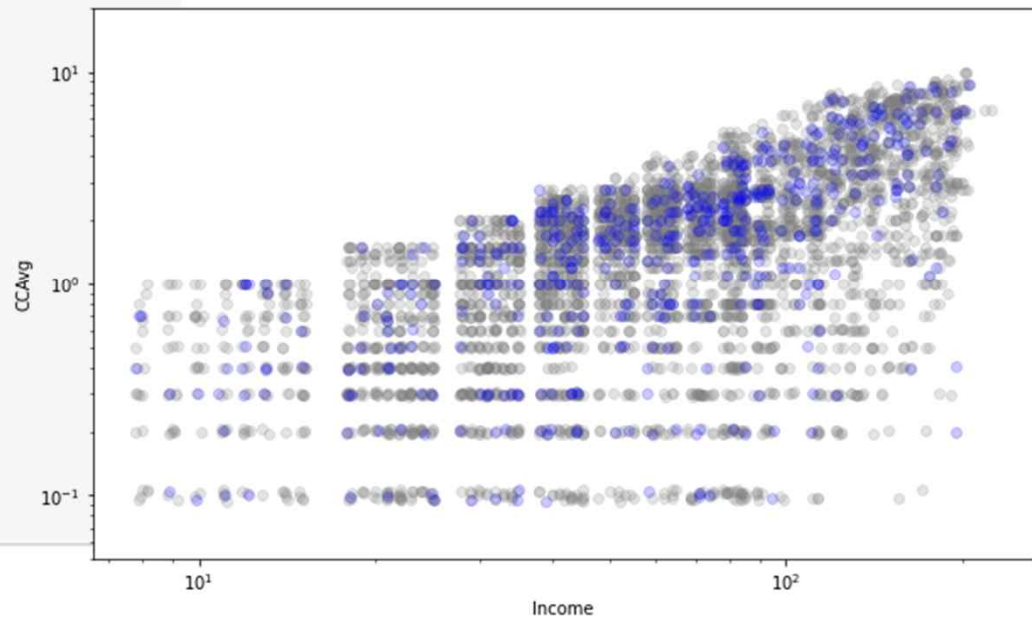
대용량 데이터셋으로 스케일 업하기

#x값에 랜덤 jitter 추가하는 함수

```
def jitter(x, factor=1):  
    sx = np.array(sorted(x))  
    delta = sx[1:] - sx[:-1]  
    minDelta = min(d for d in delta if d > 0)  
    a = factor * minDelta / 5  
    return x + np.random.uniform(-a, a, len(x))
```


```
saldx = universal_df[universal_df['Securities Account'] == 1].index
```

```
plt.figure(figsize=(10,6))  
plt.scatter(jitter(universal_df.drop(saldx).Income),  
            jitter(universal_df.drop(saldx).CCAvg),  
            marker='o', color='grey', alpha=0.2)  
plt.scatter(jitter(universal_df.loc[saldx].Income),  
            jitter(universal_df.loc[saldx].CCAvg),  
            marker='o', color='blue', alpha=0.2)  
plt.xlabel('Income')  
plt.ylabel('CCAvg')  
plt.ylim((0.05, 20))  
axes = plt.gca()  
axes.set_xscale("log")  
axes.set_yscale("log")  
plt.show()
```



대용량 데이터셋으로 스케일 업하기

```
#x값에 랜덤 jitter 추가하는 함수
def jitter(x, factor=1):
    sx = np.array(sorted(x))
    delta = sx[1:] - sx[:-1]
    minDelta = min(d for d in delta if d > 0)
    a = factor * minDelta / 5
    return x + np.random.uniform(-a, a, len(x))
```

 작성한 함수의 설계 해석 해보기

```
saldx = universal_df[universal_df['Securities Account'] == 1].index

plt.figure(figsize=(10,6))
plt.scatter(jitter(universal_df.drop(saldx).Income),
            jitter(universal_df.drop(saldx).CCAvg),
            marker='o', color='grey', alpha=0.2)
plt.scatter(jitter(universal_df.loc[saldx].Income),
            jitter(universal_df.loc[saldx].CCAvg),
            marker='o', color='blue', alpha=0.2)
plt.xlabel('Income')
plt.ylabel('CCAvg')
plt.ylim((0.05, 20))
axes = plt.gca()
axes.set_xscale("log")
axes.set_yscale("log")

plt.show()
```

다변량 플롯: 평행 좌표

■ 평행 좌표

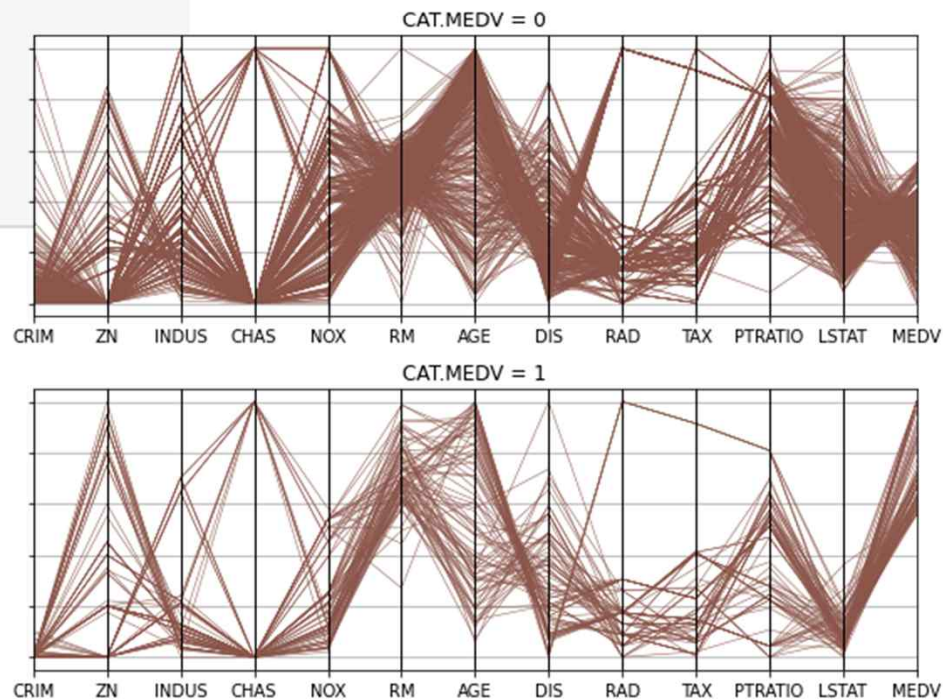
- 각 변수에 대응하는 수직 축이 하나씩 그려지고, 개별 관측들은 각 수직축 위에 있는 관측치를 연결하는 선으로 표현 ➔ 다변량 프로파일
- 비지도 학습 작업에 유용. 군집, 아웃라이어, 변수 간의 중복 정보를 가려내줌

다변량 플롯: 평행 좌표

```
# 모든 축이 동일한 범위를 갖도록 축을 변환
min_max_scaler = preprocessing.MinMaxScaler()
dataToPlot = pd.DataFrame(min_max_scaler.fit_transform(housing_df),
                           columns=housing_df.columns)

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=[8, 6])
for i in (0, 1):
    parallel_coordinates(dataToPlot.loc[dataToPlot.CAT_MEDV == i],
                        'CAT_MEDV', color='C5', ax=axes[i], linewidth=0.5)
    axes[i].set_title('CAT_MEDV = {}'.format(i))
    axes[i].set_yticklabels([])
    axes[i].legend().set_visible(False)

plt.tight_layout()
plt.show()
```



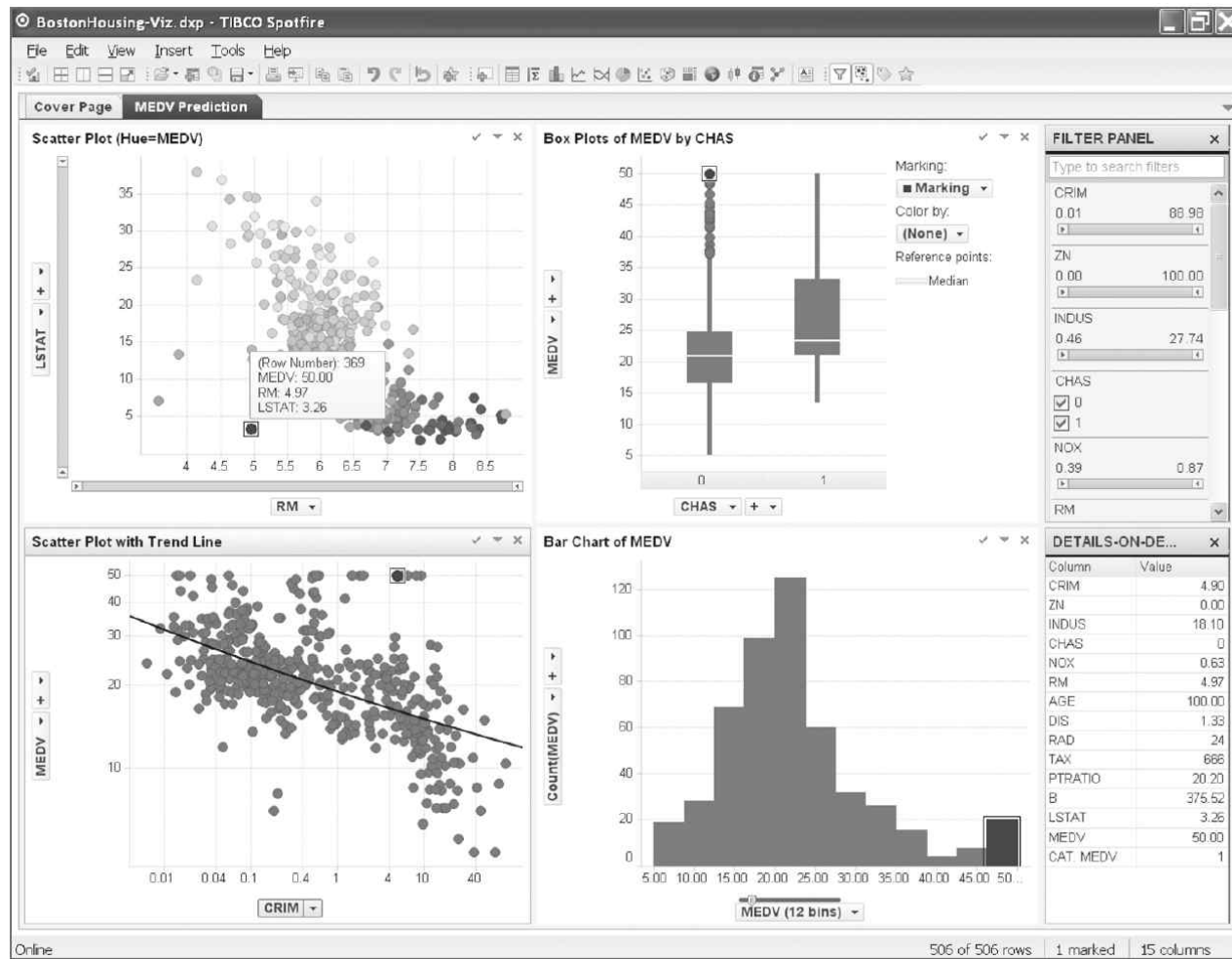
인터랙티브 시각화

■ 인터랙티브 시각화

- ① 차트 변경이 쉽고, 빠르며, 이전 상태로 복구할 수 있음
- ② 관련 있는 다중 차트가 쉽게 결합되고, 한 화면에 표시 가능
- ③ 서로 연결되는 두 차트 중 한 차트에서 동작이 일어나면, 다른 쪽 차트에도 반영됨

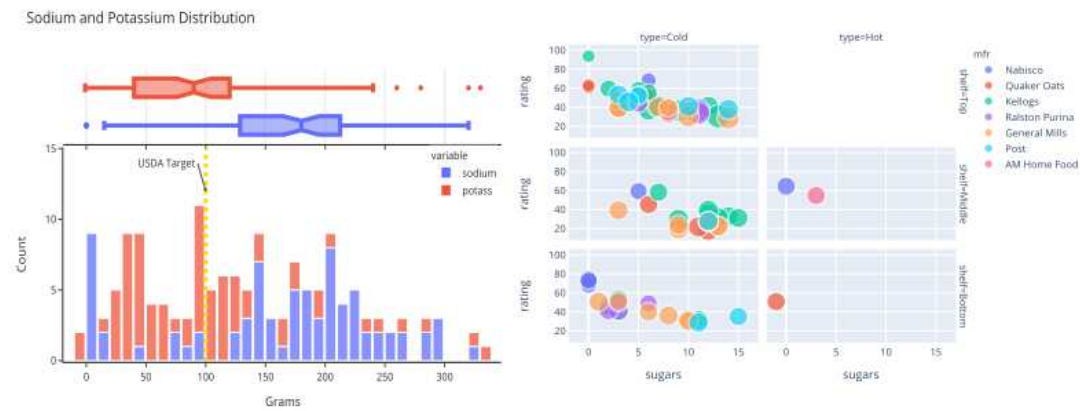
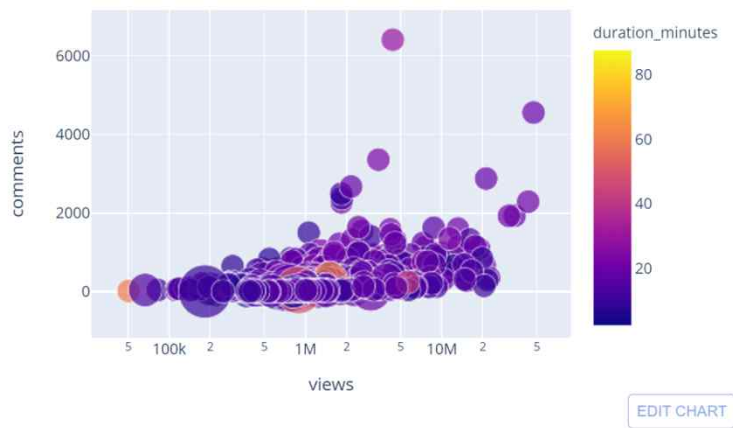


인터랙티브 시각화



인터랙티브 시각화

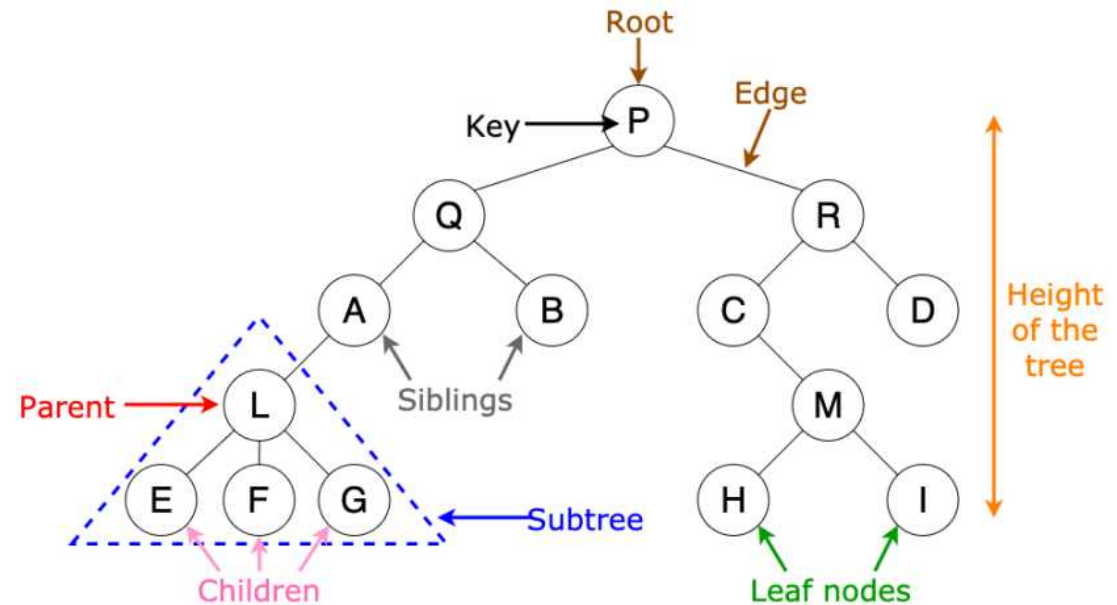
■ Python + JavaScript (plotly, Bokeh)



네트워크 데이터의 시각화

■ 네트워크 분석

- 노드와 엣지를 기반으로 데이터 간의 연결성을 확인
- networkx 패키지를 활용 (python-igraph)

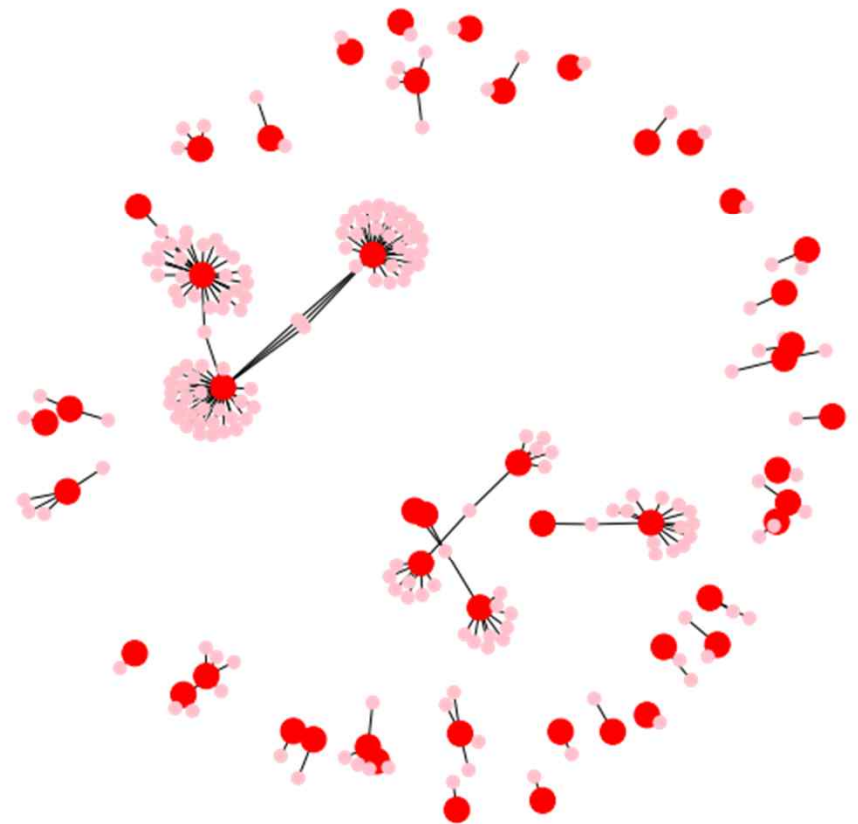


네트워크 데이터의 시각화

```
ebay_df = dmba.load_data('eBayNetwork.csv')

G = nx.from_pandas_edgelist(ebay_df, source='Seller', target='Bidder')

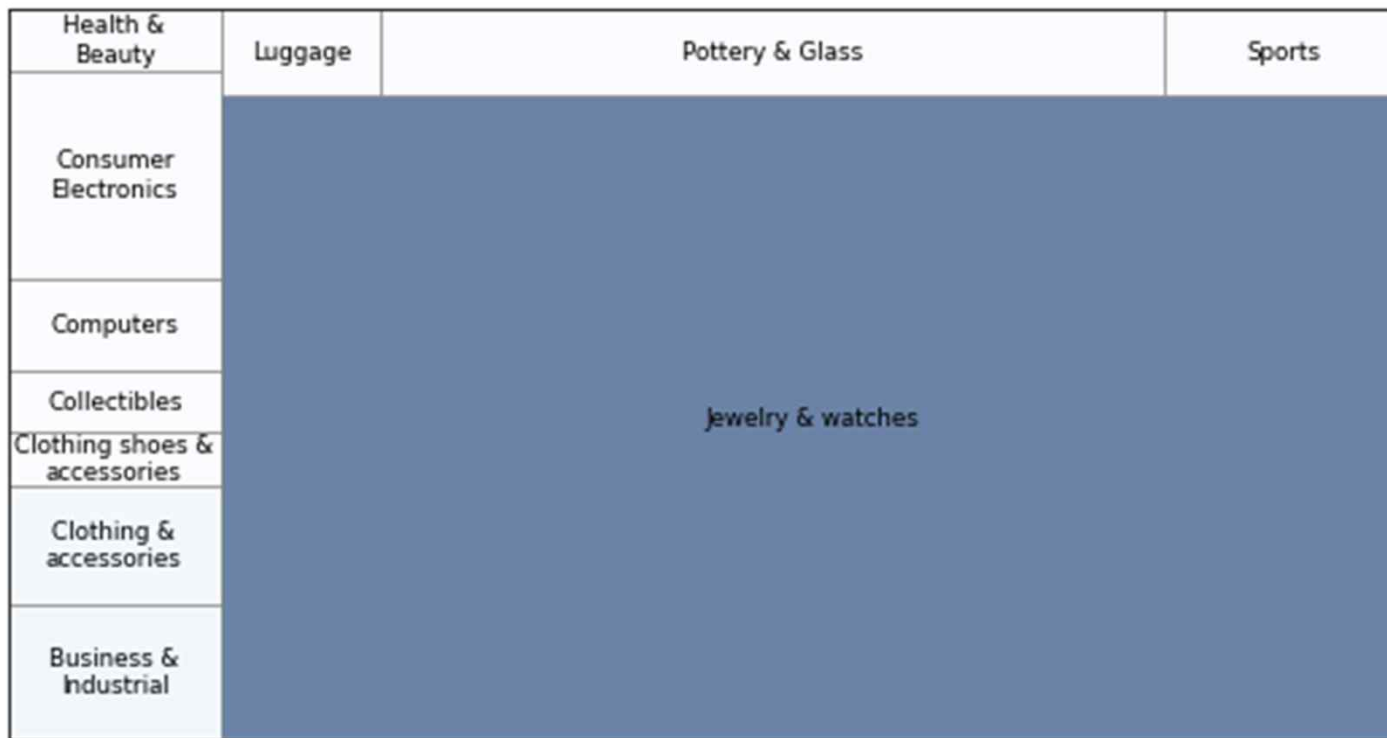
isBidder = [n in set(ebay_df.Bidder) for n in G.nodes()]
pos = nx.spring_layout(G, k=0.13, iterations=60, scale=0.5)
plt.figure(figsize=(10,10))
nx.draw_networkx(G, pos=pos, with_labels=False,
                 edge_color='black',
                 node_color=['pink' if bidder else 'red' for bidder in isBidder],
                 node_size=[50 if bidder else 200 for bidder in isBidder])
plt.axis('off')
plt.show()
```



계층 데이터의 시각화: 트리맵

■ 트리맵

- 계층구조 or 트리구조를 갖는 대규모 데이터셋의 탐색을 시각화하는데 유용함
- 트리맵은 파이썬보다 R이 효과적



계층 데이터의 시각화: 트리맵

```
import squarify
import matplotlib

ebayTreemap = dmba.load_data('EbayTreemap.csv')

grouped = []
for category, df in ebayTreemap.groupby(['Category']):
    negativeFeedback = sum(df['Seller Feedback'] < 0) / len(df)
    grouped.append({
        'category': category,
        'negativeFeedback': negativeFeedback,
        'averageBid': df['High Bid'].mean()
    })
byCategory = pd.DataFrame(grouped)

norm = matplotlib.colors.Normalize(vmin=byCategory.negativeFeedback.min(), vmax=byCategory.negativeFeedback.max())
colors = [matplotlib.cm.Blues(norm(value)) for value in byCategory.negativeFeedback]

fig, ax = plt.subplots()
fig.set_size_inches(9, 5)

renameCategories = {
    'Business & Industrial': 'Business & Industrial',
    'Health & Beauty': 'Health & Beauty',
    'Consumer Electronics': 'Consumer Electronics',
    'Clothing & accessories': 'Clothing & accessories',
    'Clothing shoes & accessories': 'Clothing shoes & accessories'
}
labels = [renameCategories.get(c, c) for c in byCategory.category]

squarify.plot(label=labels, sizes=byCategory.averageBid, color=colors,
              ax=ax, alpha=0.6, edgecolor='grey', text_kwargs={'fontsize': 8.7})

ax.get_xaxis().set_ticks([])
ax.get_yaxis().set_ticks([])

plt.subplots_adjust(left=0.1)
plt.show()
```

공간 정보 데이터의 시각화: 맵 차트



주요 시각화 작업 요약

■ 예측

- 박스 플롯, 막대그래프, 산점도의 y축에 결과 변수 배치
- 병렬 박스 플롯, 막대그래프, 다중 패널을 사용해 결과 변수와 범주형 예측 변수 간의 관계 탐색
- 산점도를 사용해 결과 수치형 예측 변수 간의 관계 탐색
- 분포도(박스 플롯이나 히스토그램)를 사용해 결과 변수(and/or 수치형 예측 변수들)의 변환 필요성 결정
- 인터랙티브 조건의 필요성을 찾기 위해 색상/패널/크기를 추가해 산점도 분석
- 여러 가지 집계 레벨과 확대/축소를 사용해 다른 행동 양식을 보이는 데이터 영역을 찾고, 전역 패턴과 지역 패턴의 레벨 평가

주요 시각화 작업 요약

■ 분류

- y축에 있는 결과 변수의 막대그래프를 사용해 범주형 예측 변수와 결과 변수의 관계 탐색
- 색상-코드화된 산점도(색상은 결과 변수를 표시)를 통해 결과 변수와 쌍별 수치형 예측 변수 간의 관계 연구
- 병렬 박스 플롯을 통해 결과 변수와 수치형 예측 변수 간의 관계 연구. 결과 변수에 관한 수치형 변수의 박스 플롯 시각화. 다른 수치형 예측 변수도 동일한 시각화 작업 수행. 가장 분리된 박스가 유용한 예측 변수임
- 평행 좌표에서는 색상을 사용해 결과 변수 표시
- 분포도(박스 플롯이나 히스토그램)를 사용해 결과 변수(and/or 수치형 예측 변수들)의 변환 필요성 결정
- 인터랙티브 조건의 필요성을 찾기 위해 색상/패널/크기를 추가해 산점도 분석
- 여러 가지 집계 레벨과 확대/축소를 사용해 다른 행동 양식을 보이는 데이터 영역을 찾고, 전역 패턴과 지역 패턴의 레벨 평가

주요 시각화 작업 요약

■ 시계열 예측

- 패턴의 종류를 결정하기 위해 다양한 시간으로 집계한 선그래프 생성
- 확대/축소와 패닝을 사용해 여러 가지 단기 시계열을 찾고, 서로 다른 행동 양식을 보이는 데이터 영역 결정
- 전역 패턴과 지역 패턴을 찾기 위해 다양한 집계 레벨 사용
- 시계열 데이터의 결측치 식별
- 적절한 모델을 선택하기 위해 여러 유형의 추세선을 겹쳐보기

주요 시각화 작업 요약

■ 비지도 학습

- 관측치의 쌍별 관계와 군집을 식별하기 위해 산점도 매트릭스 생성
- 상관관계표를 검토하기 위해 히트맵 사용
- 다른 행동 양식을 보이는 데이터의 영역을 찾기 위해 여러 집계 레벨과 확대/축소 사용
- 데이터 군집을 찾기 위해 평행 좌표 생성

퇴근 문제

■ 가전제품 출하량

- ApplianceShipments.csv 파일에는 1985~1989년의 미국 가전제품에 대한 분기 별 shipments가 있음
 1. 시계열 차트를 파이썬으로 그려보자
 2. 분기별 패턴을 자세하게 파악하기 위해서 y축 값 3500 – 5000 범위를 확대해보자
 3. Q1, Q2, Q3, Q4에 대한 차트 1개에 4개의 꺾은선 그래프를 파이썬으로 그려보자. (Hint. 분기 및 연도에 대한 열을 추가해서 할 수 있음)
 4. 데이터 프레임을 분기별로 그룹화한 다음 선그래프를 이용해 분기별 출하량을 그려보자
 5. Y축의 3500-5000 범위를 확대하고 3)과 4) 차이를 설명해보자
 6. 연도별 집계(각 연도의 shipment 합계)를 구하고 선그래프를 파이썬으로 그려보자

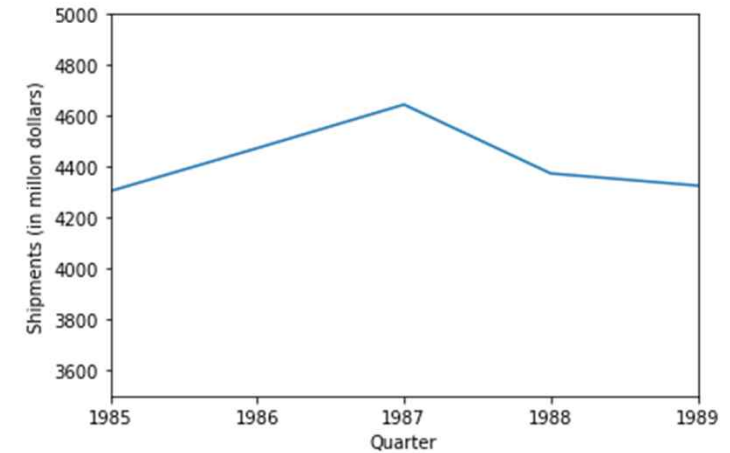
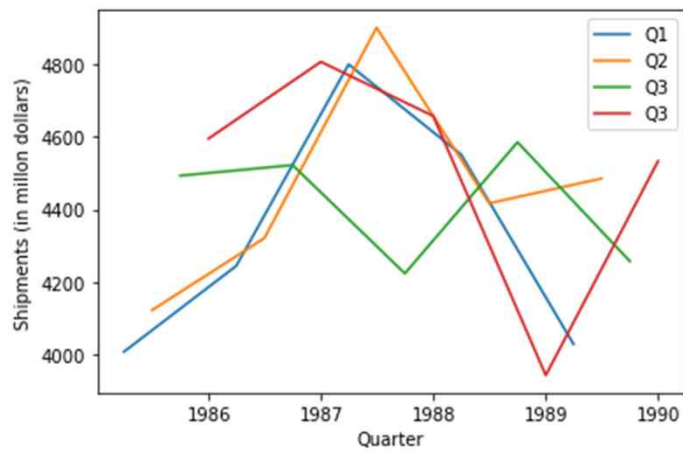
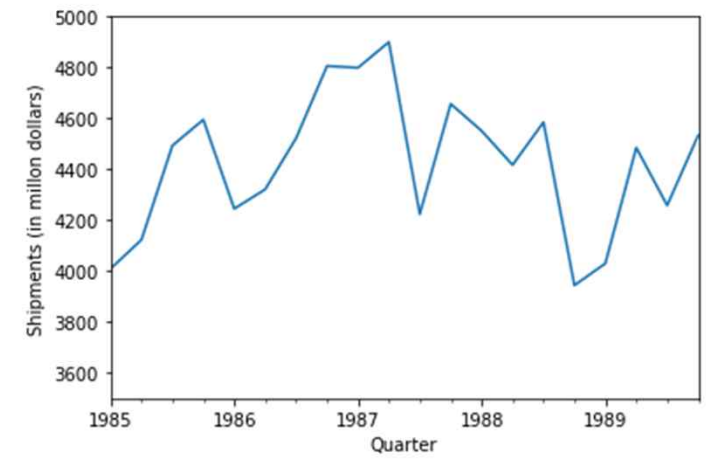
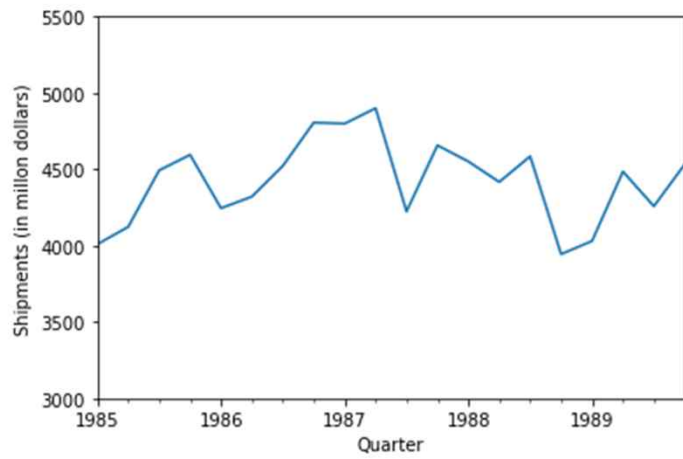
퇴근 문제



퇴근 문제



퇴근 문제



Q & A

