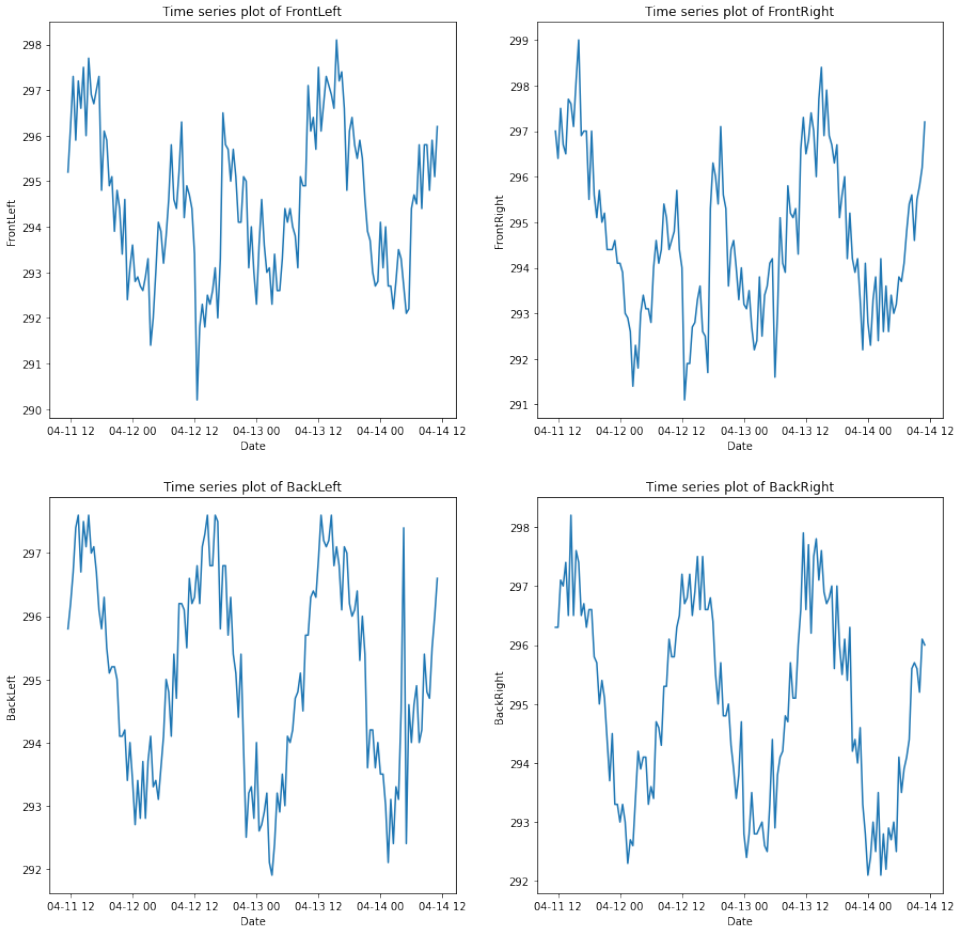
Xử Lí Da Chiều - Programming Exercise - 01

Câu 1

Sử dụng bộ dữ liệu nhiệt độ phòng. Ta sẽ vẽ các đồ thị dạng time series theo từng features bằng thư viện matplotlib.pyplot. Ta được kết quả như sau:



Câu 2

Ta sẽ thực hiện PCA vào toàn bộ bộ dữ liệu như sau:

Đầu tiên, ta sẽ lấy bộ dữ liệu trừ đi mean của nó và chia cho độ lệch chuẩn để thu được bộ dữ liệu được chuẩn hoá có trung bình bằng 0.

```
def transform(X):
  #Do the standardscaler
  X_std = (X - X.mean(axis = 0))/X.std(axis = 0, ddof = 1)
  return X_std
```

Sau đó, ta sẽ tính ma trận hiệp phương sai của bộ dữ liệu bằng thư viện numpy.

```
def calculate_covariance(X):
  #Compute covariance matrix
  return np.cov(X.T)
```

Từ đây để thực hiện PCA, ta sẽ tính cặp vectơ riêng và trị riêng của ma trận hiệp phương sai và sau đó sắp xếp nó theo 1 thứ tự tăng dần.

```
def PCA_prepare(X, k = None):
  #return the sorted list of pairs of eigenvalues and eigenvectors
  Sigma = calculate_covariance(X)
  eigenvals, eigenvecs = calculate_eigen(Sigma)
  eigenpairs = [(abs(eigenval), eigenvec) for (eigenval, eigenvec) in zip(eigenv
  eigenpairs = sorted(eigenpairs, key = lambda pair: pair[0], reverse = True)
  #We take k first eigenvalues
  if k is None:
    return eigenpairs
  else:
    return eigenpairs[:k]
```

Ta truyền vào hàm này tham số k chính là số lượng k trị riêng lớn nhất, tương ứng việc lấy k thành phần chính.
Cuối cùng ta chỉ cần thiết lập ma trận chuyển đổi W bằng cách xếp k véctơ riêng tương ứng thành cột.

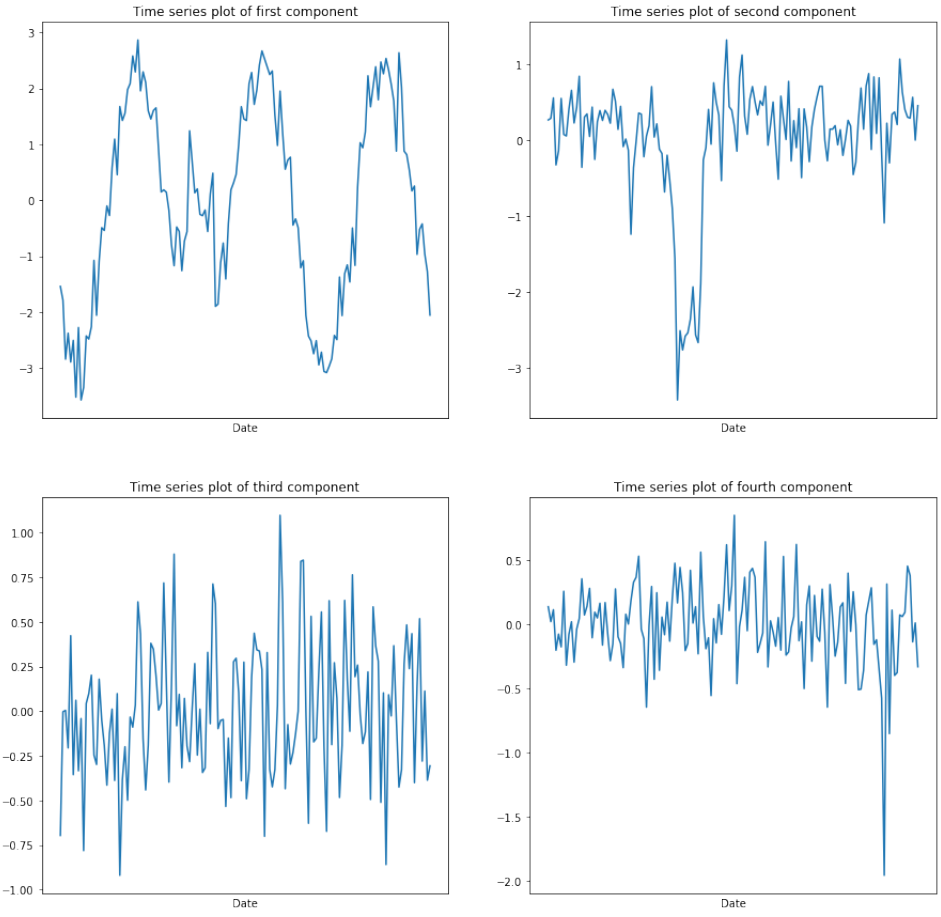
```
def construct_W(eigenpairs):
  #Contruct the transform matrix
  eigenvecs = [eigenvec for (_, eigenvec) in eigenpairs]
  W = np.array([e.T for e in eigenvecs])
  return W
```

Từ đây, class PCA sẽ gọn nhẹ hơn rất nhiều. Sau khi làm các bước trên ta chỉ cần lấy ma trận chuyển đổi và nhân với bộ dữ liệu thì sẽ ra kết quả của bộ dữ liệu đã chiếu lên các thành phần chính mà mình chọn.

```
#Class Principal Component Analysis
class PCA_algorithm:
  def __init__(self,data=df,k=4):
```

```
        self.data=data
        self.k=k

    def PCA(self):
        eigenpairs=PCA_prepare(transform(self.data),self.k)
        W = construct_W(eigenpairs)
        X_pca = W.dot(transform(self.data).T)
        X_pca = X_pca.T
        return pd.DataFrame(X_pca)
```

Ta áp dụng PCA theo các bước trên lên toàn bộ dữ liệu. Vì ta chọn số lượng thành phần chính bằng với số features nên số chiều của bộ dữ liệu chưa thay đổi. Ta plot time series trên từng thành phần chính của toàn bộ dữ liệu ta thu được kết quả như sau:



Câu 3

Ta có số lượng thông tin khi chiếu lên thành phần đó ứng với tỉ lệ của trị riêng trên tổng các

trị riêng.

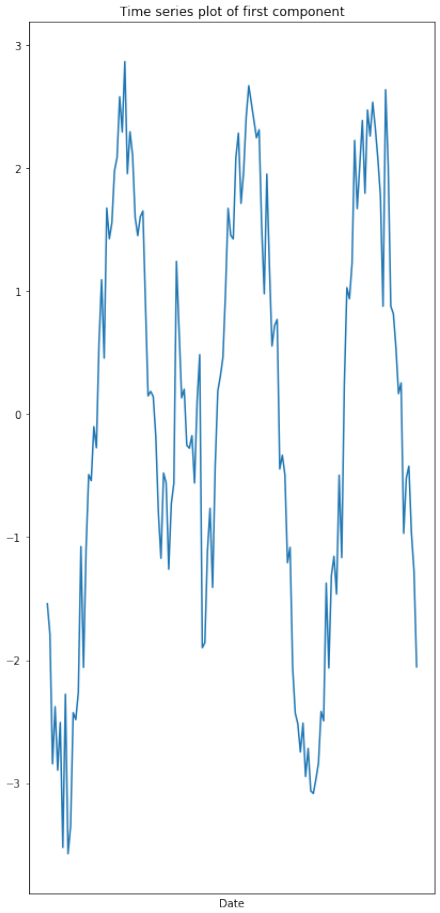
Nên ta lấy tổng tích luỹ các trị riêng để xem lượng thông tin mà thành phần đó giữ lại.

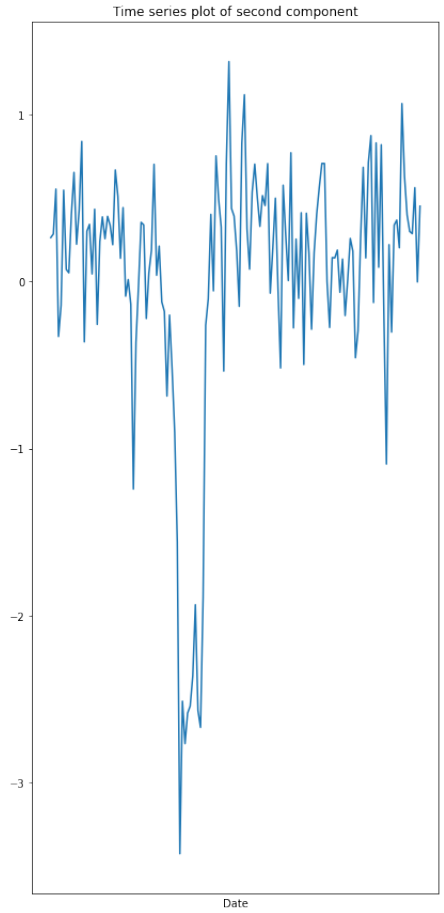
```
eigenpairs=PCA_prepare(transform(df.iloc[:,0:]),k=4)

eigenvals = [eigenval for (eigenval, _) in eigenpairs]
eigenvals = np.array(eigenvals)
cumsum = np.cumsum(eigenvals)
cumsum /= cumsum[-1]
cumsum
```

Kết quả trên ta thu được thành phần 1 giữ 73% thông tin và thành phần 1, 2 giữ 93% và 3 thành phần giữ 97% và đương nhiên 4 thành phần giữ lại toàn bộ thông tin. Ta sẽ chọn chiếu bộ dữ liệu trên 2 thành phần chính vì 2 thành phần giữ được lượng thông tin đủ lớn và các thành phần còn lại lượng thông tin chúng giữ không nhiều.

```
#Apply PCA with 2 components
model=PCA_algorithm(data=df.iloc[:,0:],k=2)
X_pca_new=model.PCA()
```

Sau khi chiếu bộ dữ liệu lên 2 thành phần chính thì ta thu được bộ dữ liệu có 2 features. Ta sẽ plot thử time series trên từng features chiếu lên thành phần chính. Ta thu được:


Time series plot of first component
3
2
1
0
−1
−2
−3
Date


Time series plot of second component
1
0
−1
−2
−3
Date