In [ ]:
```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sy
```

In [ ]:
```
import pandas as pd
```

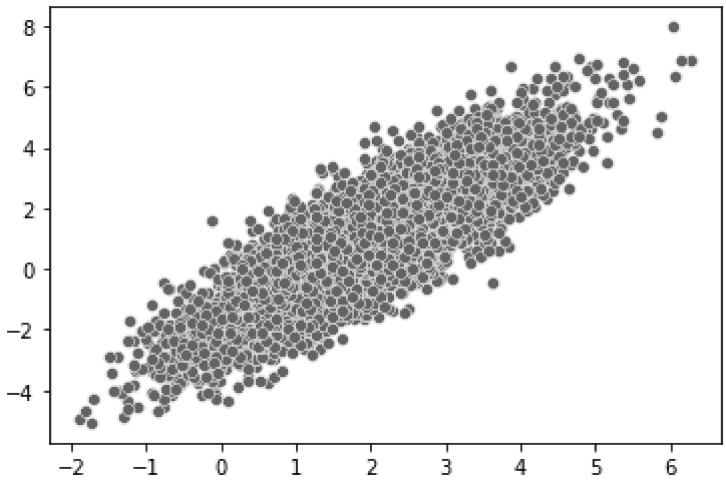
In [ ]:
```
#Question 6
```

In [ ]:
```
gauss = np.loadtxt("C:\\Users\\Joe\\Desktop\\Tufts Fall 2022\\MATH 123\\HW3\\g
aussian_noisy.csv", delimiter=",")
```

In [ ]:
```
gauss.shape
```

Out[ ]:
```
(10000, 2)
```

In [ ]:
```
sns.scatterplot(x=gauss[:,0], y = gauss[:,1])
plt.show;
```



In [ ]:
```
##6.a.
```

In [ ]:
```
def center(test):
    new_arr = np.copy(test)
    for col in range(0,new_arr.shape[1]):
        mean = new_arr[:,col].mean()
        for i in range(0,len(new_arr)):
            new_arr[i,col] = new_arr[i,col]-mean
    return new_arr
```

In [ ]:
```
def my_pca(mydata):
    cdata = center(mydata)
    #Next computer covariance matrix
    cov_mat = (1/len(cdata))*np.dot(cdata.transpose(),cdata)
    eig_vals = np.linalg.eig(cov_mat)[0] #eigenvalues un sorted
    eig_vecs = np.linalg.eig(cov_mat)[1] #eigenvectors un sorted
    eig_val_indices = (-eig_vals).argsort() #find correct order of eigs
    #print(eig_vecs)
    eig_vecs = eig_vecs[eig_val_indices] #reorder to be largest eigenval with
 its vector first
    eig_vals = eig_vals[eig_val_indices] #same
    eig_vecs = eig_vecs*-1

    return eig_vecs, eig_vals, eig_val_indices   #keep track of each item
```

In [ ]:
```
gauss_comps, gauss_vars, gauss_var_indices = my_pca(gauss)
print(gauss_comps)
gauss_vars
```

```
[[-0.4989128   0.86665219]
 [ 0.86665219  0.4989128 ]]
```

Out[ ]:
```
array([3.86620058, 0.24950899])
```

In [ ]:
```
#% Variance  for first PC
gauss_vars[0]/sum(gauss_vars)
```

Out[ ]:
```
0.9393764351535896
```

In [ ]:
```
#% Variance  for second PC
gauss_vars[1]/sum(gauss_vars)
```

Out[ ]:
```
0.060623564846410447
```

In [ ]:
```
gauss_var_indices
```

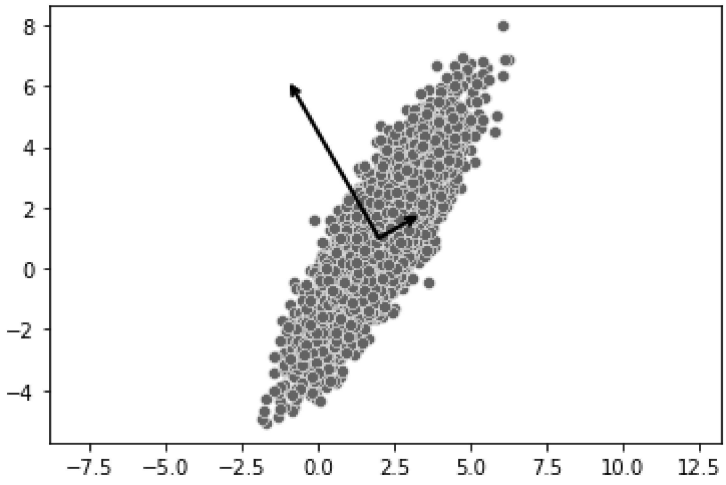
Out[ ]:
```
array([1, 0], dtype=int64)
```

In [ ]:
```
#summarize and explain observations from part B -- see my notebook
```

In [ ]:
```
#these are the right vectors but lingalg.eigs normalizes them in a random way
 so I can't systematically get them
#to point in the right direction! Would be cool if you know how to fix this! b
ut the sklearn PCA does it
#right so its NBD

def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
sns.scatterplot(x=gauss[:,0], y = gauss[:,1])
for length, vector in zip(gauss_vars, gauss_comps):
    v = vector * 3 * np.sqrt(length)
    draw_vector(gauss.mean(axis=0), gauss.mean(axis=0) + v)
plt.axis('equal');
```



In [ ]:
```
gauss_pca_check_model = PCA(n_components=2)
```

In [ ]:
```
gauss_pca_check_model.fit(gauss)
```

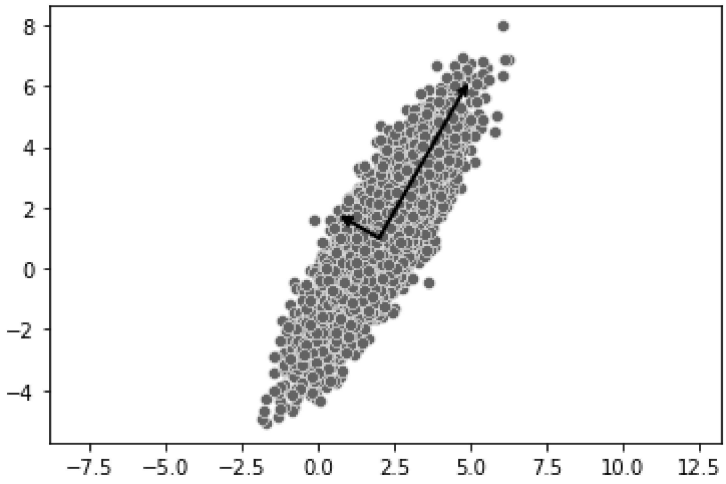
Out[ ]:
```
PCA(n_components=2)
```

In [ ]:
```
gauss_pca_check_model.components_
```

Out[ ]:
```
array([[ 0.4989128 ,  0.86665219],
       [-0.86665219,  0.4989128 ]])
```

In [ ]:
```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
sns.scatterplot(x=gauss[:,0], y = gauss[:,1])
for length, vector in zip(gauss_pca_check_model.explained_variance_, gauss_pca
_check_model.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(gauss.mean(axis=0), gauss.mean(axis=0) + v)
plt.axis('equal');
```

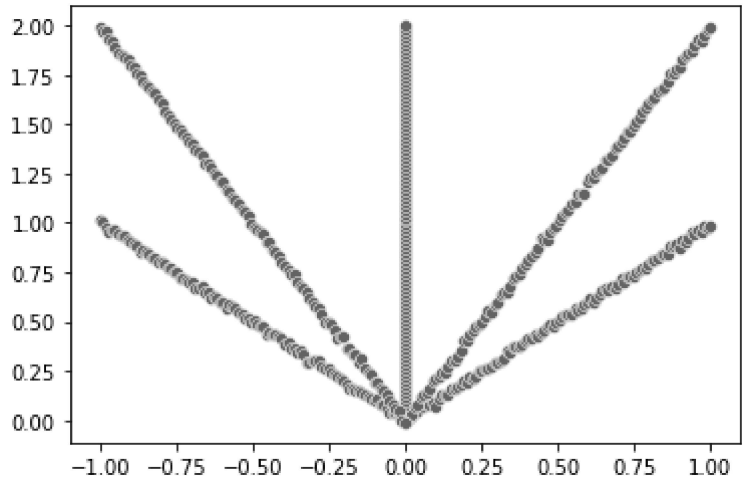


In [ ]:
```
uni = np.loadtxt("C:\\Users\\Joe\\Desktop\\Tufts Fall 2022\\MATH 123\\HW3\\uni
form_noisy.csv", delimiter=",")
```

In [ ]:
```
uni.shape
```

Out[ ]:
```
(500, 2)
```

In [ ]:
```
sns.scatterplot(x=uni[:,0],y=uni[:,1])
plt.show;
```



In [ ]:
```
uni_comps, uni_vals, uni_val_indices = my_pca(uni)
```

In [ ]:
```
uni_comps
```

Out[ ]:
```
array([[ 0.00283345,  0.99999599],
       [ 0.99999599, -0.00283345]])
```

In [ ]:
```
uni_vals
```

Out[ ]:
```
array([0.29868591, 0.26801322])
```

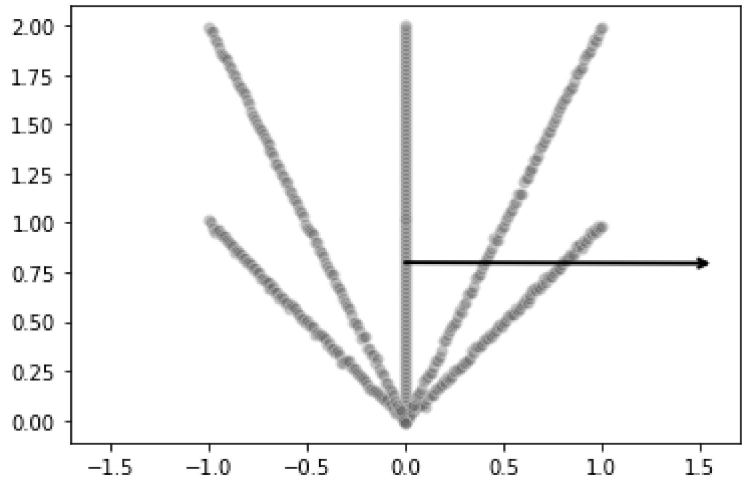
In [ ]:
```
#percentage variance for the first PC
uni_vals[0]/sum(uni_vals)
```

Out[ ]:
```
0.5270625856169509
```

In [ ]:
```
#percentage variance for second PC
uni_vals[1]/sum(uni_vals)
```

Out[ ]:
```
0.4729374143830491
```

In [ ]:
```
# plot data
sns.scatterplot(x=uni[:,0], y = uni[:,1], alpha = .4)
for length, vector in zip(uni_vals, uni_comps):
    v = vector * 3 * np.sqrt(length)
    draw_vector(uni.mean(axis=0), uni.mean(axis=0) + v)
plt.axis('equal');
```



In [ ]:
```
 #Explain and summarize observations for part C -- see my notebook
```

In [ ]:
```
#Question 7
```

In [ ]:
```
data = np.loadtxt("C:\\Users\\Joe\Desktop\\Tufts Fall 2022\\MATH 123\\HW3\\dat
a_067.csv", delimiter=",")
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(21072, 784)
```

In [ ]:
```
model = PCA(n_components=2, svd_solver='arpack', random_state=1)
```

In [ ]:
```
model.fit(data)
```

Out[ ]:
```
PCA
PCA(n_components=2, random_state=1, svd_solver='arpack')
```

In [ ]:
```
model.components_.shape
```

Out[ ]:
```
(2, 784)
```

In [ ]:
```
test_values = model.transform(data)
```

In [ ]:
```
test_values.shape
```

Out[ ]:
```
(21072, 2)
```

In [ ]:
```
V = model.components_.transpose()
V_t = model.components_
```

In [ ]:
```
V_t.shape
```

Out[ ]:
```
(2, 784)
```

In [ ]:
```
approximation = (data-model.mean_).dot(V)
```

In [ ]:
```
test_values[0]
```

Out[ ]:
```
array([ 4.08272393, -1.00021799])
```

In [ ]:
```
approximation[0]
```

Out[ ]:
```
array([ 4.08272393, -1.00021799])
```

In [ ]:
```
labels = np.loadtxt('C:\\Users\\Joe\Desktop\\Tufts Fall 2022\\MATH 123\\HW3\\l
abel_067.csv', delimiter=",")
```

In [ ]:
```
labels_a = labels.reshape(21072,1)
```

In [ ]:
```
full_data = np.hstack((data,labels_a))
```

In [ ]:
```
low_d_data = pd.DataFrame(approximation)
low_d_data.columns = ['PC1','PC2']
low_d_data.head()
```

Out[ ]:

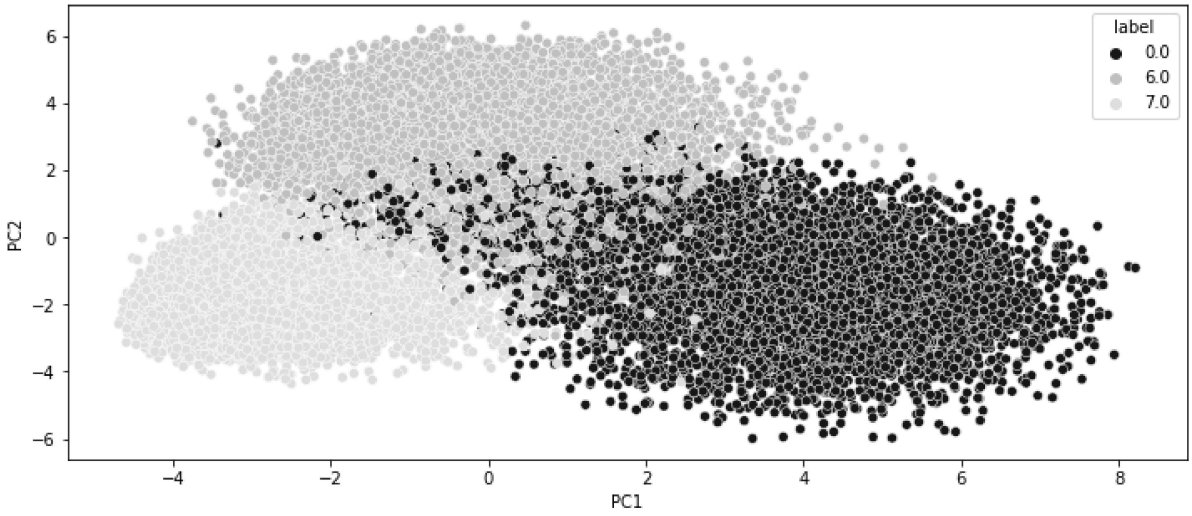
	PC1	PC2
0	4.082724	-1.000218
1	-0.139325	3.857244
2	-2.485433	-2.544350
3	-1.227092	1.991623
4	4.301781	-1.897811

In [ ]:
```
low_d_data['label'] = labels.tolist()
low_d_data.head()
```

Out[ ]:

	PC1	PC2	label
0	4.082724	-1.000218	0.0
1	-0.139325	3.857244	6.0
2	-2.485433	-2.544350	7.0
3	-1.227092	1.991623	6.0
4	4.301781	-1.897811	0.0

In [ ]:
```
plt.figure(figsize=(12,5))
sns.scatterplot(data=low_d_data,x ='PC1', y='PC2', hue='label', palette='plasm
a')
plt.show;
```



This seems like a good representation of the data. Each label seems to have its own general area in relation to the two PCs, meaning that the components are doing a good job explaining the variance between the 3 types of labels.

In [ ]:
```
approximation.shape
labels_a.shape
```

Out[ ]:
```
(21072, 1)
```

In [ ]:
```
approx_withlab = np.hstack((approximation,labels_a)) # add the labels to our l
ow-dim data
```

In [ ]:
```
sevens_pca_indicies = np.where(approx_withlab[:,2] == 7)[0] #find the indicies
of where they are 7
```

In [ ]:
```
sevens_pca = np.zeros(7293*2).reshape(7293,2) #create a new array with just th
ese
for i in range(0,len(sevens_pca_indicies)):
    sevens_pca[i] = approximation[sevens_pca_indicies[i]]
```

In [ ]:
```
sevens_full_indices = np.where(full_data[:,784] == 7) #find in the full data o
ur indicies jsut 7s
```

In [ ]:
```
sevens_full_indices = sevens_full_indices[0] #
```

In [ ]:
```
sevens_full = np.zeros(7293*785).reshape(7293,785) #create an array with just
 the 7s and label
for i in range(0,len(sevens_full_indices)):
    sevens_full[i] = full_data[sevens_full_indices[i]]
```

In [ ]:
```
sevens_full.shape
```

Out[ ]:
```
(7293, 785)
```

In [ ]:
```
sevens_full_nolab = sevens_full[:,0:784] #remove the label
```

In [ ]:
```
sevens_full_nolab.shape
```

Out[ ]:
```
(7293, 784)
```

In [ ]:
```
print(V.shape)
print(sevens_pca.shape)
```

```
(784, 2)
(7293, 2)
```

In [ ]:
```
seven_projection = model.inverse_transform(sevens_pca) #create the projection
 of the low dim data
```

In [ ]:
```
np.sum((sevens_full_nolab-seven_projection)**2,axis=1).mean() #calculate error
```

Out[ ]:
```
35.711489319847075
```

In [ ]:
```
'''
Part b
'''
```

Out[ ]:
```
'\nPart b\n'
```

In [ ]:
```
components = np.arange(0,310,10) #array of each component to tets
components[0] = 1
components
```

Out[ ]:
```
array([  1,  10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110, 120,
       130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250,
       260, 270, 280, 290, 300])
```

In [ ]:
```
error_list = np.arange(0,31)*0 #empty array to store our errors
error_list.shape
```

Out[ ]:
```
(31,)
```

In [ ]:
```
b_data = sevens_full[:,:784]  #creating new data frame because old naming conv
entions were terrible
b_data.shape
```

Out[ ]:
```
(7293, 784)
```

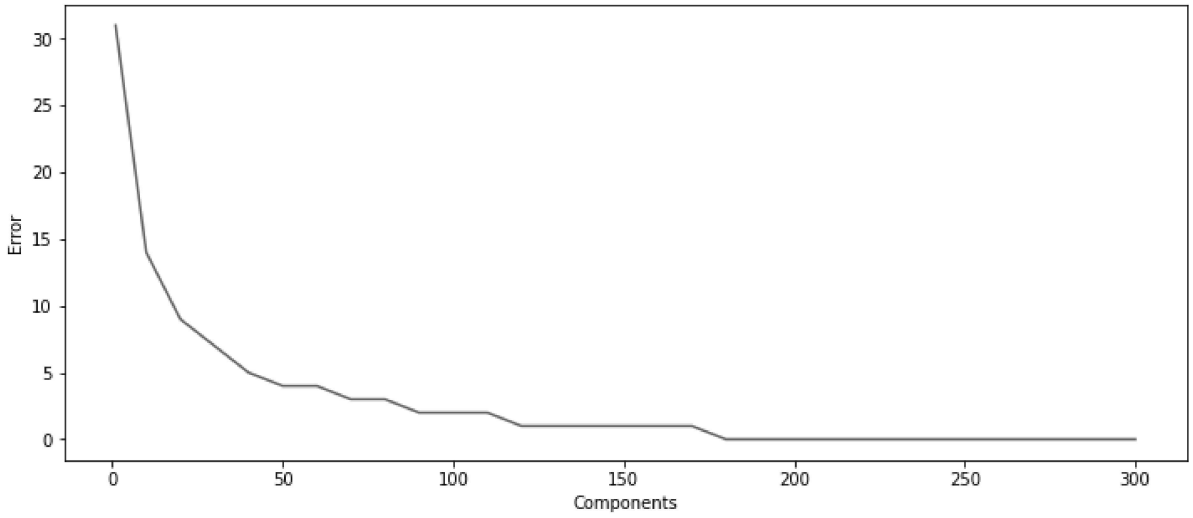
In [ ]:
```
index = 0
for comps in components:
    model = PCA(n_components=comps, svd_solver='arpack', random_state=1)
    model.fit(b_data)
    b_data_pcas = model.transform(b_data)
    projection = model.inverse_transform(b_data_pcas)
    error_item = np.sum((b_data - projection)**2, axis = 1).mean()
    error_list[index] = error_item
    index += 1
```

In [ ]:
```
error_data = pd.DataFrame([components,error_list])
```

In [ ]:
```
error_data = error_data.transpose()
```

In [ ]:
```
error_data.columns = ['Components','Error']
```

In [ ]:
```
plt.figure(figsize=(12,5))
sns.lineplot(data=error_data, x='Components', y='Error')
plt.show;
```



From this chart, we can see that with more PCs, the reconstruction error is lower since we are capturing more of the variability in the original data. Based on our results, it seems like around 180 components is enough to capture virtually all the variance.

In [ ]:
```
```