In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
```

In [4]:
```
mat = array([[1,2],[3,4],[5,6]])
mat
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [5]:
```
mean(mat)
```

```
3.5
```

In [6]:
```
M = mean(mat.T, axis = 1)
M
```

```
array([3., 4.])
```

In [7]:
```
c = mat-M
c
```

```
array([[-2., -2.],
       [ 0.,  0.],
       [ 2.,  2.]])
```

In [8]:
```
v = cov(c.T)
v
```

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

In [9]:
```
values,vectors = eig(v)
print(vectors)
print(values)
```

```
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
[8. 0.]
```

In [10]:
```
P = vectors.T.dot(c.T)
print(P.T)
```

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

In [18]:
```
class pca :
    def __init__(self,n_components):
        self.n_components = n_components
        self.components = None
        self.mean= None

    def fit(self,x):
        self.mean = np.mean(x,axis = 0)
        x = x - self.mean
        cov = np.cov(x.T)

        eigenvalues,eigenvectors = np.linalg.eig(cov)

        eigenvectors = eigenvectors.T
        idxs = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]

        self.components = eigenvectors[0:self.n_components]


    def transform(self,x):
        x = x - self.mean
        return np.dot(x,self.components.T)
```

In [14]:

```
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np

```

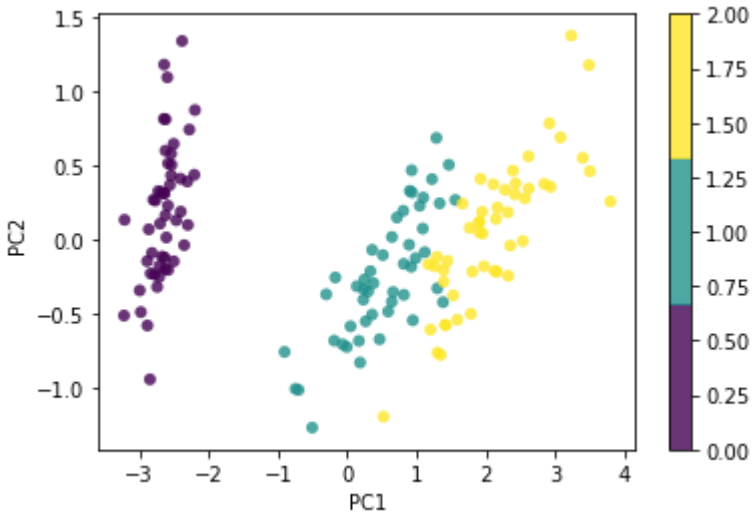
In [16]:

```
data = datasets.load_iris()
x = data.data
y = data.target

pca = PCA(2)
pca.fit(x)
x_projected = pca.transform(x)
print("shape of x : ", x.shape)
print("shape of transformed x : ", x_projected.shape)
x1 = x_projected[:,0]
x2 = x_projected[:,1]
plt.scatter(x1,x2, c=y ,edgecolor = 'none',alpha = 0.8 , cmap = plt.cm.get_cı
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.colorbar()
plt.show()

```

```
shape of x :  (150, 4)
shape of transformed x :  (150, 2)
```



In [19]:
```
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.model_selection import train_test_split
import pandas as pd
```

In [20]:
```
mnist = fetch_openml('mnist_784')
```

In [21]:
```
mnist
```

```
{'data': array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]),
 'target': array(['5', '0', '4', ..., '4', '5', '6'], dtype=object),
 'frame': None,
 'feature_names': ['pixel1',
  'pixel2',
  'pixel3',
  'pixel4',
  'pixel5',
  'pixel6',
  'pixel7',
  'pixel8',
  'pixel9',
  'pixel10',
  'pixel11',
  'pixel12',
  'pixel13',
  'pixel14',
  'pixel15'
```

In [22]:
```
mnist.data.shape
```

```
(70000, 784)
```

In [24]:
```
mnist.target.shape
```

```
(70000,)
```

In [25]:
```
train_img, test_img, train_lbl, test_lbl = train_test_split(
    mnist.data, mnist.target, test_size=1/7.0, random_state=0)
```

In [27]:
```
print(train_img.shape)
print(train_lbl.shape)
print(test_img.shape)
print(test_lbl.shape)
```

```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
```

In [28]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(train_img)
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

In [29]:
```
pca = PCA(.95)
pca.fit(train_img)
pca.n_components_
```

```
327
```

In [30]:
```
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

In [33]:
```
from sklearn.linear_model import LogisticRegression
```

In [36]:
```
logisticRegr = LogisticRegression(solver = 'lbfgs')
```

In [37]:
```
logisticRegr.fit(train_img, train_lbl)
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_m
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)


LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [38]:
```
logisticRegr.predict(test_img[0].reshape(1,-1))
```

```
array(['0'], dtype=object)
```

In [41]:
```
logisticRegr.predict(test_img[0:10])
```

```
array(['0', '4', '1', '2', '4', '7', '7', '1', '1', '7'], dtype=object)
```

In [39]:
```
score = logisticRegr.score(test_img, test_lbl)
print(score)
```

```
0.9201
```

In [40]:

```
pd.DataFrame(data = [[1.00, 784, 48.94, .9158],
                     [.99, 541, 34.69, .9169],
                     [.95, 330, 13.89, .92],
                     [.90, 236, 10.56, .9168],
                     [.85, 184, 8.85, .9156]],
             columns = ['Variance Retained',
                        'Number of Components',
                        'Time (seconds)',
                        'Accuracy'])
```

	Variance Retained	Number of Components	Time (seconds)	Accuracy
0	1.00	784	48.94	0.9158
1	0.99	541	34.69	0.9169
2	0.95	330	13.89	0.9200
3	0.90	236	10.56	0.9168
4	0.85	184	8.85	0.9156