# DIPAYAN TEWARI , ROLL:193110003

HUMAN FACE EXPRESSION RECOGNITION ON OLIVETTI DATASET USING PCA

In this project, I am going to use the face images present in the olivetti data set and try to build a model to predict the facial expressions. I will use PCA to obtain the principal components of face images.

In [50]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

#Visualization
import matplotlib.pyplot as plt

#Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import metrics

#System
import os
print(os.listdir("../input"))
```

```
['olivetti_faces.npy', 'olivetti_faces_target.npy']
```

In [51]:

```
import warnings
warnings.filterwarnings('ignore')
print("Warnings ignored!!")
```

```
Warnings ignored!!
```

In [53]:

```
data=np.load("../input/olivetti_faces.npy")
target=np.load("../input/olivetti_faces_target.npy")
```

In [62]:

```
print("There are {} images in the dataset".format(len(data)))
print("There are {} unique targets in the dataset".format(len(np.unique(target))))
print("Size of each image is {}x{}".format(data.shape[1],data.shape[2]))
print("Pixel values were scaled to [0,1] interval. e.g:{}".format(data[0][0,:4]))
```

```
There are 400 images in the dataset
There are 40 unique targets in the dataset
Size of each image is 64x64
Pixel values were scaled to [0,1] interval. e.g:[0.30991736 0.3677686  0.41735536 0.44214877]
```

In [55]:

```
print("unique target number:",np.unique(target))
```

```
unique target number: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]
```

In [57]:

```
def show_40_distinct_people(images, unique_ids):
    #Creating 4X10 subplots in  18x9 figure size
    fig, axarr=plt.subplots(nrows=4, ncols=10, figsize=(18, 9))
    #For easy iteration flattened 4X10 subplots matrix to 40 array
    axarr=axarr.flatten()

    #iterating over user ids
    for unique_id in unique_ids:
        image_index=unique_id*10
        axarr[unique_id].imshow(images[image_index], cmap='gray')
        axarr[unique_id].set_xticks([])
        axarr[unique_id].set_yticks([])
        axarr[unique_id].set_title("face id:{}".format(unique_id))
    plt.suptitle("There are 40 distinct people in the dataset")
```

In [58]:

```
show_40_distinct_people(data, np.unique(target))
```

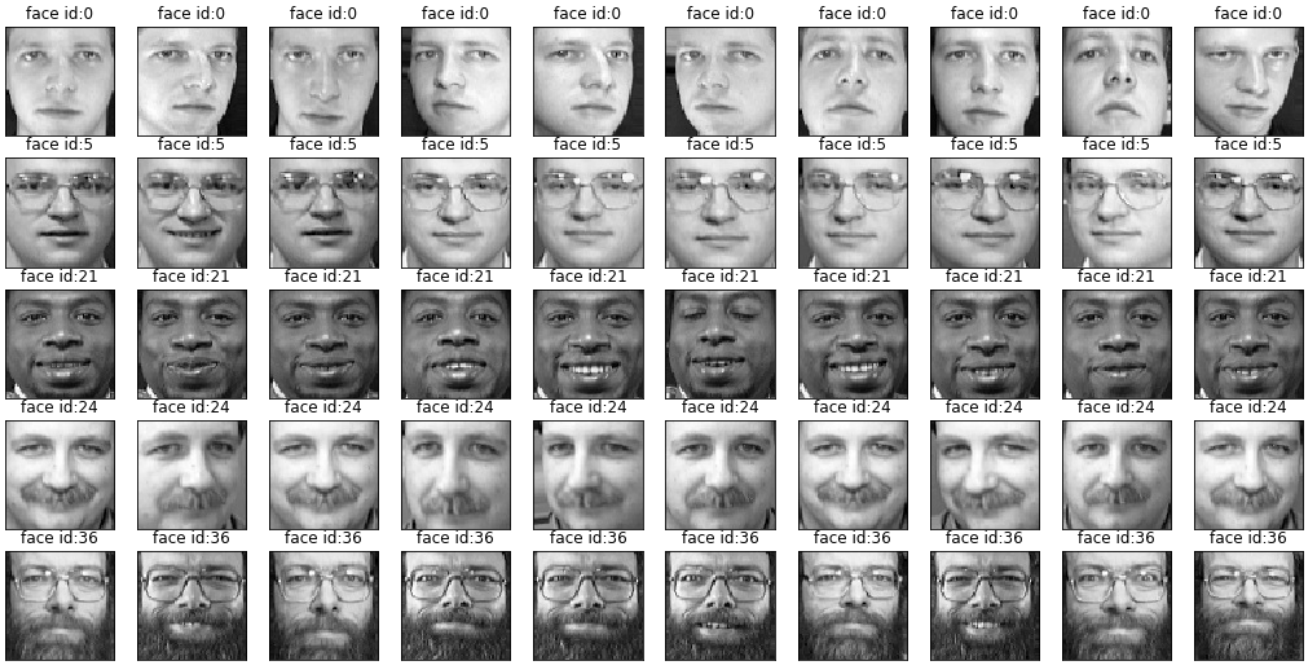

As seen in the photo gallery above, the data set has 40 different person-owned, facial images.

In [59]:

```
def show_10_faces_of_n_subject(images, subject_ids):
    cols=10# each subject has 10 distinct face images
    rows=(len(subject_ids)*10)/cols #
    rows=int(rows)

    fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(18,9))
    #axarr=axarr.flatten()

    for i, subject_id in enumerate(subject_ids):
        for j in range(cols):
            image_index=subject_id*10 + j
            axarr[i,j].imshow(images[image_index], cmap="gray")
            axarr[i,j].set_xticks([])
            axarr[i,j].set_yticks([])
            axarr[i,j].set_title("face id:{}".format(subject_id))
```

In [9]:

```
show_10_faces_of_n_subject(images=data, subject_ids=[0,5, 21, 24, 36])
```




Each face of a subject has different characteristic in context of varying lighting, facial express and facial detail(glasses, beard)

Machine Learning Model fo Face Recognition

Machine learning models can work on vectors. Since the image data is in the matrix form, it must be converted to a vector.

In [60]:

```
#We reshape images for machine learnig  model
X=data.reshape((data.shape[0],data.shape[1]*data.shape[2]))
print("X shape:",X.shape)
```

```
X shape: (400, 4096)
```

In [61]:

```
X_train, X_test, y_train, y_test=train_test_split(X, target, test_size=0.3, stratify=target, random
_state=0)
print("X_train shape:",X_train.shape)
print("y_train shape:{}".format(y_train.shape))
```

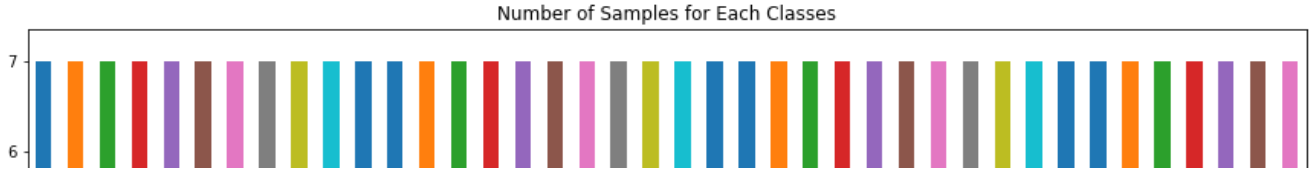
```
X_train shape: (280, 4096)
y_train shape:(280,)
```

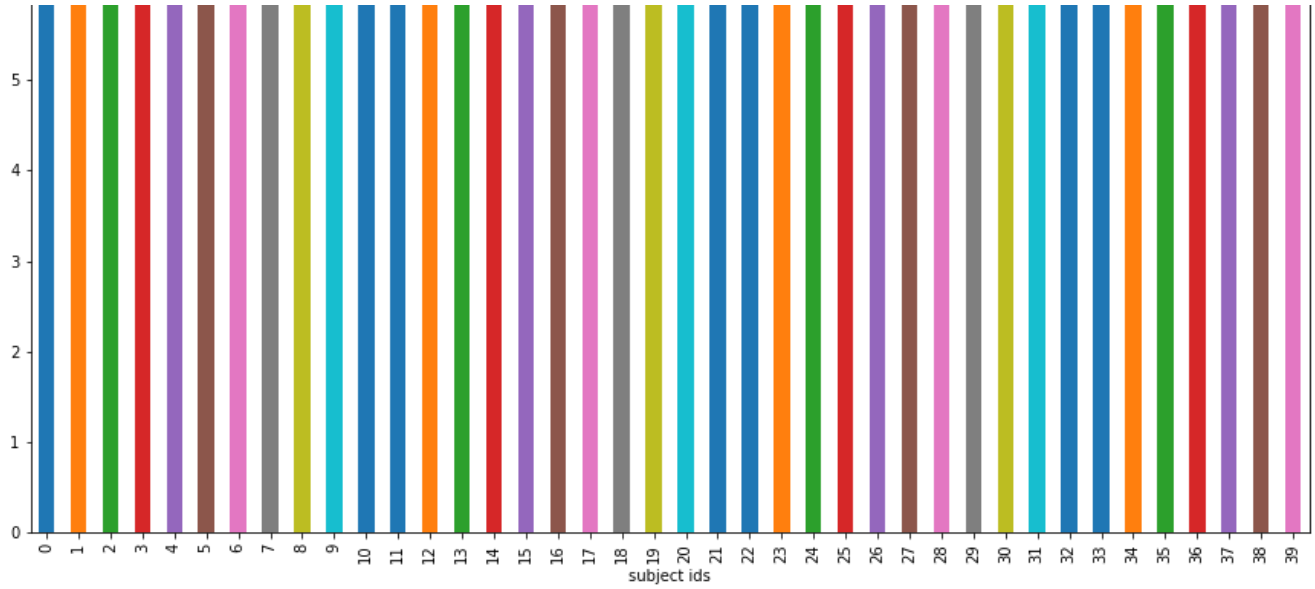
In [12]:

```
y_frame=pd.DataFrame()
y_frame['subject ids']=y_train
y_frame.groupby(['subject ids']).size().plot.bar(figsize=(15,8),title="Number of Samples for Each
Classes")
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ec2897128>
```





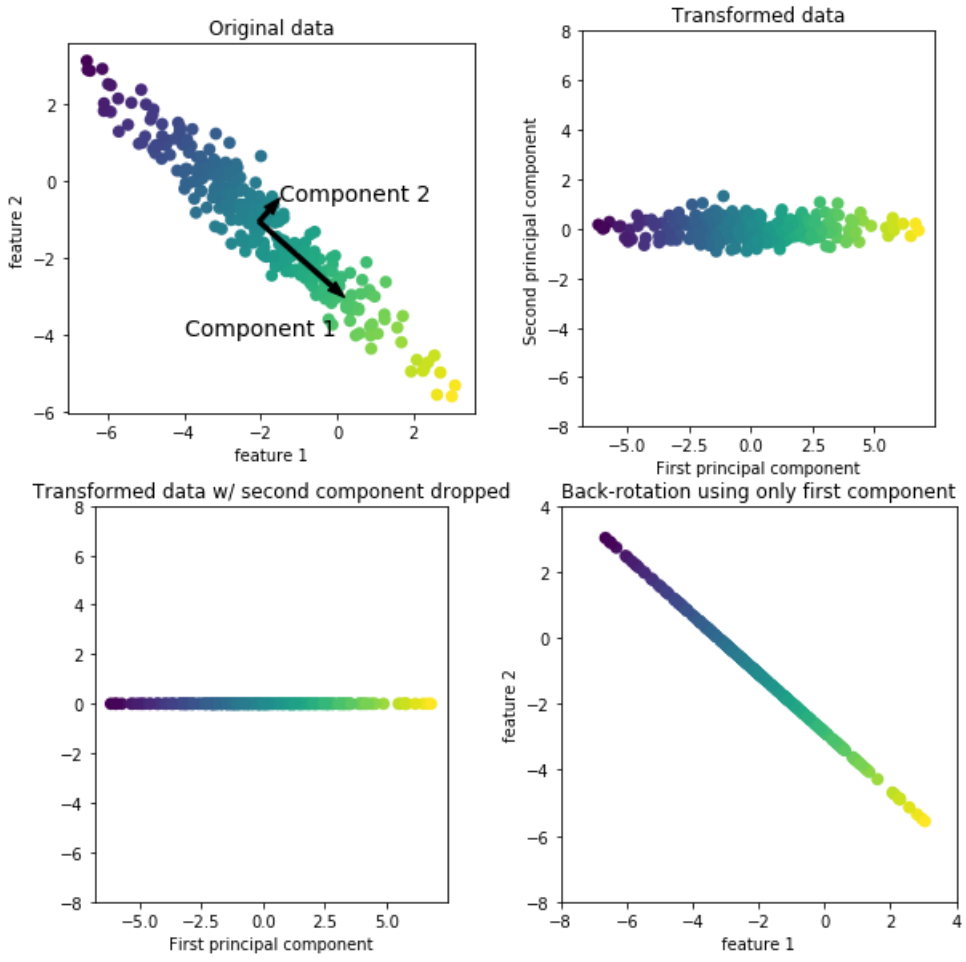

In [63]:

```
import mglearn
```

Principle Component Analysis

In [14]:

```
mglearn.plots.plot_pca_illustration()
```




The above illustration shows a simple example on a synthetic two-dimensional data set. The first drawing shows the original data points colored to distinguish points. The algorithm first proceeds by finding the direction of the maximum variance labeled "Component 1". This refers to the direction in which most of the data is associated, or in other words, the properties that are most

related to each other.

Then, when the algorithm is orthogonal (at right angle), it finds the direction that contains the most information in the first direction. There are only one possible orientation in two dimensions at a right angle, but there will be many orthogonal directions (infinite) in high dimensional spaces.

PCA Projection of Define Numberd of Target

In [49]:

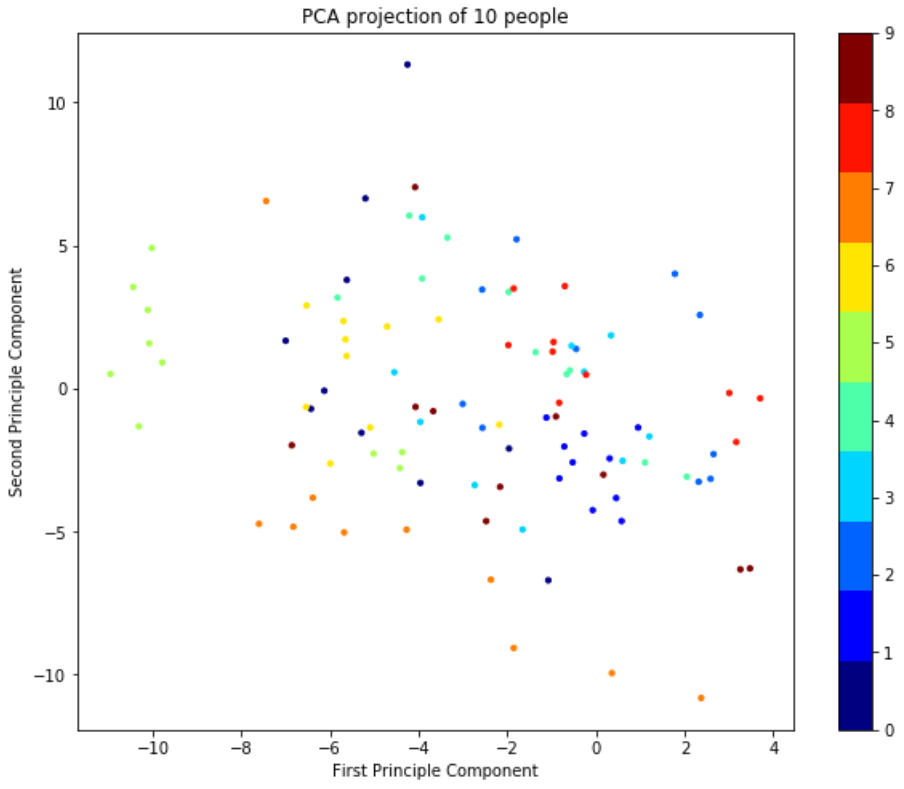
```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca.fit(X)
X_pca=pca.transform(X)
```

In [16]:

```
number_of_people=10
index_range=number_of_people*10
fig=plt.figure(figsize=(10,8))
ax=fig.add_subplot(1,1,1)
scatter=ax.scatter(X_pca[:index_range,0],
            X_pca[:index_range,1],
            c=target[:index_range],
            s=10,
           cmap=plt.get_cmap('jet', number_of_people)
          )

ax.set_xlabel("First Principle Component")
ax.set_ylabel("Second Principle Component")
ax.set_title("PCA projection of {} people".format(number_of_people))

fig.colorbar(scatter)
```

Out[16]:

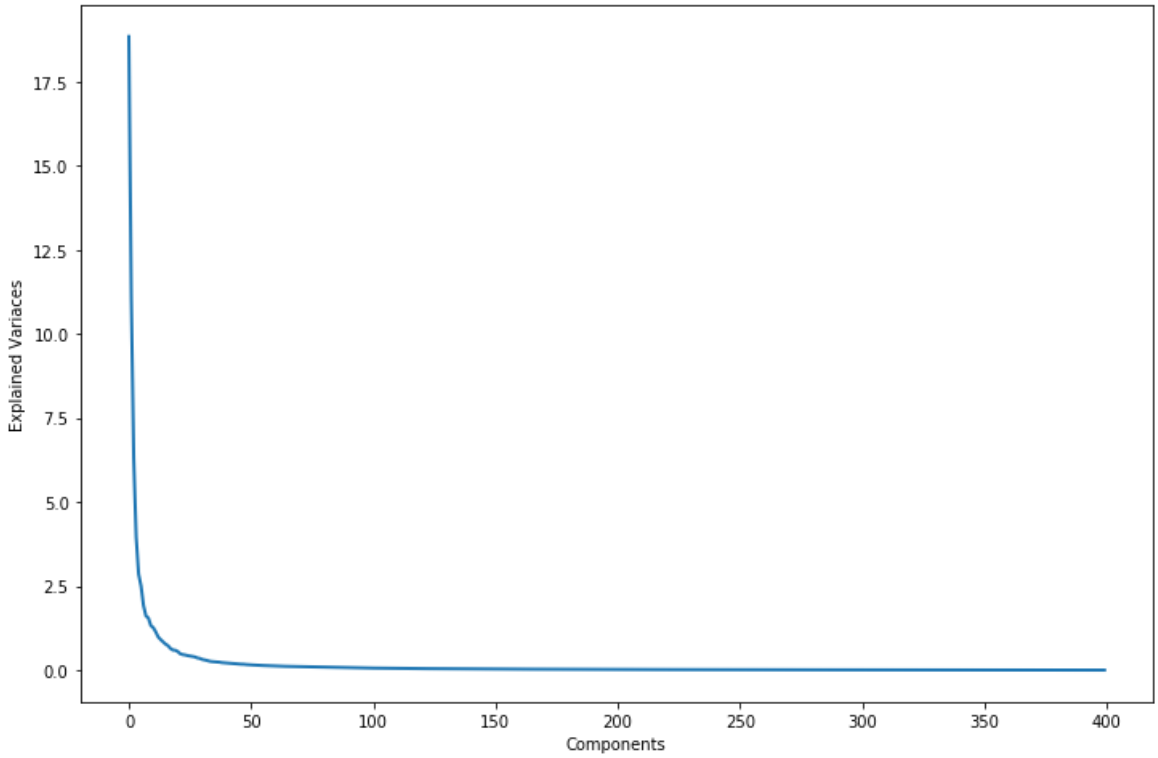
```
<matplotlib.colorbar.Colorbar at 0x7f8ebee68550>
```




Finding Optimum Number of Principle Component

In [17]:

```
pca=PCA()
pca.fit(X)

plt.figure(1, figsize=(12,8))

plt.plot(pca.explained_variance_, linewidth=2)

plt.xlabel('Components')
plt.ylabel('Explained Variaces')
plt.show()
```




In the figure above, it can be seen that 90 and more PCA components represent the same data. Now let's make the classification process using 90 PCA components.

In [18]:

```
n_components=90
```

In [19]:

```
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train)
```

Out[19]:

```
PCA(copy=True, iterated_power='auto', n_components=90, random_state=None,
  svd_solver='auto', tol=0.0, whiten=True)
```

Show Average Face

In [20]:

```
fig,ax=plt.subplots(1,1,figsize=(8,8))
ax.imshow(pca.mean_.reshape((64,64)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Face')
```

Out[20]:

```
Text(0.5,1,'Average Face')
```

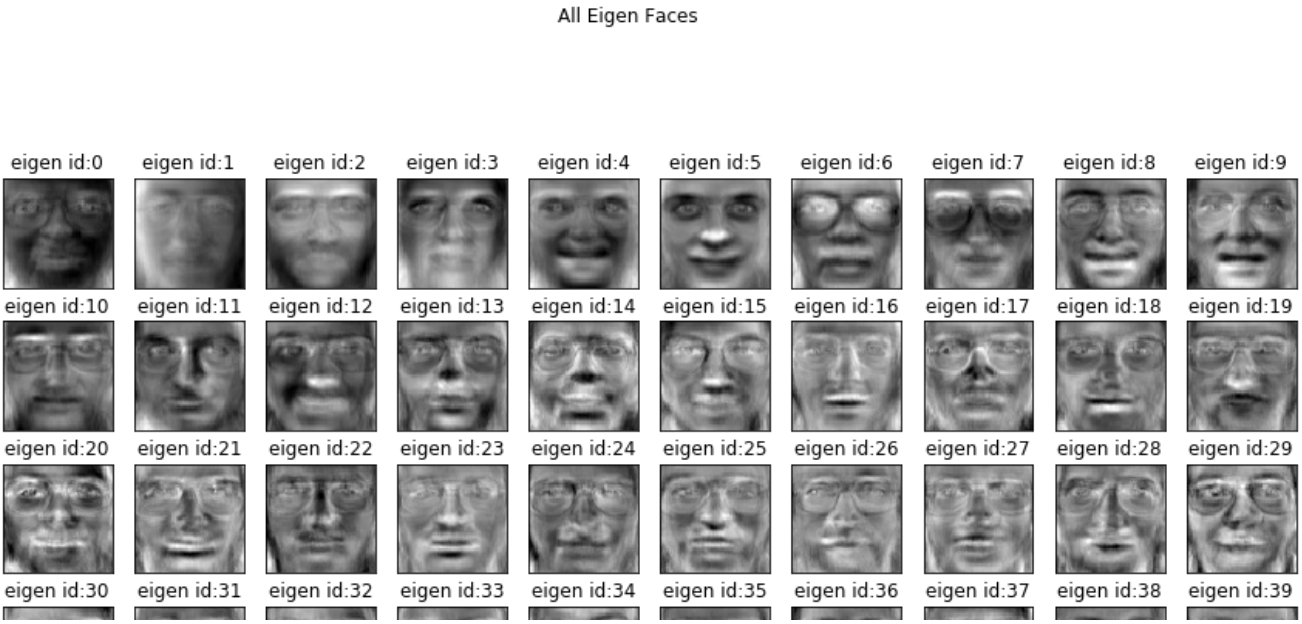

Show Eigen Faces

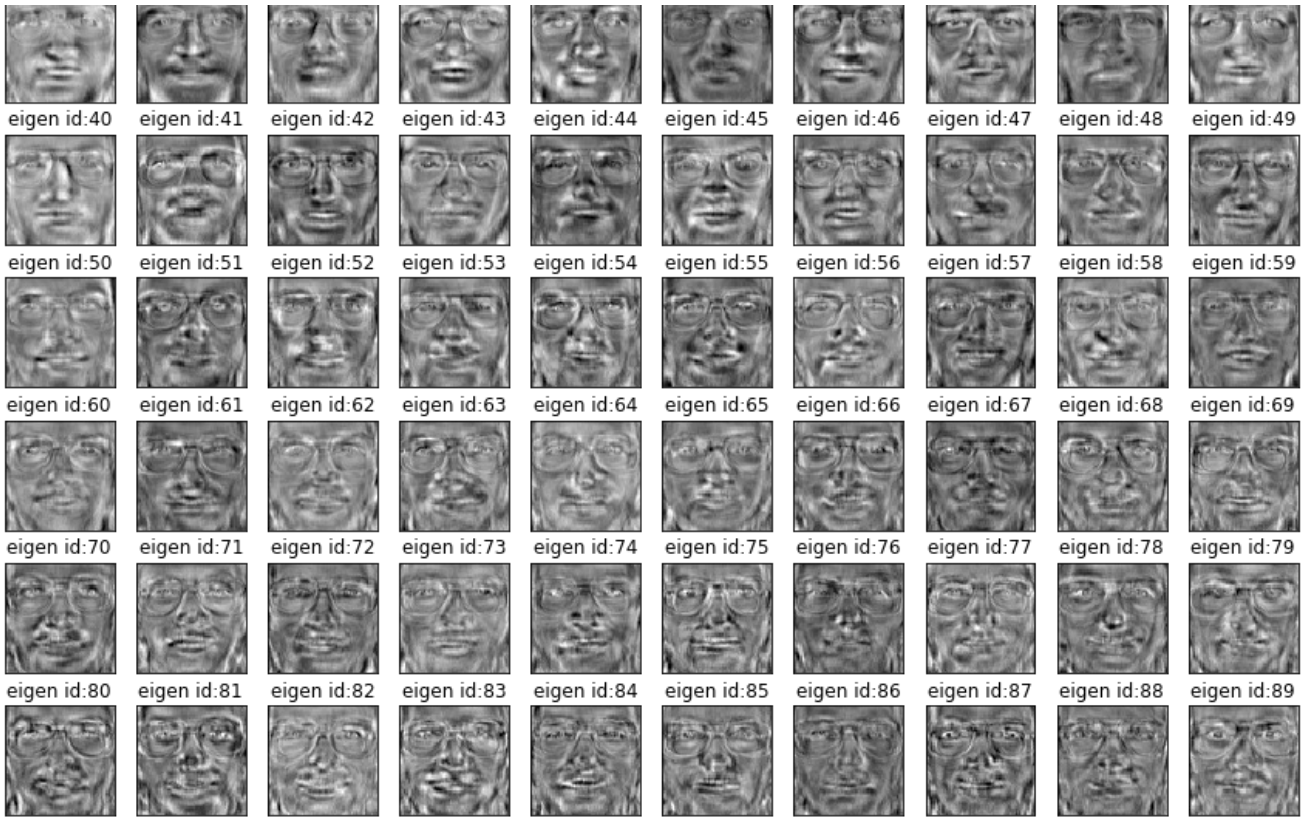
In [21]:

```
number_of_eigenfaces=len(pca.components_)
eigen_faces=pca.components_.reshape((number_of_eigenfaces, data.shape[1], data.shape[2]))

cols=10
rows=int(number_of_eigenfaces/cols)
fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(15,15))
axarr=axarr.flatten()
for i in range(number_of_eigenfaces):
    axarr[i].imshow(eigen_faces[i],cmap="gray")
    axarr[i].set_xticks([])
    axarr[i].set_yticks([])
    axarr[i].set_title("eigen id:{}".format(i))
plt.suptitle("All Eigen Faces".format(10*"=", 10*"="))
```

Out[21]:

```
Text(0.5,0.98,'All Eigen Faces')
```






Classification Results

In [22]:

```
X_train_pca=pca.transform(X_train)
X_test_pca=pca.transform(X_test)
```

In [23]:

```
clf = SVC()
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print("accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
```

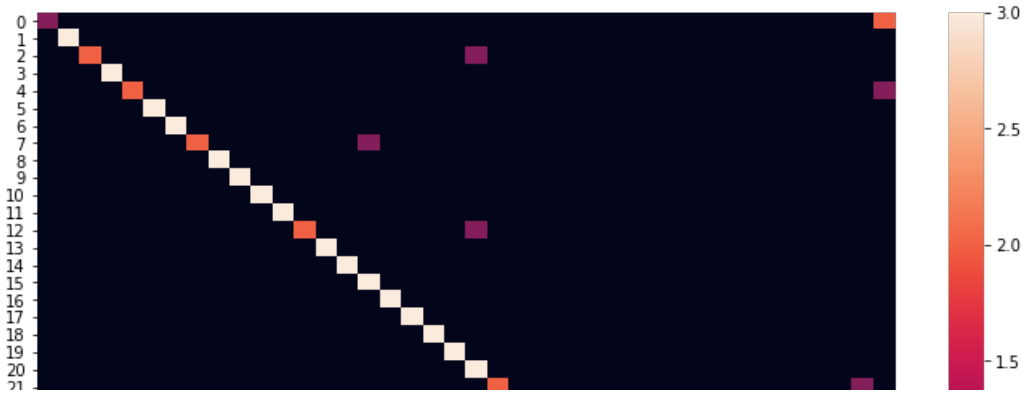
```
accuracy score:0.93
```

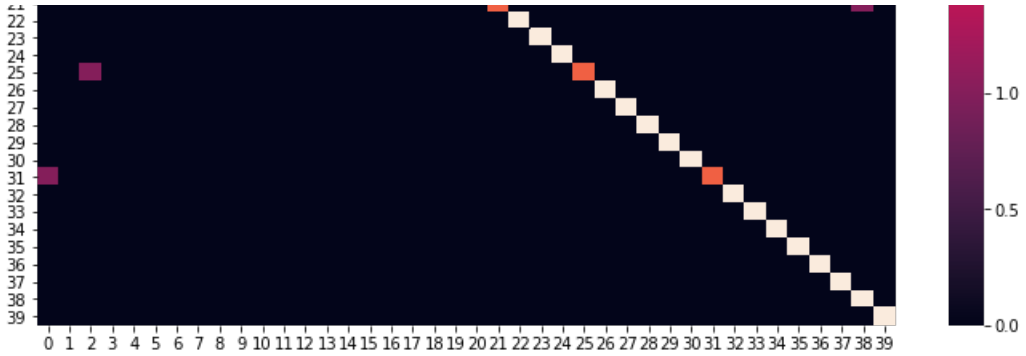
In [24]:

```
import seaborn as sns
plt.figure(1, figsize=(12,8))
sns.heatmap(metrics.confusion_matrix(y_test, y_pred))
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8eb14132e8>
```






In [25]:

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      0.33      0.40         3
           1       1.00      1.00      1.00         3
           2       0.67      0.67      0.67         3
           3       1.00      1.00      1.00         3
           4       1.00      0.67      0.80         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       1.00      1.00      1.00         3
          15       0.75      1.00      0.86         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       0.60      1.00      0.75         3
          21       1.00      0.67      0.80         3
          22       1.00      1.00      1.00         3
          23       1.00      1.00      1.00         3
          24       1.00      1.00      1.00         3
          25       1.00      0.67      0.80         3
          26       1.00      1.00      1.00         3
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00         3
          29       1.00      1.00      1.00         3
          30       1.00      1.00      1.00         3
          31       1.00      0.67      0.80         3
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         3
          34       1.00      1.00      1.00         3
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         3
          38       0.75      1.00      0.86         3
          39       0.50      1.00      0.67         3

   micro avg       0.93      0.93      0.93       120
   macro avg       0.94      0.93      0.92       120
weighted avg       0.94      0.93      0.92       120
```

More Results

We can get accuracy results of state of the art machine learning model.

In [26]:

```
models=[]
models.append(('LDA', LinearDiscriminantAnalysis()))
```

```
models.append(("LDA",LinearDiscriminantAnalysis()))
models.append(("LR",LogisticRegression()))
models.append(("NB",GaussianNB()))
models.append(("KNN",KNeighborsClassifier(n_neighbors=5)))
models.append(("DT",DecisionTreeClassifier()))
models.append(("SVM",SVC()))


for name, model in models:

    clf=model

    clf.fit(X_train_pca, y_train)

    y_pred=clf.predict(X_test_pca)
    print(10*"=","{} Result".format(name).upper(),10*"=")
    print("Accuracy score:{:0.2f}".format(metrics.accuracy_score(y_test, y_pred)))
    print()
```

```
========== LDA RESULT ==========
Accuracy score:0.93

========== LR RESULT ==========
Accuracy score:0.93

========== NB RESULT ==========
Accuracy score:0.88

========== KNN RESULT ==========
Accuracy score:0.71

========== DT RESULT ==========
Accuracy score:0.66

========== SVM RESULT ==========
Accuracy score:0.93
```

According to the above results, Linear Discriminant Analysis and Logistic Regression seems to have the best performances.

Validated Results

In [27]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X)
X_pca=pca.transform(X)
for name, model in models:
    kfold=KFold(n_splits=5, shuffle=True, random_state=0)

    cv_scores=cross_val_score(model, X_pca, target, cv=kfold)
    print("{} mean cross validations score:{:.2f}".format(name, cv_scores.mean()))
```

```
LDA mean cross validations score:0.97
LR mean cross validations score:0.95
NB mean cross validations score:0.78
KNN mean cross validations score:0.68
DT mean cross validations score:0.48
SVM mean cross validations score:0.87
```

According to the cross validation scores Linear Discriminant Analysis and Logistic Regression still have best performance

In [28]:

```
lr=LinearDiscriminantAnalysis()
lr.fit(X_train_pca, y_train)
y_pred=lr.predict(X_test_pca)
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
```

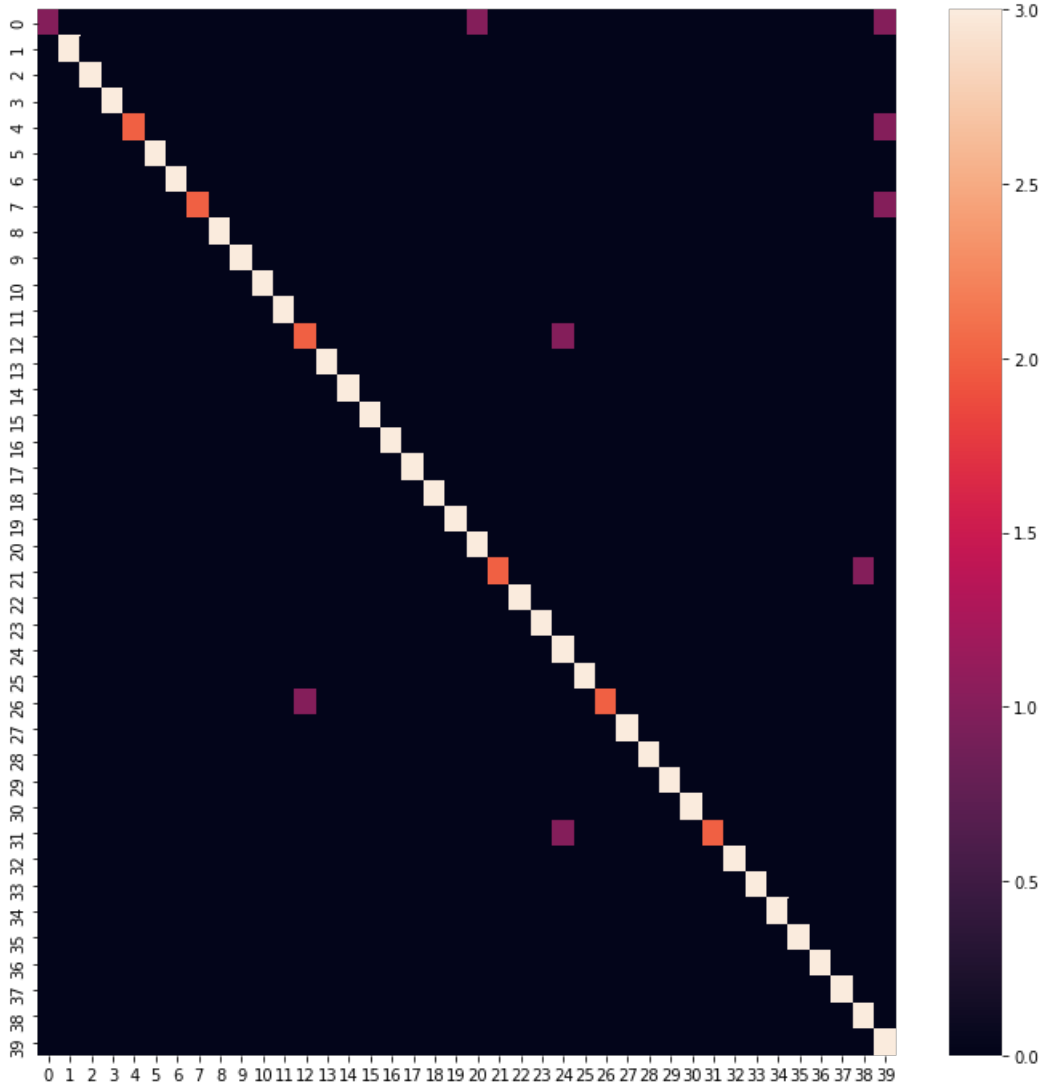
Accuracy score:0.93

In [29]:

```
cm=metrics.confusion_matrix(y_test, y_pred)

plt.subplots(1, figsize=(12,12))
sns.heatmap(cm)
```

Out[29]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f8eb018aac8>




In [30]:

```
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

Classification Results:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.33 | 0.50 | 3 |
| 1 | 1.00 | 1.00 | 1.00 | 3 |
| 2 | 1.00 | 1.00 | 1.00 | 3 |
| 3 | 1.00 | 1.00 | 1.00 | 3 |
| 4 | 1.00 | 0.67 | 0.80 | 3 |
| 5 | 1.00 | 1.00 | 1.00 | 3 |
| 6 | 1.00 | 1.00 | 1.00 | 3 |
| 7 | 1.00 | 0.67 | 0.80 | 3 |
| 8 | 1.00 | 1.00 | 1.00 | 3 |
| 9 | 1.00 | 1.00 | 1.00 | 3 |
| 10 | 1.00 | 1.00 | 1.00 | 3 |
| 11 | 1.00 | 1.00 | 1.00 | 3 |
| 12 | 0.67 | 0.67 | 0.67 | 3 |

| | | | | |
|---|---|---|---|---|
| 12 | 0.67 | 0.67 | 0.67 | 3 |
| 13 | 1.00 | 1.00 | 1.00 | 3 |
| 14 | 1.00 | 1.00 | 1.00 | 3 |
| 15 | 1.00 | 1.00 | 1.00 | 3 |
| 16 | 1.00 | 1.00 | 1.00 | 3 |
| 17 | 1.00 | 1.00 | 1.00 | 3 |
| 18 | 1.00 | 1.00 | 1.00 | 3 |
| 19 | 1.00 | 1.00 | 1.00 | 3 |
| 20 | 0.75 | 1.00 | 0.86 | 3 |
| 21 | 1.00 | 0.67 | 0.80 | 3 |
| 22 | 1.00 | 1.00 | 1.00 | 3 |
| 23 | 1.00 | 1.00 | 1.00 | 3 |
| 24 | 0.60 | 1.00 | 0.75 | 3 |
| 25 | 1.00 | 1.00 | 1.00 | 3 |
| 26 | 1.00 | 0.67 | 0.80 | 3 |
| 27 | 1.00 | 1.00 | 1.00 | 3 |
| 28 | 1.00 | 1.00 | 1.00 | 3 |
| 29 | 1.00 | 1.00 | 1.00 | 3 |
| 30 | 1.00 | 1.00 | 1.00 | 3 |
| 31 | 1.00 | 0.67 | 0.80 | 3 |
| 32 | 1.00 | 1.00 | 1.00 | 3 |
| 33 | 1.00 | 1.00 | 1.00 | 3 |
| 34 | 1.00 | 1.00 | 1.00 | 3 |
| 35 | 1.00 | 1.00 | 1.00 | 3 |
| 36 | 1.00 | 1.00 | 1.00 | 3 |
| 37 | 1.00 | 1.00 | 1.00 | 3 |
| 38 | 0.75 | 1.00 | 0.86 | 3 |
| 39 | 0.50 | 1.00 | 0.67 | 3 |
| micro avg | 0.93 | 0.93 | 0.93 | 120 |
| macro avg | 0.96 | 0.93 | 0.93 | 120 |
| weighted avg | 0.96 | 0.93 | 0.93 | 120 |

More Validated Results: Leave One Out vross-validation

The Olivetti dataset contains 10 face images for each subject. This is a small number for training and testing machine learning models. There is a recommended cross-validation method for better assessment of machine learning models where there are few examples of classes: Leave One Out cross validation. In the LOO approach, only one of the samples of a class is used for testing. Others are used for training. This procedure is repeated until each sample is used for testing.

In [31]:

```
from sklearn.model_selection import LeaveOneOut
loo_cv=LeaveOneOut()
clf=LogisticRegression()
cv_scores=cross_val_score(clf,
                          X_pca,
                          target,
                          cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy score:{:.2f}".format(clf.__class__.__name__
,
                                                                            cv_scores.mean()))
```

```
LogisticRegression Leave One Out cross-validation mean accuracy score:0.96
```

In [32]:

```
from sklearn.model_selection import LeaveOneOut
loo_cv=LeaveOneOut()
clf=LinearDiscriminantAnalysis()
cv_scores=cross_val_score(clf,
                          X_pca,
                          target,
                          cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy score:{:.2f}".format(clf.__class__.__name__
,
                                                                            cv_scores.mean()))
```

```
LinearDiscriminantAnalysis Leave One Out cross-validation mean accuracy score:0.98
```

Hyperparameter Tunning: GridSearchCV

Hyperparameter Tunning: GridSearcCV

In [33]:

```
from sklearn.model_selection import GridSearchCV
```

In [34]:

```
from sklearn.model_selection import LeaveOneOut

#This process takes long time. You can use parameter:{'C': 1.0, 'penalty': 'l2'}
#grid search cross validation score:0.93
"""
params={'penalty':['l1', 'l2'],
                'C':np.logspace(0, 4, 10)
                }
clf=LogisticRegression()
#kfold=KFold(n_splits=3, shuffle=True, random_state=0)
loo_cv=LeaveOneOut()
gridSearchCV=GridSearchCV(clf, params, cv=loo_cv)
gridSearchCV.fit(X_train_pca, y_train)
print("Grid search fitted..")
print(gridSearchCV.best_params_)
print(gridSearchCV.best_score_)
print("grid search cross validation score:{:.2f}".format(gridSearchCV.score(X_test_pca, y_test)))
"""
```

Out[34]:

```
'\nparams={\'penalty\':[\'l1\', \'l2\'],\n                \'C\':np.logspace(0, 4, 10)\n
}\nclf=LogisticRegression()\n#kfold=KFold(n_splits=3, shuffle=True,
random_state=0)\nloo_cv=LeaveOneOut()\ngridSearchCV=GridSearchCV(clf, params,
cv=loo_cv)\ngridSearchCV.fit(X_train_pca, y_train)\nprint("Grid search
fitted..")\nprint(gridSearchCV.best_params_)\nprint(gridSearchCV.best_score_)\nprint("grid search
cross validation score:{:.2f}".format(gridSearchCV.score(X_test_pca, y_test)))\n'
```

In [35]:

```
lr=LogisticRegression(C=1.0, penalty="l2")
lr.fit(X_train_pca, y_train)
print("lr score:{:.2f}".format(lr.score(X_test_pca, y_test)))
```

```
lr score:0.93
```

Precision-Recall-ROC Curves

Precision-recall curves are for binary classification. In Olivetti dataset, there are 40 distinct classes. Dont worry, sklearn allows us to illustrate the precision-recall in multi-label settings

In [36]:

```
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier

Target=label_binarize(target, classes=range(40))
print(Target.shape)
print(Target[0])

n_classes=Target.shape[1]
```

```
(400, 40)
[1 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 0
 0 0 0 0]
```

In [37]:

```
X_train_multiclass, X_test_multiclass, y_train_multiclass, y_test_multiclass=train_test_split(X,
                                                                                              Targe
                                                                                             test_s

=0.3,
                                                                                             strati
```

```
Target,
                                                                    random_state=(
```

In [38]:

```
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train_multiclass)

X_train_multiclass_pca=pca.transform(X_train_multiclass)
X_test_multiclass_pca=pca.transform(X_test_multiclass)
```

In [39]:

```
oneRestClassifier=OneVsRestClassifier(lr)

oneRestClassifier.fit(X_train_multiclass_pca, y_train_multiclass)
y_score=oneRestClassifier.decision_function(X_test_multiclass_pca)
```

In [40]:

```
# For each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = metrics.precision_recall_curve(y_test_multiclass[:, i],
                                                        y_score[:, i])
    average_precision[i] = metrics.average_precision_score(y_test_multiclass[:, i], y_score[:, i])

# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = metrics.precision_recall_curve(y_test_multiclass.ravel(),
    y_score.ravel())
average_precision["micro"] = metrics.average_precision_score(y_test_multiclass, y_score,
                                                     average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

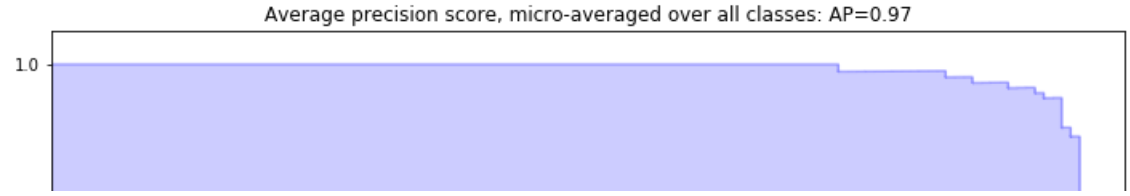
```
Average precision score, micro-averaged over all classes: 0.97
```

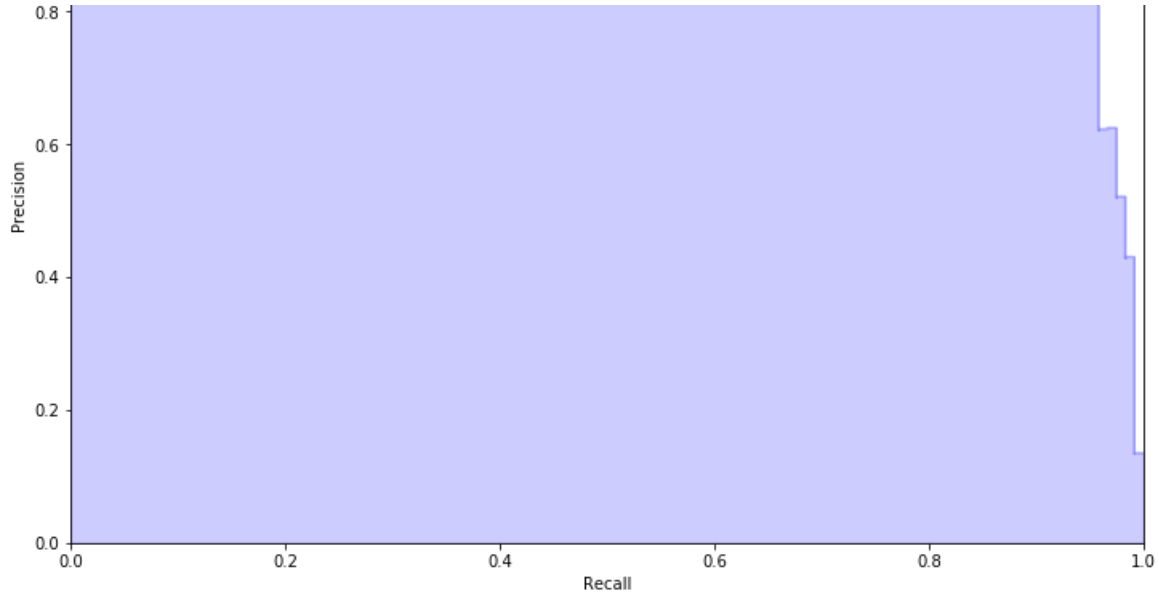
In [41]:

```
from sklearn.utils.fixes import signature

step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})
plt.figure(1, figsize=(12,8))
plt.step(recall['micro'], precision['micro'], color='b', alpha=0.2,
         where='post')
plt.fill_between(recall["micro"], precision["micro"], alpha=0.2, color='b',
                 **step_kwargs)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title(
    'Average precision score, micro-averaged over all classes: AP={0:0.2f}'
    .format(average_precision["micro"]))
```

Out[41]:

```
Text(0.5,1,'Average precision score, micro-averaged over all classes: AP=0.97')
```






Linear Discriminant Analysis İle Boyut Azaltma

In [42]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [43]:

```
lda = LinearDiscriminantAnalysis(n_components=n_components)
X_train_lda = lda.fit(X_train, y_train).transform(X_train)
X_test_lda=lda.transform(X_test)
```

In [44]:

```
lr=LogisticRegression(C=1.0, penalty="l2")
lr.fit(X_train_lda,y_train)
y_pred=lr.predict(X_test_lda)
```

In [45]:

```
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

```
Accuracy score:0.94
Classification Results:
              precision    recall  f1-score   support

           0       0.50      0.33      0.40         3
           1       1.00      1.00      1.00         3
           2       0.60      1.00      0.75         3
           3       1.00      1.00      1.00         3
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       0.75      1.00      0.86         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       1.00      1.00      1.00         3
          15       1.00      1.00      1.00         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       1.00      1.00      1.00         3
          21       1.00      0.67      0.80         3
```

```
           22       1.00      1.00      1.00         3
           23       0.75      1.00      0.86         3
           24       1.00      1.00      1.00         3
           25       1.00      0.67      0.80         3
           26       1.00      0.67      0.80         3
           27       1.00      1.00      1.00         3
           28       1.00      1.00      1.00         3
           29       1.00      1.00      1.00         3
           30       1.00      1.00      1.00         3
           31       1.00      1.00      1.00         3
           32       1.00      1.00      1.00         3
           33       1.00      1.00      1.00         3
           34       1.00      1.00      1.00         3
           35       1.00      1.00      1.00         3
           36       1.00      1.00      1.00         3
           37       1.00      1.00      1.00         3
           38       0.75      1.00      0.86         3
           39       0.75      1.00      0.86         3

   micro avg       0.94      0.94      0.94       120
   macro avg       0.95      0.94      0.94       120
weighted avg       0.95      0.94      0.94       120
```

Machine Learning Automated Workflow: Pipeline

Application of machine learning on data sets has a standard workflow. Sklearn offers the Pipeline object to automate this workflow. Pipeline allows standard work flows for performing machine learning operations such as scaling, feature extraction and modeling. The Pipeline guarantees the same operation in the entire data set, ensuring that the training and test data are consistent.

In [46]:

```
from sklearn.pipeline import Pipeline
```

In [47]:

```
work_flows_std = list()
work_flows_std.append(('lda', LinearDiscriminantAnalysis(n_components=n_components)))
work_flows_std.append(('logReg', LogisticRegression(C=1.0, penalty="l2")))
model_std = Pipeline(work_flows_std)
model_std.fit(X_train, y_train)
y_pred=model_std.predict(X_test)
```

In [48]:

```
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

```
Accuracy score:0.94
Classification Results:
              precision    recall  f1-score   support

           0       0.50      0.33      0.40         3
           1       1.00      1.00      1.00         3
           2       0.60      1.00      0.75         3
           3       1.00      1.00      1.00         3
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       0.75      1.00      0.86         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       1.00      1.00      1.00         3
          15       1.00      1.00      1.00         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
```

```
           20       1.00      1.00      1.00         3
           21       1.00      0.67      0.80         3
           22       1.00      1.00      1.00         3
           23       0.75      1.00      0.86         3
           24       1.00      1.00      1.00         3
           25       1.00      0.67      0.80         3
           26       1.00      0.67      0.80         3
           27       1.00      1.00      1.00         3
           28       1.00      1.00      1.00         3
           29       1.00      1.00      1.00         3
           30       1.00      1.00      1.00         3
           31       1.00      1.00      1.00         3
           32       1.00      1.00      1.00         3
           33       1.00      1.00      1.00         3
           34       1.00      1.00      1.00         3
           35       1.00      1.00      1.00         3
           36       1.00      1.00      1.00         3
           37       1.00      1.00      1.00         3
           38       0.75      1.00      0.86         3
           39       0.75      1.00      0.86         3

   micro avg       0.94      0.94      0.94       120
   macro avg       0.95      0.94      0.94       120
weighted avg       0.95      0.94      0.94       120
```

So, I have completed the project and now i will conclude with the gist of the entire project. Data preprocessed and reshaped using pandas & sklearn and principal components of images were obtained by PCA. Dataset trained with Logistic Regression, Decision Tree, KNN, SVM, and Linear Discriminant Analysis.LDA and LR show the best perfomance. we get 93% validation accuracy with Linear Discriminant Analysis and Logistic Regression. Cross validation scores are 97% & 95% respectively.