In [1]:
```
# Import `datasets` from `sklearn`
from sklearn import datasets

# Load in the `digits` data
digits = datasets.load_digits()

# Print the `digits` data
print(digits)
```

```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]]), 'target': array([0, 1, 2,
..., 8, 9, 8]), 'frame': None, 'feature_names': ['pixel_0_0', 'pixel_0_
1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5', 'pixel_0_6', 'pix
el_0_7', 'pixel_1_0', 'pixel_1_1', 'pixel_1_2', 'pixel_1_3', 'pixel_1_4',
'pixel_1_5', 'pixel_1_6', 'pixel_1_7', 'pixel_2_0', 'pixel_2_1', 'pixel_2
_2', 'pixel_2_3', 'pixel_2_4', 'pixel_2_5', 'pixel_2_6', 'pixel_2_7', 'pi
xel_3_0', 'pixel_3_1', 'pixel_3_2', 'pixel_3_3', 'pixel_3_4', 'pixel_3_
5', 'pixel_3_6', 'pixel_3_7', 'pixel_4_0', 'pixel_4_1', 'pixel_4_2', 'pix
el_4_3', 'pixel_4_4', 'pixel_4_5', 'pixel_4_6', 'pixel_4_7', 'pixel_5_0',
'pixel_5_1', 'pixel_5_2', 'pixel_5_3', 'pixel_5_4', 'pixel_5_5', 'pixel_5
_6', 'pixel_5_7', 'pixel_6_0', 'pixel_6_1', 'pixel_6_2', 'pixel_6_3', 'pi
xel_6_4', 'pixel_6_5', 'pixel_6_6', 'pixel_6_7', 'pixel_7_0', 'pixel_7_
1', 'pixel_7_2', 'pixel_7_3', 'pixel_7_4', 'pixel_7_5', 'pixel_7_6', 'pix
el_7_7'], 'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), 'image
s': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ..., 15.,  5.,  0.],
        [ 0.,  3., 15., ..., 11.,  8.,  0.],
        ...,
        [ 0.,  4., 11., ..., 12.,  7.,  0.],
        [ 0.,  2., 14., ..., 12.,  0.,  0.],
        [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

       [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
        [ 0.,  0.,  0., ...,  9.,  0.,  0.],
        [ 0.,  0.,  3., ...,  6.,  0.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

       [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
        [ 0.,  0.,  3., ..., 14.,  0.,  0.],
        [ 0.,  0.,  8., ..., 16.,  0.,  0.],
        ...,
        [ 0.,  9., 16., ...,  0.,  0.,  0.],
        [ 0.,  3., 13., ..., 11.,  5.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

       ...,

       [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
```

```
        [ 0.,  0., 13., ...,  2.,  1.,  0.],
        [ 0.,  0., 16., ..., 16.,  5.,  0.],
        ...,
        [ 0.,  0., 16., ..., 15.,  0.,  0.],
        [ 0.,  0., 15., ..., 16.,  0.,  0.],
        [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

       [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
        [ 0.,  0., 14., ..., 15.,  1.,  0.],
        [ 0.,  4., 16., ..., 16.,  7.,  0.],
        ...,
        [ 0.,  0.,  0., ..., 16.,  2.,  0.],
        [ 0.,  0.,  4., ..., 16.,  2.,  0.],
        [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

       [[ 0.,  0., 10., ...,  1.,  0.,  0.],
        [ 0.,  2., 16., ...,  1.,  0.,  0.],
        [ 0.,  0., 15., ..., 15.,  0.,  0.],
        ...,
        [ 0.,  4., 16., ..., 16.,  6.,  0.],
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]), 'DESCR': ".. _digits_data
set:\n\nOptical recognition of handwritten digits dataset\n--------------
----------------------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 1797\n    :Number of Attributes: 64\n    :Attribute
Information: 8x8 image of integer pixels in the range 0..16.\n    :Missin
g Attribute Values: None\n    :Creator: E. Alpaydin (alpaydin '@' boun.ed
u.tr)\n    :Date: July; 1998\n\nThis is a copy of the test set of the UCI
ML hand-written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/
Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images
of hand-written digits: 10 classes where\neach class refers to a digit.\n
\nPreprocessing programs made available by NIST were used to extract\nnor
malized bitmaps of handwritten digits from a preprinted form. From a\ntot
al of 43 people, 30 contributed to the training set and different 13\nto
the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x
4 and the number of on pixels are counted in each block. This generates\n
an input matrix of 8x8 where each element is an integer in the range\n0..
16. This reduces dimensionality and gives invariance to small\ndistortion
s.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Bl
ue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet,
and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5
469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of C
ombining Multiple Classifiers and Their\n    Applications to Handwritten
Digit Recognition, MSc Thesis, Institute of\n    Graduate Studies in Scie
nce and Engineering, Bogazici University.\n  - E. Alpaydin, C. Kaynak (19
98) Cascading Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthurai N.
Suganthan and Xi Yao and A. Kai Qin.\n    Linear dimensionalityreduction
using relevance weighted LDA. School of\n    Electrical and Electronic En
gineering Nanyang Technological University.\n    2005.\n  - Claudio Genti
le. A New Approximate Maximal Margin Classification\n    Algorithm. NIPS.
2000.\n"}
```

In [2]:
```
# Get the keys of the `digits` data
print(digits.keys())

# Print out the data
print(digits.data)

# Print out the target values
print(digits.target)

# Print out the description of the `digits` data
print(digits.DESCR)
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'ima
ges', 'DESCR'])
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
[0 1 2 ... 8 9 8]
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+
Digits (https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Hand
written+Digits)

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
```

```
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

In [4]:

```
import numpy as np
# Isolate the `digits` data
digits_data = digits.data

# Inspect the shape
print(digits_data.shape)

# Isolate the target values with `target`
digits_target = digits.target

# Inspect the shape
print(digits_target.shape)

# Print the number of unique labels
number_digits = len(np.unique(digits.target))
```

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

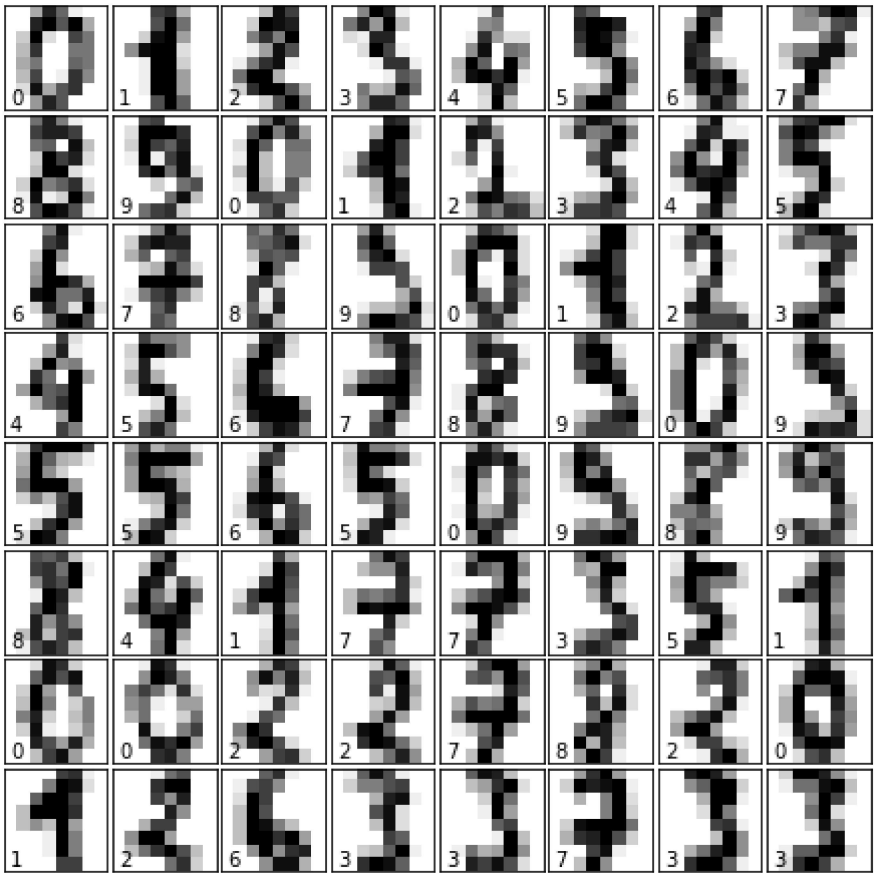
In [5]:

```
print(np.all(digits.images.reshape((1797,64)) == digits.data))
```

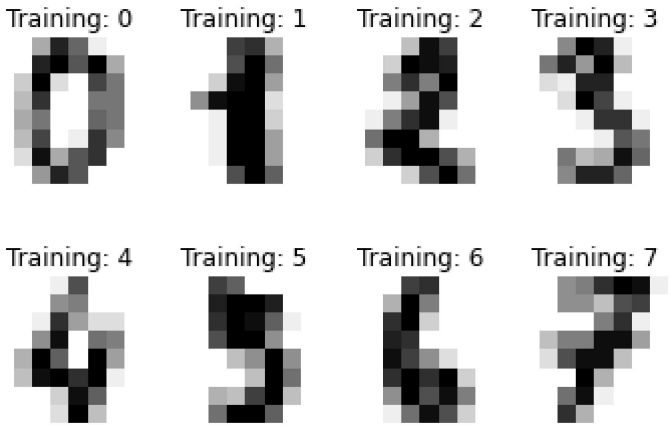
```
True
```

In [6]:

```
# Import matplotlib
import matplotlib.pyplot as plt

# Figure size (width, height) in inches
fig = plt.figure(figsize=(6, 6))

# Adjust the subplots
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.0

# For each of the 64 images
for i in range(64):
    # Initialize the subplots: add a subplot in the grid of 8 by 8, at the i+
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
    # Display an image at the i-th position
    ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='nearest')
    # label the image with the target value
    ax.text(0, 7, str(digits.target[i]))

# Show the plot
plt.show()
```



In [7]:
```
# Import matplotlib
import matplotlib.pyplot as plt

# Join the images and target labels in a list
images_and_labels = list(zip(digits.images, digits.target))

# for every element in the list
for index, (image, label) in enumerate(images_and_labels[:8]):
    # initialize a subplot of 2X4 at the i+1-th position
    plt.subplot(2, 4, index + 1)
    # Don't plot any axes
    plt.axis('off')
    # Display images in all subplots
    plt.imshow(image, cmap=plt.cm.gray_r,interpolation='nearest')
    # Add a title to each subplot
    plt.title('Training: ' + str(label))

# Show the plot
plt.show()
```



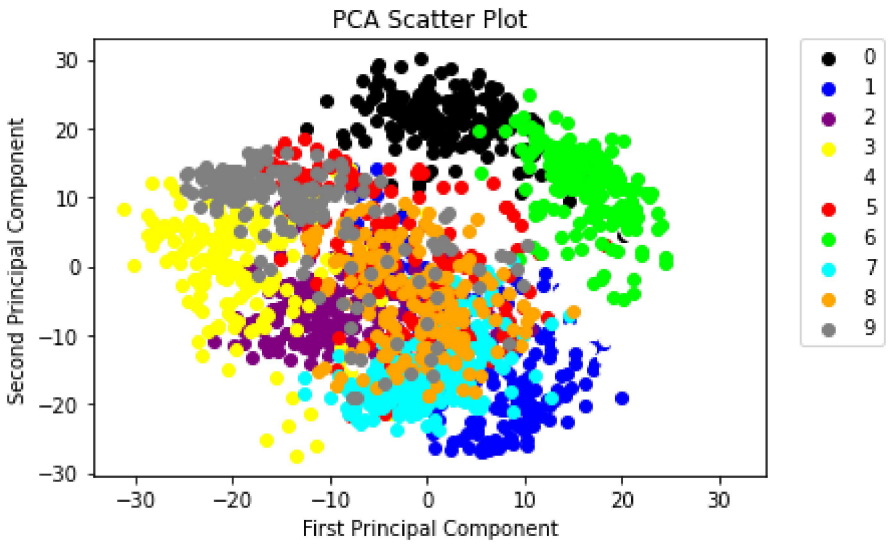
In [14]:
```
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.decomposition import PCA
```

In [15]:

```
# Create a Randomized PCA model that takes two components
randomized_pca = RandomizedPCA(n_components=2)

# Fit and transform the data to the model
reduced_data_rpca = randomized_pca.fit_transform(digits.data)

# Create a regular PCA model
pca = PCA(n_components=2)

# Fit and transform the data to the model
reduced_data_pca = pca.fit_transform(digits.data)

# Inspect the shape
reduced_data_pca.shape

# Print out the data
print(reduced_data_rpca)
print(reduced_data_pca)
```

```
[[ -1.2594652   21.27488808]
 [  7.95760898 -20.76870486]
 [  6.99192317  -9.95598785]
 ...
 [ 10.8012847   -6.96025543]
 [ -4.87210264  12.4239528 ]
 [ -0.34438698   6.36555293]]
[[ -1.25946699  21.27488189]
 [  7.95761462 -20.76868999]
 [  6.99192223  -9.95598793]
 ...
 [ 10.80128106  -6.96026198]
 [ -4.87209739  12.42396467]
 [ -0.3443948    6.36553332]]
```

In [16]:
```
colors = ['black', 'blue', 'purple', 'yellow', 'white', 'red', 'lime', 'cyan'
for i in range(len(colors)):
    x = reduced_data_rpca[:, 0][digits.target == i]
    y = reduced_data_rpca[:, 1][digits.target == i]
    plt.scatter(x, y, c=colors[i])
plt.legend(digits.target_names, bbox_to_anchor=(1.05, 1), loc=2, borderaxespa
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title("PCA Scatter Plot")
plt.show()
```



In [17]:
```
# Import
from sklearn.preprocessing import scale

# Apply `scale()` to the `digits` data
data = scale(digits.data)
```

In [19]:
```
# Import `train_test_split`
from sklearn.model_selection import train_test_split

# Split the `digits` data into training and test sets
X_train, X_test, y_train, y_test, images_train, images_test = train_test_spli
```

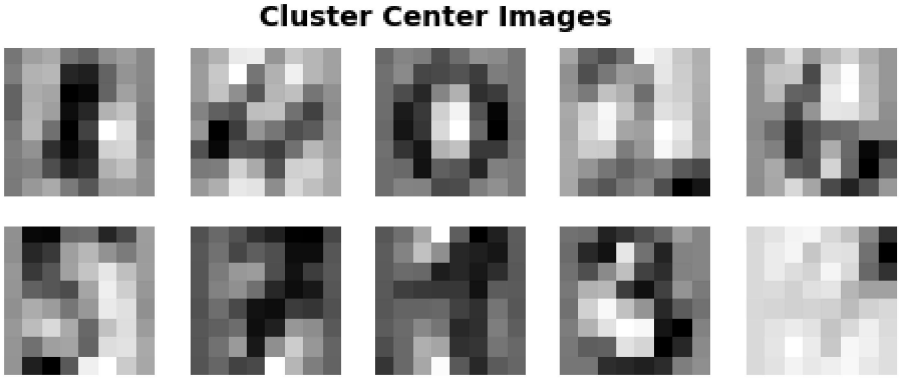
In [20]:
```
# Number of training features
n_samples, n_features = X_train.shape

# Print out `n_samples`
print(n_samples)

# Print out `n_features`
print(n_features)

# Number of Training labels
n_digits = len(np.unique(y_train))

# Inspect `y_train`
print(len(y_train))
```

```
1347
64
1347
```

In [21]:
```
# Import the `cluster` module
from sklearn import cluster

# Create the KMeans model
clf = cluster.KMeans(init='k-means++', n_clusters=10, random_state=42)

# Fit the training data to the model
clf.fit(X_train)
```

Out[21]:
```
KMeans(n_clusters=10, random_state=42)
```

In [22]:
```
# Import matplotlib
import matplotlib.pyplot as plt

# Figure size in inches
fig = plt.figure(figsize=(8, 3))

# Add title
fig.suptitle('Cluster Center Images', fontsize=14, fontweight='bold')

# For all labels (0-9)
for i in range(10):
    # Initialize subplots in a grid of 2X5, at i+1th position
    ax = fig.add_subplot(2, 5, 1 + i)
    # Display images
    ax.imshow(clf.cluster_centers_[i].reshape((8, 8)), cmap=plt.cm.binary)
    # Don't show the axes
    plt.axis('off')

# Show the plot
plt.show()
```



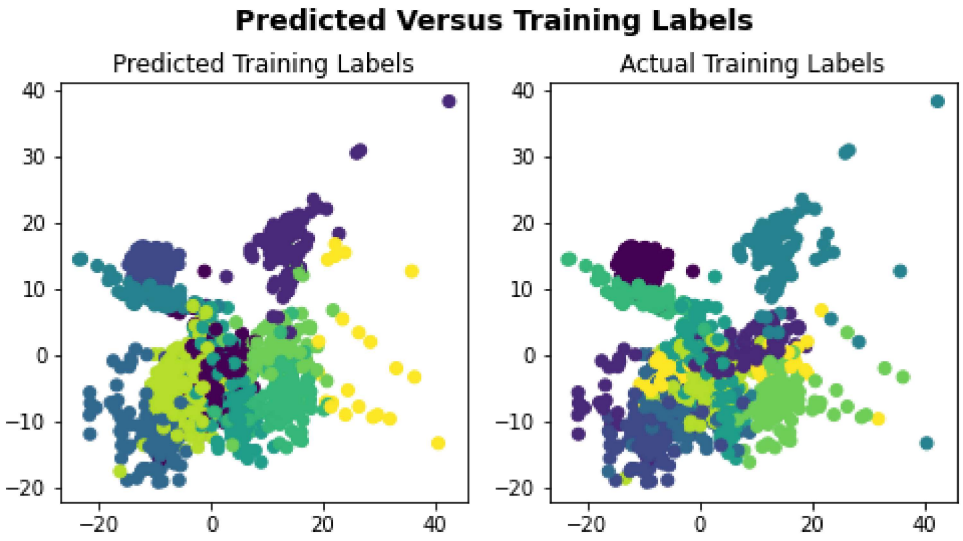
In [23]:
```
# Predict the labels for `X_test`
y_pred=clf.predict(X_test)

# Print out the first 100 instances of `y_pred`
print(y_pred[:100])

# Print out the first 100 instances of `y_test`
print(y_test[:100])

# Study the shape of the cluster centers
clf.cluster_centers_.shape
```

```
[4 8 8 9 3 3 5 8 5 3 0 7 1 2 1 3 8 6 8 8 1 5 8 6 5 4 8 5 4 8 1 8 3 1 1 4 8
 1 6 4 4 8 0 8 4 7 8 2 4 5 5 0 8 5 4 2 8 2 2 7 2 1 5 3 1 5 6 2 6 8 8 8 8 6
 6 2 1 5 8 8 8 2 3 8 8 2 4 1 1 8 0 3 7 8 8 3 8 2 1 1]
[6 9 3 7 2 1 5 2 5 2 1 9 4 0 4 2 3 7 8 8 4 3 9 7 5 6 3 5 6 3 4 9 1 4 4 6 9
 4 7 6 6 9 1 3 6 1 3 0 6 5 5 1 9 5 6 0 9 0 0 1 0 4 5 2 4 5 7 0 7 5 9 5 5 4
 7 0 4 5 5 9 9 0 2 3 8 0 6 4 4 9 1 2 8 3 5 2 9 0 4 4]
```

Out[23]: (10, 64)

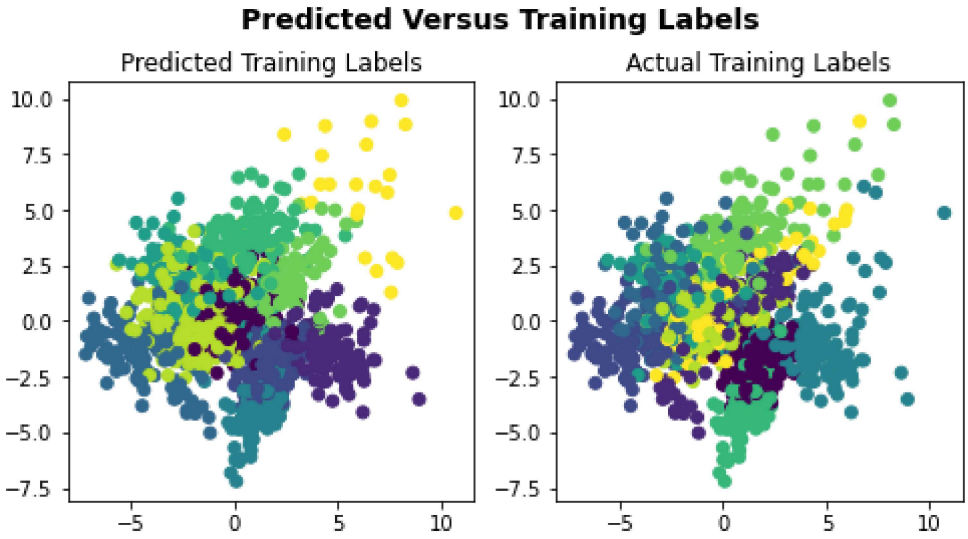
In [24]:

```
# Import `Isomap()`
from sklearn.manifold import Isomap

# Create an isomap and fit the `digits` data to it
X_iso = Isomap(n_neighbors=10).fit_transform(X_train)

# Compute cluster centers and predict cluster index for each sample
clusters = clf.fit_predict(X_train)

# Create a plot with subplots in a grid of 1X2
fig, ax = plt.subplots(1, 2, figsize=(8, 4))

# Adjust layout
fig.suptitle('Predicted Versus Training Labels', fontsize=14, fontweight='bol
fig.subplots_adjust(top=0.85)

# Add scatterplots to the subplots
ax[0].scatter(X_iso[:, 0], X_iso[:, 1], c=clusters)
ax[0].set_title('Predicted Training Labels')
ax[1].scatter(X_iso[:, 0], X_iso[:, 1], c=y_train)
ax[1].set_title('Actual Training Labels')

# Show the plots
plt.show()
```



In [25]:

```
# Import `PCA()`
from sklearn.decomposition import PCA

# Model and fit the `digits` data to the PCA model
X_pca = PCA(n_components=2).fit_transform(X_train)

# Compute cluster centers and predict cluster index for each sample
clusters = clf.fit_predict(X_train)

# Create a plot with subplots in a grid of 1X2
fig, ax = plt.subplots(1, 2, figsize=(8, 4))

# Adjust layout
fig.suptitle('Predicted Versus Training Labels', fontsize=14, fontweight='bol
fig.subplots_adjust(top=0.85)

# Add scatterplots to the subplots
ax[0].scatter(X_pca[:, 0], X_pca[:, 1], c=clusters)
ax[0].set_title('Predicted Training Labels')
ax[1].scatter(X_pca[:, 0], X_pca[:, 1], c=y_train)
ax[1].set_title('Actual Training Labels')

# Show the plots
plt.show()
```



In [26]:
```
# Import `metrics` from `sklearn`
from sklearn import metrics

# Print out the confusion matrix with `confusion_matrix()`
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[ 0  0 43  0  0  0  0  0  0  0]
 [20  0  0  7  0  0  0 10  0  0]
 [ 5  0  0 31  0  0  0  1  1  0]
 [ 1  0  0  1  0  1  4  0 39  0]
 [ 1 50  0  0  0  0  1  2  0  1]
 [ 1  0  0  0  1 41  0  0 16  0]
 [ 0  0  1  0 44  0  0  0  0  0]
 [ 0  0  0  0  0  1 34  1  0  5]
 [21  0  0  0  0  3  1  2 11  0]
 [ 0  0  0  0  0  2  3  3 40  0]]
```

In [27]:
```
from sklearn.metrics import homogeneity_score, completeness_score, v_measure_
print('% 9s' % 'inertia    homo   compl  v-meas     ARI AMI  silhouette')
print('%i   %.3f   %.3f   %.3f   %.3f   %.3f    %.3f'
          %(clf.inertia_,
      homogeneity_score(y_test, y_pred),
      completeness_score(y_test, y_pred),
      v_measure_score(y_test, y_pred),
      adjusted_rand_score(y_test, y_pred),
      adjusted_mutual_info_score(y_test, y_pred),
      silhouette_score(X_test, y_pred, metric='euclidean')))
```

```
inertia    homo   compl  v-meas     ARI AMI  silhouette
54276   0.688   0.733   0.710   0.567   0.697    0.146
```

In [29]:
```
# Import `train_test_split`
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
X_train, X_test, y_train, y_test, images_train, images_test = train_test_spli

# Import the `svm` model
from sklearn import svm

# Create the SVC model
svc_model = svm.SVC(gamma=0.001, C=100., kernel='linear')

# Fit the data to the SVC model
svc_model.fit(X_train, y_train)
```

Out[29]:
```
SVC(C=100.0, gamma=0.001, kernel='linear')
```

In [32]:
```
# Split the `digits` data into two equal sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.targe

# Import GridSearchCV'
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV, train_t



# Set the parameter candidates
parameter_candidates = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
]

# Create a classifier with the parameter candidates
clf = GridSearchCV(estimator=svm.SVC(), param_grid=parameter_candidates, n_jo

# Train the classifier on training data
clf.fit(X_train, y_train)

# Print out the results
print('Best score for training data:', clf.best_score_)
print('Best `C`:',clf.best_estimator_.C)
print('Best kernel:',clf.best_estimator_.kernel)
print('Best `gamma`:',clf.best_estimator_.gamma)
```

```
Best score for training data: 0.9866480446927375
Best `C`: 10
Best kernel: rbf
Best `gamma`: 0.001
```

In [33]:
```
# Apply the classifier to the test data, and view the accuracy score
clf.score(X_test, y_test)

# Train and score a new classifier with the grid search parameters
svm.SVC(C=10, kernel='rbf', gamma=0.001).fit(X_train, y_train).score(X_test,
```

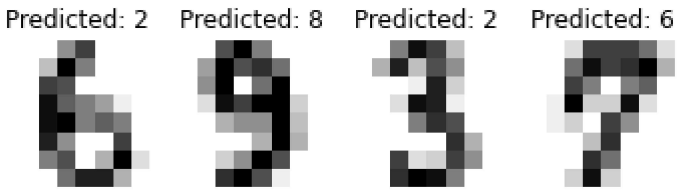
Out[33]:
```
0.9911012235817576
```

In [34]:
```
# Predict the label of `X_test`
print(svc_model.predict(X_test))

# Print `y_test` to check the results
print(y_test)
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 2 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 6 5 2 5 8 4 8 7 0 6 1 5 9 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 8 7 6 7 6 5 6 0 8 8 9 8 6 1 0 4 1 6 3 8 6 7 4 5 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 9 8 5 4 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 1 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 2 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 8 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 3 3 4 3 1 8 9 5 3 6 2 1 6 2
 1 7 5 5 1 9 2 8 9 7 2 1 4 9 3 2 6 2 5 9 6 5 8 2 0 7 8 0 5 8 4 1 8 6 4 3 4
 2 0 4 5 8 3 3 1 8 3 4 5 0 8 5 6 3 0 6 9 1 5 2 2 1 9 8 4 3 3 0 7 8 8 1 1 3
 5 5 8 4 9 7 8 4 4 9 0 1 6 9 3 6 1 7 0 6 2 9 9 9 6 1 5 1 8 9 8 4 1 7 2 8 0
 6 2 1 0 6 1 6 5 2 8 6 2 1 4 6 8 2 2 7 5 9 1 9 5 0 2 5 5 6 8 9 5 7 0 5 2 1
 1 2 8 8 2 1 5 5 2 7 1 4 4 1 3 5 6 7 4 6 7 3 5 7 1 5 5 3 8 5 2 3 5 1 4 6 5
 1 4 1 8 3 3 1 4 8 9 2 9 4 9 4 8 0 3 3 2 0 6 6 8 6 5 9 0 3 2 5 6 9 5 4 3 3
 9 9 2 7 9 9 1 1 5 9 1 0 4 1 7 4 3 8 6 5 5 5 3 5 0 0 5 0 0 3 5 2 5 4 6 5 8
 5 0 9 9 1 5 0 5 0 8 7 7 5 2 7 8 6 5 3 0 2 9 7 7 4 0 0 6 1 1 3 4 1 8 7 5 1
 0 8 0 9 0 9 3 0 0 6 4 5 3 8 0 9 0 3 2 0 4 0 3 3 2 2 0 5 3 2 3 6 5 0 4 2 7
 9 1 5 8 2 1 2 2 6 0 1 1 3 6 2 3 7 3 8 3 8 0 3 5 5 8 5 1 3 1 1 0 1 2 5 0 0
 4 8 0 5 7 9 5 0 2 5 2 7 3 6 8 7 9 5 0 0 7 0 9 3 6 2 9 1 1 9 1 6 8 3 3 2 4
 4 8 9 6 6 6 8 9 2 4 5 0 0 7 7 4 4 7 3 1 1 9 7 0 4 9 2 9 3 5 3 6 3 8 2 0 0
 2 4 9 8 5 2 2 9 5 9 5]
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 2 6 9 1 8 3 5 1
 2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
 4 8 9 7 9 8 2 6 5 2 5 8 4 8 7 0 6 1 5 9 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 8 7 6 7 6 5 6 0 8 8 9 8 6 1 0 4 1 6 3 8 6 7 4 5 6 3 0
 3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 9 8 5 4 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 1 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8 6 5 3 4 4 4 8 8 7 0
 9 6 3 5 2 3 0 8 3 3 1 3 3 0 0 4 6 0 7 7 6 2 0 4 4 2 3 7 8 9 8 6 8 5 6 2 2
 3 1 7 7 8 0 3 3 2 1 5 5 9 1 3 7 0 0 7 0 4 5 9 3 3 4 3 1 8 9 8 3 6 2 1 6 2
 1 7 5 5 1 9 2 8 9 7 2 1 4 9 3 2 6 2 5 9 6 5 8 2 0 7 8 0 5 8 4 1 8 6 4 3 4
 2 0 4 5 8 3 9 1 8 3 4 5 0 8 5 6 3 0 6 9 1 5 2 2 1 9 8 4 3 3 0 7 8 8 1 1 3
 5 5 8 4 9 7 8 4 4 9 0 1 6 9 3 6 1 7 0 6 2 9 9 3 6 1 5 1 8 9 8 4 1 7 2 8 0
 6 2 1 0 6 1 6 5 2 8 6 2 1 4 6 8 2 2 7 5 9 1 9 5 0 2 5 5 6 8 9 5 7 0 5 2 1
 1 2 8 8 2 1 5 5 2 7 1 4 4 1 3 5 6 7 4 6 7 3 5 7 1 5 5 3 8 5 2 3 5 4 4 6 5
 1 4 1 8 3 3 1 4 8 9 2 9 4 9 4 8 0 3 3 2 0 6 6 8 6 5 9 0 3 2 5 6 9 5 4 3 3
 9 9 2 7 9 9 1 1 5 9 1 0 4 1 7 4 3 8 6 5 5 5 3 5 0 0 5 0 0 3 5 2 3 4 6 5 8
 5 0 9 9 1 5 0 5 0 8 7 7 5 2 7 8 6 5 3 0 2 9 7 7 4 0 0 6 1 1 3 4 1 8 7 5 1
 0 8 0 9 0 9 3 0 0 6 4 5 3 8 0 9 0 3 2 0 4 0 3 3 2 2 0 5 3 2 3 6 5 0 4 2 7
 9 1 5 8 2 1 2 2 6 0 1 1 3 6 2 3 7 3 8 3 8 0 3 5 5 8 5 1 3 1 1 0 1 2 5 0 0
 4 8 0 5 7 9 5 0 2 5 2 7 3 6 8 7 9 5 0 0 7 0 9 3 6 2 9 1 1 9 1 6 8 3 3 2 4
 4 8 9 6 6 6 8 9 2 4 5 0 0 7 7 4 4 7 3 1 1 9 7 0 4 9 2 9 3 5 3 6 3 8 2 0 0
 2 4 9 8 5 2 2 9 5 9 5]
```

In [35]:

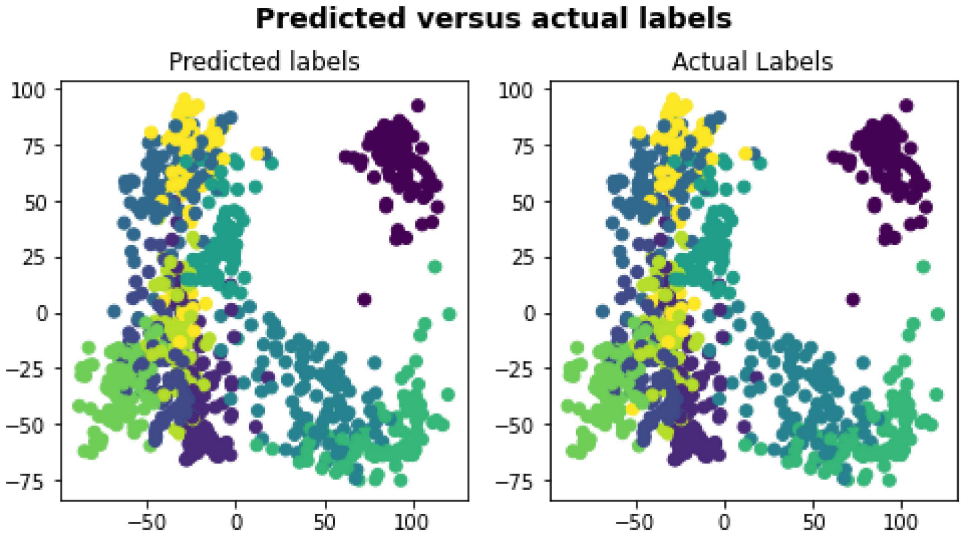
```
# Import matplotlib
import matplotlib.pyplot as plt

# Assign the predicted values to `predicted`
predicted = svc_model.predict(X_test)

# Zip together the `images_test` and `predicted` values in `images_and_predic
images_and_predictions = list(zip(images_test, predicted))

# For the first 4 elements in `images_and_predictions`
for index, (image, prediction) in enumerate(images_and_predictions[:4]):
    # Initialize subplots in a grid of 1 by 4 at positions i+1
    plt.subplot(1, 4, index + 1)
    # Don't show axes
    plt.axis('off')
    # Display images in all subplots in the grid
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    # Add a title to the plot
    plt.title('Predicted: ' + str(prediction))

# Show the plot
plt.show()
```



In [36]:
```
# Import `metrics`
from sklearn import metrics

# Print the classification report of `y_test` and `predicted`
print(metrics.classification_report(y_test, predicted))

# Print the confusion matrix
print(metrics.confusion_matrix(y_test, predicted))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        89
           1       0.99      1.00      0.99        90
           2       0.99      1.00      0.99        92
           3       0.99      0.97      0.98        93
           4       1.00      0.99      0.99        76
           5       0.98      1.00      0.99       108
           6       1.00      1.00      1.00        89
           7       1.00      1.00      1.00        78
           8       1.00      0.99      0.99        92
           9       0.99      0.99      0.99        92

    accuracy                           0.99       899
   macro avg       0.99      0.99      0.99       899
weighted avg       0.99      0.99      0.99       899

[[ 89   0   0   0   0   0   0   0   0   0]
 [  0  90   0   0   0   0   0   0   0   0]
 [  0   0  92   0   0   0   0   0   0   0]
 [  0   0   1  90   0   1   0   0   0   1]
 [  0   1   0   0  75   0   0   0   0   0]
 [  0   0   0   0   0 108   0   0   0   0]
 [  0   0   0   0   0   0  89   0   0   0]
 [  0   0   0   0   0   0   0  78   0   0]
 [  0   0   0   0   0   1   0   0  91   0]
 [  0   0   0   1   0   0   0   0   0  91]]
```

In [37]:
```
# Import `Isomap()`
from sklearn.manifold import Isomap

# Create an isomap and fit the `digits` data to it
X_iso = Isomap(n_neighbors=10).fit_transform(X_train)

# Compute cluster centers and predict cluster index for each sample
predicted = svc_model.predict(X_train)

# Create a plot with subplots in a grid of 1X2
fig, ax = plt.subplots(1, 2, figsize=(8, 4))

# Adjust the layout
fig.subplots_adjust(top=0.85)

# Add scatterplots to the subplots
ax[0].scatter(X_iso[:, 0], X_iso[:, 1], c=predicted)
ax[0].set_title('Predicted labels')
ax[1].scatter(X_iso[:, 0], X_iso[:, 1], c=y_train)
ax[1].set_title('Actual Labels')


# Add title
fig.suptitle('Predicted versus actual labels', fontsize=14, fontweight='bold'

# Show the plot
plt.show()
```



In [ ]:
```
You'll see that this visualization confirms your classification report, which
```