ch.10 Working with Data

```
from collections import Counter, defaultdict
from functools import partial, reduce
from linear_algebra import shape, get_row, get_column, make_matrix, \
    vector_mean, vector_sum, dot, magnitude, vector_subtract, scalar_multiply
from stats import correlation, standard_deviation, mean
from probability import inverse_normal_cdf
from gradient_descent import maximize_batch
import math, random, csv
import matplotlib.pyplot as plt
import dateutil.parser
```

데이터 탐색

1차원 데이터 탐색

- 가장 간단한 경우는 1차원 데이터 세트가 있는 경우입니다
- 가장 작은, 가장 큰, 평균 및 표준 편차 히스토그램

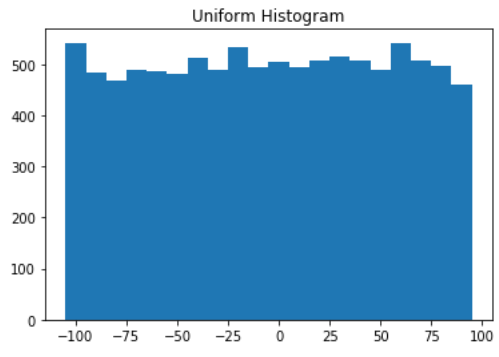
```
def bucketize(point, bucket_size):
    return bucket_size * math.floor(point / bucket_size)

def make_histogram(points, bucket_size):
    return Counter(bucketize(point, bucket_size) for point in points)

def plot_histogram(points, bucket_size, title=""):
    histogram = make_histogram(points, bucket_size)
    plt.bar(histogram.keys(), histogram.values(), width=bucket_size)
    plt.title(title)
    plt.show()
```

- **데이터 포인트 버킷화**: `bucketize` 함수는 주어진 포인트를 버킷 크기(`bucket_size`)로 나눈 후 내림하여, 버킷의 시작 값을 계산합니다. 예를 들어, 포인트 5와 버킷 크기 2가 주어지면 4로 버킷화합니다.
- **히스토그램 데이터 생성**: `make_histogram` 함수는 모든 데이터 포인트를 버킷화하고, `Counter`를 사용하여 각 버킷에 몇 개의 포인트가 있는지 셉니다.
- **히스토그램 그리기**: `plot_histogram` 함수는 `make_histogram`을 호출하여 얻은 버킷별 포인트 수를 바 그래프로 그립니다. `plt.bar`를 사용하여 각 버킷의 시작 값과 해당 버킷에 속하는 포인트 수로 히스토그램을 생성합니다.

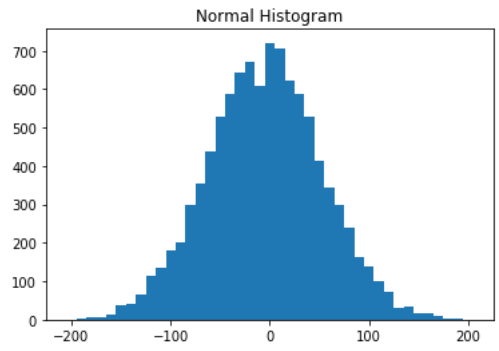
```
import random

# 난수 생성기 시드 설정
random.seed(0)

# -100에서 100 사이의 균등 분포 샘플 생성
uniform = [200 * random.random() - 100 for _ in range(10000)]

# 평균 0, 표준편차 57인 정규 분포 샘플 생성
normal = [57 * inverse_normal_cdf(random.random()) for _ in range(10000)]
```

- 두 개의 서로 다른 확률 분포에서 샘플을 생성하는 과정을 보여줍니다:
- 균등 분포와 정규 분포. 먼저, `random.seed(0)`을 사용하여 난수 생성기를 초기화합니다.
- 그런 다음, 각각의 분포에서 10,000개의 샘플을 생성합니다.

```
plot_histogram(uniform, 10, "Uniform Histogram")
```



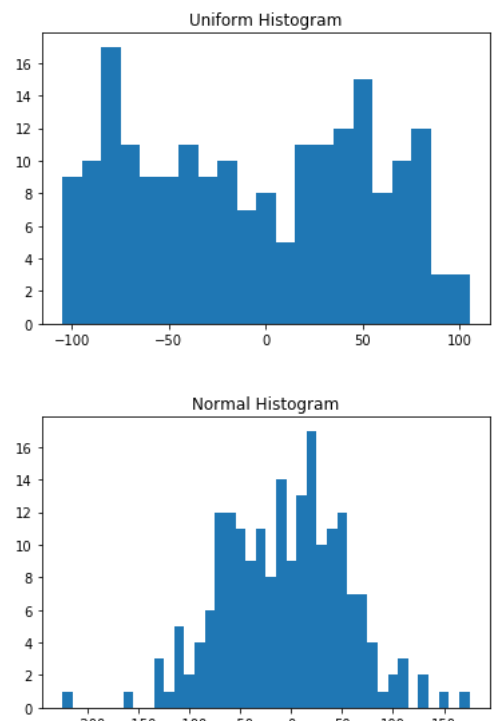

```
plot_histogram(normal, 10, "Normal Histogram")
```




```
def compare_two_distributions():
    random.seed(0)

    # 균등 분포 샘플 생성
    uniform = [random.randrange(-100, 101) for _ in range(200)]

    # 정규 분포 샘플 생성
    normal = [57 * inverse_normal_cdf(random.random()) for _ in range(200)]

    # 균등 분포 히스토그램 그리기
    plot_histogram(uniform, 10, "Uniform Histogram")

    # 정규 분포 히스토그램 그리기
    plot_histogram(normal, 10, "Normal Histogram")
```

1. **난수 생성기 시드 설정**: `random.seed(0)`을 통해 난수 생성기의 초기 상태를 설정합니다.
2. **균등 분포 샘플 생성**: `random.randrange(-100, 101)`을 사용하여 -100에서 100 사이의 정수 200개를 생성합니다.
3. **정규 분포 샘플 생성**: `inverse_normal_cdf(random.random())`을 사용하여 평균 0, 표준편차 57인 정규 분포에서 200개의 샘플을 생성합니다.
4. **히스토그램 그리기**: `plot_histogram` 함수를 통해 각각의 분포에 대한 히스토그램을 그립니다.

```
compare_two_distributions()
```



Two Dimensions

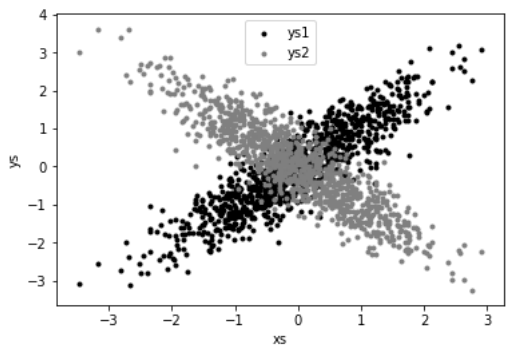
- 이제 2차원 데이터 세트가 있습니다.
- 먼저 각 차원을 개별적으로 이해하고 싶습니다.
- 그런 다음 데이터를 산포하려고 할 수도 있습니다.

```
# 누적 분포 함수(CDF)의 역함수 (구현이 필요함)
def inverse_normal_cdf(p):
    """표준 정규 분포에 대한 누적 분포 함수(CDF)의 역함수를 근사합니다."""
    # 이 함수는 자리 표시자 함수입니다. 실제 구현이 필요합니다.
    return math.sqrt(2) * math.erfinv(2 * p - 1)

def random_normal():
    """표준 정규 분포에서 무작위 값을 반환합니다."""
    return inverse_normal_cdf(random.random())

# 표준 정규 분포에서 1000개의 샘플을 생성하여 xs 리스트에 저장
xs = [random_normal() for _ in range(1000)]

# xs 값을 기반으로 ys1과 ys2 값을 생성 (잡음 추가)
```

```
ys1 = [x + random_normal() / 2 for x in xs]
ys2 = [-x + random_normal() / 2 for x in xs]
```

```
def scatter():
    """(xs, ys1)과 (xs, ys2)의 산점도를 그립니다."""
    plt.scatter(xs, ys1, marker='.', color='black', label='ys1')
    plt.scatter(xs, ys2, marker='.', color='gray', label='ys2')
    plt.xlabel('xs')
    plt.ylabel('ys')
    plt.legend(loc=9)  # 범례를 상단 중앙에 위치시킴
    plt.show()

# 산점도 함수 호출하여 플롯 그리기
scatter()
```

- **정규 분포 샘플 생성**: `random_normal` 함수는 표준 정규 분포에서 샘플을 생성합니다.
- **데이터 생성**:
  - `xs`: 표준 정규 분포에서 생성된 1000개의 샘플.
  - `ys1`: `xs` 값에 표준 정규 분포에서 생성된 값을 2로 나눈 값을 더해 생성.
  - `ys2`: `xs` 값에 표준 정규 분포에서 생성된 값을 2로 나눈 값을 빼서 생성.
- **산점도 플롯**:
  - `scatter` 함수는 `xs`와 `ys1`, `xs`와 `ys2`의 산점도를 그립니다.
  - 각 산점도는 다른 색상과 레이블로 구분됩니다.
  - x축과 y축에 레이블을 추가하고, 범례를 상단 중앙에 위치시킵니다




```
# xs와 ys1, xs와 ys2의 상관 계수를 출력
print(correlation(xs, ys1))
print(correlation(xs, ys2))
```

0.891585944012268
-0.8939937075957362

Many Dimensions

- 모든 차원이 서로 어떻게 관련되어 있는지 알고 싶습니다.
- 간단한 접근법은 상관 행렬을 보는 것입니다
- 더 시각적인 접근법(차원이 너무 많지 않은 경우)은 모든 쌍별 산점도를 나타내는 산점도 행렬을 만드는 것입니다.

```
def correlation_matrix(data):
    """데이터의 각 열 간 상관 계수를 구하여 행렬로 반환합니다."""
    _, num_columns = shape(data)

    def matrix_entry(i, j):
        return correlation(get_column(data, i), get_column(data, j))

    return make_matrix(num_columns, num_columns, matrix_entry)
```

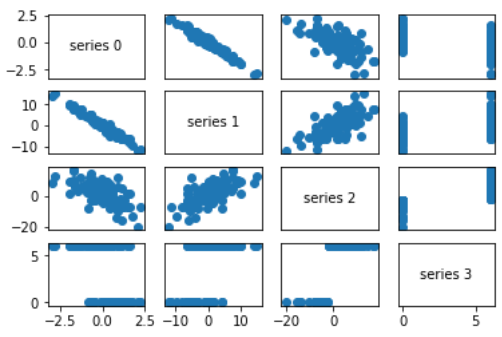
- `shape`: 행렬의 형태(행 수, 열 수)를 반환.
- `get_column`: 행렬의 특정 열을 추출.
- `make_matrix`: 주어진 함수로 행렬을 생성.
- `correlation_matrix`: 데이터의 각 열 간 상관 계수를 구하여 상관 행렬을 생성.
- 데이터의 각 열에 대해 상관 계수를 계산하고, 이를 행렬 형태로 반환.

```
def make_scatterplot_matrix():
    """산점도 행렬을 생성합니다."""

    # 먼저, 무작위 데이터를 생성합니다.
    num_points = 100

    def random_row():
        row = [None, None, None, None]
        row[0] = random_normal()
        row[1] = -5 * row[0] + random_normal()
        row[2] = row[0] + row[1] + 5 * random_normal()
        row[3] = 6 if row[2] > -2 else 0
        return row
```

```
random.seed(0)
data = [random_row() for _ in range(num_points)]

# 그런 다음, 데이터를 플로팅합니다.
_, num_columns = shape(data)
fig, ax = plt.subplots(num_columns, num_columns)

for i in range(num_columns):
    for j in range(num_columns):

        # x축에 column_j, y축에 column_i를 산점도로 그립니다.
        if i != j:
            ax[i][j].scatter(get_column(data, j), get_column(data, i))
        # i == j인 경우, 시리즈 이름을 표시합니다.
        else:
            ax[i][j].annotate("series " + str(i), (0.5, 0.5),
                              xycoords='axes fraction',
                              ha="center", va="center")

        # 왼쪽과 아래쪽 차트를 제외한 축 라벨을 숨깁니다.
        if i < num_columns - 1:
            ax[i][j].xaxis.set_visible(False)
        if j > 0:
            ax[i][j].yaxis.set_visible(False)

# 텍스트만 포함된 차트의 축 라벨을 수정합니다.
ax[-1][-1].set_xlim(ax[0][-1].get_xlim())
ax[0][0].set_ylim(ax[0][1].get_ylim())

plt.show()

# 상관 행렬을 출력합니다.
print('### Correlation Matrix ###')
print(np.array(correlation_matrix(data)))
print()
print('### Correlation Matrix using numpy.corrcoef() ###')
print(np.corrcoef(np.array(data).T))
```

- **무작위 데이터 생성**:
  - `random_row` 함수는 4개의 값을 가지는 리스트를 생성합니다.
    - `row[0]`은 표준 정규 분포에서 추출된 값.
    - `row[1]`은 `row[0]`의 선형 변환과 잡음을 더한 값.
    - `row[2]`은 `row[0]`과 `row[1]`의 합에 잡음을 더한 값.
    - `row[3]`은 `row[2]`가 -2보다 크면 6, 그렇지 않으면 0.
  - `random.seed(0)`을 통해 난수 생성기를 초기화하고, 100개의 무작위 행을 생성합니다.
- **산점도 행렬 생성**:
  - 각 열 간의 산점도를 플로팅합니다.
  - 대각선에는 해당 시리즈의 이름을 표시합니다.
  - 왼쪽과 아래쪽 차트를 제외한 나머지 차트의 축 라벨을 숨깁니다.
- **상관 행렬 출력**:
  - 생성된 데이터의 상관 행렬을 계산하고 출력합니다.
  - `numpy.corrcoef`를 사용하여 상관 행렬을 계산하고 이를 비교합니다.

```
make_scatterplot_matrix()
```




```
### Correlation Matrix ###
[[ 1.         -0.98113582 -0.66174938 -0.51732231]
 [-0.98113582  1.          0.68181854  0.54211527]
 [-0.66174938  0.68181854  1.          0.74614437]
 [-0.51732231  0.54211527  0.74614437  1.        ]]

### Correlation Matrix using numpy.corrcoef() ###
[[ 1.         -0.98113582 -0.66174938 -0.51732231]
 [-0.98113582  1.          0.68181854  0.54211527]
 [-0.66174938  0.68181854  1.          0.74614437]
 [-0.51732231  0.54211527  0.74614437  1.        ]]
```

Cleaning and Munging

- 실제 데이터는 더럽습니다.
- 종종 사용하기 전에 작업을 수행해야 합니다.
- 없음을 사용하여 "이 열에 대해 아무것도 수행하지 않음"을 나타냅니다
- 실제 데이터 처리를 처리할 때 다음 코드와 유사한 파서를 작성합니다
- Munge(컴퓨터 슬랭): (원 데이터를) 조작하고, 특히 한 형식에서 다른 형식으로 (데이터를) 변환합니다.

Example: comma_delimited_stock_prices.csv

```
def parse_row(input_row, parsers):
    """주어진 파서 리스트를 이용해 input_row의 각 요소에 적절한 파서를 적용합니다."""
    return [parser(value) if parser is not None else value
            for value, parser in zip(input_row, parsers)]

def parse_rows_with(reader, parsers):
    """reader를 래핑하여 각 행에 파서를 적용합니다."""
    for row in reader:
        yield parse_row(row, parsers)
```

- `parse_row` **함수**:
  - `input_row`의 각 요소에 대해 `parsers` 리스트에 있는 적절한 파서를 적용합니다.
  - `parsers` 리스트는 `input_row`의 각 요소에 대해 사용할 파서를 정의합니다.
  - `parser(value) if parser is not None else value` 구문을 통해, 파서가 `None`이 아닌 경우 해당 파서를 적용하고, `None`인 경우 원래 값을 그대로 사용합니다.
  - `zip(input_row, parsers)`를 통해 `input_row`와 `parsers`를 동시에 순회하며 파서를 적용합니다.
- `parse_rows_with` **함수**:
  - `reader`의 각 행(row)에 대해 `parsers`를 적용하여 파싱된 행을 생성합니다.
  - `for row in reader` 루프를 통해 `reader`의 각 행을 순회하며, `parse_row(row, parsers)`를 사용하여 파싱된 행을 생성합니다.
  - `yield`를 사용하여 파싱된 행을 하나씩 반환합니다. 이를 통해 `parse_rows_with` 함수는 제너레이터로 작동합니다.

```
!type comma_delimited_stock_prices.csv
```

6/20/2014,AAPL,90.91
6/20/2014,MSFT,41.68
6/20/3014,FB,64.5
6/19/2014,AAPL,91.86
6/19/2014,MSFT,n/a
6/19/2014,FB,64.34

```
import csv
import dateutil.parser

data = []

# CSV 파일을 열고 데이터를 읽습니다.
with open("comma_delimited_stock_prices.csv", "r") as f:
    reader = csv.reader(f)
    # 각 행에 대해 날짜 파싱, 두 번째 열 그대로, 세 번째 열은 float로 변환
    for line in parse_rows_with(reader, [dateutil.parser.parse, None, float]):
        data.append(line)

# 각 행을 순회하며 None 값이 있는 행을 출력합니다.
for row in data:
    if any(x is None for x in row):
        print(row)
```

[datetime.datetime(2014, 6, 19, 0, 0), 'MSFT', None]

```
def try_or_none(f):
    """f를 래핑하여 f가 예외를 발생시키면 None을 반환합니다.
    f는 하나의 입력만 받는다고 가정합니다."""

    def f_or_none(x):
        try:
            return f(x)  # f를 호출하고 결과를 반환
        except:
            return None  # 예외가 발생하면 None을 반환

    return f_or_none  # f_or_none 함수를 반환
```

```
def parse_row(input_row, parsers):
    """주어진 파서 리스트를 이용해 input_row의 각 요소에 적절한 파서를 적용합니다.
    파서가 예외를 발생시키면 None을 반환합니다."""
```

```
    return [try_or_none(parser)(value) if parser is not None else value
            for value, parser in zip(input_row, parsers)]
```

- `parse_row` 함수는 입력된 행(row)의 각 요소에 대해 지정된 파서를 적용하고, 파서가 예외를 발생시킬 경우 `None`을 반환

Stocks.txt

```
def try_parse_field(field_name, value, parser_dict):
    """parser_dict에서 적절한 함수를 사용해 value를 파싱하려고 시도합니다."""
    parser = parser_dict.get(field_name)  # parser_dict에서 field_name에 해당하는 파서를 가져옵니다. 없으면 None
    if parser is not None:
        return try_or_none(parser)(value)  # 파서가 있으면 try_or_none으로 안전하게 호출
    else:
        return value  # 파서가 없으면 원래 값을 반환

def parse_dict(input_dict, parser_dict):
    """input_dict의 각 필드에 대해 parser_dict의 파서를 적용하여 파싱된 딕셔너리를 반환합니다."""
    return { field_name : try_parse_field(field_name, value, parser_dict)
             for field_name, value in input_dict.items() }  # 각 필드에 대해 try_parse_field를 호출하여 새로운 딕셔너리를 만듦
```

- 주어진 입력 딕셔너리(`input_dict`)에서 각 필드에 대해 지정된 파서(`parser_dict`)를 적용하여 파싱된 결과를 반환합니다.
- `try_parse_field` 함수는 필드별로 적절한 파서를 적용하며, 예외 발생 시 `None`을 반환하도록 안전하게 처리

```
!type stocks.txt
```

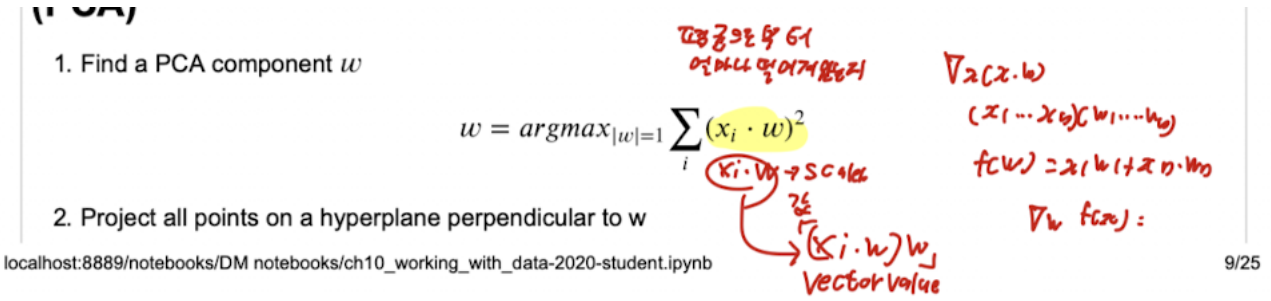
```
symbol  date    closing_price
AAPL    2015-01-23  112.98
AAPL    2015-01-22  112.4
AAPL    2015-01-21  109.55
AAPL    2015-01-20  108.72
AAPL    2015-01-16  105.99
AAPL    2015-01-15  106.82
AAPL    2015-01-14  109.8
AAPL    2015-01-13  110.22
AAPL    2015-01-12  109.25
AAPL    2015-01-09  112.01
AAPL    2015-01-08  111.89
AAPL    2015-01-07  107.75
AAPL    2015-01-06  106.26
AAPL    2015-01-05  106.25
AAPL    2015-01-02  109.33
AAPL    2014-12-31  110.38

....

FB  2012-06-04  26.9
FB  2012-06-01  27.72
FB  2012-05-31  29.6
FB  2012-05-30  28.19
FB  2012-05-29  28.84
FB  2012-05-25  31.91
FB  2012-05-24  33.03
FB  2012-05-23  32
FB  2012-05-22  31
FB  2012-05-21  34.03
FB  2012-05-18  38.23
```

```
import csv
import dateutil.parser

# "stocks.txt" 파일을 열고 데이터를 읽습니다.
with open("stocks.txt", "r", encoding='utf8', newline='') as f:
    reader = csv.DictReader(f, delimiter="\t")  # 탭으로 구분된 파일을 읽음
    # 각 행(row)을 파싱하여 데이터 리스트를 생성합니다.
    data = [parse_dict(row, { 'date' : dateutil.parser.parse,
                              'closing_price' : float })
            for row in reader]

# 데이터의 총 길이를 출력합니다.
print(len(data))
# 처음 두 개의 행을 출력합니다.
print(data[:2])

# AAPL의 최대 종가(closing_price)를 계산합니다.
max_aapl_price = max(row["closing_price"]
                     for row in data
                     if row["symbol"] == "AAPL")
# 최대 종가를 출력합니다.
print("max aapl price", max_aapl_price)
```

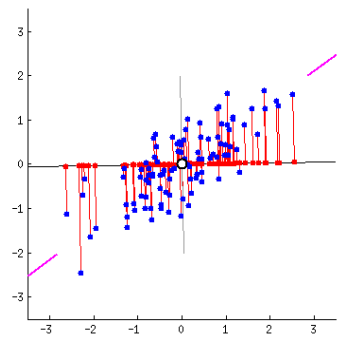
- "stocks.txt" 파일에서 주식을 읽어와 파싱한 후, 'AAPL' 주식의 최대 종가를 계산하고 출력

```
16555
[{'symbol': 'AAPL', 'date': datetime.datetime(2015, 1, 23, 0, 0), 'closing_price': 112.98}, {'symbol': 'AAPL', 'date': datetime.datetime(2015, 1
max aapl price 119.0
```

Dimensionality Reduction: Principal Component Analysis (PCA)



1. PCA 구성 요소 w 찾기
   $w=argmax|w|=1\sum i$
   $(xi\cdot w)2$
   - $(xi\cdot w)2$: 평균으로 부터 얼마나 떨어져 있는지 → Vector Value
2. 모든 점을 w에 수직인 초평면에 투영
3. 1, 2단계를 반복합니다

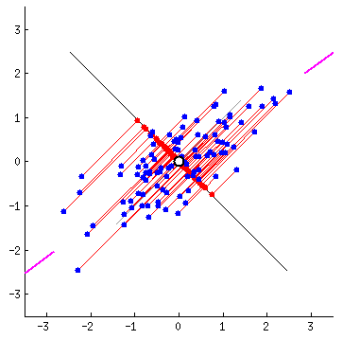


```
def scale(data_matrix):
    """데이터 행렬을 스케일링하기 위해 각 열의 평균과 표준편차를 계산합니다."""
    num_rows, num_cols = shape(data_matrix)  # 행렬의 행 수와 열 수를 구합니다
    means = [mean(get_column(data_matrix, j))  # 각 열의 평균을 계산합니다
             for j in range(num_cols)]
    stdevs = [standard_deviation(get_column(data_matrix, j))  # 각 열의 표준편차를 계산합니다
              for j in range(num_cols)]
    return means, stdevs  # 평균과 표준편차 리스트를 반환합니다
```

```
def shape(matrix):
    """행렬의 형태를 반환합니다: (행 수, 열 수)."""
    num_rows = len(matrix)
    num_cols = len(matrix[0]) if matrix else 0
    return num_rows, num_cols

def mean(xs):
    """리스트 xs의 평균을 계산합니다."""
    return sum(xs) / len(xs)

def standard_deviation(xs):
    """리스트 xs의 표준편차를 계산합니다."""
    return math.sqrt(variance(xs))

def variance(xs):
    """리스트 xs의 분산을 계산합니다."""
    assert len(xs) >= 2, "분산을 계산하려면 2개 이상의 데이터가 필요합니다."
    mean_xs = mean(xs)
    return sum((x - mean_xs) ** 2 for x in xs) / (len(xs) - 1)

def dot(v, w):
    """벡터 v와 w의 내적을 계산합니다."""
    return sum(v_i * w_i for v_i, w_i in zip(v, w))

def scalar_multiply(c, v):
    """스칼라 c와 벡터 v의 곱을 계산합니다."""
    return [c * v_i for v_i in v]

def vector_sum(vectors):
    """벡터들의 합을 계산합니다."""
    return [sum(vector[i] for vector in vectors) for i in range(len(vectors[0]))]

def vector_subtract(v, w):
    """벡터 v에서 벡터 w를 뺍니다."""
```

```
    return [v_i - w_i for v_i, w_i in zip(v, w)]

def make_matrix(num_rows, num_cols, entry_fn):
    """주어진 함수 entry_fn으로 행렬을 생성합니다."""
    return [[entry_fn(i, j) for j in range(num_cols)]
            for i in range(num_rows)]

def de_mean_matrix(A):
    """각 열의 평균을 빼서 행렬 A를 평균 0으로 만듭니다."""
    nr, nc = shape(A)
    column_means, _ = scale(A)
    return make_matrix(nr, nc, lambda i, j: A[i][j] - column_means[j])

def direction(w):
    """벡터 w의 방향을 반환합니다."""
    mag = magnitude(w)
    return [w_i / mag for w_i in w]

def magnitude(v):
    """벡터 v의 크기를 계산합니다."""
    return math.sqrt(dot(v, v))

def directional_variance_i(x_i, w):
    """벡터 x_i의 방향 w에 대한 분산을 계산합니다."""
    return dot(x_i, direction(w)) ** 2

def directional_variance(X, w):
    """데이터 X의 방향 w에 대한 분산을 계산합니다."""
    return sum(directional_variance_i(x_i, w) for x_i in X)

def directional_variance_gradient_i(x_i, w):
    """벡터 x_i가 방향 w의 분산에 기여하는 바를 계산합니다."""
    projection_length = dot(x_i, direction(w))
    return [2 * projection_length * x_ij for x_ij in x_i]

def directional_variance_gradient(X, w):
    """데이터 X의 방향 w에 대한 분산의 기울기를 계산합니다."""
    return vector_sum(directional_variance_gradient_i(x_i, w) for x_i in X)

def first_principal_component(X):
    """첫 번째 주성분을 계산합니다."""
    guess = [1 for _ in X[0]]
    unscaled_maximizer = maximize_batch(
        partial(directional_variance, X),           # w에 대한 함수로 변환
        partial(directional_variance_gradient, X),  # w에 대한 함수로 변환
        guess)
    return direction(unscaled_maximizer)

def first_principal_component_sgd(X):
    """확률적 경사 하강법을 사용하여 첫 번째 주성분을 계산합니다."""
    guess = [1 for _ in X[0]]
    unscaled_maximizer = maximize_stochastic(
        lambda x, _, w: directional_variance_i(x, w),
        lambda x, _, w: directional_variance_gradient_i(x, w),
        X, [None for _ in X], guess)
    return direction(unscaled_maximizer)

def project(v, w):
    """벡터 v를 벡터 w에 사영(projection)합니다."""
    coefficient = dot(v, w)
    return scalar_multiply(coefficient, w)

def remove_projection_from_vector(v, w):
    """벡터 v를 w에 사영한 후 결과를 v에서 뺍니다."""
    return vector_subtract(v, project(v, w))

def remove_projection(X, w):
    """데이터 X의 각 행을 벡터 w에 사영한 후 결과를 뺍니다."""
    return [remove_projection_from_vector(x_i, w) for x_i in X]

def principal_component_analysis(X, num_components):
    """주성분 분석을 수행하여 num_components 개의 주성분을 반환합니다."""
    components = []
    for _ in range(num_components):
        component = first_principal_component(X)
        components.append(component)
        X = remove_projection(X, component)
    return components

def transform_vector(v, components):
    """벡터 v를 주성분 공간으로 변환합니다."""
    return [dot(v, w) for w in components]

def transform(X, components):
    """데이터 X를 주성분 공간으로 변환합니다."""
    return [transform_vector(x_i, components) for x_i in X]
```

경사 하강법을 사용하여 주성분 찾기



```
print("PCA")

X = DATA
Y = de_mean_matrix(X)
components = principal_component_analysis(Y, 2)
print("principal components", components)
print("first point", Y[0])
print("first point transformed", transform_vector(Y[0], components))
```

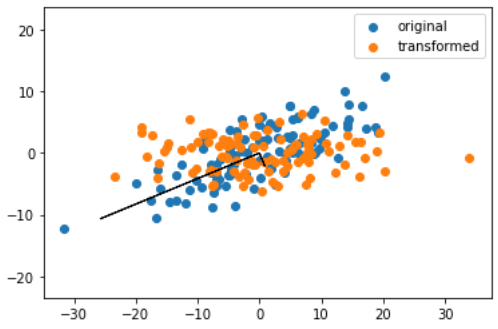
```
PCA
principal components [[0.9238554090431896, 0.382741666377781], [-0.3827224539579983, 0.9238633682728025]]
first point [0.6663708720254604, 1.6869418499129427]
first point transformed [1.2612932692676448, 1.3034686841532082]
```

PCA using sklearn module

```
from sklearn.decomposition import PCA
import numpy as np

# 데이터 초기화
# X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
X = np.array(DATA)

# PCA 수행
pca = PCA(n_components=2)
pca.fit(X)

# 주성분 설명 분산 비율 출력
print(pca.explained_variance_ratio_)

# 주성분 출력
print(pca.components_)
```

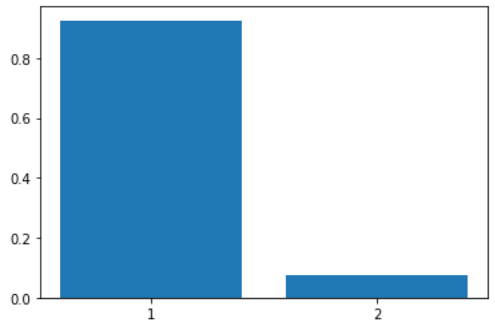
```
[0.9260347 0.0739653]
[[-0.92391028 -0.38260919]
 [ 0.38260919 -0.92391028]]
```

```
# 데이터 중앙 정렬(평균 0으로 만듦)
TX = X - np.mean(X, axis=0)
x, y = TX[:,0], TX[:,1]

# 원본 데이터 산점도 그리기
plt.scatter(x, y, label='original')
plt.axis([-25, 25, -25, 25])

# 주성분 벡터 계산 및 그리기
p0 = pca.components_[0] * pca.explained_variance_ratio_[0] * 30
p1 = pca.components_[1] * pca.explained_variance_ratio_[1] * 30
plt.arrow(0, 0, p0[0], p0[1], head_width=0.05, head_length=0.1, fc='k', ec='k')
plt.arrow(0, 0, p1[0], p1[1], head_width=0.05, head_length=0.1, fc='k', ec='k')

# 변환된 데이터 산점도 그리기
x, y = pca.transform(X)[:,0], pca.transform(X)[:,1]
plt.scatter(x, y, label='transformed')

# 축 설정 및 레전드 추가
plt.axis('equal')
plt.legend(loc=1)
plt.show()
```



- 성분 분석(PCA)을 통해 데이터를 변환하고, 원본 데이터와 변환된 데이터를 시각화하여 비교합니다.
- 주성분 벡터를 화살표로 표시하여 데이터의 주요 방향을 나타냅니다.

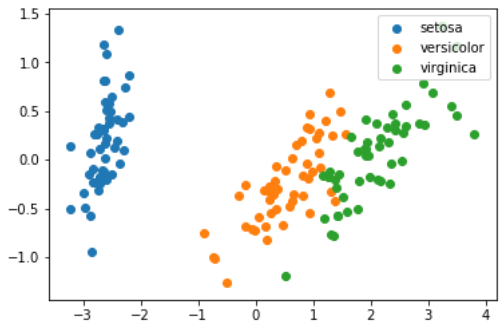
```
# 주성분 설명 분산 비율 막대 그래프 그리기
ratio = pca.explained_variance_ratio_
plt.bar([1, 2], ratio)
plt.xticks([1, 2], ['PC1', 'PC2'])
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio of Principal Components')
plt.show()

# [0.92461872 0.05306648 0.01710261 0.00521218]
```



- 주성분 분석(PCA)을 통해 데이터를 변환하고, 원본 데이터와 변환된 데이터를 시각화하여 비교합니다.
- 또한, 각 주성분이 설명하는 분산 비율을 막대 그래프로 시각화 합니다.

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np

def plot_pca(data, target, target_names):
    """
    주성분 분석(PCA)을 수행하고, 데이터를 2차원으로 시각화합니다.

    Parameters:
    data (array-like): 입력 데이터.
    target (array-like): 각 데이터 포인트의 타겟 레이블.
    target_names (list): 타겟 레이블의 이름들.
    """
    # PCA 모델 생성 및 적합
    model = PCA(n_components=2)
    embedded = model.fit_transform(data)

    # 주성분 좌표
    xs = embedded[:,0]
    ys = embedded[:,1]

    # 각 타겟 레이블에 대해 산점도 그리기
    for t in np.unique(target):
        i = np.where(target == t)
        plt.scatter(xs[i], ys[i], label=target_names[t])

    # 레전드 추가
    plt.legend(loc=1)

    # 플롯 표시
    plt.show()
```

```
from sklearn import datasets

iris = datasets.load_iris()
plot_pca(iris.data, iris.target, iris.target_names)
```



Example: Breast Cancer Wisconsin (Diagnostic) Data Set

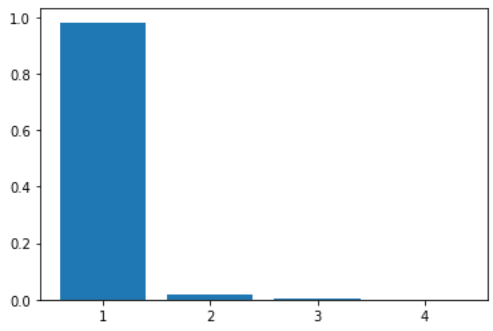
- Kaggle Competition: 암이 양성인지 악성인지 예측
- https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

```
import pandas as pd

bc = pd.read_csv('breast_cancer_data.csv')
bc[:50]
```

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.147100
1	842517	M	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.070170
2	84300903	M	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.127900
3	84348301	M	11.420	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.105200
4	84358402	M	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.104300
5	843786	M	12.450	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.080890
6	844359	M	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.074000
7	84458202	M	13.710	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.059850
8	844981	M	13.000	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.093530
9	84501001	M	12.460	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.085430
10	845636	M	16.020	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.033230
11	84610002	M	15.780	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.066060
12	846226	M	19.170	24.80	132.40	1123.0	0.09740	0.24580	0.20650	0.111800
13	846381	M	15.850	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.053640
14	84667401	M	13.730	22.61	93.60	578.3	0.11310	0.22930	0.21280	0.080250
15	84799002	M	14.540	27.54	96.73	658.8	0.11390	0.15950	0.16390	0.073640
16	848406	M	14.680	20.13	94.74	684.5	0.09867	0.07200	0.07395	0.052590
17	84862001	M	16.130	20.68	108.10	798.8	0.11700	0.20220	0.17220	0.102800
18	849014	M	19.810	22.15	130.00	1260.0	0.09831	0.10270	0.14790	0.094980
19	8510426	B	13.540	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.047810
20	8510653	B	13.080	15.71	85.63	520.0	0.10750	0.12700	0.04568	0.031100
21	8510824	B	9.504	12.44	60.34	273.9	0.10240	0.06492	0.02956	0.020760
22	8511133	M	15.340	14.26	102.50	704.4	0.10730	0.21350	0.20770	0.097560
23	851509	M	21.160	23.04	137.20	1404.0	0.09428	0.10220	0.10970	0.086320
24	852552	M	16.650	21.38	110.00	904.6	0.11210	0.14570	0.15250	0.091700
25	852631	M	17.140	16.40	116.00	912.7	0.11860	0.22760	0.22290	0.140100
26	852763	M	14.580	21.53	97.41	644.8	0.10540	0.18680	0.14250	0.087830
27	852781	M	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.14900	0.077310
28	852973	M	15.300	25.27	102.40	732.4	0.10820	0.16970	0.16830	0.087510
29	853201	M	17.570	15.05	115.00	955.1	0.09847	0.11570	0.09875	0.079530
30	853401	M	18.630	25.11	124.80	1088.0	0.10640	0.18870	0.23190	0.124400
31	853612	M	11.840	18.70	77.93	440.6	0.11090	0.15160	0.12180	0.051820
32	85382601	M	17.020	23.98	112.80	899.3	0.11970	0.14960	0.24170	0.120300
33	854002	M	19.270	26.47	127.90	1162.0	0.09401	0.17190	0.16570	0.075930
34	854039	M	16.130	17.88	107.00	807.2	0.10400	0.15590	0.13540	0.077520
35	854253	M	16.740	21.59	110.10	869.5	0.09610	0.13360	0.13480	0.060180
36	854268	M	14.250	21.72	93.63	633.0	0.09823	0.10980	0.13190	0.055980
37	854941	B	13.030	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.029230
38	855133	M	14.990	25.20	95.54	698.8	0.09387	0.05131	0.02398	0.028990
39	855138	M	13.480	20.82	88.40	559.2	0.10160	0.12550	0.10630	0.054390
40	855167	M	13.440	21.58	86.18	563.0	0.08162	0.06031	0.03110	0.020310
41	855563	M	10.950	21.35	71.90	371.1	0.12270	0.12180	0.10440	0.056690
42	855625	M	19.070	24.81	128.30	1104.0	0.09081	0.21900	0.21070	0.099610
43	856106	M	13.280	20.28	87.32	545.2	0.10410	0.14360	0.09847	0.061580
44	85638502	M	13.170	21.81	85.42	531.5	0.09714	0.10470	0.08259	0.052520
45	857010	M	18.650	17.60	123.70	1076.0	0.10990	0.16860	0.19740	0.100900
46	85713702	B	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.005917
47	85715	M	13.170	18.66	85.98	534.6	0.11580	0.12310	0.12260	0.073400
48	857155	B	12.050	14.63	78.04	449.3	0.10310	0.09092	0.06592	0.027490
49	857156	B	13.490	22.30	86.91	561.0	0.08752			

```
import numpy as np

# 예제 데이터 (bc)에서 필요한 부분 추출
bc_data = np.array(bc)[:, 2:-1]  # 'bc' 배열에서 2번째 열부터 마지막 전 열까지 추출
diagnosis = np.array(bc)[:, 1]   # 'bc' 배열에서 1번째 열 추출

# 첫 두 행의 데이터를 출력 (주석 처리됨)
# print(bc_data[:2])

# 진단 데이터를 출력 (주석 처리됨)
# print(diagnosis)
```

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np

# 주성분 분석(PCA) 모델 생성
bc_pca = PCA()

# 데이터에 PCA 모델 적합
bc_pca.fit(bc_data)

# 첫 4개의 주성분에 대한 설명 분산 비율 추출
ratio = bc_pca.explained_variance_ratio_[:4]

# 설명 분산 비율 출력
print(ratio)

# 막대 그래프 그리기
plt.bar(range(1, len(ratio) + 1), ratio)
plt.xticks(range(1, len(ratio) + 1))
plt.show()

# [9.82044672e-01 1.61764899e-02 1.55751075e-03 1.20931964e-04]
```

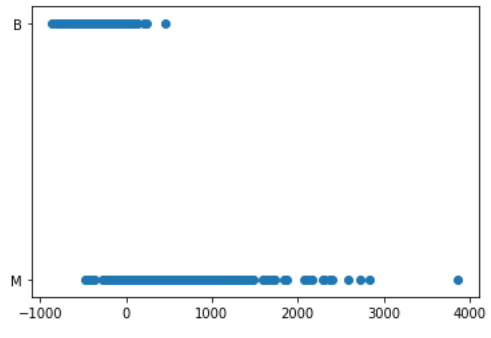


- 주성분 분석(PCA)을 통해 데이터를 분석하고, 첫 4개의 주성분에 대한 설명 분산 비율을 시각화합니다.
- 막대 그래프를 통해 각 주성분이 데이터의 분산을 얼마나 설명하는지 쉽게 확인할 수 있습니다.

```
# 첫 번째 주성분으로 데이터 변환
transf_data_1 = bc_pca.transform(bc_data)[:, 0]

# 변환된 데이터의 형태 출력
print(transf_data_1.shape)

# (569,)
```

- `transf_data_1 = bc_pca.transform(bc_data)[:, 0]` 로 첫 번째 주성분으로 데이터를 변환합니다.
- `bc_pca.transform(bc_data)` 는 `bc_data` 를 주성분 공간으로 변환하며, `[:, 0]` 은 첫 번째 주성분에 해당하는 값을 추출합니다.

```
plt.scatter(transf_data_1, diagnosis)
plt.show()
```

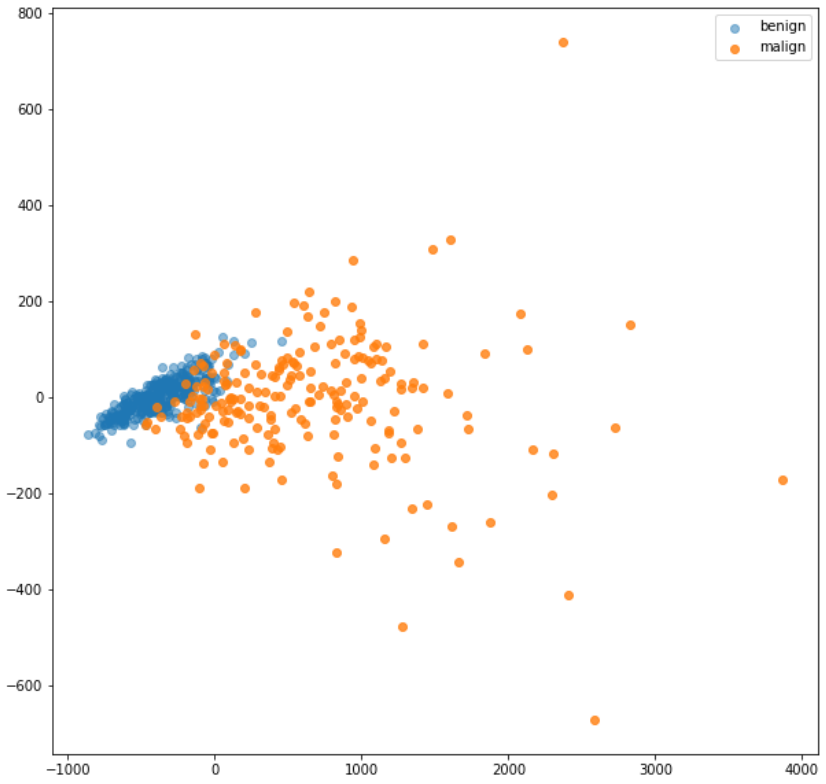


- 우리는 '**Malignan(**악성)'과 '**Benign(음성)**'을 구분하는 단 하나의 판별자를 찾을 수 없습니다.

```
transf_data_2 = bc_pca.transform(bc_data)[:,:2]
transf_data_2.shape
```

`(569, 2)`

```
malign = transf_data_2[diagnosis == 'M']
benign = transf_data_2[diagnosis == 'B']

mx, my = malign[:,0], malign[:,1]
bx, by = benign[:,0], benign[:,1]
plt.figure(figsize=(10,10))
plt.scatter(bx, by, label = 'benign', alpha=0.5)
plt.scatter(mx, my, label = 'malign', alpha=0.8)
plt.legend(loc=1)
plt.show()
```



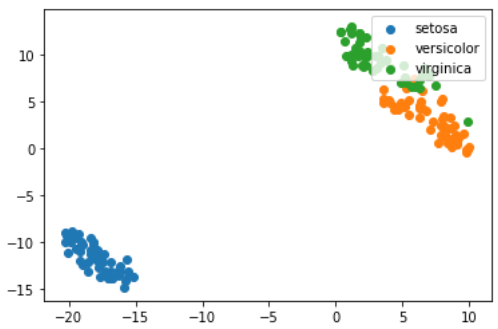
Manifold Learning: t-SNE

- t-분산 확률적 이웃 임베딩(t-SNE)은 고차원 데이터 세트의 시각화에 특히 적합한 차원 축소를 위한 (상수상) 기술입니다.
- 당신은 정말로 다음을 이해할 필요가 없습니다:
- t-SNE 작동 방식
- 첫째, t-SNE는 유사한 객체는 선택될 확률이 높은 반면 다른 점은 선택될 확률이 극히 작은 방식으로 고차원 객체 쌍에 대한 확률 분포를 구성합니다.
- 둘째, t-SNE는 저차원 지도의 점들에 대한 유사한 확률 분포를 정의하고, 지도의 점들의 위치에 대한 두 분포 사이의 쿨백-라이블러 차이를 최소화합니다.

```
# 필요한 라이브러리 임포트
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import numpy as np

def plot_tsne(data, target, target_names, learning_rate=100, perplexity=30):
    """
    t-SNE를 사용하여 데이터를 2차원으로 시각화합니다.

    Parameters:
    data (array-like): 입력 데이터.
    target (array-like): 각 데이터 포인트의 타겟 레이블.
    target_names (list): 타겟 레이블의 이름들.
    learning_rate (float): t-SNE 학습률.
    perplexity (float): t-SNE 퍼플렉서티.
    """
    # t-SNE 모델 생성 및 데이터 적합
    model = TSNE(learning_rate=learning_rate, perplexity=perplexity)
    embedded = model.fit_transform(data)

    # 2차원 좌표 추출
    xs = embedded[:,0]
    ys = embedded[:,1]

    # 각 타겟 레이블에 대해 산점도 그리기
    for t in np.unique(target):
        i = np.where(target == t)
        plt.scatter(xs[i], ys[i], label=target_names[t])

    # 레전드 추가
    plt.legend(loc=1)

    # 플롯 표시
    plt.show()
```

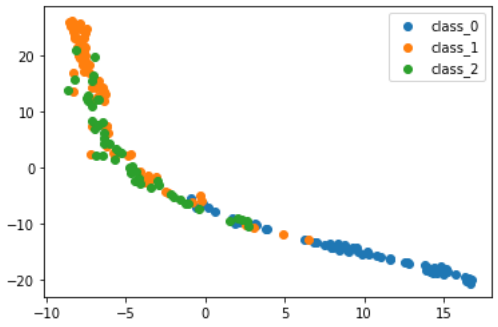
```
from sklearn import datasets
```

```
iris = datasets.load_iris()
plot_tsne(iris.data, iris.target, iris.target_names)
```



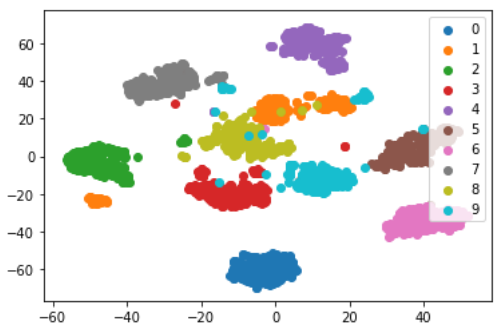
Example t-SNE: Wine classes

```
from sklearn import datasets

wine = datasets.load_wine()
plot_tsne(wine.data, wine.target, wine.target_names, perplexity=20)
```



Example t-SNE: Digits

```
from sklearn import datasets

digits = datasets.load_digits()
plot_tsne(digits.data, digits.target, digits.target_names)
```

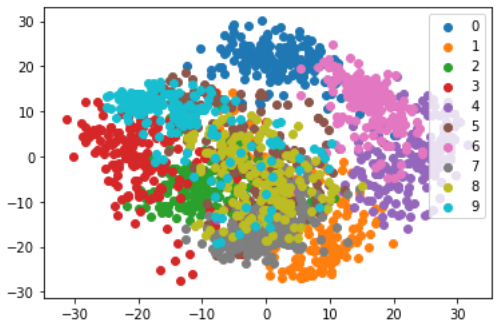



```
digits.data[1]
```

```
array([ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.,  0.,  0.,  0., 11., 16.,
        9.,  0.,  0.,  0.,  0.,  3., 15., 16.,  6.,  0.,  0.,  0.,  7.,
       15., 16., 16.,  2.,  0.,  0.,  0.,  0.,  1., 16., 16.,  3.,  0.,
        0.,  0.,  0.,  1., 16., 16.,  6.,  0.,  0.,  0.,  0.,  1., 16.,
       16.,  6.,  0.,  0.,  0.,  0.,  0., 11., 16., 10.,  0.,  0.])
```

```
plt.figure(figsize=(3,3))
plt.imshow(digits.data[111].reshape(8,8))
plt.axis('off')
plt.show()
```

```
from sklearn import datasets

digits = datasets.load_digits()
plot_pca(digits.data, digits.target, digits.target_names)
```




```
from sklearn import datasets
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# 손글씨 데이터셋 로드
digits = datasets.load_digits()

# PCA 모델 생성 및 데이터 적합 (20개의 주성분)
digits_pca = PCA(n_components=20)
digits_pca.fit(digits.data)

# 각 주성분의 설명 분산 비율 추출
ratio = digits_pca.explained_variance_ratio_

# 설명 분산 비율 출력
print(ratio)

# 총 분산 중 몇 퍼센트가 설명되는지 출력
print('{:.2f}% of total variance is explained'.format(sum(ratio) * 100))

# 막대 그래프 그리기
plt.bar(range(1, len(ratio) + 1), ratio)
plt.xticks(range(1, len(ratio) + 1))
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Principal Component')
plt.title('Explained Variance Ratio of Principal Components')
plt.show()
```

- 손글씨 데이터셋에 대해 주성분 분석(PCA)을 수행하고, 첫 20개의 주성분에 대한 설명 분산 비율을 시각화합니다.
- 이를 통해 각 주성분이 데이터의 분산을 얼마나 설명하는지 쉽게 확인할 수 있습니다.

```
[0.14890594 0.13618771 0.11794594 0.08409979 0.05782415 0.0491691
 0.04315987 0.03661373 0.03353248 0.03078806 0.0237234  0.02272696
 0.01821863 0.01773854 0.01467083 0.01409711 0.01318568 0.0124812
 0.01017626 0.00905282]
89.43% of total variance is expalined
```