Hands-on Practice for Module 1: Exploratory Data Analysis

0. Importing necessary packages

In [ ]:
```
# data loading and computing functionality
import pandas as pd
import numpy as np
import scipy as sp

# datasets in sklearn package
from sklearn.datasets import load_digits

# visualization packages
import seaborn as sns
import matplotlib.pyplot as plt

#PCA, SVD, LDA
from sklearn.decomposition import PCA
from scipy.linalg import svd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

1. Loading data, determining samples, attributes, and types of attributes

Question: Where is the data obtained from?

Answer: Data is obtained from the URL https://raw.githubusercontent.com/plotly/datasets/master/iris.csv (https://raw.githubusercontent.com/plotly/datasets/master/iris.csv) that is originally part of the UCI Machine Learning repository. https://archive.ics.uci.edu/ml/datasets/iris (https://archive.ics.uci.edu/ml/datasets/iris)

In [2]:
```
import pandas as pd
iris_df = pd.read_csv('https://raw.githubusercontent.com/plotly/dataset
s/master/iris.csv')
```

In [3]:
```
type(iris_df)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

Question: What does the data capture?

Answer: Data captures four properties (Sepal Length, Sepal Width, Petal Length, Petal Width) of three types of Iris plants.

Question: How many data points are there?

```
In [4]: iris_df.shape
```

```
Out[4]: (150, 5)
```

Answer: There are 50 instances/data points for each type. Collectively, 150 data points.

Question: What is the dimensionality?

```
In [5]: iris_df.dtypes
```

```
Out[5]: SepalLength    float64
        SepalWidth     float64
        PetalLength    float64
        PetalWidth     float64
        Name            object
        dtype: object
```

```
In [6]: iris_df.head()
```

Out[6]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Answer: There are four attributes (Sepal Length, Sepal Width, Petal Length, Petal Width) and one label (Name).

Question: What type of attributes are present in the dataset?

```
In [7]: iris_df.dtypes
```

```
Out[7]: SepalLength    float64
        SepalWidth     float64
        PetalLength    float64
        PetalWidth     float64
        Name            object
        dtype: object
```

Answer: All four attributes are continuous-valued.

2. Generating summary statistics

Question: What are range of values these numeric attributes take?

```
In [8]: iris_df.describe()
```

Out[8]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Question: What are the mean values for each of the attributes?

In [9]:
```python
from pandas.api.types import is_numeric_dtype

for col in iris_df.columns:
    if is_numeric_dtype(iris_df[col]):
        print('%s:' % (col))
        print('\t Mean = %.2f' % iris_df[col].mean())
```

```
SepalLength:
         Mean = 5.84
SepalWidth:
         Mean = 3.05
PetalLength:
         Mean = 3.76
PetalWidth:
         Mean = 1.20
```

Question: What is the variance for each of the attributes?

In [10]:
```python
from pandas.api.types import is_numeric_dtype

for col in iris_df.columns:
    if is_numeric_dtype(iris_df[col]):
        print('%s:' % (col))
        print('\t Variance = %.2f' % iris_df[col].var())
```

```
SepalLength:
         Variance = 0.69
SepalWidth:
         Variance = 0.19
PetalLength:
         Variance = 3.11
PetalWidth:
         Variance = 0.58
```

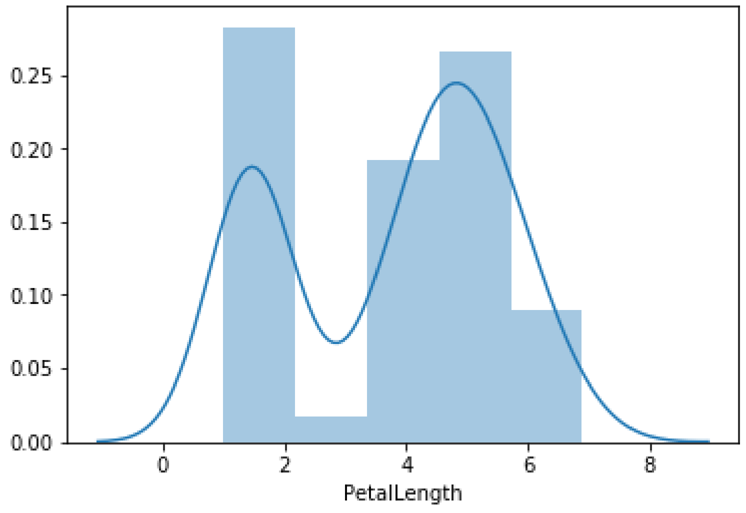
Question: Visually examine how the attribute PetalLength is distributed and comment if the data is Normally distributed?

Introducing **Seaborn**, a statistical data visualization library

Visualizing a histogram for a numerical attribute using distplot function in seaborn

```
In [11]: sns.distplot(iris_df['PetalLength']);
```

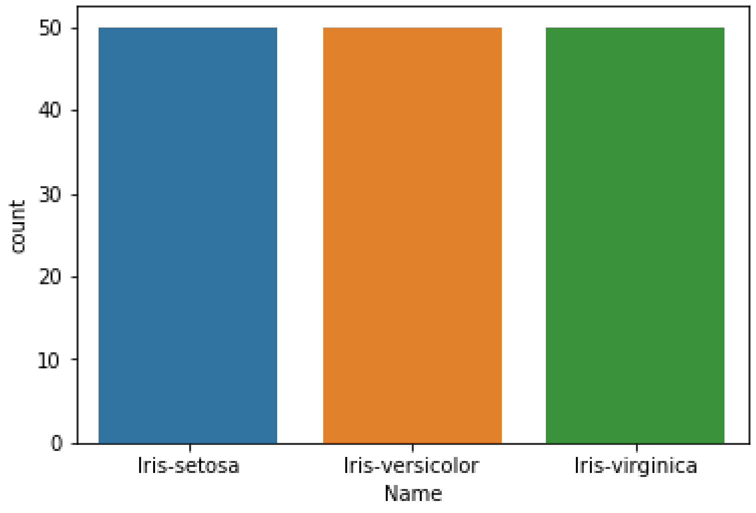
```
/usr/local/python/2.7-conda5.2/lib/python2.7/site-packages/scipy/stats/
stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimen
sional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[se
q]`. In the future this will be interpreted as an array index, `arr[np.
array(seq)]`, which will result either in an error or a different resul
t.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Answer: PetalLength is not Normally distributed. The distribution appears to be bimodal.

Question: Visually examine how the label (Name) attribute is distributed and comment if the data is uniformly distributed?

```
In [12]: sns.countplot(iris_df['Name']);
```



Answer: The label is uniformly distributed as all classes have exactly 50 data points.

3. Geometric view and Probabilistic view

For this part, we will restrict to SepalLength and SepalWidth attributes as we can only visualize 2D space.

Question: Show the Geometric view of the data on a 2D space along with the mean.

In [13]:
```
iris_df_new = iris_df[['SepalLength','SepalWidth']]
```

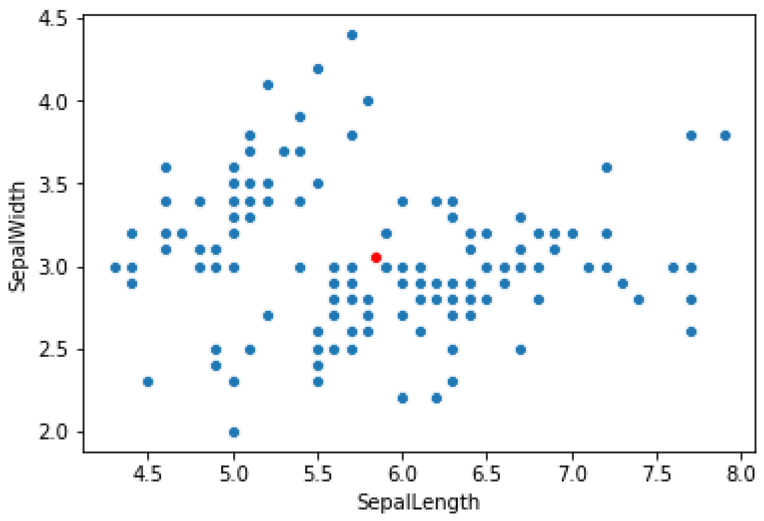
In [14]:
```
iris_df_new.head()
```

Out[14]:

	SepalLength	SepalWidth
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

In [15]:
```
fig, ax = plt.subplots()
sns.scatterplot(x='SepalLength',y='SepalWidth',data=iris_df_new,ax=ax)
mu = np.mean(iris_df_new.values,0)
sns.scatterplot(x=[mu[0], mu[0]],y=[mu[1], mu[1]],color='r',ax=ax)
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2afc522f3c50>
```



Question: Based on the geometric view of the data, which of the points [6.5, 3.0], [7.5, 3.0] are more closer to the mean?

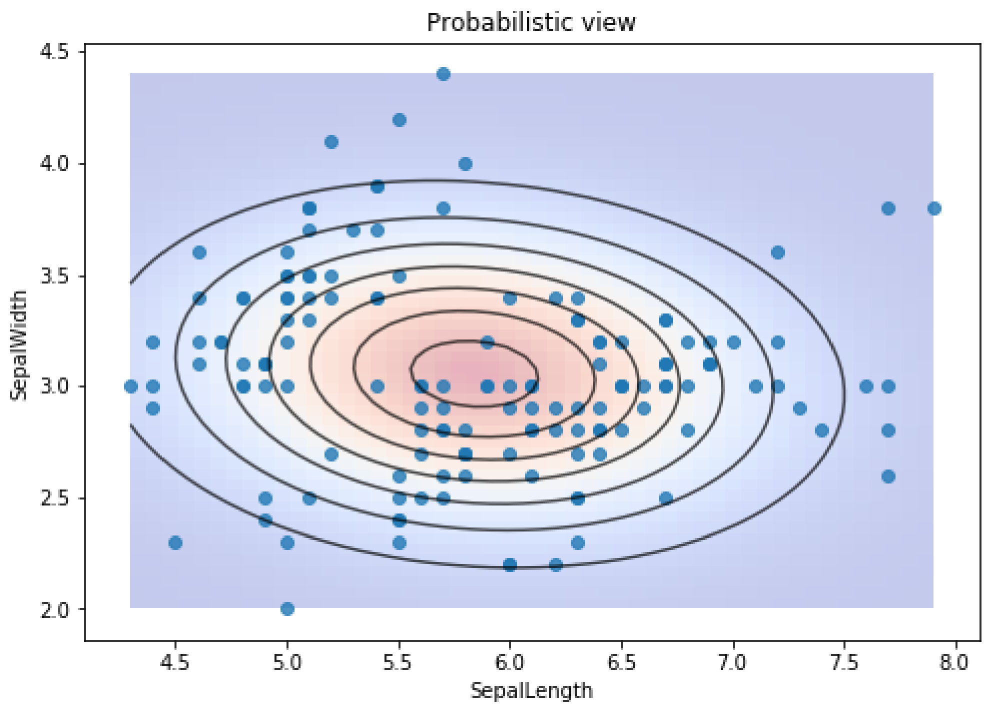
Answer: [6.5, 3.0] is more closer to the mean shown in red circle.

Question: Show the probabilistic view of the data. Assume that the data is drawn from a 2D distribution.

In [16]:
```
from scipy.stats import multivariate_normal

mu = np.mean(iris_df_new.values,0)
Sigma = np.cov(iris_df_new.values.transpose())

min_length = np.min(iris_df_new.values[:,0]);
min_width = np.min(iris_df_new.values[:,1]);
max_length = np.max(iris_df_new.values[:,0]);
max_width = np.max(iris_df_new.values[:,1]);
x, y = np.mgrid[min_length:max_length:50j, min_width:max_width:50j]

positions = np.empty(x.shape + (2,))
positions[:, :, 0] = x;
positions[:, :, 1] = y

F = multivariate_normal(mu, Sigma)
Z = F.pdf(positions)
```

```
In [17]: fig = plt.figure(figsize=(8,8))
         ax = fig.gca()
         ax.imshow(np.rot90(Z), cmap='coolwarm', extent=[min_length,max_length, m
         in_width,max_width], alpha=0.3)
         cset = ax.contour(x, y, Z, colors='k', alpha=0.7)
         plt.scatter(iris_df_new.values[:,0],iris_df_new.values[:,1],alpha=0.8)
         ax.set_xlabel('SepalLength')
         ax.set_ylabel('SepalWidth')
         plt.title('Probabilistic view')
```

```
Out[17]: Text(0.5,1,'Probabilistic view')
```



Question: Based on the probabilistic view of the data, which of the points [5.8, 3.0], [6.5, 3.0] are more likely to be generated from the bivariate normal distribution? Provide your reason.

Answer: [5.8, 3.0]. The probability density at this point is higher than that of [6.5, 3.0].

3. Understanding the (in)dependencies among attributes using Covariance matrix

Question: What is the covariance matrix?

Selecting the relevant data...

In [18]:
```
data = iris_df.values[:,0:4]
data[1:10,:]
```

Out[18]:
```
array([[4.9, 3.0, 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5.0, 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5.0, 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]], dtype=object)
```

In [19]:
```
def mycov(data, col_a, col_b):
    mu = np.mean(data, axis=0) #compute mean
    sum = 0;
    for i in range(0, len(data)):
        sum += ((data[i,col_a] - mu[col_a]) * (data[i,col_b] - mu[col_b
]))

    return sum/(len(data)-1)
```

In [20]:
```
[mycov(data,0,0), mycov(data,0,1), mycov(data,0,2), mycov(data,0,3)]
```

Out[20]:
```
[0.6856935123042505,
 -0.03926845637583892,
 1.2736823266219242,
 0.5169038031319912]
```

In [21]:
```
print('Covariance:')
iris_df.cov()
```

```
Covariance:
```

Out[21]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
SepalLength	0.685694	-0.039268	1.273682	0.516904
SepalWidth	-0.039268	0.188004	-0.321713	-0.117981
PetalLength	1.273682	-0.321713	3.113179	1.296387
PetalWidth	0.516904	-0.117981	1.296387	0.582414

Question: Which pairs of attributes co-vary in the same direction?

Answer:

SepalLength, PetalLength

SepalLength, PetalWidth

PetalLength, PetalWidth

Question: Which pairs of attributes are highly correlated? List all such pairs.

```
In [22]: print('Correlation:')
         iris_df.corr()
```

```
Correlation:
```

Out[22]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
SepalLength	1.000000	-0.109369	0.871754	0.817954
SepalWidth	-0.109369	1.000000	-0.420516	-0.356544
PetalLength	0.871754	-0.420516	1.000000	0.962757
PetalWidth	0.817954	-0.356544	0.962757	1.000000

Answer: Highly correlated pairs listed in decreasing order of correlation.

PetalLength, PetalWidth

SepalLength, PetalLength

SepalLength, PetalWidth

Question: Which pairs of attributes are uncorrelated/weakly correlated?

Answer: Highly correlated pairs listed in decreasing order of correlation.

SepalLength, SepalWidth

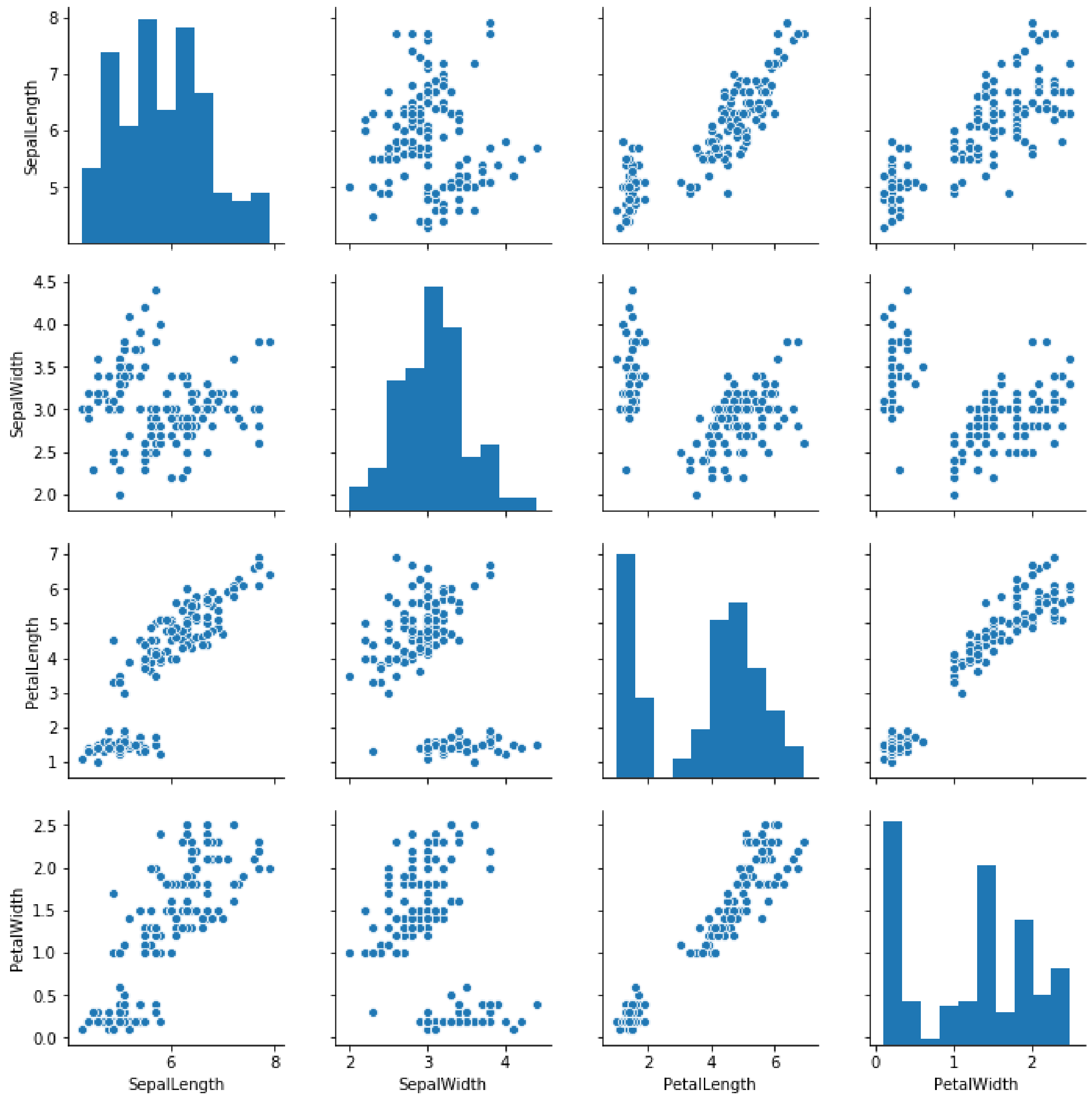
4. Visualizing relationships between attributes

Question: Visualize the iris dataset using a pairplot and comment if PetalLength and PetalWidth have positive covariance.

pairplot function in seaborn library simultaneously generates histograms for individual attributes and pairwise scatter plots.

In [23]:
```
sns.pairplot(iris_df)
```

Out[23]:
```
<seaborn.axisgrid.PairGrid at 0x2afc52340290>
```



Answer:

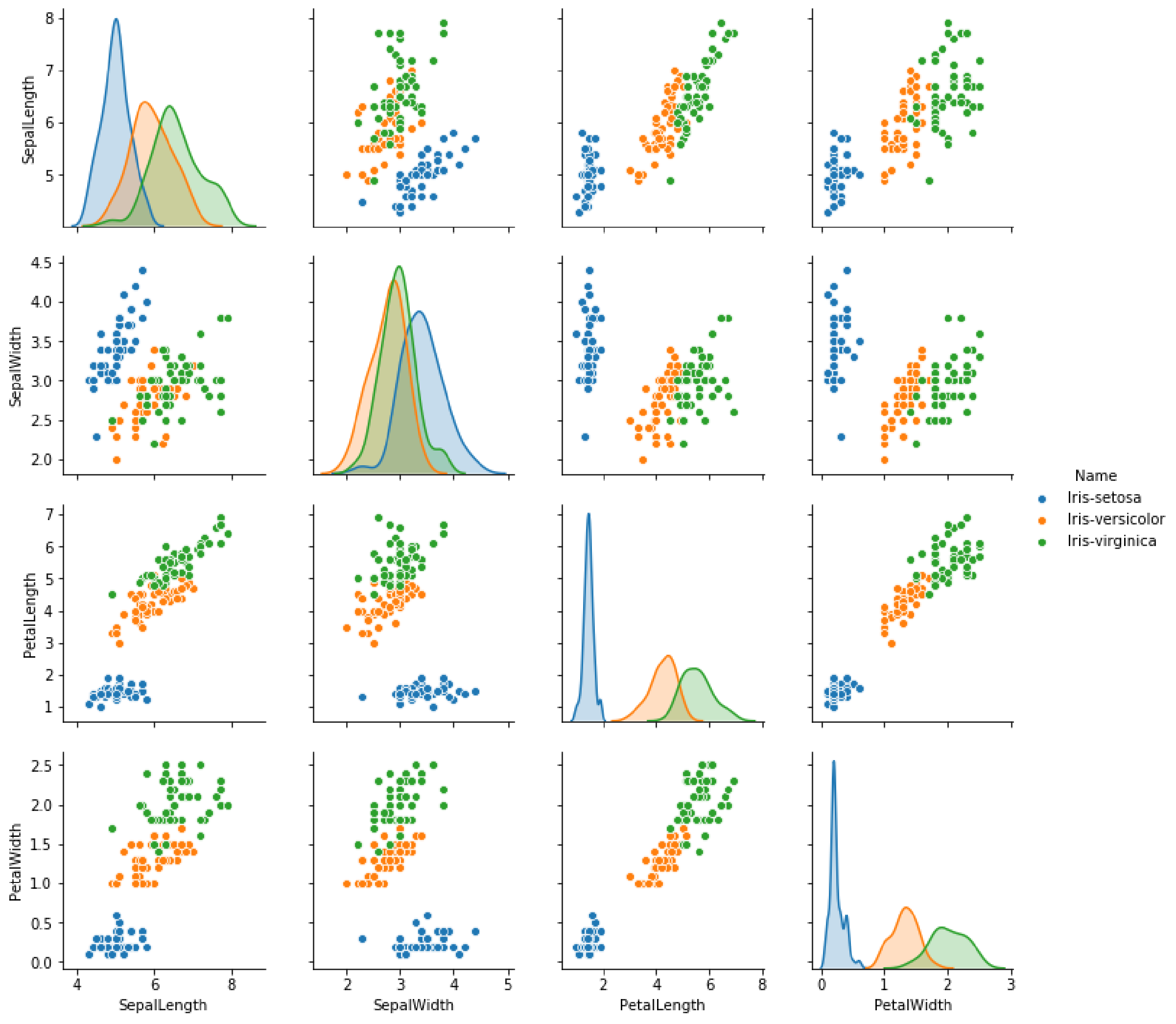
From the above plot, PetalLength and PetalWidth have positive covariance.

Question: Visualize the iris dataset using a pairplot and comment if the three classes can be separated if SepalLength and SepalWidth are the only variables used.

Pair plots allows you to do separate histograms and color scatter plots based on a categorical attribute.

```
In [24]: import seaborn as sns
         sns.pairplot(iris_df, hue="Name")
```

```
Out[24]: <seaborn.axisgrid.PairGrid at 0x2afc585c0490>
```



Answer:

From the above plot, we can observe that for SepalLength and SepalWidth attributes the orange and green datapoints overlap. Hence, SepalLength and SepalWidth attributes cannot separate the three classes.

5. Dimensionality Reduction: PCA

Question: Project points in the digits dataset onto a two-dimensional space using PCA.

Steps involved in PCA

1. Input data: set of points in R^d
2. Compute covariance matrix Σ (a $d \times d$ matrix)
3. Compute Eigenvectors of Σ
4. Select r Eigenvectors (based on a parameter or based on variance explained) corresponding to the highest eigenvalues
5. Project data on to the new r dimensional space

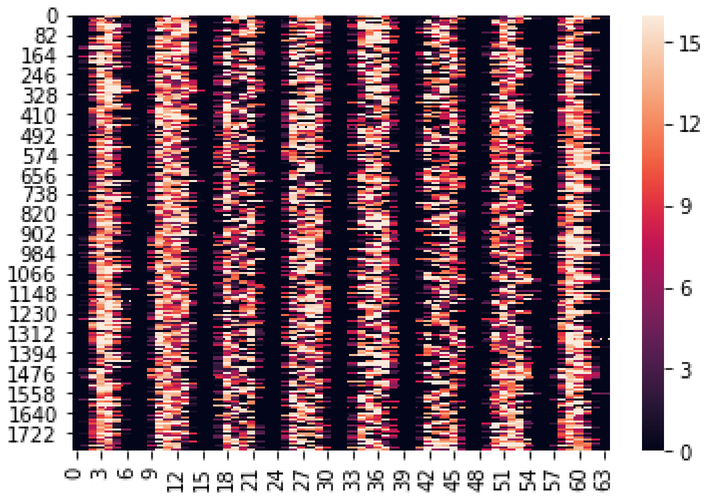
Step 1: Load data

```
In [25]: digits = load_digits()
         digits.data.shape
```

```
Out[25]: (1797, 64)
```

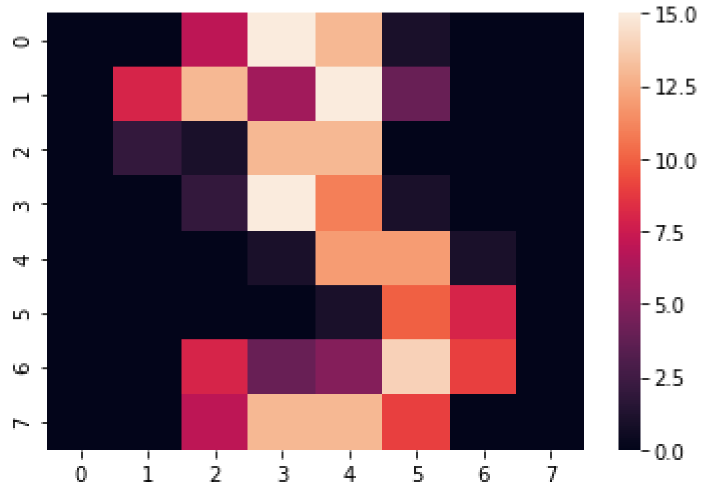
```
In [26]: sns.heatmap(digits.data)
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc59921890>
```



Plotting one data point as an 8x8 image

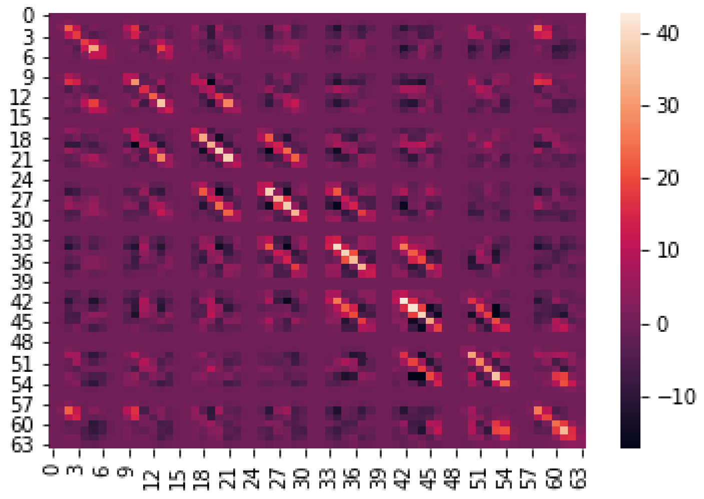
```
In [27]: sns.heatmap(np.reshape(digits.data[3,:],[8,8]))

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5aa0bbd0>
```



Step 2: Compute covariance matrix Σ (a $d \times d$ matrix)

```
In [28]: digits_cov = np.empty([np.size(digits.data,1), np.size(digits.data,1)]);
         for i in range (0, np.size(digits.data,1)):
             for j in range (i, np.size(digits.data,1)):
                 digits_cov[i,j] = mycov(digits.data,i,j);
                 digits_cov[j,i] = digits_cov[i,j];
```

```
In [29]: sns.heatmap(digits_cov)

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5aae72d0>
```



Step 3: Compute Eigenvectors of Σ

```
In [30]: w,v = np.linalg.eig(digits_cov)
In [31]: w
```

```
Out[31]: array([1.79006930e+02, 1.63717747e+02, 1.41788439e+02, 1.01100375e+02,
                6.95131656e+01, 5.91085249e+01, 5.18845391e+01, 4.40151067e+01,
                4.03109953e+01, 3.70117984e+01, 2.85190412e+01, 2.73211698e+01,
                2.19014881e+01, 2.13243565e+01, 1.76367222e+01, 1.69468639e+01,
                1.58513899e+01, 1.50044602e+01, 1.22344732e+01, 1.08868593e+01,
                1.06935663e+01, 9.58259779e+00, 9.22640260e+00, 8.69036872e+00,
                8.36561190e+00, 7.16577961e+00, 6.91973881e+00, 6.19295508e+00,
                5.88499123e+00, 5.15586690e+00, 4.49129656e+00, 4.24687799e+00,
                4.04743883e+00, 3.94340334e+00, 3.70647245e+00, 3.53165306e+00,
                3.08457409e+00, 2.73780002e+00, 2.67210896e+00, 2.54170563e+00,
                2.28298744e+00, 1.90724229e+00, 1.81716569e+00, 1.68996439e+00,
                1.40197220e+00, 1.29221888e+00, 1.15893419e+00, 9.31220008e-01,
                6.69850594e-01, 4.86065217e-01, 2.52350432e-01, 9.91527944e-02,
                6.31307848e-02, 6.07377581e-02, 3.96662297e-02, 1.49505636e-02,
                8.47307261e-03, 3.62365957e-03, 1.27705113e-03, 6.61270906e-04,
                4.12223305e-04, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00])
```

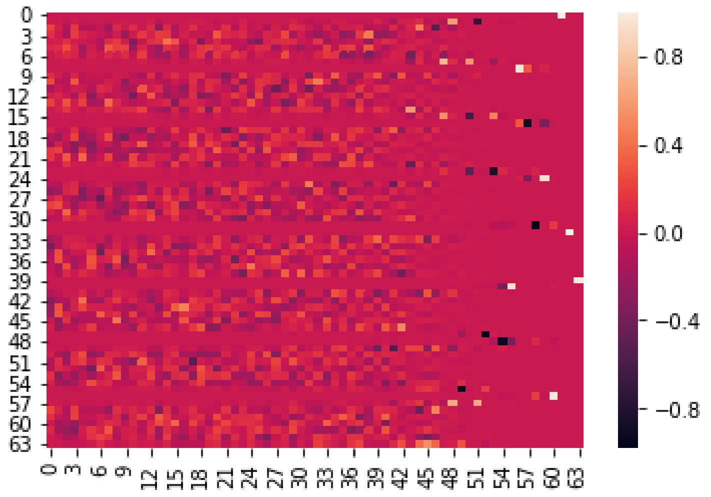
```
In [32]: np.shape(v)
```

```
Out[32]: (64, 64)
```

```
In [33]: np.matmul(v,  np.transpose(v))
```

```
Out[33]: array([[ 1.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
                [ 0.00000000e+00,  1.00000000e+00,  7.50029130e-16, ...,
                 -3.70266621e-16,  3.49763506e-15, -5.05616000e-15],
                [ 0.00000000e+00,  7.50029130e-16,  1.00000000e+00, ...,
                 -5.99586784e-15,  3.70508844e-15,  5.89051975e-15],
                ...,
                [ 0.00000000e+00, -3.70266621e-16, -5.99586784e-15, ...,
                  1.00000000e+00, -4.11493649e-16, -9.40544892e-15],
                [ 0.00000000e+00,  3.49763506e-15,  3.70508844e-15, ...,
                 -4.11493649e-16,  1.00000000e+00,  5.87643883e-15],
                [ 0.00000000e+00, -5.05616000e-15,  5.89051975e-15, ...,
                 -9.40544892e-15,  5.87643883e-15,  1.00000000e+00]])
```

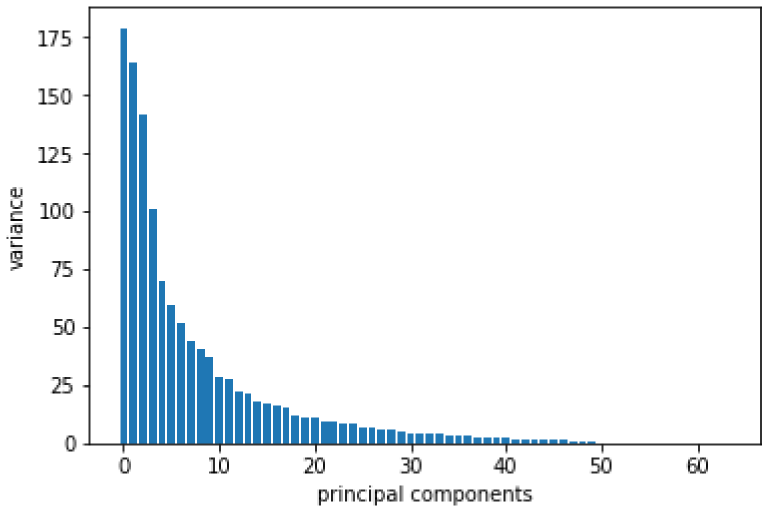
In [34]:
```
sns.heatmap(v)
```
Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2afc5b148a10>
```



Step 4: Select r Eigenvectors (based on a parameter or based on variance explained) corresponding to the highest eigenvalues

Variance captured by each of the principal directions

In [35]:
```
plt.bar(np.arange(64),w)
plt.xlabel('principal components')
plt.ylabel('variance')
```
Out[35]:
```
Text(0,0.5,'variance')
```



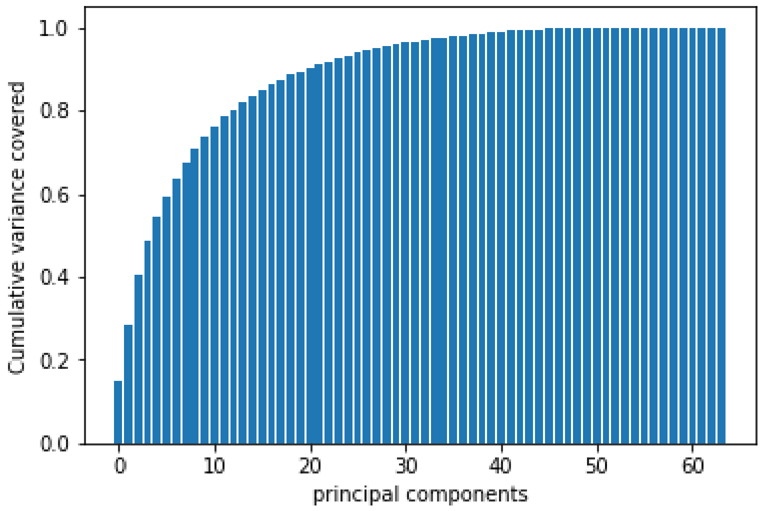
Cumulative variance captured

```
In [36]: np.cumsum(w)/np.sum(w)
```

```
Out[36]: array([0.14890594, 0.28509365, 0.40303959, 0.48713938, 0.54496353,
                0.59413263, 0.6372925 , 0.67390623, 0.70743871, 0.73822677,
                0.76195018, 0.78467714, 0.80289578, 0.82063433, 0.83530534,
                0.84940249, 0.86258838, 0.87506976, 0.88524694, 0.89430312,
                0.9031985 , 0.91116973, 0.91884467, 0.9260737 , 0.93303259,
                0.9389934 , 0.94474955, 0.94990113, 0.95479652, 0.9590854 ,
                0.96282146, 0.96635421, 0.96972105, 0.97300135, 0.97608455,
                0.97902234, 0.98158823, 0.98386565, 0.98608843, 0.98820273,
                0.99010182, 0.99168835, 0.99319995, 0.99460574, 0.99577196,
                0.99684689, 0.99781094, 0.99858557, 0.99914278, 0.99954711,
                0.99975703, 0.99983951, 0.99989203, 0.99994255, 0.99997555,
                0.99998798, 0.99999503, 0.99999804, 0.99999911, 0.99999966,
                1.        , 1.        , 1.        , 1.        ])
```

```
In [37]: plt.bar(np.arange(64),np.cumsum(w)/np.sum(w))
         plt.xlabel('principal components')
         plt.ylabel('Cumulative variance covered')
```

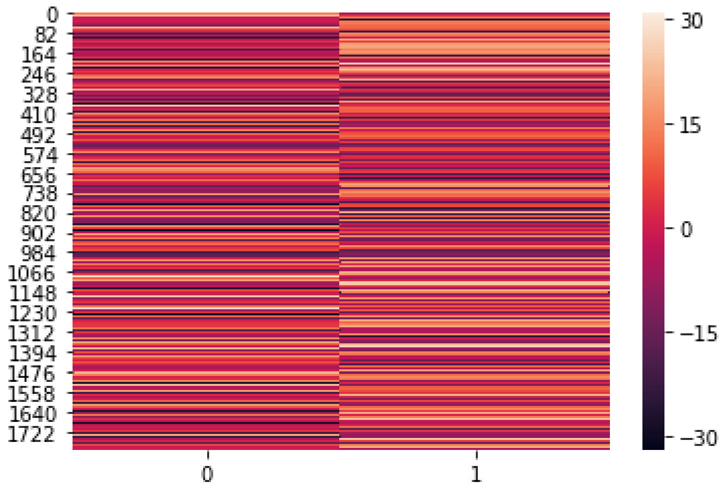
```
Out[37]: Text(0,0.5,'Cumulative variance covered')
```



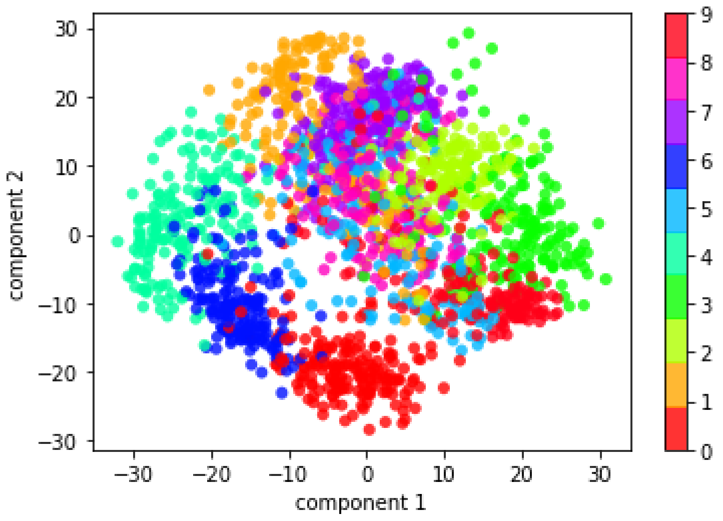
Step 5: Project data on to the new r dimensional space

```
In [38]: projected_data = np.matmul(digits.data,v[:,0:2])
In [39]: sns.heatmap(projected_data)
```

```
Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5b806750>
```



```
In [40]: plt.scatter(projected_data[:, 0], projected_data[:, 1],
                     c=digits.target, edgecolor='none', alpha=0.8,
                     cmap=plt.cm.get_cmap('hsv', 10))
         plt.xlabel('component 1')
         plt.ylabel('component 2')
         plt.colorbar();
```

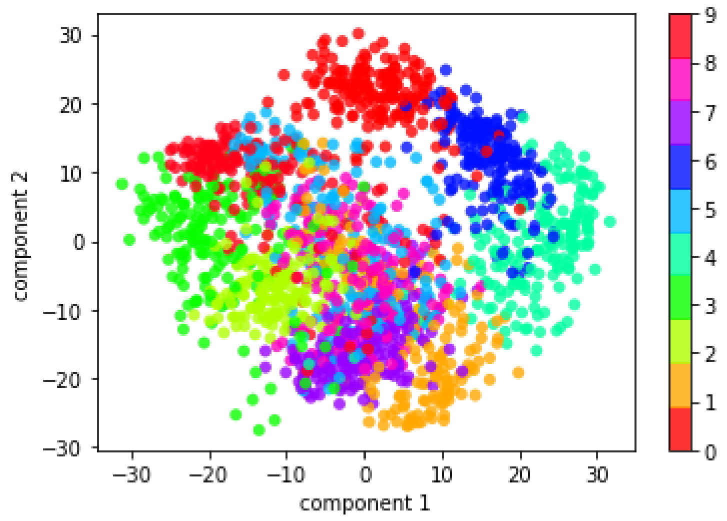


A much simpler way to do PCA using library function from sklearn:

```
In [41]: pca = PCA(2)  # project from 64 to 2 dimensions
         projected = pca.fit_transform(digits.data)
         print(digits.data.shape)
         print(projected.shape)
```

```
(1797, 64)
(1797, 2)
```

In [42]:
```
plt.scatter(projected[:, 0], projected[:, 1],
            c=digits.target, edgecolor='none', alpha=0.8,
            cmap=plt.cm.get_cmap('hsv', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```



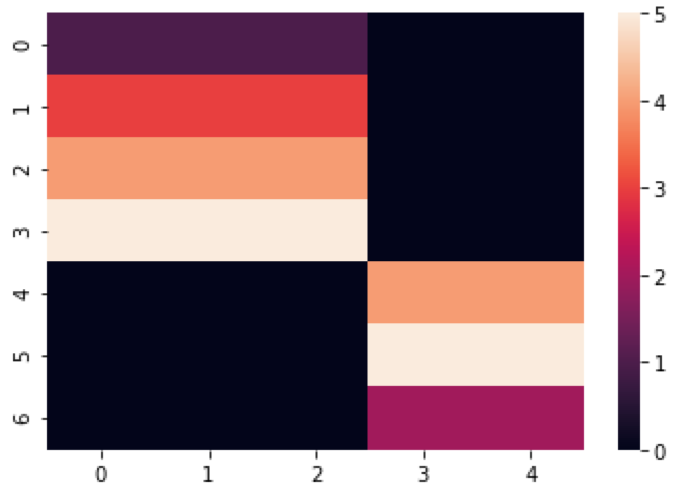
6. Singular Value Decomposition

Question: Compute SVD on the following matrix A using svd() function from scipy library.

In [43]:
```
A = np.array([
    [1, 1, 1, 0, 0],
    [3, 3, 3, 0, 0],
    [4, 4, 4, 0, 0],
    [5, 5, 5, 0, 0],
    [0, 0, 0, 4, 4],
    [0, 0, 0, 5, 5],
    [0, 0, 0, 2, 2]])
```

```
In [44]: sns.heatmap(A)
```

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5bb35bd0>
```

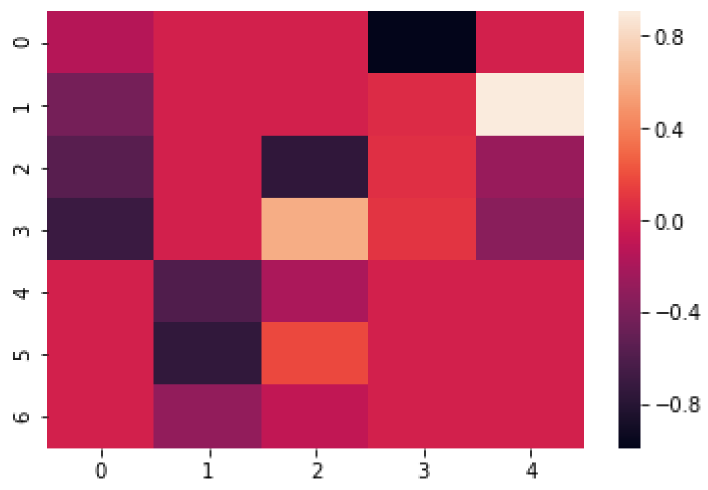


Computing SVD using scipy library function svd()

```
In [45]: U, S, V = svd(A, full_matrices = False)
```

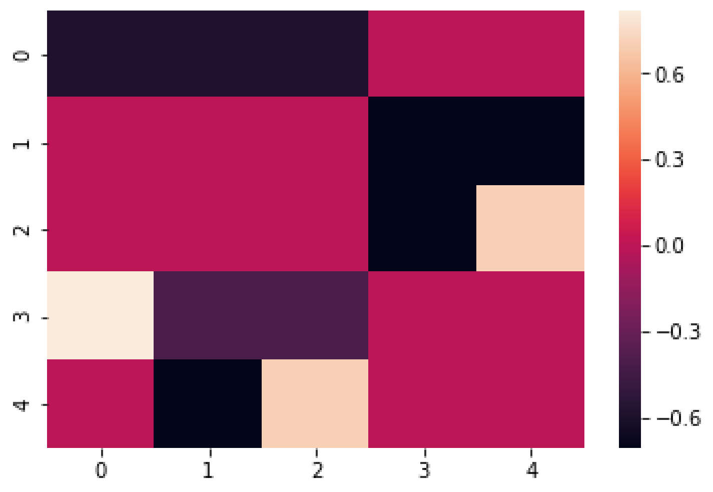
```
In [46]: sns.heatmap(U)
```

```
Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5bc0fc90>
```



```
In [47]: sns.heatmap(V)
```

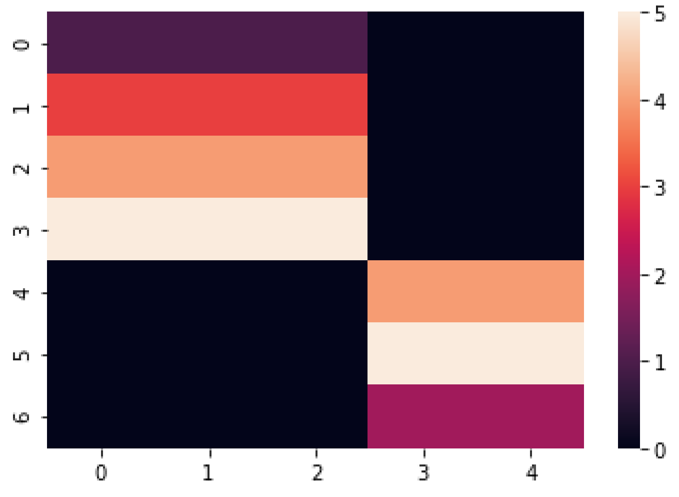
```
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5bcae050>
```



Reconstructing the matrix from the factors

```
In [48]: sns.heatmap(np.matmul(np.matmul(U,np.diag(S)), V))
```

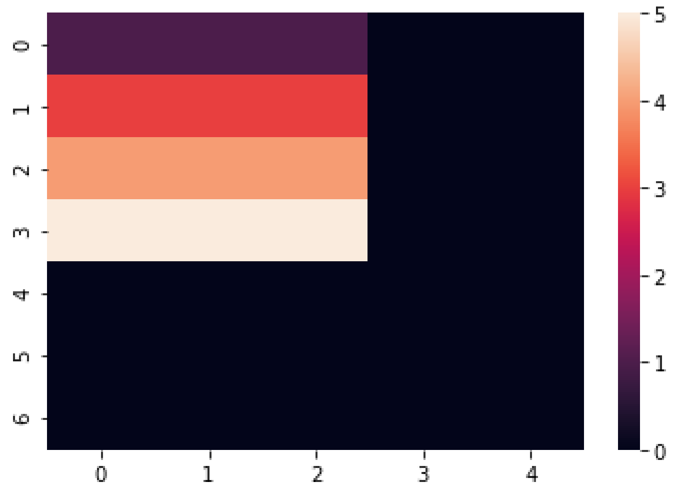
```
Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5bd41f10>
```



Visualizing the spectral decomposition from U and V. The first element in spectral decomposition is $\delta_1 u_1 v_1^T$ is

```
In [49]: sns.heatmap(S[0]*np.outer(U[:,0],V[0,:]))
```

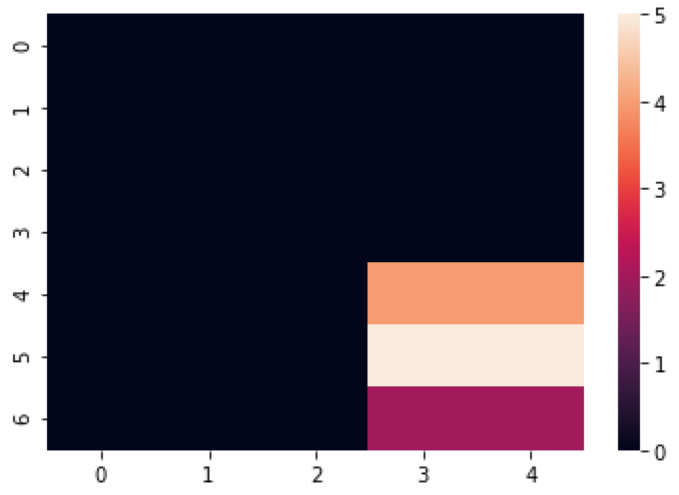
```
Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5be37050>
```



The second element $\delta_2 u_2 v_2^T$ is

```
In [50]: sns.heatmap(S[1]*np.outer(U[:,1],V[1,:]))
```

```
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x2afc5bee6610>
```



Question: Determine number of spectral values that must be used to capture 90% of the data.

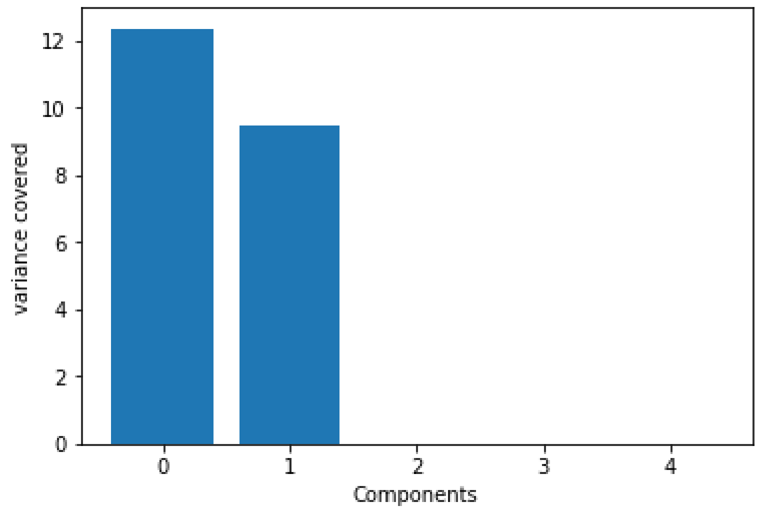
```
In [51]: S
```

```
Out[51]: array([1.23693169e+01, 9.48683298e+00, 3.16341117e-16, 2.88717587e-16,
                3.64663694e-32])
```

First, plotting the variance captured by each spectral value.

```
In [52]: plt.bar(np.arange(5),S)
         plt.xlabel('Components')
         plt.ylabel('variance covered')
```

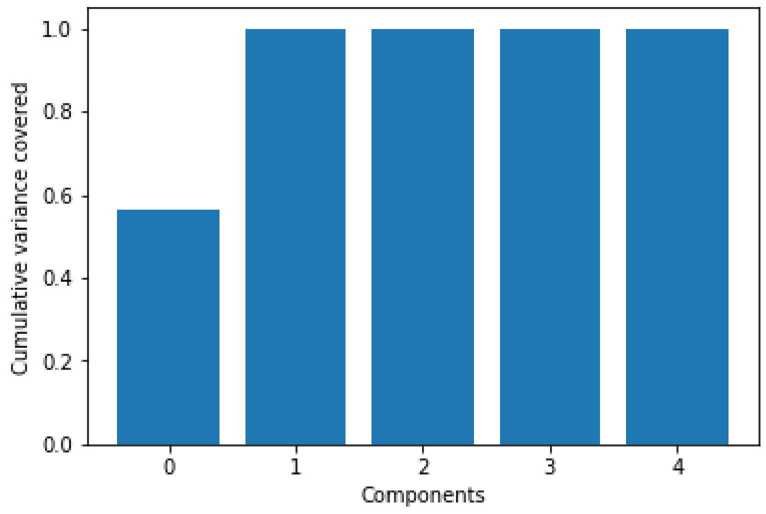
```
Out[52]: Text(0,0.5,'variance covered')
```



Second, plotting the fraction of variance captured by first r spectral values.

```
In [53]: plt.bar(np.arange(5),np.cumsum(S)/np.sum(S))
         plt.xlabel('Components')
         plt.ylabel('Cumulative variance covered')
```

```
Out[53]: Text(0,0.5,'Cumulative variance covered')
```



Answer:

First two spectral values are needed to capture 90% of the variance.

7. Linear Discriminant Analysis

We will use iris data to study LDA.

```
In [54]: X = iris_df.values[:,0:4]
         y = iris_df.values[:,4]
```

We will use the first 100 samples. The first 50 are of the class 'Iris-setosa' and the rest are of the class 'Iris-versicolor'.

```
In [55]: X = X[0:100,:]
```

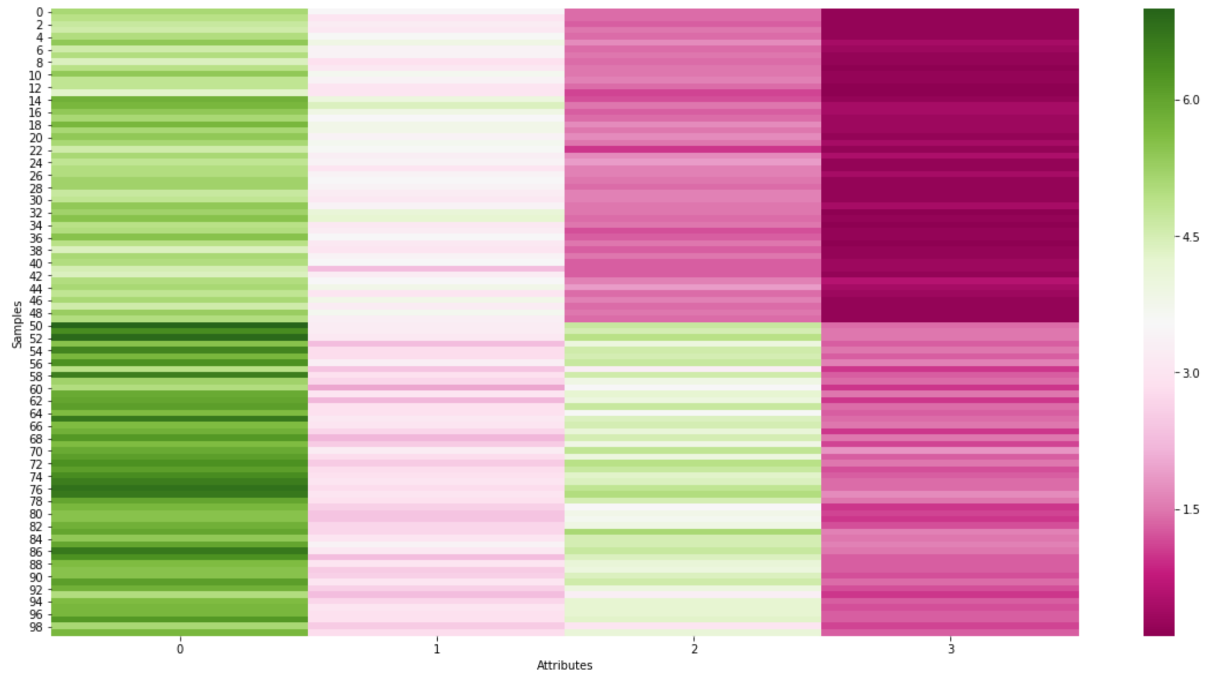
```
In [56]: X = X.astype(float)
```

```
In [57]: y = y[0:100]
```

Question: Plot the heatpmap of the data. And determine which attributes can be used for projection so the two classes are well separated.

```
In [58]: plt.figure(figsize=(20,10))
         ax = sns.heatmap(X,cmap='PiYG')
         ax.set(xlabel='Attributes', ylabel='Samples')
```

```
Out[58]: [Text(159,0.5,'Samples'), Text(0.5,69,'Attributes')]
```



Answer:

The last two attributes are useful to separate the two classes.

Question: Using only the first two attributes, project the selected points in X (below) from the iris dataset using LDA. Determine if the two classes are separated despite choosing the first two attributes. Compute the absolute difference between the two means in the projected space. Hint: Use LinearDiscriminantAnalysis function from scikit library.

Steps involved in performing LDA

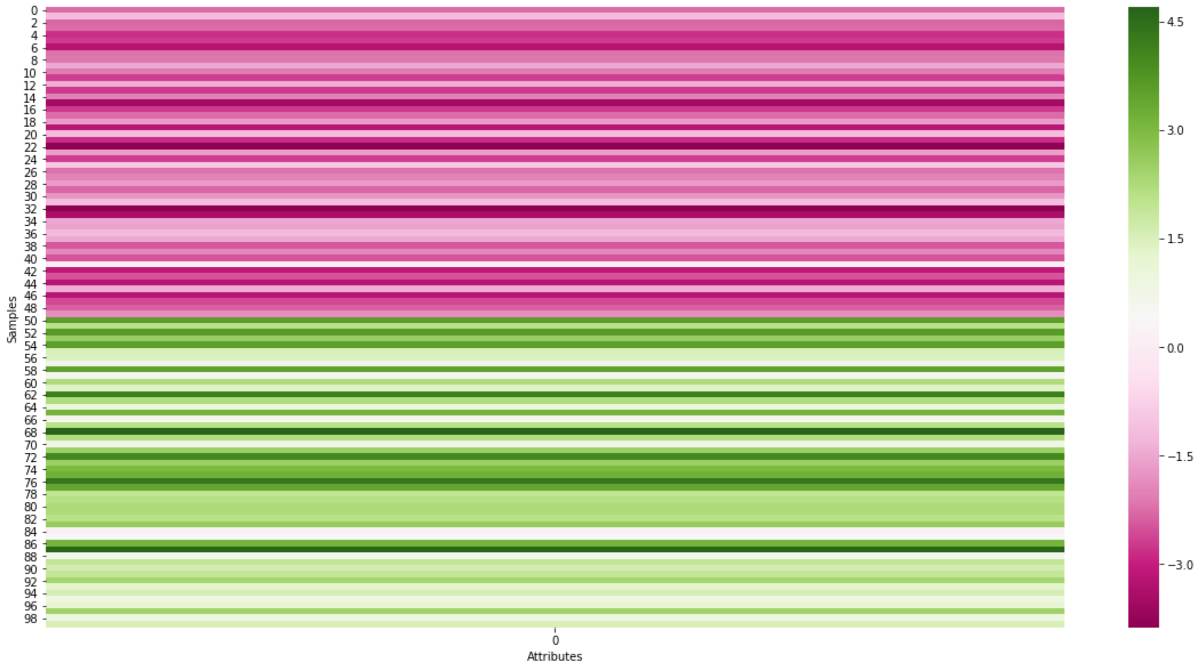
1. Input data: set of points in R^d
2. Compute mean vectors μ_1 and μ_2
3. Compute between class scatter matrix S_B
4. Compute within class scatter matrix S_W
5. Compute the matrix $S_W^{-1} S_B$
6. Compute the first eigenvector (v_1) of the matrix $S_W^{-1} S_B$
7. Project data on to this eigenvector $X v_1$

LinearDiscriminantAnalysis function accomplishes all of this.

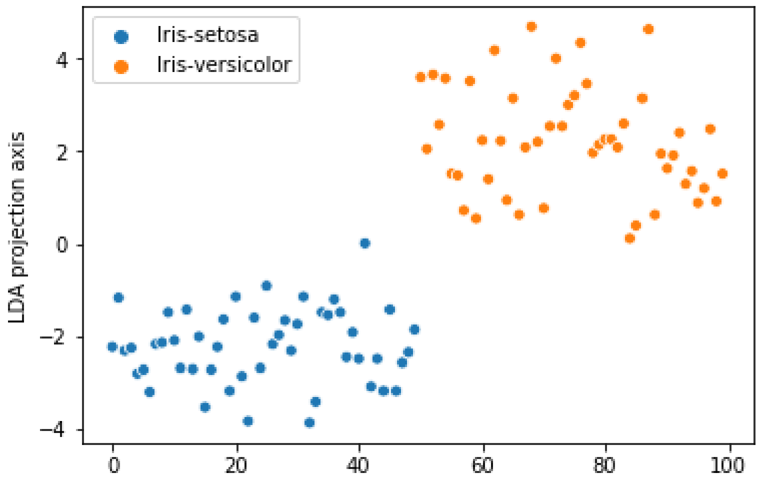
In [59]:
```
lda = LinearDiscriminantAnalysis(n_components=2)
X_r1 = lda.fit(X[:,0:2], y).transform(X[:,0:2])
```

```
In [60]: plt.figure(figsize=(20,10))
         ax = sns.heatmap(X_r1,cmap='PiYG')
         ax.set(xlabel='Attributes', ylabel='Samples')
```

```
Out[60]: [Text(159,0.5,'Samples'), Text(0.5,69,'Attributes')]
```



```
In [61]: fig = sns.scatterplot(x=np.arange(np.size(X_r1)),y=X_r1[:,0],hue=y)
         plt.ylabel('LDA projection axis')
         plt.show(fig)
```



Computing the absolute difference between the means in the projected space.

```
In [62]: abs(np.mean(X_r1[y=='Iris-setosa',0]) - np.mean(X_r1[y=='Iris-versicolo
         r',0]))
```

```
Out[62]: 4.436979775442079
```

```
In [ ]:
```