In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
```

Exercise 1

Determine PCA of a 3x2 matrix

define a matrix

In [2]:

```
A = np.array([[1, 2], [3, 4], [5, 6]])
print(A)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

First do it manually!

1. Subtract the mean of each variable

In [3]:

```
M = np.mean(A.T, axis=1)
print(M)
C = A - M
print(C)
```

```
[3. 4.]
[[-2. -2.]
 [ 0.  0.]
 [ 2.  2.]]
```

2. Calculate the Covariance Matrix

In [4]:

```
CV = np.cov(C.T)
CV
```

Out[4]:

```
array([[4., 4.],
       [4., 4.]])
```

3. Compute the Eigenvalues and Eigenvectors

In [5]:

```
vals,vecs = np.linalg.eig(CV)
vals,vecs
```

Out[5]:

```
(array([8., 0.]),
 array([[ 0.70710678, -0.70710678],
```

```
       [ 0.70710678,  0.70710678]]))
```

4. project data of the original matrix to the new basis

In [6]:

```
P = vecs.T.dot(C.T)       # C
print(P.T)
```

```
[[-2.82842712  0.        ]
 [ 0.          0.        ]
 [ 2.82842712  0.        ]]
```

conclusion?

Since one of the eigenvalues is zero, the first principal component can be used to explain 100% of the data (i.e., the variance in data) and the second component is not needed i.e., it can be ignored and no data will be lost.

Exercise 2

Ok Let's do it again but for a larger matrix 20x5

Generate a dummy dataset.

In [7]:

```
X = np.random.randint(10,50,100).reshape(20,5)
print(X)
```

```
[[12 34 41 45 11]
 [27 41 45 29 15]
 [16 37 49 40 46]
 [30 24 45 35 44]
 [15 41 39 29 49]
 [47 22 18 26 42]
 [28 17 18 13 40]
 [35 14 12 21 49]
 [49 13 10 44 35]
 [23 25 18 43 34]
 [13 44 22 30 17]
 [22 42 31 46 47]
 [38 39 11 14 39]
 [42 43 27 14 19]
 [42 33 40 47 19]
 [21 27 18 49 40]
 [18 17 29 26 10]
 [17 17 34 34 26]
 [45 48 13 32 48]
 [16 30 15 27 21]]
```

1. Subtract the mean of each variable

Subtract the mean of each variable from the dataset so that the dataset should be centered on the origin. Doing this proves to be very helpful when calculating the covariance matrix.

In [8]:

```
M = X - np.mean(X, axis=0)
print(M)
```

```
[[-15.8    3.6   14.25  12.8  -21.55]
 [ -0.8   10.6   18.25  -3.2  -17.55]
 [-11.8    6.6   22.25   7.8   -13.45]
```

```
[-11.8    6.6   22.25   7.8   15.45]
 [  2.2   -6.4   18.25   2.8   11.45]
 [-12.8   10.6   12.25  -3.2   16.45]
 [ 19.2   -8.4   -8.75  -6.2    9.45]
 [  0.2  -13.4   -8.75 -19.2    7.45]
 [  7.2  -16.4  -14.75 -11.2   16.45]
 [ 21.2  -17.4  -16.75  11.8    2.45]
 [ -4.8   -5.4   -8.75  10.8    1.45]
 [-14.8   13.6   -4.75  -2.2  -15.55]
 [ -5.8   11.6    4.25  13.8   14.45]
 [ 10.2    8.6  -15.75 -18.2    6.45]
 [ 14.2   12.6    0.25 -18.2  -13.55]
 [ 14.2    2.6   13.25  14.8  -13.55]
 [ -6.8   -3.4   -8.75  16.8    7.45]
 [ -9.8  -13.4    2.25  -6.2  -22.55]
 [-10.8  -13.4    7.25   1.8   -6.55]
 [ 17.2   17.6  -13.75  -0.2   15.45]
 [-11.8   -0.4  -11.75  -5.2  -11.55]]
```

2. Calculate the Covariance Matrix

Calculate the Covariance Matrix of the mean-centered data.

In [9]:

```
CV = np.cov(M.T)
print(CV)
```

```
[[152.90526316 -14.54736842 -66.84210526 -28.74736842  43.53684211]
 [-14.54736842 126.98947368  38.31578947   1.28421053  -6.65263158]
 [-66.84210526  38.31578947 166.72368421  45.68421053 -39.53947368]
 [-28.74736842   1.28421053  45.68421053 129.95789474  -0.22105263]
 [ 43.53684211  -6.65263158 -39.53947368  -0.22105263 190.36578947]]
```

Note: the matrix is symmetrical

3. Compute the Eigenvalues and Eigenvectors

Now, compute the Eigenvalues and Eigenvectors for the calculated Covariance matrix.

In [10]:

```
vals, vecs = np.linalg.eig(CV)
print(vals, vecs)
```

```
[290.21460109  78.66795941 101.05503055 128.10783592 168.89667829] [[ 0.5281115  -0.32587
311 -0.7549703  -0.19663541 -0.07911595]
 [-0.21138344  0.43849267 -0.14009677 -0.8380584   0.20266285]
 [-0.60669024 -0.70814683 -0.12076168 -0.12030215  0.31843215]
 [-0.27004608  0.43670647 -0.55149089  0.4937686   0.4340656 ]
 [ 0.48519826 -0.09667691  0.30276586 -0.02669449  0.81415997]]
```

Note: The Eigenvectors of the Covariance matrix we get are Orthogonal to each other and each vector represents a principal axis. A Higher Eigenvalue corresponds to a higher variability. Hence the principal axis with the higher Eigenvalue will be an axis capturing higher variability in the data.

4. Sort Eigenvalues in descending order

Sort the Eigenvalues in the descending order along with their corresponding Eigenvector.

In [11]:

```
ST = vecs[:,np.argsort(vals)[::-1]]
print(ST)
```

```
[[ 0.5281115  -0.07911595 -0.19663541 -0.7549703  -0.32587311]
```

```
[[ 0.5281115  -0.07911595 -0.19663541 -0.7549703  -0.32587311]
 [-0.21138344  0.20266285 -0.8380584  -0.14009677  0.43849267]
 [-0.60669024  0.31843215 -0.12030215 -0.12076168 -0.70814683]
 [-0.27004608  0.4340656   0.4937686  -0.55149089  0.43670647]
 [ 0.48519826  0.81415997 -0.02669449  0.30276586 -0.09667691]]
```

Note: Each column in the Eigen vector-matrix corresponds to a principal component, so arranging them in descending order of their Eigenvalue will automatically arrange the principal component in descending order of their variability. Hence the first column in our rearranged Eigen vector-matrix will be a principal component that captures the highest variability.

5. Select a subset from the rearranged Eigenvalue matrix

Select a subset of n first eigenvectors from the rearranged Eigenvector matrix as per our need, n is desired dimension of your final reduced data. i.e. "n_components=2" means you selected the first two principal components.

In [12]:

```
PC2 = ST[:,:2]
print(PC2)
```

```
[[ 0.5281115  -0.07911595]
 [-0.21138344  0.20266285]
 [-0.60669024  0.31843215]
 [-0.27004608  0.4340656 ]
 [ 0.48519826  0.81415997]]
```

Note: The final dimensions of X_reduced will be (20, 2) and originally the data was of higher dimensions (20, 5).

6. Transform the data

Finally, transform the data by having a dot product between the Transpose of the Eigenvector subset and the Transpose of the mean-centered data. By transposing the outcome of the dot product, the result we get is the data reduced to lower dimensions from higher dimensions.

In [13]:

```
np.matmul(M, PC2)
```

Out[13]:

```
array([[-31.66309038,  -5.47183128],
       [-21.38633262,  -7.6546117 ],
       [-16.70614696,  23.69242165],
       [ -3.75800642,  14.87780478],
       [ -7.58678815,  19.06562582],
       [ 23.48331054,  -1.00507049],
       [ 17.04631184,  -7.78635437],
       [ 27.22379981,  -0.05878295],
       [ 23.03828911,  -3.42066426],
       [  1.70211479,   2.36753629],
       [-14.75981797, -11.20055377],
       [ -4.80904912,  21.91781503],
       [ 21.1685784 ,  -6.72805066],
       [  3.02448152, -17.42214808],
       [-11.66017784,  -0.98499386],
       [  1.51403802,  10.4204473 ],
       [-12.97494275, -22.27438755],
       [-10.93370185,  -4.10402652],
       [ 21.25548235,  10.31958811],
       [ -3.21835232, -14.54976346]])
```

In [14]:

```
vals.sort()
```

```
vals = vals[::-1]
vals
```

Out[14]:

```
array([290.21460109, 168.89667829, 128.10783592, 101.05503055,
        78.66795941])
```

In [15]:

```
explained_variance  = (sum(vals[:2])/sum(vals))*100
print("The explained variance is:", explained_variance)
```

```
The explained variance is: 59.862573228411634
```

Here as we can see from from the above calculation, 52% of the variance can be explained using the first 2 principal components.

Exercise 3

Now, let's just combine everything above by making a function and try our Principal Component analysis from scratch on an example.

Create a PCA function accepting data matrix and the number of components as input arguments.

In [16]:

```
def PCA(X, n_components=2):
    M = np.mean(X.T, axis=1)
    C = X - M
    CV = np.cov(C.T)
    vals,vecs = np.linalg.eig(CV)
    vecs_sorted = vecs[:,np.argsort(vals)[::-1]]
    return (vals, np.matmul(C, vecs_sorted[:,:n_components]))
```

Let's use the IRIS dataset to test our PCA function, and by the same way see if we can classify the dataset in the projected space

In [17]:

```
#Get the IRIS dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
data = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal width'
,'target'])
data.head()
```

Out[17]:

	sepal length	sepal width	petal length	petal width	target
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

1. prepare the dataset & target set for classification

In [18]:

```
X = np.array(data.drop('target', axis=1))
```

2. Apply the PCA function

In [19]:

```
eigen_vals,res = PCA(X, n_components=2)
```

3. Create a Pandas Dataframe of reduced Dataset with target data

In [20]:

```
res = pd.DataFrame(res, columns=['c1','c2'])
res['target'] = data['target']
res.head()
```

Out[20]:

	c1	c2	target
0	**-2.684207**	**-0.326607**	**Iris-setosa**
1	**-2.715391**	**0.169557**	**Iris-setosa**
2	**-2.889820**	**0.137346**	**Iris-setosa**
3	**-2.746437**	**0.311124**	**Iris-setosa**
4	**-2.728593**	**-0.333925**	**Iris-setosa**

4. Vizualize the data with one and two principal components

In [21]:

```
res['target'].unique()
```

Out[21]:

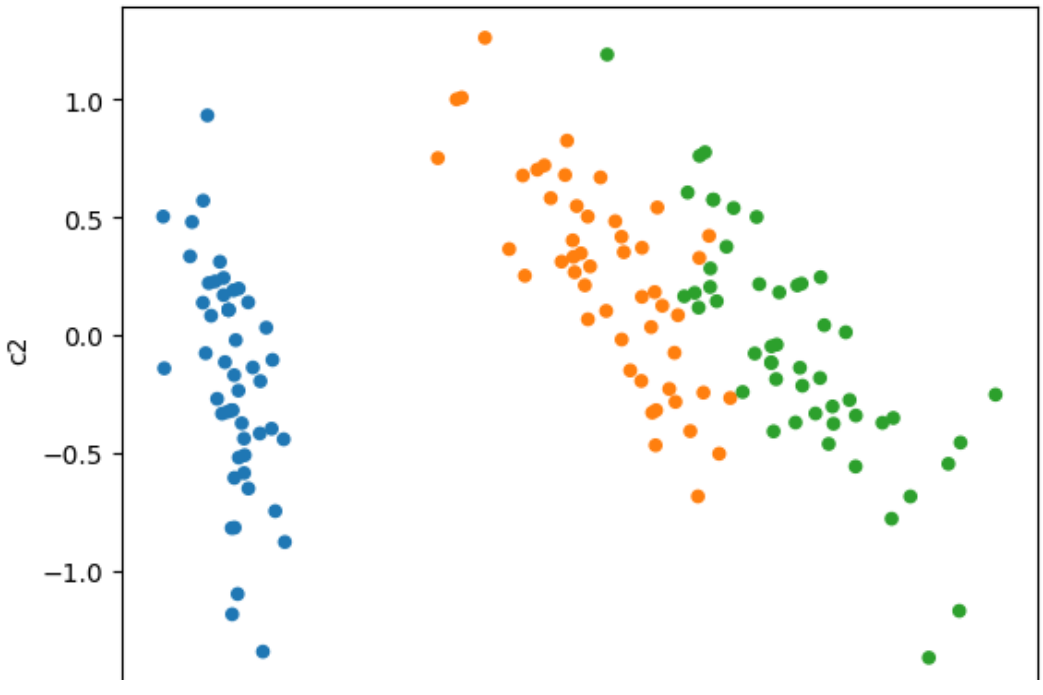
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

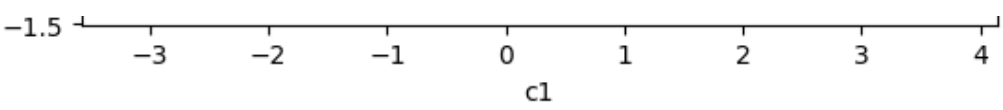
In [22]:

```
colormap = {'Iris-setosa':'tab:blue', 'Iris-versicolor':'tab:orange', 'Iris-virginica':'
tab:green'}
res.plot(kind='scatter',x='c1',y='c2', c=res['target'].map(colormap))
```

Out[22]:

```
<AxesSubplot:xlabel='c1', ylabel='c2'>
```



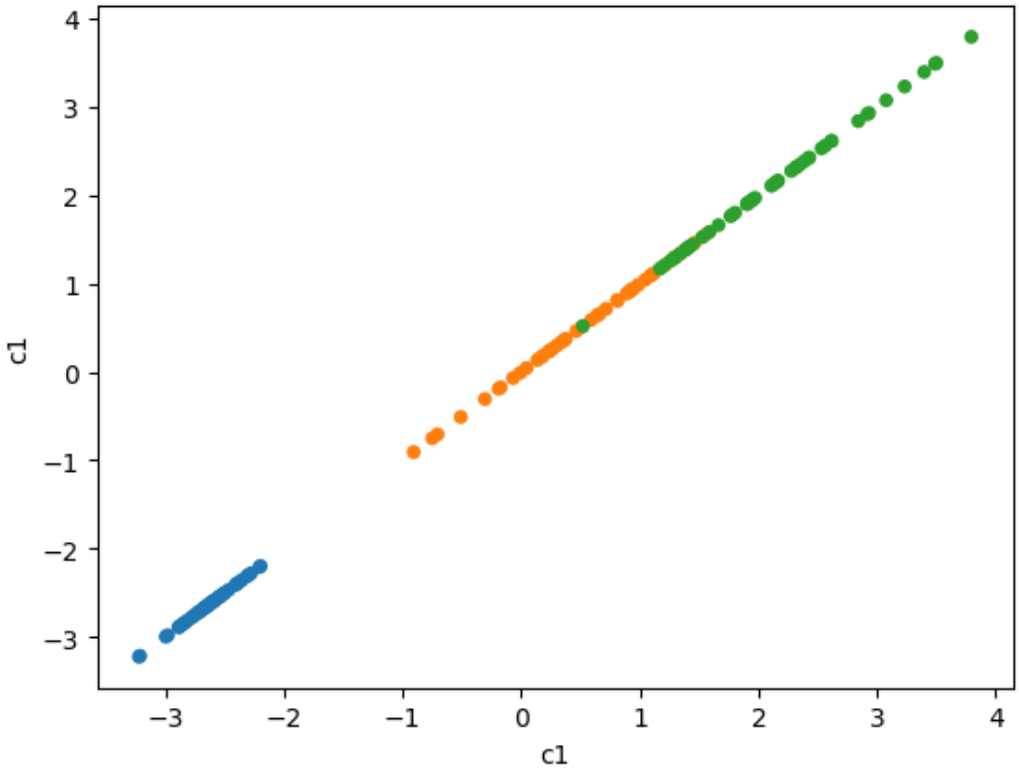


In [23]:

```
colormap = {'Iris-setosa':'tab:blue', 'Iris-versicolor':'tab:orange', 'Iris-virginica':'
tab:green'}
res[['c1','target']].plot(kind='scatter',x='c1',y='c1', c=res['target'].map(colormap))
```

Out[23]:

```
<AxesSubplot:xlabel='c1', ylabel='c1'>
```



In [24]:

```
explained_variance_1  = (sum(eigen_vals[:1])/sum(eigen_vals))*100
print("The explained variance for PC1 is:", explained_variance_1)
explained_variance_2  = (sum(eigen_vals[:2])/sum(eigen_vals))*100
print("The explained variance for PC1+PC2 is:", explained_variance_2)
```

```
The explained variance for PC1 is: 92.46162071742684
The explained variance for PC1+PC2 is: 97.76317750248035
```

Here we can clearly see that with PC1 92.46% of the data can be explained.
**With PC1 and PC2, 97.76% of the data can be explained. **

More?

Go to: https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

In [ ]:

```

```