In [1]:

```
#do not change
import sys
import os
from time import time
%matplotlib inline
from urllib.request import urlopen
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np
#from scipy.misc import imsave
import math
from tqdm import tqdm

def load_mnist():
    images_url = 'https://github.com/guptashvm/Data/blob/master/data/train-image
s-idx3-ubyte?raw=true'
    with urlopen(images_url) as urlopened:
      fd = urlopened.read()
      loaded = np.frombuffer(fd,dtype=np.uint8)
      trX = loaded[16:].reshape((60000,28*28)).astype(float)

    labels_url = 'https://github.com/guptashvm/Data/blob/master/data/train-label
s-idx1-ubyte?raw=true'
    with urlopen(labels_url) as urlopened:
      fd = urlopened.read()
      loaded = np.frombuffer(fd,dtype=np.uint8)
      trY = loaded[8:].reshape((60000))

    trY = np.asarray(trY)

    X = trX / 255.
    y = trY

    subset  = [i for i, t in enumerate(y) if t in [1, 0, 2, 3]]
    X, y = X.astype('float32')[subset], y[subset]
    return X[:1000], y[:1000]
```

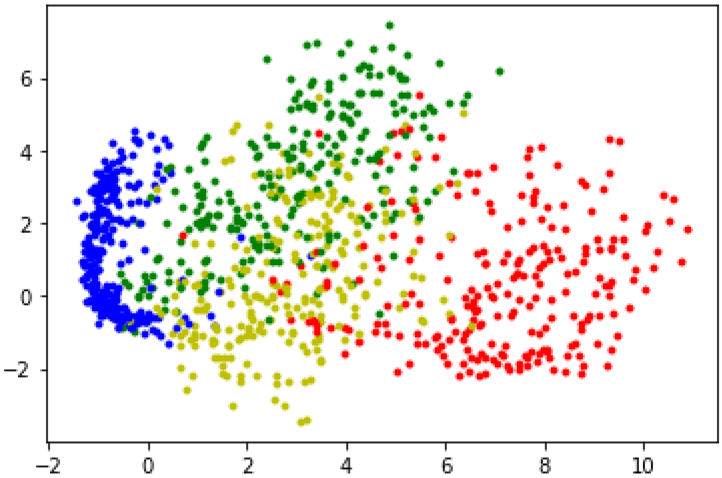
Run the following code to load the data we need. Here, **X** is the array of images, **y** is the array of labels, and **X2d** is the array of projections of the images into two-dimensional space using PCA. The two-dimensional scatterplot of the top two principal components of the images is displayed, where the color of each point represents its label.

In [2]:

```
X, y = load_mnist()
pca = PCA(n_components=2)
pca.fit(X)
X2d = X.dot(pca.components_.T)

def plot_with_colors(Xs, ys):
  for i, _ in enumerate(ys):
    if ys[i] == 0:
      plt.plot([Xs[i, 0]], [Xs[i, 1]], 'r.')
    elif ys[i] == 1:
      plt.plot([Xs[i, 0]], [Xs[i, 1]], 'b.')
    elif ys[i] == 2:
      plt.plot([Xs[i, 0]], [Xs[i, 1]], 'g.')
    elif ys[i] == 3:
      plt.plot([Xs[i, 0]], [Xs[i, 1]], 'y.')
  plt.show()

plot_with_colors(X2d, y)
```



(a) Implement the standard k-means algorithm. Please complete the function **kmeans** defined below. You are NOT allowed to use any existing code of **kmeans** for this problem.

In [3]:

```
def kmeans(X, k = 4, max_iter = 500, random_state=0):
    """
    Inputs:a
      X: input data matrix, numpy array with shape (n * d), n: number of data po
ints, d: feature dimension
      k: number of clusters
      max_iters: maximum iterations
    Output:
      clustering label for each data point
    """
    assert len(X) > k, 'illegal inputs'
    np.random.seed(random_state)

    # randomly select k data points as centers
    idx = np.random.choice(len(X), k, replace=False)
    centers = X[idx]

    # please complete the following code:

    from scipy.spatial import distance
    for i in range(max_iter):
        H = distance.cdist(X, centers, 'euclidean')
        labels = np.argmin(H, axis=1)
        centers = np.array([np.mean(X[labels==i], axis=0) for i in range(k)])

    return labels
```

(b) Run your **kmeans** function on the dataset (of the top two PCA components given by array X2d). Set the number of clusters to 4. Visualize the result by coloring the 2D points in (a) according to their **clustering labels** returned by your **kmeans** algorithm. Because **kmeans** is sensitive to initialization, repeat your **kmeans** at least 5 times with different random initializations and show the plot of each initialization.

To quantitatively evaluate the clustering performance, we evaluate the $\textit{unsupervised clustering accuracy}$, which can be written as follows, $$ \text{accuracy} = \max_{\mathcal M} \frac{\sum_{i=1}^{n} \mathbb{I}(y_i = \mathcal M(z_i))}{n}, n = 1000, $$ where y_i is the ground-truth label, z_i is the cluster assignment produced by the algorithm, and $\mathcal M$ ranges over all possible one-to-one mapping between clusters and labels and $\mathbb{I}(x)$ is a indicator function ($\mathbb{I}(x) = 1 \text{if}~x=1; \text{otherwise} ~0$). Please use the **accuracy_score** function defined below to calculate the accuracy. Report the best clustering accuracy you get out of 10 random initializations.

In [4]:

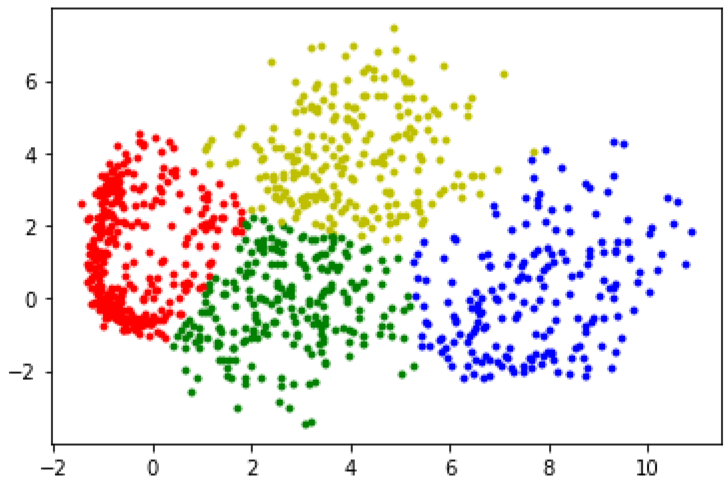
```
# do not change
def accuracy_score(y_true, y_pred):
    """
    Calculate clustering accuracy.
        y_true: true labels, numpy.array with shape `(n_samples,)`
        y_pred: predicted labels, numpy.array with shape `(n_samples,)`
    # Return
        accuracy, in [0,1]
    """
    y_true = np.squeeze(y_true)
    y_pred = np.squeeze(y_pred)
    assert y_true.shape == y_pred.shape, 'illegal inputs'

    y_true = y_true.astype(np.int64)
    assert y_pred.size == y_true.size
    D = max(y_pred.max(), y_true.max()) + 1
    w = np.zeros((D, D), dtype=np.int64)
    for i in range(y_pred.size):
        w[y_pred[i], y_true[i]] += 1
    from scipy.optimize import linear_sum_assignment
    row_ind, col_ind = linear_sum_assignment(w.max() - w)
    return sum([w[row_ind[i], col_ind[i]] for i in range(len(row_ind))]) * 1.0 /
y_pred.size
```

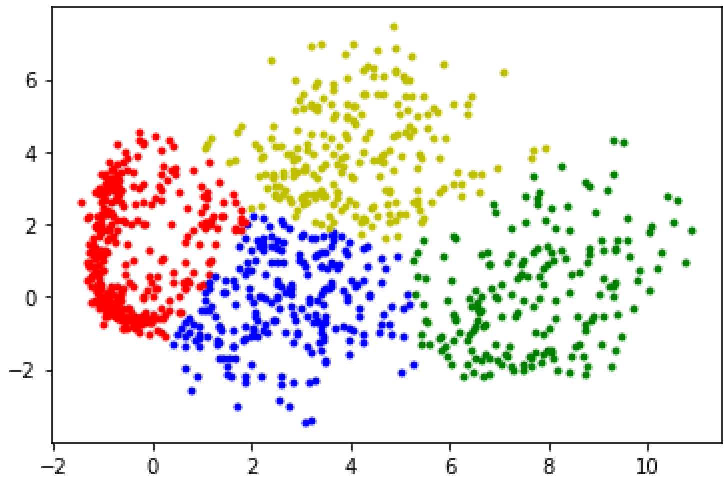
In [5]:

```
random_seeds = [1, 19, 1777, 51, 29, 314, 272, 2023, 13, 17, 1993]
for rs in random_seeds:
    labels = kmeans(X2d, random_state=rs, max_iter=10) # set max_iter=10 in orde
r to see differences with
                                                        # different random initia
lization
    print(round(accuracy_score(labels, y)*100, 4))
    plot_with_colors(X2d, labels)
```

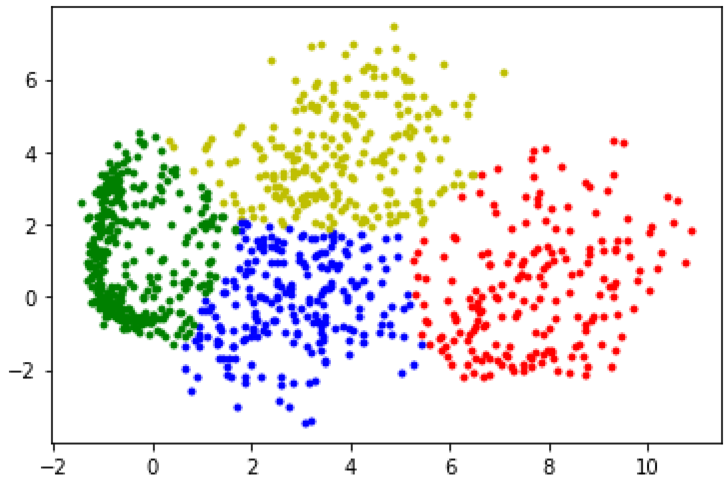
```
74.4
```



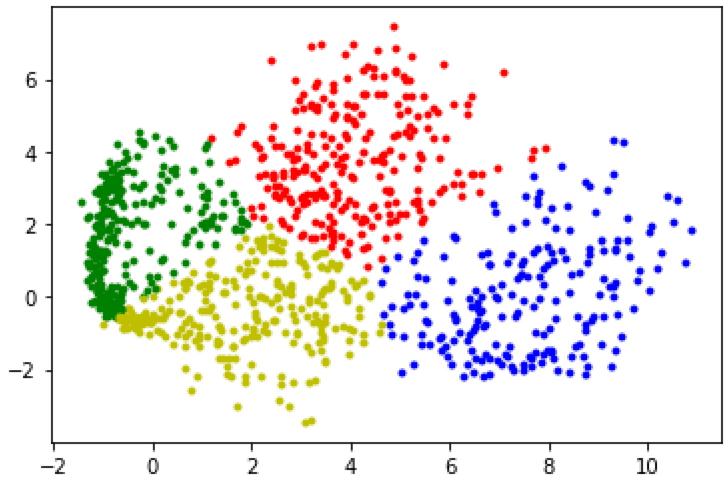
```
74.3
```



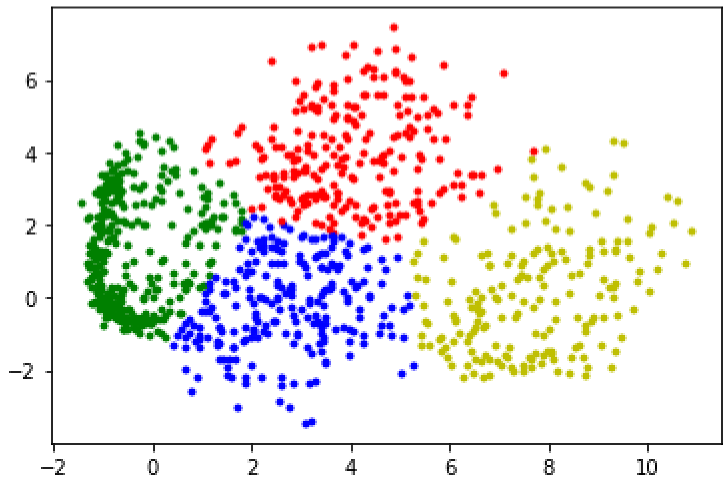
```
75.7
```



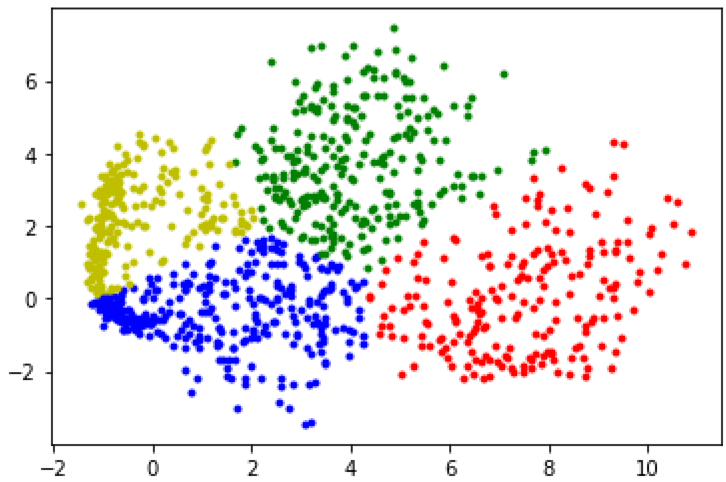
```
70.3
```



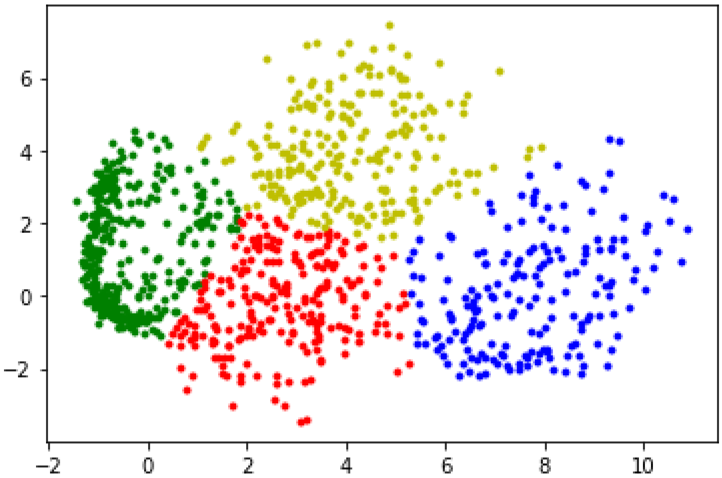
74.4



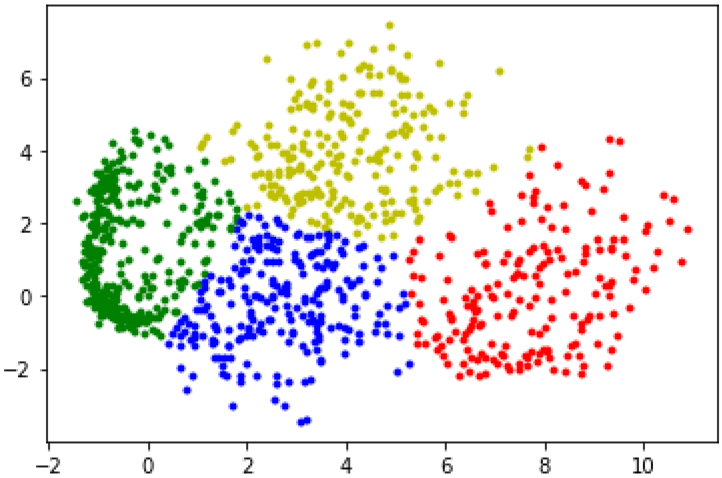
65.8



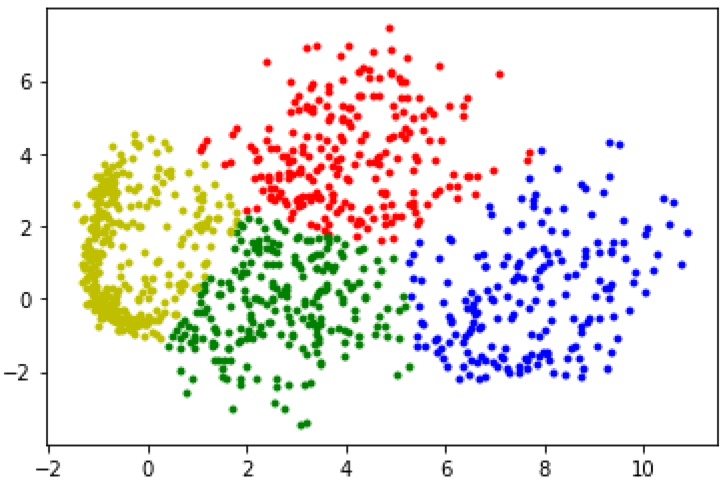
74.3



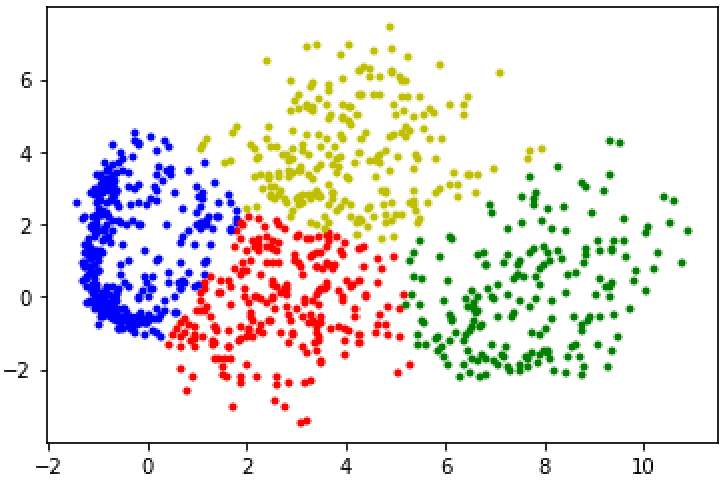
74.3



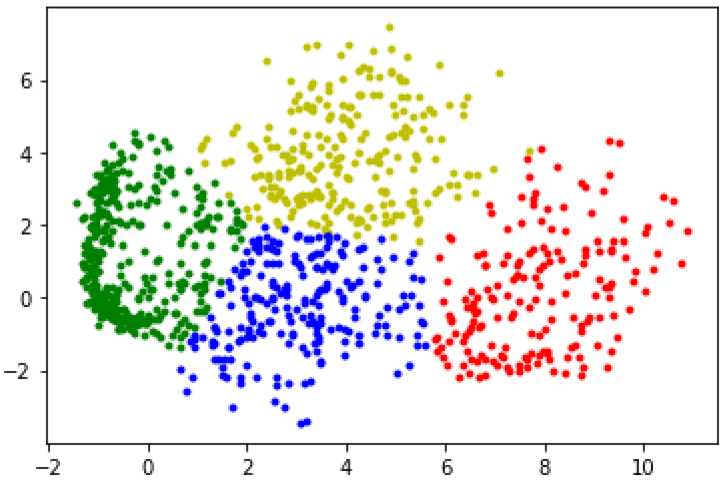
74.3



74.4




```
73.8
```



We have been testing $\textit{k}$-means on the top two principal components for the purpose of visualization. Please run $\textit{k}$-means on the (784 dimensional) original image dataset (again using 4 clusters). Try at least 10 different random initializations and report the best accuracy as above.

In [10]:

```
best_acc = -1
for rs in tqdm(random_seeds):
    labels = kmeans(X, random_state=rs)
    acc = accuracy_score(labels, y)
    if acc > best_acc:
        best_acc = acc
print(f"Best accuracy on the MNIST dataset using the k-means algorithm is {best_
acc*100:.2f}")
```

```
100%|██████████| 11/11 [00:29<00:00,  2.70s/it]

Best accuracy on the MNIST dataset using the k-means algorithm is 8
6.10
```