머신러닝 사례

붓꽃의 품종 판별-분류

제주대학교 머신러닝 연구실
고지영 (박사과정)

분류(Classification)

- 지도학습(Supervised Learning)
- 주어진 데이터를 **미리 정의된 클래스 레이블 중 하나로 분류**하는 문제
- **입력 데이터**와 해당 데이터의 **정답인 클래스 레이블 사이의 관계를 학습**하여 새로운 데이터에 대한 클래스를 **예측**

사진 속 동물을 보고 **고양이인지, 개인지**

어떤 사람의 병원 데이터를 보고, **병에 걸렸는지 안 걸렸는지**

이메일을 분류할 때, **스팸인지 아닌지**

붓꽃 데이터 분류 예측을 위한 과정

1. 데이터 준비 : 분류 모델을 학습하기 위해 입력 데이터와 클래스 레이블을 준비

2. 데이터셋 분리 : 전체 데이터를 학습 / 테스트 데이터로 분리

3. 모델 선택 & 학습 : 적절한 모델을 선택 & 학습 데이터로부터 모델 학습

4. 모델 예측 : 학습된 모델을 이용해 테스트 데이터의 분류를 예측

5. 모델 평가 : 예측된 결괏값과 테스트 데이터의 실제 결괏값을 비교해 모델 성능 평가

붓꽃이란?

- **꽃받침(sepal)과 꽃잎(petal)** 으로 구성되어 있음,
- 이러한 특징을 기반으로 붓꽃의 품종을 식별

1.Setosa (세토사): Setosa는 붓꽃 중에서 가장 작은 꽃잎과 꽃받침을 가지고 있다.
꽃잎과 꽃받침이 비교적 짧고 뾰족한 모습을 갖고 있으며, 주로 흰색 또는 연한 분홍색.

2.Versicolor (버시컬러): Versicolor는 Setosa보다 크고 긴 꽃잎과 꽃받침을 가지고 있다.
꽃잎의 색은 보통 연한 보라색이며, 중간 크기의 붓꽃.

3.Virginica (버지니카): Virginica는 붓꽃 중에서 가장 크고 긴 꽃잎과 꽃받침을 가지고 있다.
꽃잎의 색은 주로 짙은 보라색이며, 다른 품종들에 비해 상대적으로 더 큰 크기를 갖고 있다.

라이브러리 설치

파이썬 패키지 관리자인 pip를 사용하여 설치

```
!pip install pandas
```

```
!pip install scikit-learn
```

-pandas 라이브러리 : 데이터를 구조화하고 조작하는 데에 효과적인 도구를 제공하는 라이브러리

-scikit-learn 라이브러리 : 오픈소스로 공개된 머신러닝 라이브러리

- matplotlib 라이브러리 : 파이썬에서 2D 형태의 그래프, 이미지 등을 그릴 때 사용하는 라이브러리

라이브러리 불러오기

```
from sklearn.datasets import load_iris # 붓꽃 데이터 생성 라이브러리
from sklearn.model_selection import train_test_split # 데이터셋을 학습, 테스트, 예측 데이터로 분리하는 라이브러리
from sklearn.tree import DecisionTreeClassifier # 트리기반 ML 알고리즘(의사결정나무)을 불러오는 라이브러리
from sklearn.metrics import accuracy_score # 정확도 평가 라이브러리
import pandas as pd # pandas 라이브러리
```

```
from 파일명(라이브러리) import 함수이름
```

데이터 준비

```
iris = pd.read_csv('/kaggle/input/iris-data/Iris.csv')
#pd.read_csv () 함수를 사용하여 경로에 있는 데이터를 가져오기
```

```
iris.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

꽃 받침 길이 (cm), 꽃 받침 너비(cm), 꽃 잎 길이(cm), 꽃잎 너비(cm), 붓꽃 종류

데이터 분석

shape :행과 열의 개수를 반환

```
iris.shape
```

```
(150, 5)
```

dtypes : 열을 기준으로 데이터 형태 반환

```
iris.dtypes
```

```
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

Describe() : 데이터프레임의 숫자형 열에 대한 통계 요약.각 열의 개수, 평균, 표준편차, 최솟값, 1사분위수, 중앙값 (2사분위수), 3사분위수, 최댓값 등을 반환
T: 전치

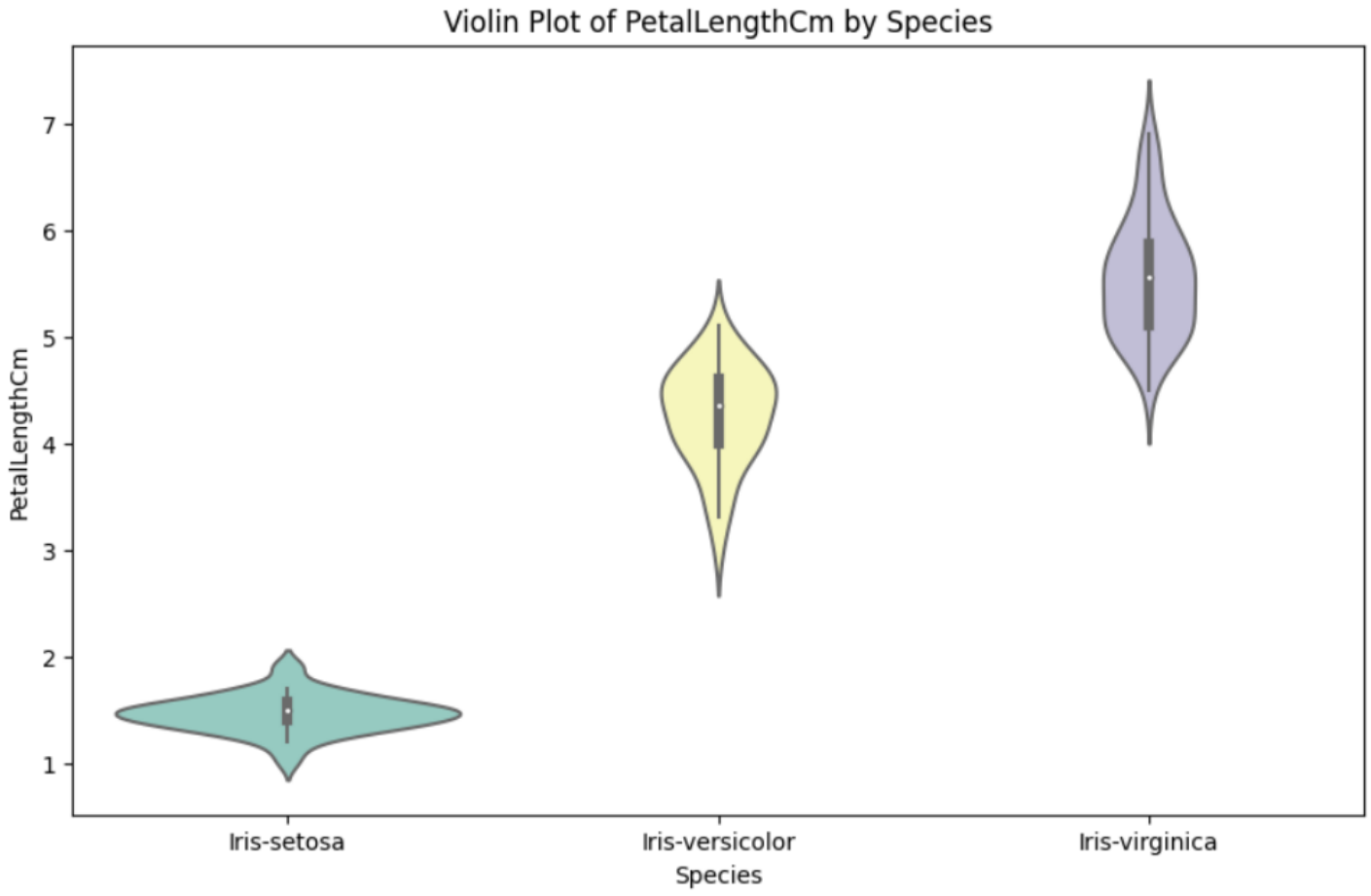
```
#statistical information about the data
iris.describe().T
```

	count	mean	std	min	25%	50%	75%	max
SepalLengthCm	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
SepalWidthCm	150.0	3.054000	0.433594	2.0	2.8	3.00	3.3	4.4
PetalLengthCm	150.0	3.758667	1.764420	1.0	1.6	4.35	5.1	6.9
PetalWidthCm	150.0	1.198667	0.763161	0.1	0.3	1.30	1.8	2.5

데이터 분석

#품종별 꽃받침 길이

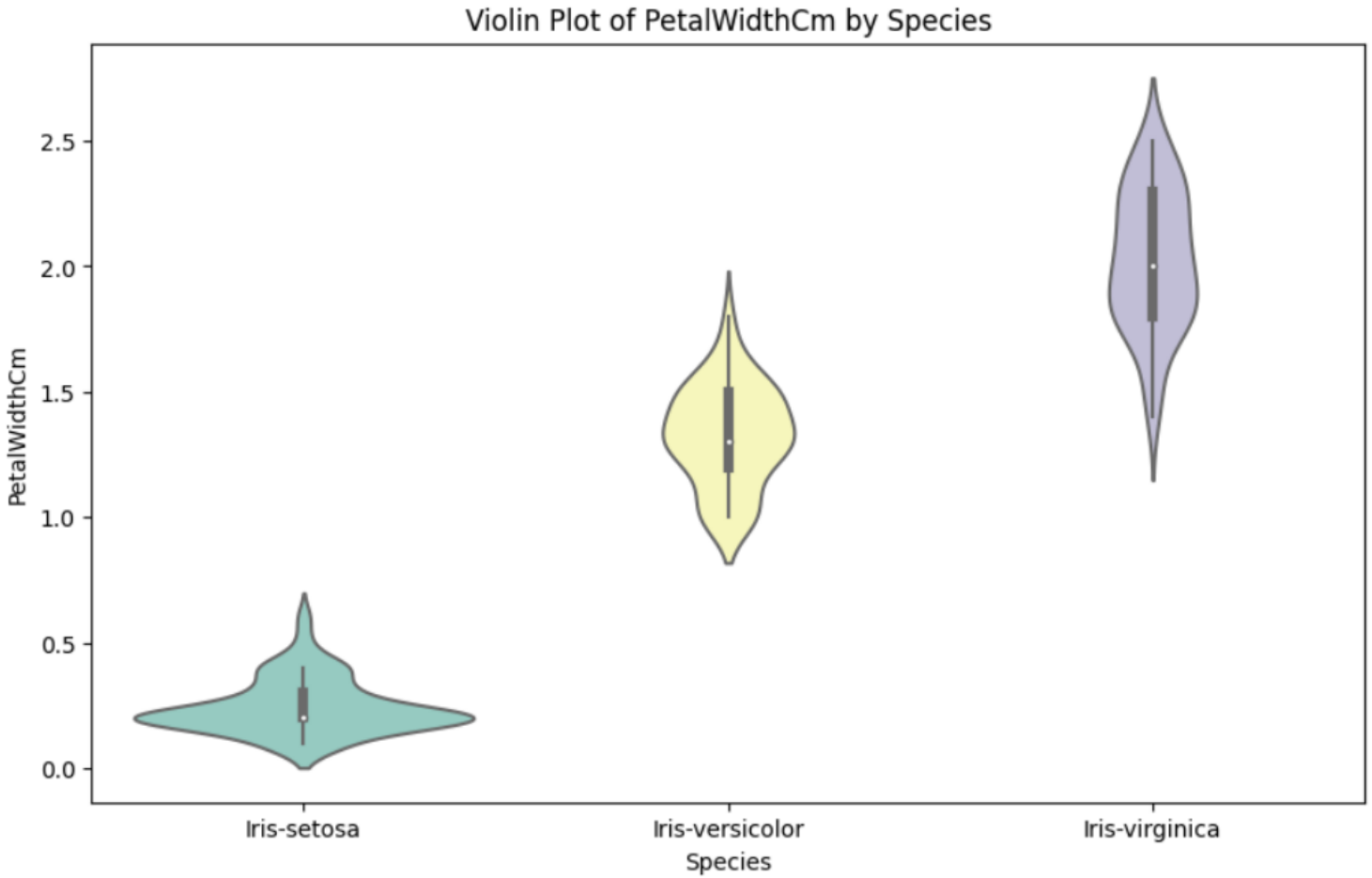
```
plt.figure(figsize = (10, 6))
sns.violinplot(x = 'Species', y = 'SepalLengthCm', data = iris, palette = 'Set3')
plt.title('Violin Plot of SepalLengthCm by Species')
plt.xlabel('Species')
plt.ylabel('SepalLengthCm')
plt.show()
```



데이터 분석

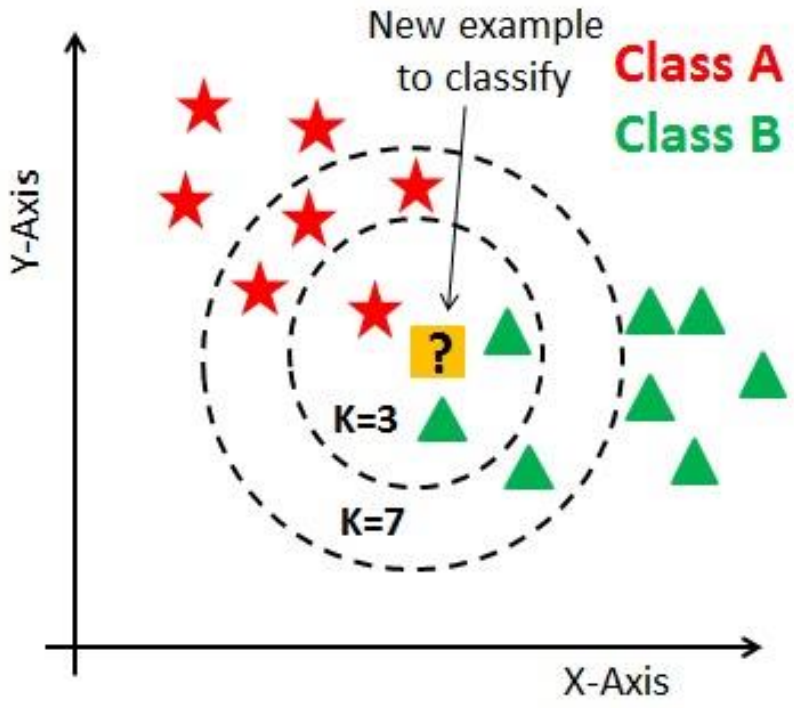
#품종별 꽃잎 너비

```
plt.figure(figsize = (10, 6))
sns.violinplot(x = 'Species', y = 'PetalWidthCm', data = iris, palette = 'Set3')
plt.title('Violin Plot of PetalWidthCm by Species')
plt.xlabel('Species')
plt.ylabel('PetalWidthCm')
plt.show()
```



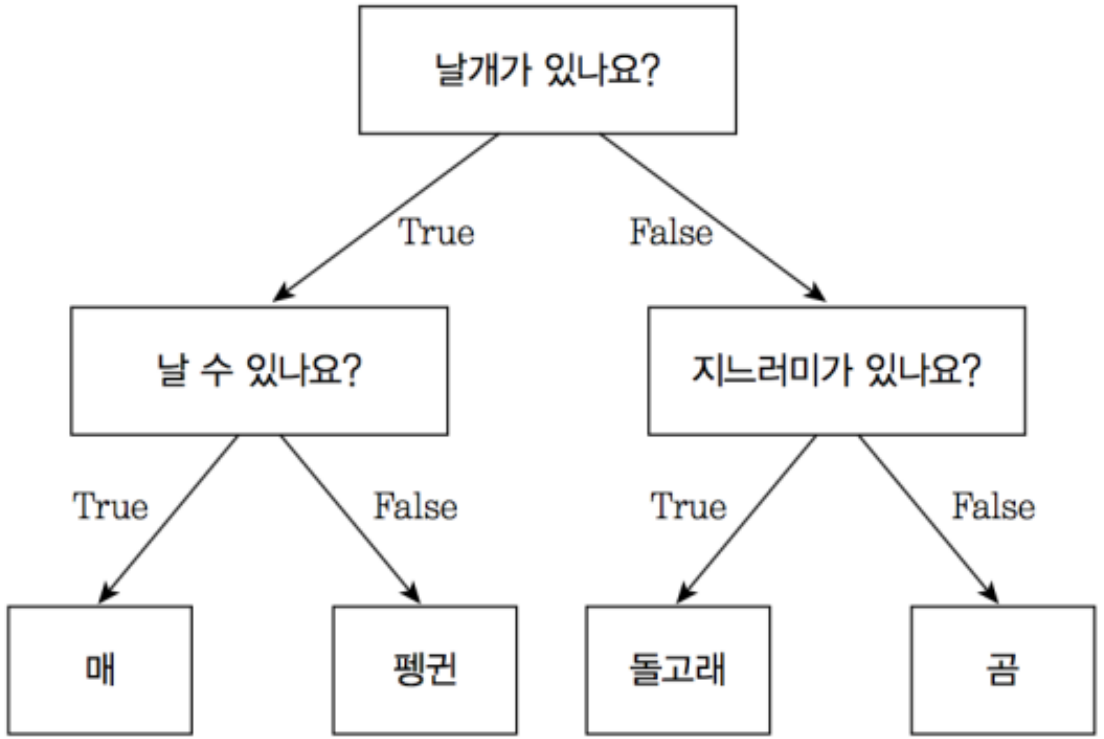
K-Nearest Neighbor (KNN)

- 어떤 데이터가 주어지면 그 주변(이웃)의 데이터를 살펴본 뒤 더 많은 데이터가 포함되어 있는 범주로 분류하는 방식
- KNN은 훈련이 따로 필요 없다.



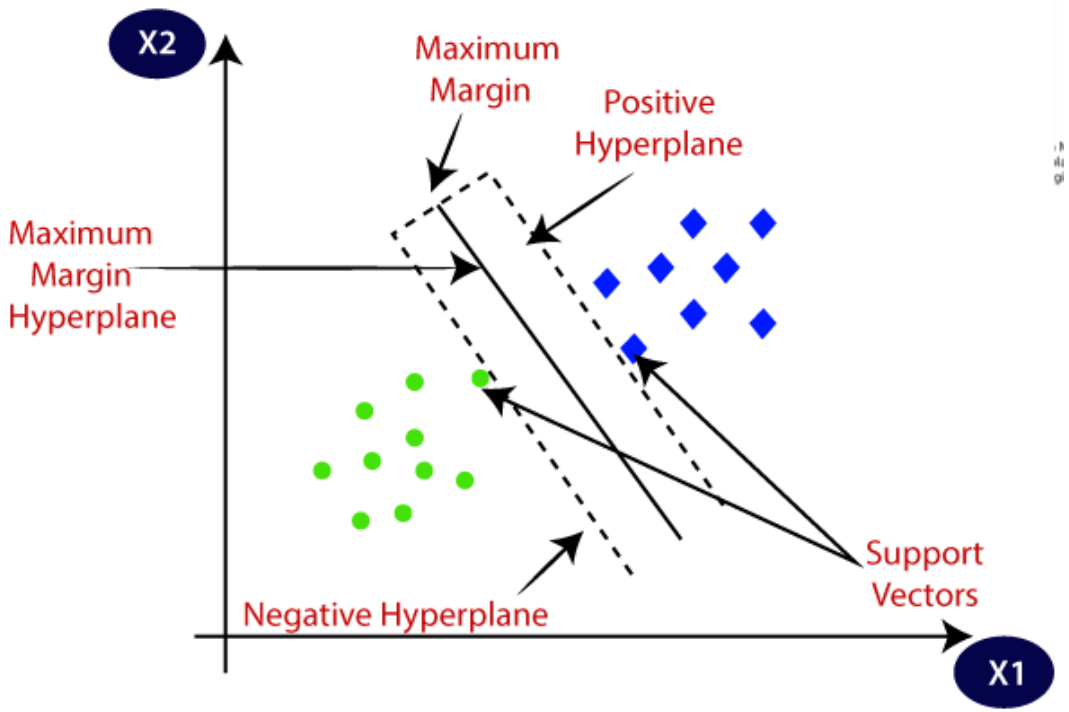
Decision Tree

- 결정 트리는 스무고개 하듯이 예/아니오 질문을 이어가며 학습
- 한번의 분기 때마다 변수 영역을 두 개로 구분
- 질문이나 정답을 담은 네모 상자를 노드(Node)



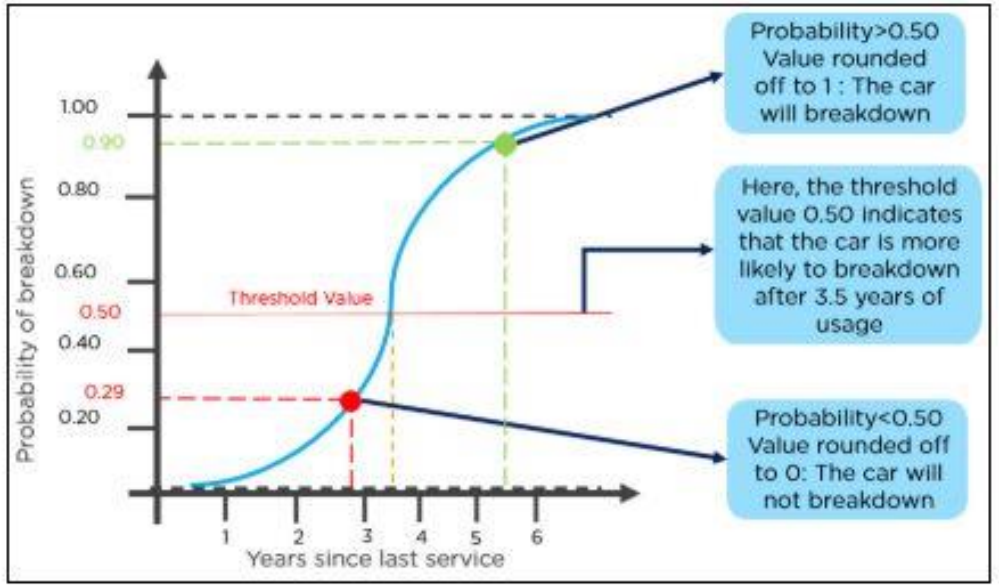
Support Vector Machine (SVM)

- 주어진 데이터가 어느 카테고리에 속할지 판단하는 이진 선형 분류 모델
- 가운데 선이 두 데이터를 더 적절히 구분하는 선
- 가운데 선이 Margin(선과 가장 가까운 양 옆 데이터와의 거리)을 최대
- 선과 가장 가까운 포인트를 서포트 벡터(Support vector)



Logistic Regression

- 로지스틱 회귀 모델은 일종의 확률 모델로서 독립 변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는데 사용되는 통계기법
- 종속 변수가 범주형 데이터를 대상으로 하며 입력 데이터가 주어졌을 때 해당 데이터의 결과가 특정 분류로 나뉘기 때문에 일종의 분류(classification) 기법
- 로지스틱 함수는 시그모이드 함수라고도 불리며, 이 함수의 값은 0과 1 사이에 위치



데이터 전처리

중복 데이터 제거

```
#check if there are any duplicated rows
iris.duplicated().sum()
```

```
iris.drop_duplicates(inplace = True)
```

```
iris.duplicated().sum()
```

```
0
```

결측치 확인

```
#number of nulls in each column
iris.isnull().sum()
```

```
#check unbalance in target
iris['Species'].value_counts()
```

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

데이터 전처리

- **Scaling**

```
from sklearn.preprocessing import MinMaxScaler

#features of the dataset
features = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']

#scale features using Min-Max scaler
scaler = MinMaxScaler()
iris[features] = scaler.fit_transform(iris[features])
```

-변수의 범위를 조절해주는 과정
-기본값은 0~1
-X : 데이터 셋 / x : 데이터 샘플

$$z = \frac{x - mean(X)}{std(X)}$$

결과

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	0.222222	0.625000	0.067797	0.041667	Iris-setosa
1	0.166667	0.416667	0.067797	0.041667	Iris-setosa
2	0.111111	0.500000	0.050847	0.041667	Iris-setosa
3	0.083333	0.458333	0.084746	0.041667	Iris-setosa
4	0.194444	0.666667	0.067797	0.041667	Iris-setosa
...	...	...	...	...	...
145	0.666667	0.416667	0.711864	0.916667	Iris-virginica
146	0.555556	0.208333	0.677966	0.750000	Iris-virginica
147	0.611111	0.416667	0.711864	0.791667	Iris-virginica
148	0.527778	0.583333	0.745763	0.916667	Iris-virginica
149	0.444444	0.416667	0.694915	0.708333	Iris-virginica

147 rows × 5 columns

데이터 전처리

- **Lable Encoding**

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

```
#encode the target variable using LabelEncoder
label_encoder = LabelEncoder()
iris['Species'] = label_encoder.fit_transform(iris['Species'])
```

-문자를 0부터 시작하는 정수형 숫자로 바꿔주는 기능

결과

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	0.222222	0.625000	0.067797	0.041667	0
1	0.166667	0.416667	0.067797	0.041667	0
2	0.111111	0.500000	0.050847	0.041667	0
3	0.083333	0.458333	0.084746	0.041667	0
4	0.194444	0.666667	0.067797	0.041667	0
...	...	...	...	...	...
145	0.666667	0.416667	0.711864	0.916667	2
146	0.555556	0.208333	0.677966	0.750000	2
147	0.611111	0.416667	0.711864	0.791667	2
148	0.527778	0.583333	0.745763	0.916667	2
149	0.444444	0.416667	0.694915	0.708333	2

147 rows × 5 columns

모델 불러오기

```
from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
from sklearn.model_selection import train_test_split #to split the dataset for training and testing
from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbours
from sklearn import svm #for Support Vector Machine (SVM) Algorithm
from sklearn import metrics #for checking the model accuracy
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
```

학습 테스트 세트 분리

```
train, test = train_test_split(iris, test_size = 0.3) # in this our main data is split into train and test
# the attribute test_size=0.3 splits the data into 70% and 30% ratio. train=70% and test=30%
print(train.shape)
print(test.shape

train_X = train[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
# taking the training data features
train_y=train.Species# output of our training data
test_X= test[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
# taking test data features
test_y =test.Species    #output value of test data
```

학습 테스트 세트 분리

```
train_X.head(2)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
128	0.583333	0.333333	0.779661	0.833333
109	0.805556	0.666667	0.864407	1.000000

```
test_X.head(2)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
75	0.638889	0.416667	0.576271	0.541667
25	0.194444	0.416667	0.101695	0.041667

```
train_y.head()  ##output of the training data
```

```
128    2
109    2
22     0
107    2
127    2
Name: Species, dtype: int64
```

모델 예측 & 성능 평가

```
model = svm.SVC() #select the algorithm
model.fit(train_X,train_y) # train the algorithm with the training data and the training output
prediction=model.predict(test_X) #pass the testing data to the trained algorithm
print('The accuracy of the SVM is:',metrics.accuracy_score(prediction,test_y))
#check the accuracy of the algorithm.
```

결과

The accuracy of the SVM is : 0.9555555555555556

모델 예측 & 성능 평가

```
model = LogisticRegression()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction,test_y))
```

```
The accuracy of the Logistic Regression is 0.7555555555555555
```

```
model=DecisionTreeClassifier()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
print('The accuracy of the Decision Tree is',metrics.accuracy_score(prediction,test_y))
```

```
The accuracy of the Decision Tree is 0.9555555555555556
```

데이터 분석 및 시각화

데이터 시각화를 해야 하는 이유

1. 많은 양의 데이터를 한눈에 볼 수 있다.
2. 데이터 분석에 대한 전문 지식이 없어도, 누구나 쉽게 데이터 인사이트를 찾을 수 있다.
3. 요약 통계보다 정확한 데이터 분석 결과를 도출 할 수 있다.
4. 효과적인 데이터 인사이트공유로 데이터 기반의 의사결정을 할 수 있다.


matplotlib

막대 차트 (Column Charts)

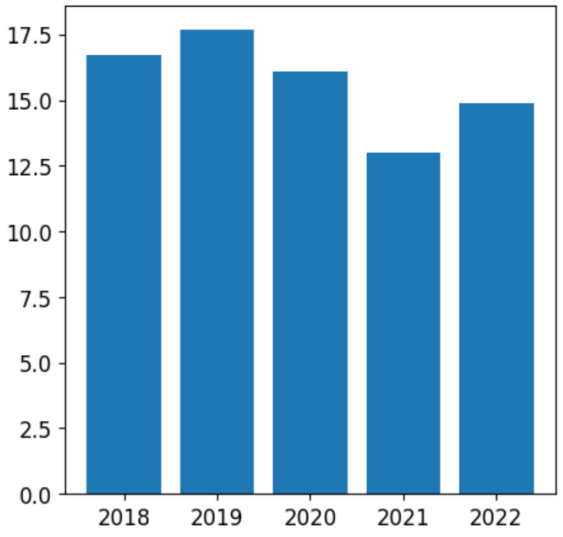
#막대그래프
plt.bar(x축,y축, 너비, *align='center'*)

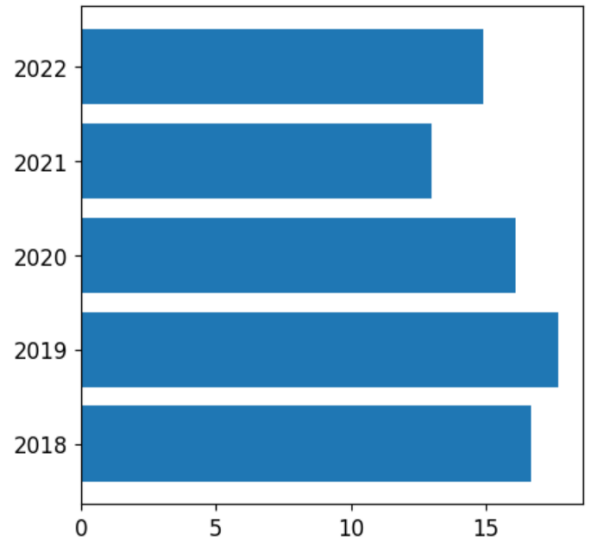
#폰트 설정
plt.rc('font', family = 'Malgun Gothic')
plt.rc('font', size = 12)

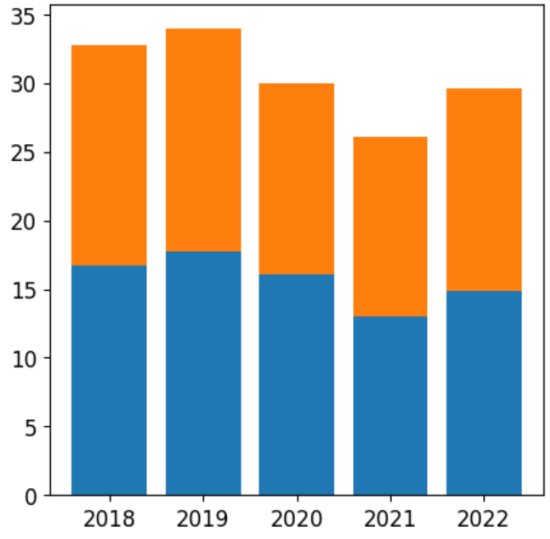
#누적 시키기
bottom = 데이터프레임[칼럼이름]

#x축, y축 바꾸기
plt.barh(x, y, align = 'center')

✓ 카테고리별로 값을 **비교**할 때 유용
예)각 과목별 성적 비교, 판매량 비교, 인구 수 비교



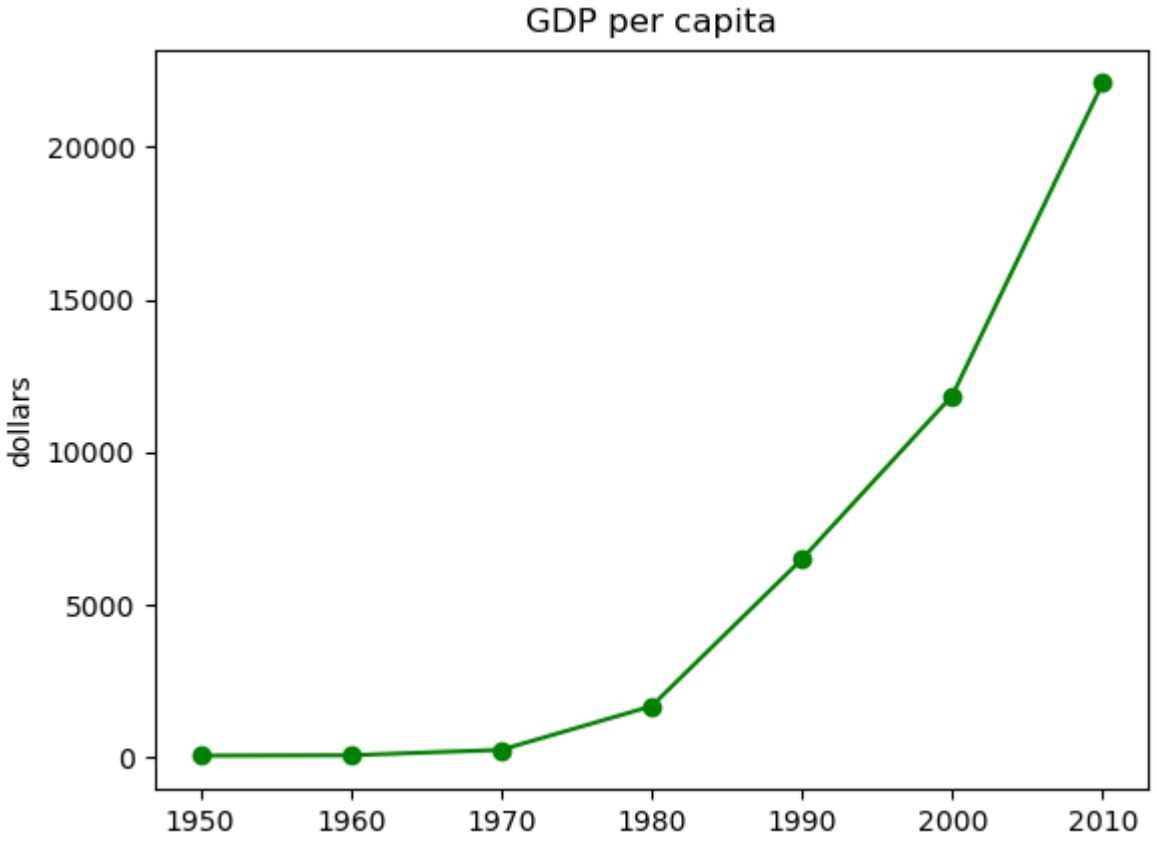




선 차트 (Line Charts)

```
#선그래프 그리기
plt.plot(years, gdp, color='green’,
 marker='o', linestyle='solid’)
```

- ✓ 연속 데이터 세트가 있는 경우 선 차트를 사용
- ✓ 데이터 포인트 수가 매우 많을 때 (20 개 이상) 일정 기간 동안의 추세 기반 데이터 시각화에 가장적합.
- ✓ 꺾은 선형 차트의 경우 값의 연속 또는 흐름 (추세)에 중점을 두지 만 데이터 마커를 사용하는 단일 값 비교에 대한 일부 지원이 있습니다 (데이터 포인트가 20 개 미만인 경우에만).
- ✓ 선 차트는 차트가 작을 때 세로 막대 차트에 대한 좋은 대안.

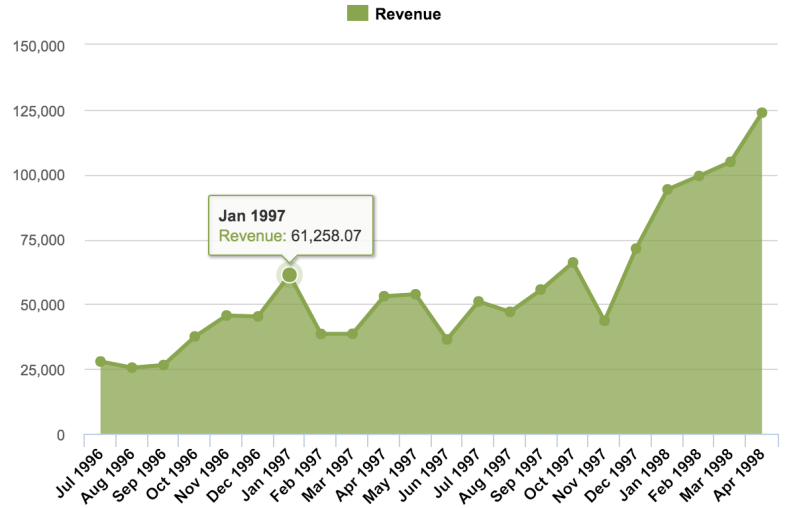


날씨 변화, 성적 변화, 주가 변동

영역 차트 (Area Charts)

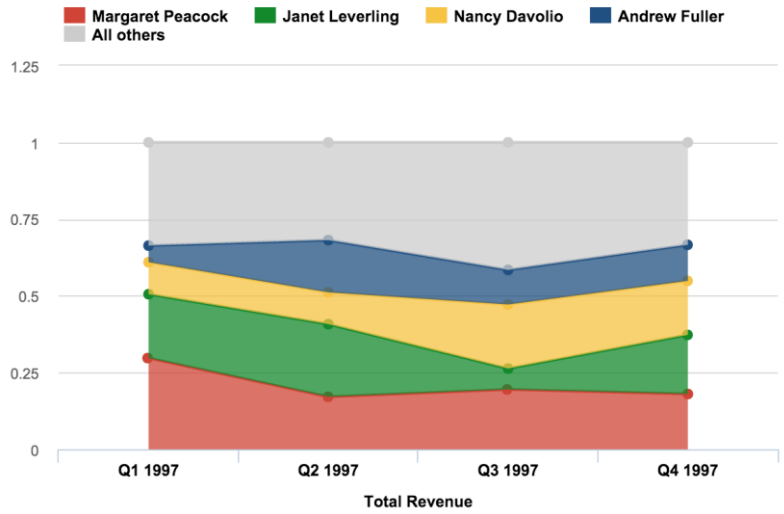
```
# 영역차트
fig, ax = plt.subplots()
ax.fill_between(x = x, y1 = y)

# 누적영역차트
plt.stackplot(beernum, price, amount, country)
```



- ✓ 영역 차트는 기본적으로 꺾은 선형 차트로 추세 및 일부 비교에 적합.
- ✓ 영역 차트는 선 아래 영역을 채우는 그래프.
- ✓ 시간에 따른 변화를 보여주는 동시에, 누적된 양을 강조

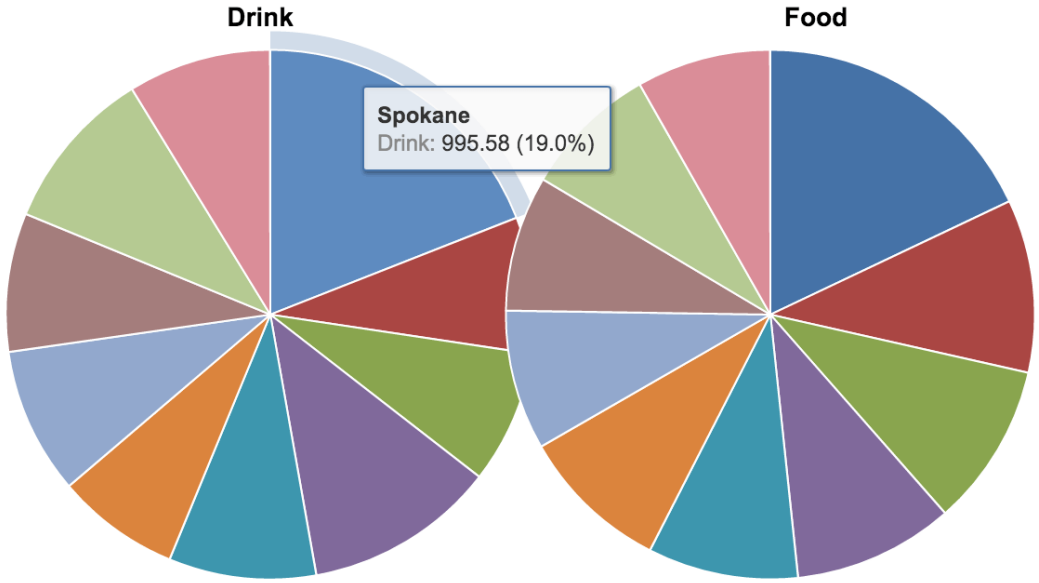
웹사이트 방문자 수, 매출 증가, 자원 소비량



파이 차트

ax.pie(frequency, ## 파이차트 출력 labels=labels, ## 라벨 출력 startangle=90, ## 시작점을 90도(degree)로 지정 counterclock=True, ## 반시계 방향으로 파이차트를 그린다. autopct=lambda p : '{:.2f}%'.format(p) ## 퍼센티지 출력)

- ✓ **파이 차트는 데이터를 원 모양으로 나누어, 각 부분이 전체 중에서 차지하는 비율을 시각적으로 보여주는 그래프**
- ✓ **원형 차트는 개별 섹션을 서로 비교하거나 정확한 값을 나타 내기위한 것이 아님. (바 차트를 사용해야 함).**
- ✓ **모든 세그먼트의 총합이 100 % 인지 확인 .**
- ✓ **집중하고 싶은 명확한 승자 가없는 한, 카테고리가 6 개 미만인 경우에만 원형 차트를 사용.**

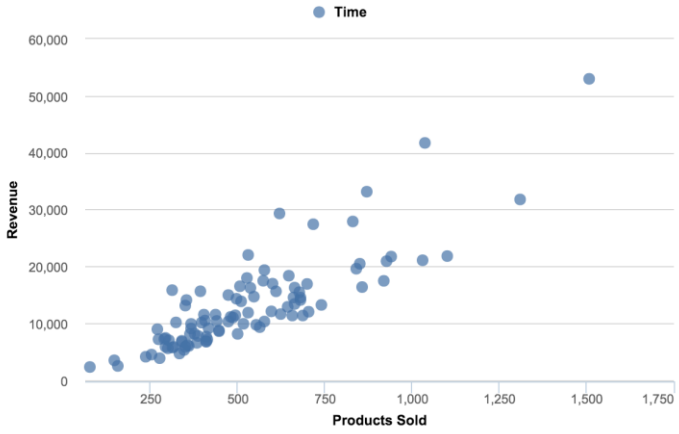


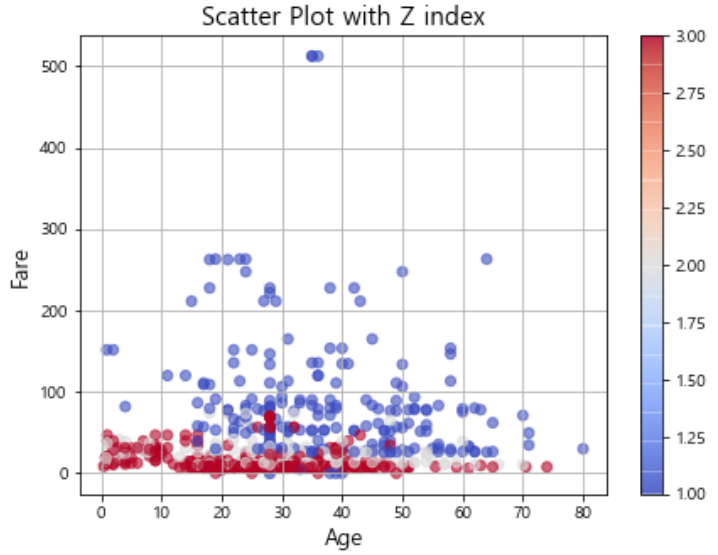
학교 반별 학생 수, 설문조사 결과, 예산 분포

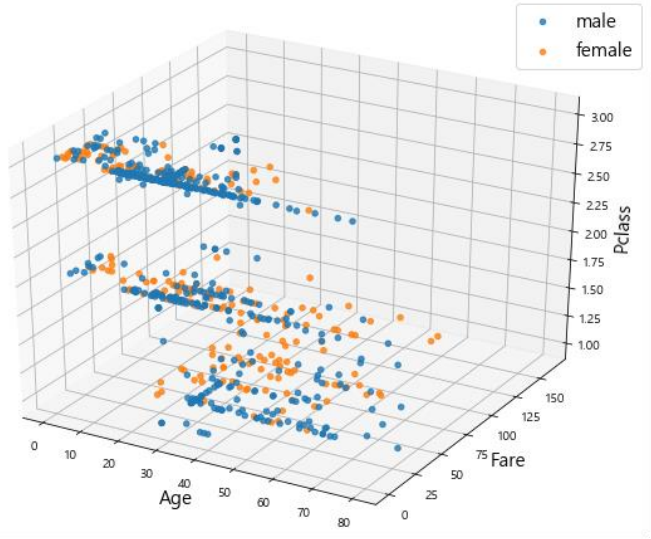
분산 차트 (Scatter Charts)

plt.scatter(x, y, color='orange', s=30)

- ✓ **분산 차트는 주로 상관 관계 및 분포 분석에 사용된다.**
- ✓ **하나가 다른 변수와 연관되거나 연관되지 않는 두 개의 서로 다른 변수 간의 관계를 표시하는 데 좋은 그래프다.**
- ✓ **분산 차트는 데이터 분포 또는 클러스터링 추세를 표시하고 이상 점이나 이상 값을 파악하는 데 도움이 된다.**
- ✓ **마케팅 지출과 수익을 보여 줄때 좋은 방식이다.**





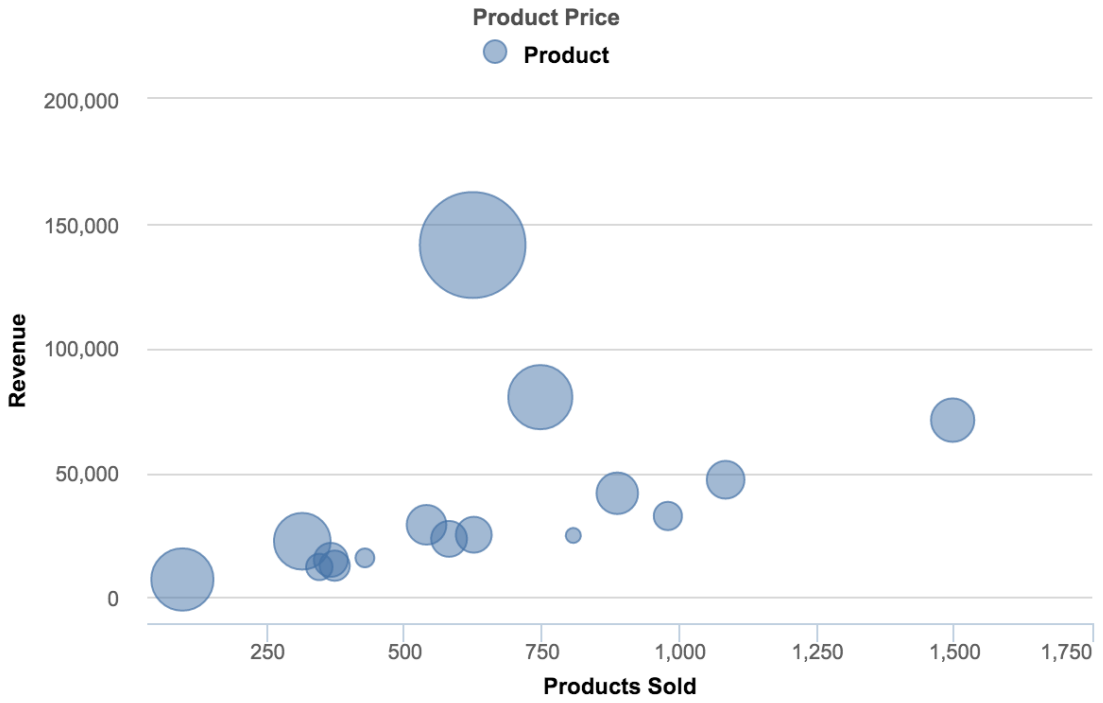


공부 시간(, 수면 시간)과 성적, 나이와 소득, 키와 몸무게(,나이)

버블 차트 (Bubble Charts)

plt.scatter(x, y, s=z*1000,c=colors)

- ✓ **버블 차트는 분산 차트에 다른 차원을 추가해야하는 경우 좋은 옵션입니다.**
- ✓ **분산은 두 값을 비교하지만 버블의 크기를 세 번째 변수로 추가하여 비교할 수 있습니다.**
- ✓ **버블의 크기가 매우 비슷하면 라벨을 사용 할 수 있다.**

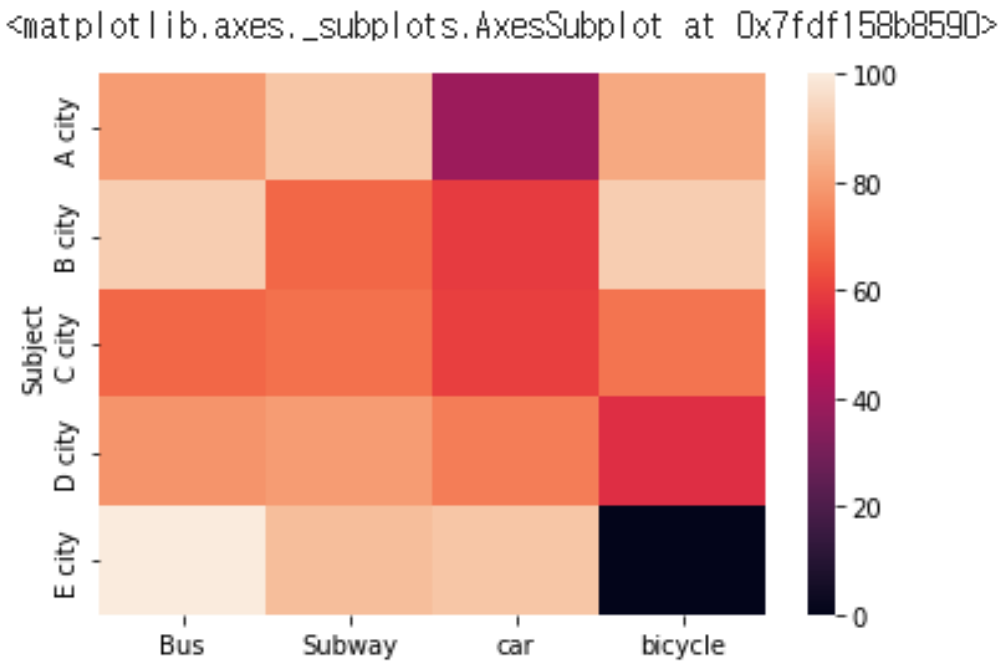


도시 별 인구, 평균 소득, 공기 오염도 / 상품 가격, 판매량, 이익률 /학생의 공부 시간, 수면 시간, 성적

히트맵(Heatmap)

sns.heatmap(df)

- ✓ **Heatmap은 열을 뜻하는 히트와 지도를 뜻하는 맵을 결합시킨 그래프로 3차원 데이터를 2차원의 보기 쉬운 형태로 나타낼 수 있다.**
- ✓ **값의 크거나 낮음을 한 눈에 알아보기 쉽고, 어디에 집중되어 있는지 파악하기 쉽다.**
- ✓ **이상치(outlier) 또는 비정상적인 값을 쉽게 발견할 수 있다.**
- ✓ **행과 열이 되는 데이터는 보통 명목변수나 이산형 변수가 많이 쓰이고 색상으로 표현할 수 있는 데이터는 연속형도 가능하다.**



https://www.kaggle.com/

ChatGPT 

 Copilot