Principal Component Analysis

Load the Data and Libraries

In [1]:
```
#matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```
plt.style.use("ggplot")
plt.rcParams["figure.figsize"] = (12,8)
```

Iris data can be found at the URL: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data (https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data)
PCA is used for dimensionality reduction. For example: in high dimensional images, bag of words etc -of which a large part is not of use. Distribution of the data is relatively concentrated and not densely occupying the space. High dimensional data is converted to low dimensional data using Linear Discriminant Analysis and Principal Component Analysis. PCA returns the principal components that maximize the variance of the data. It projects the vectors into another feature space.

In [3]:
```
iris = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databa
iris.head()
```

Out[3]:

	0	1	2	3	4
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

In [4]:
```
# setting the column names as described in the iris dataset
iris.columns = ["sepal_length", "sepal_width", "petal_length", "petal_wid
iris.dropna(how= 'all', inplace = True)
iris.head()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
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

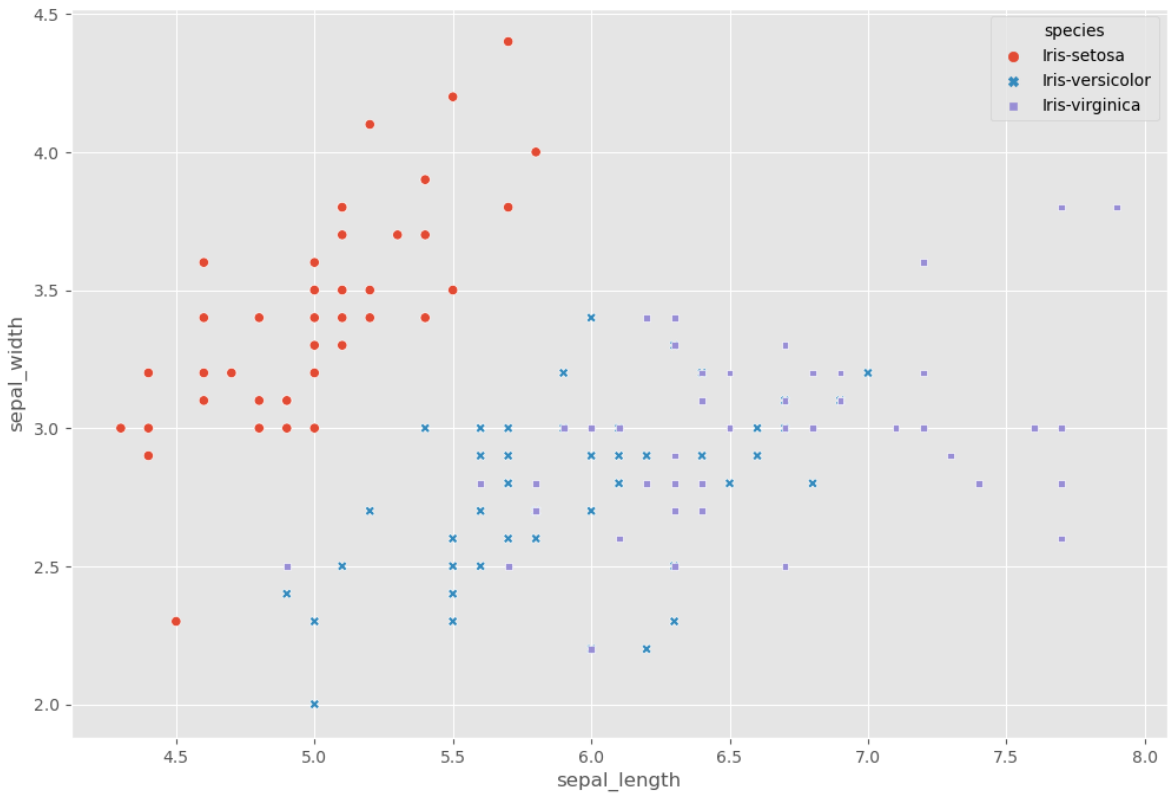
In [5]:
```
1 iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Visualize the Data

In [6]:
```
1 # A scatterplot easily shows if there is any separability in the dataset,
2 # hue sets each species to a unique color and style argument gives legend
3
4 sns.scatterplot(x = iris.sepal_length, y = iris.sepal_width,
5                 hue = iris.species, style = iris.species
6                 )
```

Out[6]: <AxesSubplot:xlabel='sepal_length', ylabel='sepal_width'>



In [7]:
```
1 #Note that two species in blue and purple are not separable. We hope to s
```

Standardize the Data

In [8]:
```
# Aim is to predict the species using the petal and sepal lengths and wid
X = iris.iloc[:,0:4].values
y = iris.species.values
```

In [9]:
```
# Next we standardize the values: i.e. feature_value - feature(mean)/sd(f
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)
```

Compute the Eigenvectors and Eigenvalues

Covariance: $\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^{N} (x_{ij} - \bar{x_j})(x_{ik} - \bar{x_k})$

Coviance matrix: $\Sigma = \frac{1}{n-1}((X - \bar{x})^T (X - \bar{x}))$

In [10]:
```
covariance_matrix = np.cov(X.T)
print("Covariance matrix: \n", covariance_matrix)
```

```
Covariance matrix:
 [[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
```

It is a positive, symmetric, semi-definite matrix. It has the property that it is symmetric. We also constrain each of the columns (eigenvectors) such that the values sum to one. Thus, they are orthonormal to each other.

Eigen vector decomposition of the covriance matrix: $\Sigma = W \wedge W^{-1}$

W has eigen vectors/ basis of the new space that gives the orthogonal directions in the new space and lambda is a measure of how far we need to go in that direction.

In [11]:
```
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
print("Eigenvectors: \n", eigen_vectors, "\n") # each column below is an 
print("Eigenvalues: \n", eigen_values)
```

```
Eigenvectors:
 [[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]

Eigenvalues:
 [2.93035378 0.92740362 0.14834223 0.02074601]
```

Singular Value Decomposition (SVD)

In [12]:
```
# another method to calculate the eigenvalues and eigenvectors is:
eigen_vec_svd, s, v = np.linalg.svd(X.T)
# sign may be different, but it is the same axis, and hence another basis
eigen_vec_svd
```

Out[12]:
```
array([[-0.52237162, -0.37231836,  0.72101681,  0.26199559],
       [ 0.26335492, -0.92555649, -0.24203288, -0.12413481],
       [-0.58125401, -0.02109478, -0.14089226, -0.80115427],
       [-0.56561105, -0.06541577, -0.6338014 ,  0.52354627]])
```

Picking Principal Components Using the Explained Variance

In [13]:
```
for val in eigen_values:
    print(val)
```

```
2.930353775589313
0.9274036215173417
0.1483422264816399
0.020746013995595784
```

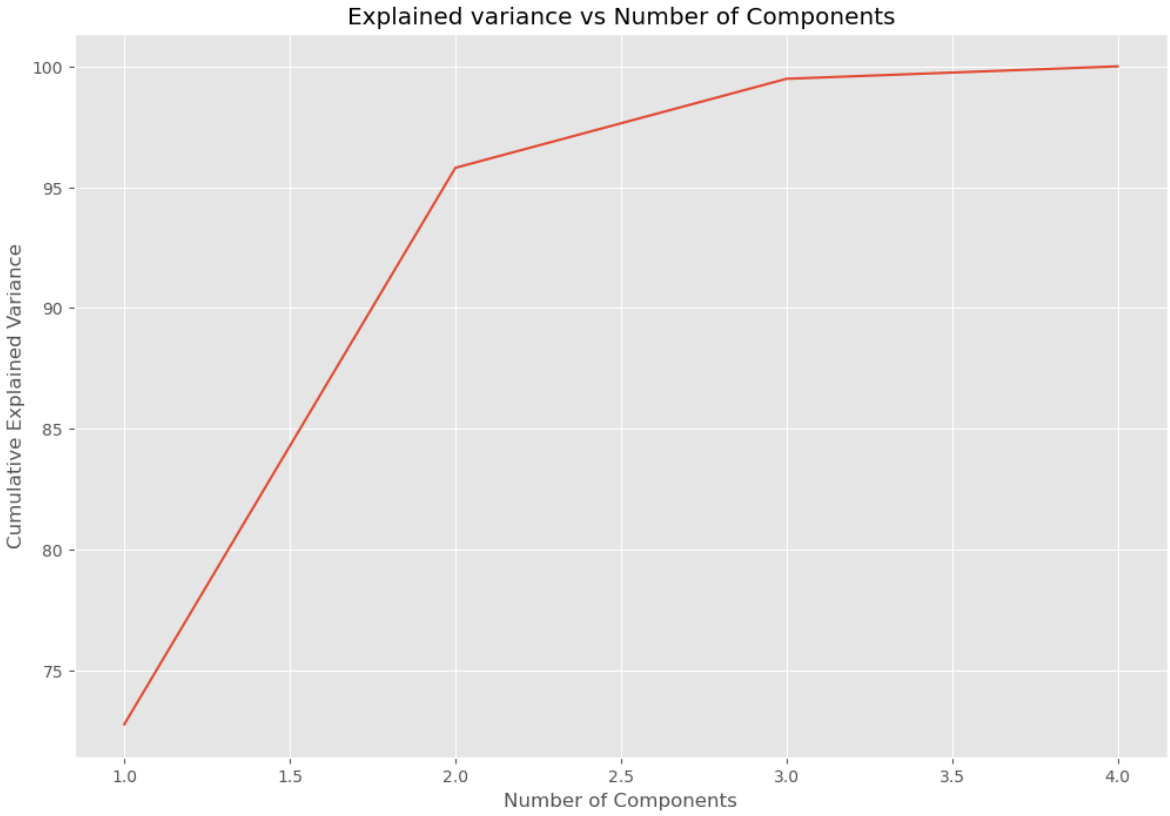
In [14]:
```
# calculate variance explained by each one of these components
variance_explained = [(i/sum(eigen_values))*100 for i in eigen_values]
variance_explained
```

Out[14]:
```
[72.77045209380131, 23.030523267680664, 3.6838319576273926, 0.51519268089062
9]
```

In [15]:
```
# this means that 72.7 percent of the variance is explained by the first
# calculate the cumulative variance
cumulative_variance_explained = np.cumsum(variance_explained)
cumulative_variance_explained
```

Out[15]:
```
array([ 72.77045209,  95.80097536,  99.48480732, 100.        ])
```

In [16]:
```
sns.lineplot(x = [1,2,3,4], y = cumulative_variance_explained)
# x = n => first n components are used to explain the variance.
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Explained variance vs Number of Components")
plt.show()
```



Project Data Onto Lower-Dimensional Linear Subspace

$$X_{pca} = X.W$$

In [17]:
```
eigen_vectors
```

Out[17]:
```
array([[ 0.52237162, -0.37231836, -0.72101681,  0.26199559],
       [-0.26335492, -0.92555649,  0.24203288, -0.12413481],
       [ 0.58125401, -0.02109478,  0.14089226, -0.80115427],
       [ 0.56561105, -0.06541577,  0.6338014 ,  0.52354627]])
```

In [18]:
```
projection_matrix = (eigen_vectors.T[:][:])[:2].T
print("Projection_matrix: \n", projection_matrix)
```

```
Projection_matrix:
 [[ 0.52237162 -0.37231836]
 [-0.26335492 -0.92555649]
 [ 0.58125401 -0.02109478]
 [ 0.56561105 -0.06541577]]
```

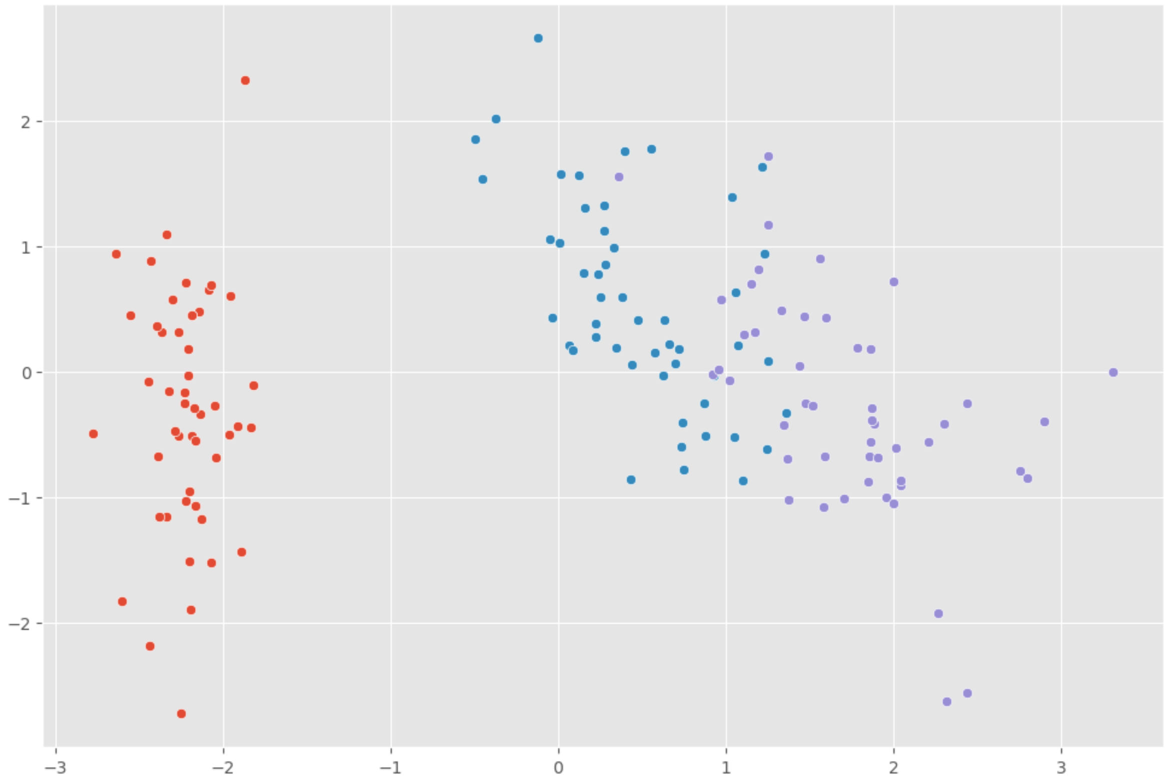
In [19]:
```
X_pca = X.dot(projection_matrix)
```

In [20]:
```
X.shape, X_pca.shape
```

Out[20]:
```
((150, 4), (150, 2))
```

In [22]:
```
for species in ('Iris-setosa','Iris-versicolor', 'Iris-virginica'):
    sns.scatterplot(X_pca[y==species,0],
                    X_pca[y==species,1])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variables as keyword args: x, y. From version 0.
12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```



The two components in each row are plotted against each other for all the species. In the transformed data plotted above, there is a clearer difference in each species. In conclusion, transforming the measurements of a flower helps in deciding which species it belongs to in a better way, given its measurements.

In [ ]:
```
1
```

In [ ]:
```
1
```