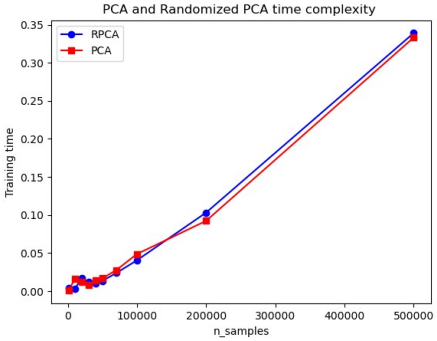
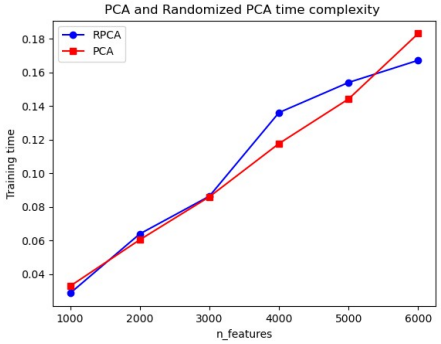
Q1-5. Time Complexity of PCA and IPCA

```
import numpy as np
import os

import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA

import time

#########################################################################################################################################################
### Prepare MNIST dataset ###############################################################################################################################
#########################################################################################################################################################
mnist = fetch_openml('mnist_784', version=1)
mnist.target = mnist.target.astype(np.uint8)

X = mnist['data']       # MNIST 데이터셋의 데이터를 불러옴
y = mnist['target']     # MNIST 데이터셋의 Target 값을 불러옴

X_train, X_test, y_train, y_test = train_test_split(X, y)   # 데이터셋을 Training Set과 Test Set으로 분리함

#########################################################################################################################################################
### PCA, IPCA, Randomized PCA under Different Number of Principle Components ############################################################################
#########################################################################################################################################################
for n_components in (2, 10, 154):   # Principle Component의 개수를 변경하면서 PCA를 수행함

    print('n_components = ', n_components)

    regular_pca = PCA(n_components=n_components)        # Principle Component 개수를 변경하면서 Regular PCA 수행함
    inc_pca = IncrementalPCA(n_components=n_components, batch_size=500)     # Principle Component 개수를 변경하면서 점진적인 PCA 수행함
    rnd_pca = PCA(n_components=n_components, random_state=42, svd_solver='randomized')      # Principle Component 개수를 변경하면서 랜덤 PCA 수행함

    for pca in (regular_pca, inc_pca, rnd_pca):

        # PCA 수행에 소요되는 시간을 측정함
        t1 = time.time()    # PCA 전 시간 측정
        pca.fit(X_train)    # 현재 MNIST Training Set에 대한 Principle Component를 구함
        t2 = time.time()    # PCA 후 시간 측정

        print('     {} : {:.1f} seconds'.format(pca.__class__.__name__, t2-t1))

#########################################################################################################################################################
### PCA vs Randomized PCA under Different Number of Samples #############################################################################################
#########################################################################################################################################################
times_rpca = []     # 랜덤 PCA 소요시간을 저장하는 리스트
times_pca = []      # PCA 소요시간을 저장하는 리스트
sizes = [1000, 10000, 20000, 30000, 40000, 50000, 70000, 100000, 200000, 500000]    # 샘플 개수

# 샘플 개수를 변경하면서 PCA, 랜덤 PCA 소요시간을 측정함
for n_samples in sizes:

    X = np.random.randn(n_samples, 5)       # 5개의 Feature로 구성된 랜덤 데이터셋을 준비함

    pca = PCA(n_components=2, svd_solver='randomized', random_state=42)     # 2개의 상위 Principle Component를 생성하는 랜덤 PCA 객체
    t1 = time.time()            # 랜덤 PCA 전 시간 측정
    pca.fit(X)                  # 5개의 Feature로 구성된 현재 랜덤 데이터셋에 대한 Principle Component를 구함
    t2 = time.time()            # 랜덤 PCA 후 시간 측정
    times_rpca.append(t2-t1)    # 랜덤 PCA 소요시간을 저장함

    pca = PCA(n_components=2)   # 2개의 상위 Principle Component를 생성하는 PCA 객체
    t1 = time.time()            # PCA 전 시간 측정
    pca.fit(X)                  # 5개의 Feature로 구성된 현재 랜덤 데이터셋에 대한 Principle Component를 구함
    t2 = time.time()            # PCA 후 시간 측정
    times_pca.append(t2-t1)     # PCA 소요시간을 저장함

# PCA와 랜덤 PCA의 시간 복잡도를 그래프로 그림
plt.plot(sizes, times_rpca, "b-o", label="RPCA")
plt.plot(sizes, times_pca, "r-s", label="PCA")
plt.xlabel("n_samples")
plt.ylabel("Training time")
plt.legend(loc="upper left")
plt.title("PCA and Randomized PCA time complexity ")
plt.show()

#########################################################################################################################################################
### PCA vs Randomized PCA under Different Number of Features ############################################################################################
#########################################################################################################################################################
times_rpca = []     # 랜덤 PCA 소요시간을 저장하는 리스트
times_pca = []      # PCA 소요시간을 저장하는 리스트
sizes = [1000, 2000, 3000, 4000, 5000, 6000]        # Feature 개수

# Feature 개수를 변경하면서 PCA, 랜덤 PCA 소요시간을 측정함
for n_features in sizes:

    X = np.random.randn(2000, n_features)       # Feature 개수를 변경하면서 랜덤 데이터셋을 준비함

    pca = PCA(n_components=2, random_state=42, svd_solver='randomized')     # 2개의 상위 Principle Component를 생성하는 랜덤 PCA 객체
    t1 = time.time()                # 랜덤 PCA 전 시간 측정
    pca.fit(X)                      # 변경된 Feature로 구성된 현재 랜덤 데이터셋에 대한 Principle Component를 구함
    t2 = time.time()                # 랜덤 PCA 후 시간 측정
    times_rpca.append(t2-t1)        # 랜덤 PCA 소요시간을 저장함

    pca = PCA(n_components=2)       # 2개의 상위 Principle Component를 생성하는 PCA 객체
    t1 = time.time()                # PCA 전 시간 측정
    pca.fit(X)                      # 변경된 Feature로 구성된 현재 랜덤 데이터셋에 대한 Principle Component를 구함
    t2 = time.time()                # PCA 후 시간 측정
    times_pca.append(t2-t1)         # PCA 소요시간을 저장함

# PCA와 랜덤 PCA의 시간 복잡도를 그래프로 그림
plt.plot(sizes, times_rpca, "b-o", label="RPCA")
plt.plot(sizes, times_pca, "r-s", label="PCA")
plt.xlabel("n_features")
plt.ylabel("Training time")
plt.legend(loc="upper left")
plt.title("PCA and Randomized PCA time complexity ")
plt.show()
```



Sample 개수에 따른
PCA, 랜덤 PCA 시간 복잡도 비교



Feature 개수에 따른
PCA, 랜덤 PCA 시간 복잡도 비교