Principal component analysis

Input: a matrix X, unlabeled data matrix

Output: k directions of maximum variation

```
import numpy as np


def scratch_PCA(X, k):
    demeaned_X= X - np.mean(X, axis=0)
    #we now have to caculate the covariance matrix
    cov_X= np.matmul(np.transpose(demeaned_X), demeaned_X)
    sigma,V= np.linalg.eig(cov_X)
    # you then select the k biggest eignevalues and their corresponding eigenvectors
    return sigma[:k], V[:k]


def svd_PCA(X,k):
  U, S, V = np.linalg.svd(X - np.mean(X, axis=0))
  return S[:k]**2, V[:k]
```

Explanation

In PCA we choose the first k components of the SVD of a matrix

In a nutshell, SVD represents the breakdown of the input into a sum of rank 1 matrices.

By ordering the variance (the sigma values) by high to low,

we capture the largest amount of variance in the first k components.

As we increase k to the rank of the input, we fully capture the data matrix.

Mathematically:

SVD is represented as U * S * V.T

- U - eigenvectors of X @ X.T
- V - eigenvectors of X.T @ X
- S - sigma squared = eigenvalues of both

PCA uses the first k values of S and the first k vectors of V

```
from sklearn.decomposition import PCA
```

In practice we would use modules such as sci-kit learn for running methods such as PCA.

Example extrapolated method below:

```
def sk_PCA(X, k):
  # initialize pca with num of components (k)
  pca = PCA(n_components=k)

  # run pca on our data set
  pca.fit(X)

  # get our first k components from our solutions
  variance = pca.explained_variance_[:k]
  vectors = pca.components_[:k]
  vectors = [np.asarray(vectors[i]) for i in range(len(vectors))]

  return variance, vectors
```